KI-gestütztes Reverse Requirements Engineering bei Legacy-Software

Masterarbeit an der Hochschule Neu-Ulm

Masterarbeit

Master of Business Administration

Autor: Christoph Schwörer

Betreuer: Prof. Dr. Daniel Schallmo

Abgabedatum: XX 2026

University of Applied Sciences Neu-Ulm

Eigenständigkeitserklärung

Hiermit versichere ich, die vorliegende Masterarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet zu haben. Alle wörtlich oder sinngemäß übernommenen Textstellen sind als solche gekennzeichnet.

Neu-Ulm, _____

Unterschrift: _____

Abstract (ca. 1 Seite)

Zusammenfassung der Arbeit.

Inhaltsverzeichnis

Abstract (ca. 1 Seite)
Zusammenfassung der Arbeit.

1 Einleitung (ca. 8 Seiten)

1.1 Ausgangssituation und Motivation
In den vergangenen Jahren hat die digitale Transformation mittelständische Softwareanbieter gezwungen, ihre Produkte neu zu bewerten. Betroffen sind vor allem Systeme die über lange Jahre nur in Windows-Umgebungen vertrieben wurden. Diese stoßen bei Cloud-, Web- und Mobile-Szenarien an technische sowie organisatorische Grenzen und fallen in der technologischen Schuld immer weiter Zurück. Eine technologische Weiterentwicklung ist nicht möglich und an einer Neuentwicklung führt oft kein Weg vorbei. Dokumentierte Anforderungen und Code sind allerdings selten. Das meiste Wissen steckt implizit im Code oder in der Köpfen der verlbeibenden Entwickler.

Die c-entron GmbH in Ulm ist von diesem Szenario voll Betroffen. Das Unternehmen betreibt seit über zwanzig Jahren eine Windows-basierte ERP-Suite für IT-Systemhäuser. Die Lösung deckt Auftragsabwicklung, Lager, Fakturierung und Projektabrechnung ab, ist aber eng mit der bisherigen Client/Server-Architektur gekoppelt. Kunden erwarten zwischenzeitlich aber plattformunabhängige Oberflächen, Self-Service-Funktionen und flexible Betriebsmodelle wie z.B. SaaS (Software as a Service). Die bestehende Anwendung ist aber in ihrer Skalierung, Deployment und Abrechnung limitiert. Eine Migration auf eine webbasierte Plattform ist somit zwingend erforderlich.

Eine einfache Neuimplementierung auf Basis vorhandener Anfoderungs oder Code Dokumentation ist aber aus oben geschilderten Gründen nicht einfach möglich. Die Herausforderung die sich stellt ist mit möglichst geringem Aufwand eine vollständige Beschreibung für ein vollständiges ERP System zu erarbeiten. Eine Manuelle auswertung des Codes oder Obfläche auf Funktionalitäten ist aufgrund der extremen Komplexität, geschuldet der langjährgien Weiterententwicklung, nur mit sehr hohem Personalaufwand möglich und daher nicht realsierbar.

In den letzen Jahren hat sich hierzu nun ein neues Instrumentarium etabliert. Large Language Models wie Chat GPT-5 oder Claude.ai können durch agentische CLIs (Codex, Claude Code) große Mengen an Quellcode analysieren, Anforderungen erarbeiten und textuell beschreiben. Damit entsteht die Chance, fehlende Anforderungsdokumentationen zumindest teilweise aus dem Code heraus zu rekonstruieren. Die praktische Nutzung dieses Potenzials ist bislang kaum erforscht. Diese Arbeit adressiert dies und untersucht, wie KI-gestützte Verfahren für eine systematische Anforderunganalyse eingesetzt werden können.

1.2 Problemstellung
Für das ERP Software Produkt der c-entron fehlen strukturierte und dokumentierte Requirements. Die Analyse der bestehenden Codebasis ist zeitintensiv, ressourcenintensiv und anfällig für Insel- und Metawissen. Daraus ergeben sich mehrere Risiken:

- Re-Implementationsfehler: Edge Cases, Workarounds und kundenindividuelle Anpassungen sind nur im Code sichtbar. Ohne vollständige Erfassung drohen Funktionsverluste nach der Migration. Zeitgleich sind Workaround of Symptom einer mangelhaften Erfassung der Anfoderungen bei originalen Implementierung und ein zeichen fehlender Weitsicht.
- Technische Schuld: Entwickler investieren viel Zeit in das Verständnis historischer Strukturen, statt aktiv an der neuen Plattform zu arbeiten. Veraltete Muster werden unreflektiert übernommen. Neue Mitarbeiter sind auch nicht bewandt in alten technolgien und es fehlt das Verständnis für historische Zwänge und Zusammenhänge.

- Implizites Wissen: Domänenwissen liegt bei wenigen langjährigen Mitarbeitenden. Personalwechsel führen zu Wissensverlust und Verzögerungen. Gleichzeitig führt die langjährige arbeit mit dem bestehenden System zu eingeschränkter Offenheit beim design neuer Lösungen ("Das haben wir schon immer so umgestzt").
- Komplexität der Codebasis: Verschachtelte Abhängigkeiten, unterschiedliche Stile und technologiebedingte Zwänge erschweren eine modulare Anforderungsableitung.
- Fehlende Traceability: Ohne Zuordnung zwischen Code und Geschäftsprozess fehlt die Grundlage für Priorisierung, Testkonzeption und spätere Wartung. Große Teile des Codes sind auch generisch und lassen sich nicht einer konkreten Anforderung zuordnen, wie zum Beispiel das anzeigen eine Tabelle. Konkrete Datenflüsse lassen sich hier nur am laufenden System beobachten was die Analyse nochmal um eine Größenordnung Resourcenintensiver macht.

Eine rein manuelle Rekonstruktion aller Anforderungen wäre wirtschaftlich kaum tragbar. Deshalb soll geprüft werden, ob KI-gestützte Verfahren Requirements so extrahieren können, dass sie als belastbare Basis für die Modernisierung dienen.

1.3 Zielsetzung

Diese Arbeit verfolgt das Ziel, ein vollständiges Vorgehen für KI-gestütztes Reverse Requirements Engineering im Umfeld eines mittelständischen ERP-Herstellers zu entwickeln und zu bewerten. Die Teilziele lauten:

- Entwicklung eines Prozessmodells, das Vorbereitung, Analyse, Validierung und Übergabe strukturiert.
- Evaluation aktueller LLMs hinsichtlich Kontextfenster, Codeverständnis, Steuerbarkeit, Kosten und Datenschutz.
- Durchführung und Vergleich von drei Claude-Code basierten Versuchen mit unterschiedlicher Tooling-Tiefe (Prompt-only, Agenten, Agenten+MCP).
- Definition eines Evaluationsrahmens mit quantitativen und qualitativen Kriterien (Vollständigkeit, Verständlichkeit, Redundanzfreiheit, Aufwandseinsparung).
- Integration von Stakeholder-Wissen durch Interviews, um die Qualität der KI ergebnisse zu bewerten und nicht direkt aus dem Code ableitbare Anforderungen zu ergänzen .

1.4 Forschungsleitfragen

Die Zielsetzung wird über vier Forschungsleitfragen strukturiert:

1. **Einsatz von LLMs im Reverse Requirements Engineering:** Welche Prozessschritte, Steuerungsmechanismen und Kontrollpunkte sind notwendig, um LLMs reproduzierbar einzusetzen?
2. **Kombination von KI-Analyse und Stakeholder-Input:** Welche funktionalen und nicht-funktionalen Anforderungen lassen sich aus Code extrahieren, und welche Informationen müssen über Interviews ergänzt werden?
3. **Qualitätsbewertung der generierten Requirements:** Wie beurteilen Fachexperten Vollständigkeit, Verständlichkeit, Nützlichkeit und Aufwandseinsparung der KI-Ergebnisse?
4. **Chancen und Grenzen des Ansatzes:** Welche Effizienzgewinne sind realistisch, wo liegen technische oder organisatorische Limitierungen, und welche Risiken (z. B. Halluzinationen, Datenschutz) müssen adressiert werden?

1.5 Aufbau der Arbeit

Die Arbeit ist in acht Kapitel gegliedert:

1. **Einleitung:** Kontext, Problemstellung, Ziele und Forschungsfragen.

2. **Theoretische Grundlagen:** Requirements Engineering, Reverse Engineering, Large Language Models sowie Qualitätssicherungskriterien.
3. **Fallstudie c-entron GmbH:** Unternehmensprofil, Produktarchitektur, Migrationsdruck und Rahmenbedingungen.
4. **Konzeption und methodisches Vorgehen:** Prozessmodell, Technologieauswahl, Stakeholder-Einbindung und Datenbasis.
5. **Ergebnisse:** Vollständige Ergebnisdarstellung der drei Versuche inkl. Artefaktlisten und beispielhafter Requirements/Use Cases aus den Ergebnisverzeichnissen.
6. **Evaluation:** Vorgehen, Metriken, Ergebnisse und Expertenfeedback.
7. **Diskussion:** Interpretation der Resultate, Limitationen und Implikationen für Forschung und Praxis.
8. **Fazit und Ausblick:** Zusammenfassung, Beantwortung der Forschungsfragen und Perspektiven für weitere Arbeiten.

2 Theoretische Grundlagen (ca. 12 Seiten)

Dieses Kapitel beschreibt die theoretischen Grundlagen, die für die Konzeption und Bewertung eines KI-gestützten Reverse Requirements Engineering in Legacy-Umgebungen benötigt werden. Zunächst werden zentrale Themen des Requirements Engineerings sowie die Idee des reverse Requirements Engineerings auf Basis bestehender Systeme eingeordnet. Anschließend werden Large Language Models und deren Einsatz im Software Engineering beschrieben. Inklusive typischer Leistungsgrenzen und Absicherungsmechanismen. Abschließend werden Grundlagen der Legacy-Modernisierung sowie etablierte Migrationsstrategien zusammengefasst, um den Kontext der Fallstudie und die Zielrichtung einzuordnen.

2.1 Requirements Engineering und Reverse Requirements Engineering

2.1.1 Begriff und Zielsetzung des Requirements Engineering

der Begriff Requirements Engineering (RE) umfasst die systematische Erhebung, Analyse, Spezifikation, Validierung und Verwaltung von Anforderungen an ein System über dessen Lebenszyklus. In Standards und Lehrwerken (IEEE, 1998; ISO/IEC/IEEE, 2018) wird Reqirements Engineering als eigenständiger Prozess verstanden, der sowohl fachliche Ziele (z. B. unterstützte Geschäftsprozesse) als auch technische und organisatorische Randbedingungen (z. B. Sicherheitsvorgaben, Betriebsmodelle) in überprüfbare Aussagen überführt.

Im Kern adressiert das Requirments Engineering zwei Themen:

- **Kommunikation zwischen Domäne und Technik:** Anforderungen müssen fachlich verständlich und gleichzeitig so präzise sein, dass sich daraus eine Software-Architektur ableiten lässt, die implementiert, getestet und geändert werden kann.
- **Umgang mit Unsicherheit und Wandel:** Anforderungen sind zu Projektbeginn selten vollständig. Requriements Engingeeing ist daher nicht nur Dokumentation, sondern auch ein iterativer Klärungs- und Abstimmungsprozess.

Ein etablierter Ansatz zur Strukturierung von diversen Sichtweisen ist das Viewpoint-Konzept (Kotonya & Sommerville, 1996), bei dem Anforderungen aus unterschiedlichen Perspektiven modelliert und anschließend konsolidiert werden .

Für diese Arbeit ist die Perspektivenorientierung relevant, weil implementierter Code typischerweise keine expliziten Stakeholder-Sichten enthält. Für eine Migration auf Basis eines Reverse Engeneering Ansatzes sind diese aber relevant für die implementierung und architekturlle Entscheidungen (z. B. Nutzerrollen, kundenspezifische Varianten, regulatorische Vorgaben).

2.1.2 Arten von Requirements und Qualitätskriterien

In der Literatur wird häufig zwischen funktionalen Anforderungen (Was soll das System tun?) und Qualitäts- bzw. nicht-funktionalen Anforderungen (Welche Eigenschaften und Randbedingungen gelten?) unterschieden. Die Praxis zeigt jedoch, dass diese Trennung nicht immer scharf ist. Eigenschaften können sowohl als Systemverhalten (z. B. „Audit-Log erzeugen“) als auch als Qualitätsziel (z. B. „Nachvollziehbarkeit”) formuliert werden (Glinz, 2007). Für das Reverse Requirements Engineering ist diese Unschärfe besonders relevant, weil Quellcode meist Verhalten konkretisiert, Qualitätsziele aber häufig implizit bleiben (z. B. Performance-Workarounds, Sicherheitsannahmen).

Für die Qualität einzelner Requirements gibt es etablierte Standards. (ISO/IEC/IEEE, 2018) nennt unter anderem Eindeutigkeit, Konsistenz, Vollständigkeit, Verifizierbarkeit und Nachvollziehbarkeit als zentrale Eigenschaften. (IEEE, 1998) formuliert ähnliche Prinzipien für Software Requirements Specifications, mit stärkerem Fokus auf Dokumentstruktur und Lesbarkeit.

Für die Bewertung von KI-extrahierten Requirements sind drei Kriterien maßgeblich relevant:

- **Verifizierbarkeit:** Ein Requirement ist so formuliert, dass ein Test oder eine Prüfmethode ableitbar ist (z. B. Messkriterium, Akzeptanzbedingung).
- **Eindeutigkeit:** Formulierungen vermeiden Mehrdeutigkeiten und definieren Begriffe, die in der Domäne unterschiedlich interpretiert werden können
  (z.B. „Das System soll Aufträge schnell verarbeiten“ vs. „Das System soll einen Auftrag innerhalb von 2 Sekunden validieren und bestätigen”)
- **Nachvollziehbarkeit (Traceability):** Es ist erkennbar, aus welchem Requirement das Artefakt (Code, Konfiguration, Datenbank, Ticket, Interview) abgeleitet wurde.

Nicht funktionale Anforderungen (z.B. Qualitätsanforderungen) bedürfen einer besonderen Betrachtung, weil sie über die reine Funktionsgleichheit hinaus die Zielarchitektur bestimmen. Glinz (2008) argumentiert, dass Qualitätsanforderungen risikobasiert und wertorientiert priorisiert werden sollten. Für Legacy-Migrationen ist dies nachvollziehbar: Ein „vollständiges“ Requirements-Set ist praktisch schwer erreichbar, gleichzeitig sind bestimmte Non-Functional Requirements (z. B. Datenschutz, Verfügbarkeit, Rollout-Fähigkeit) hochkritisch, weil sie Architekturentscheidungen dominieren.

Für die inhaltliche Strukturierung von Qualitätsanforderungen ist das Qualitätsmodell ISO/IEC 25010:2011 verbreitet, das Qualitätsmerkmale wie Performance-Effizienz, Zuverlässigkeit, Sicherheit oder Wartbarkeit systematisch ordnet. Für Reverse Requirements Engineering ist dies hilfreich, weil aus Code häufig nur Teilaspekte sichtbar werden (z. B. Caching-Mechanismen als Hinweis auf Performance-Annahmen), während andere Qualitätsziele (z. B. „Maintainability“) eher indirekt über Architekturentscheidungen und Entwicklungspraktiken wirksam werden (ISO/IEC, 2011).

Die Relevanz saubere formulierter Requirements zeigt sich auch in der Risikoperspektive. B et al. (2001) beschreiben Requirements Engineering als primäres Risiko, wenn Anforderungen unklar, instabil oder unvollständig sind.

Für diese Arbeit folgt daraus, dass KI-gestütztes Reverse-Requirements-Engineering nicht nur „mehr Text“ erzeugen darf, sondern gezielt die Risiken der Unklarheit und der Fehlinterpretation reduzieren muss.

2.1.3 Spezifikationsformen und Grad der Formalisierung

Requirements werden in unterschiedlichen Repräsentationsformen dokumentiert. Standards wie IEEE 830-1998 und ISO/IEC/IEEE 29148:2018 fokussieren auf strukturierte Spezifikationen (z. B. SRS)

und definieren typische Kapitel (Zweck, Systemkontext, funktionale Anforderungen, Schnittstellen, Qualitätsanforderungen, Annahmen). Zudem existieren weniger formale Formen wie User Stories, Use-Case-Beschreibungen oder Backlog-Einträge, die in agilen Settings verwendung finden. (IEEE, 1998; ISO/IEC/IEEE, 2018).

Für Reverse Requirements Engineering sind zwei Punkte entscheidend:

- **Form** beeinflusst Interpretierbarkeit: Eine kurze User Story („Als Nutzer möchte ich …“) ist leicht verständlich, transportiert aber weniger Randbedingungen, Datenregeln oder Fehlerfälle. Eine SRS-Formulierung kann präziser sein, erfordert aber mehr Kontext und Definitionen.
- **Grad** der Formalisierung beeinflusst Prüfbarkeit: Je stärker Requirements mit Akzeptanzkriterien, Beispielen oder Messgrößen verknüpft sind, desto einfacher sind Reviews und Tests. Pohl (2010) betont Anforderungen-Validierung als eigene Disziplin.

Für diese Arbeit wir daher ein hybrider Stil gewählt: Requirements werden als kurze, klare Soll-Aussagen formuliert und jeweils um Kontext (Akteur/Prozess), Randbedingungen (Vorbedingungen, Datenobjekte) und mindestens eine Prüfidee ergänzt.

2.1.4 Traceability als Verbindung zwischen Code und Requirement

Traceability bezeichnet die Verknüpfung von Requirements und Artefakten, wie z.B. Code oder Test. Gotel & Finkelstein (n.d.) bezeichnen Traceability als wiederkehrendes Problem, vor allem bei unterschiedlichen Arten von Artefakten.

Beim Reverse Requirements Engineering ist Traceability nicht nur ein „Nice-to-have“, sondern eine Grundvoraussetzung.
Ein Requirement lässt sich gegen konkrete Codeausschnitte oder Laufzeitbeobachtungen prüfen da da Requirement ja aus dem Code selbst entsteht.

In Legacy-Systemen ist die Ursprüngliche Traceability typischerweise unvollständig, falls überhaupt vorhanden: Hinweise finden sich in Commit-Messages, Branch-Namen, Datenbankskripten, Konfigurationsdateien, UI-Texten oder in impliziten Konventionen. Der Anspruch dieser Arbeit besteht daher nicht darin, „perfekte“ Traceability wiederherzustellen, sondern eine minimal belastbare, reproduzierbare Verknüpfung zwischen extrahierten Requirements und Artefakten zu etablieren.

2.1.5 Abgrenzung von Reverse Engineering zu Reverse Requirements Engineering

Reverse Engineering wird klassisch als Analyseprozess verstanden, der aus einem bestehenden System Wissen über Struktur, Verhalten und Designentscheidungen rekonstruiert. Chikofsky & Cross (1990) prägen hierfür die Bennung und grenzen Reverse Engineering von Reengineering sowie Design Recovery ab. Für Requirements-nahe Fragestellungen ist hier relevant, dass Reverse Engineering nicht automatisch auch Requirements liefert, sondern erstmal nur technische Fakten (z. B. Abhängigkeiten, Datenflüsse, Zustandsautomaten).

Reverse Requirements Engineering (RRE) fokussiert sich dagegen auf die rückwärtsgerichtete Gewinnung von Requirements aus bestehenden Artefakten. Dabei kann das Ziel unterschiedlich interpretiert werden:

- **Rekonstruktion eines Soll-Zustands:** Welche fachlichen Anforderungen werden durch die aktuelle Implementierung implizit erfüllt? Was war das ursprüngliche Ziel der Imepementierung?
- **Rekonstruktion eines Ist-Zustands:** Welche Funktionen und Regeln sind dagegen tatsächlich implementiert?

Gerade im Legacy Umfeld ist diese Unterscheidung entscheidend. Die Codebasis enthält oft historisch entstandene Workarounds oder kundenspezifische Anpassungen. Ohne zusätzliche Validierung besteht das Risiko, dass RRE den Ist-Zustand als Soll-Zustand fehlinterpretiert.

Frühe Ansätze zur Brücke zwischen Reverse Engineering und Requirements liefern beispielsweise Yu et al. (n.d.) mit „RETR: Reverse Engineering to Requirements“. Der Beitrag betont, dass Requirements-Rückgewinnung eine methodische Kette aus Artefaktsichtung, Strukturierung und Validierung benötigt.

Methodisch lassen sich dabei grob zwei Analysestränge unterscheiden:

- **Statische Analyse:** Ableitung von Struktur- und Datenflussinformationen aus Code und Artefakten ohne Ausführung (z. B. Abhängigkeiten, SQL-Statements, Aufrufketten). Statische Analyse ist skaliert gut, erkennt aber nicht zuverlässig Laufzeitbedingungen (z. B. Feature Flags, Konfigurationsvarianten).
- **Dynamische Analyse:** Beobachtung von Laufzeitverhalten durch Logging, Tracing oder instrumentierte Tests (z. B. welche Regeln bei bestimmten Eingaben greifen). Dynamische Analyse ist näher am realen Verhalten, benötigt aber reproduzierbare Szenarien und Testdaten.

Reverse Requirements Engineering in einem Migrationsprojekt profitiert typischerweise von einer Kombination beider Stränge. Ohne dynamische Belege steigt das Risiko, dass nicht offensichtliche Bedingungen (z. B. kundenspezifische Schalter) übersehen werden; ohne statische Analyse bleibt die Abdeckung häufig zu gering.

Da eine Dynamische Analyse durch ein LLM mit den gegebenen technischen Möglichkeiten derzeit unpraktikabel ist, fokussiert sich diese Arbeit auf die statische Analyse von Artefakten, ergänzt manuell erstellte Artefakte zur Laufzeit (z.B. Screenshots).

2.1.6 Typische Methodenkette für Requirements-Rückgewinnung aus Code

Aus Sicht dieser Arbeit lässt sich Reverse Requirements Engineering einer Legacy-Codebasis als wiederholbarer Ablauf darstellen. Die konkrete Implementierung hängt vom System und den verfügbaren Artefakten ab, die grundlegenden Schritte sind jedoch weitgehend stabil:

1. **Scope und Domänenabgrenzung:** Auswahl relevanter Module, Datenobjekte und Prozesse (z. B. Auftragsabwicklung, Fakturierung).
2. **Artefakterhebung:** Quellcode, Konfiguration, UI-Texte, Datenbankschemata, Schnittstellenbeschreibungen, Change-Historie.
3. **Technische Analyse:** Struktur- und Abhängigkeitsanalyse, Identifikation von Kernkomponenten, Regeln und Integrationspunkten.
4. **Semantische Interpretation:** Ableitung fachlicher Aussagen aus technischen Implementierungen (z. B. Statusübergänge, Berechtigungsprüfungen).
5. **Formalisierung als Requirements:** Überführung in klare, testbare Anforderungen mit Kontext (Akteur, Vorbedingung, Ergebnis).
6. **Traceability-Anreicherung:** Verknüpfung jedes Requirements mit Belegen (Datei, Klasse, Methode, SQL-Statement, UI-String).
7. **Validierung:** Review durch Fachexperten und Abgleich mit Laufzeitverhalten, Tickets oder Kundenwissen.

In der Praxis unterscheiden sich Artefakte darin, wie direkt sie fachliche Aussagen stützen. Quellcode, der eine Regel hart erzwingt (z. B. „Update nur bei Status X“), ist als Beleg stärker als Kommentare oder UI-Texte, die lediglich Absichten ausdrücken. Für eine belastbare Requirementsbasis ist es daher sinnvoll, Belege zu klassifizieren und die Aussagekraft zu kennzeichnen, beispielsweise:

- **Primärbelege:** Durchgesetzte Regeln im Code oder in Datenbankconstraints (z. B. Statusmaschinen, Validierungslogik, Berechtigungschecks).
- **Sekundärbelege:** Indirekte Hinweise wie UI-Labels, Fehlermeldungen, Report-Layouts, Mappingtabellen oder Konfigurationsschalter.
- **Kontextbelege:** Ticketbeschreibungen, Commit-Messages oder Interviewaussagen, die Motivation und Ausnahmen erklären, aber nicht zwingend im Code sichtbar sind.

Diese Einteilung dient der Risikobewertung: Requirements, die überwiegend auf Sekundär- oder Kontextbelegen beruhen, sind anfälliger für Fehlinterpretation und sollten priorisiert validiert werden. Datenbankschemata und SQL-Statements sind häufig besonders aussagekräftig, weil sie Domänenobjekte, Kardinalitäten und Geschäftsregeln (z. B. referentielle Integrität, historisierte Tabellen) abbilden.

Für diese Arbeit werden in der Schrittkette lediglich Punk 1 und 7 manuell durchgeführt, während die Schritte 2-6 durch KI-gestützte Analyse automatisiert werden sollen.

2.2 Large Language Models im Software Engineering

2.2.1 Künstliche Intelligenz, Machine Learning und Einordnung von LLMs

Die Einordnung von Large Language Models folgt einer hierarchischen Begriffsstruktur, die sich in vier Ebenen gliedern lässt (vgl. Figure 1):

Künstliche Intelligenz (KI) ist der Oberbegriff für Verfahren, die Aufgaben bearbeiten, welche in der Praxis typischerweise kognitive Fähigkeiten erfordern (z. B. Klassifikation, Planung, Sprachverarbeitung) (Bishop, 2006).

Machine Learning (ML) ist ein Teilgebiet der KI, das Modelle aus Daten lernt, anstatt Regeln vollständig manuell zu spezifizieren (sogennante Expertensysteme). in der Praxis wird zwischen Systemen mit überwachtem Lernen (mit Zielwerten), unüberwachtem Lernen (ohne Zielwerte, System erkennt selbst eine Struktur) und Reinforcement Learning (Lernen über Rückmeldung, z.B. Ergebnis Richtig / Falsch) unterschieden (Bishop, 2006; Goodfellow et al., 2016). In der heutigen Welt sind Neuronale Netze die dominierende ML-Architektur, insbesondere für unstrukturierte Daten wie Text und Code.

Deep Learning ist wiederum ein Teilbereich von ML, der neuronale Netze mit vielen Parametern und vielen (tiefen) Verarbeitungsebenen nutzt, um geeignete Repräsentationen aus Rohdaten zu lernen. Charakteristisch ist, dass Merkmalsextraktion und Modellanpassung gemeinsam über Optimierung (typischerweise Gradientenverfahren) erfolgen.

Large Language Models (LLMs) sind eine spezielle Ausprägung von Neuronalen Netzen, die auf sehr großen Text- und Codemengen vortrainiert werden. Zudem verfügen sie über sehr große Mengen (>Milliarden) von Eingabeparametern (Tokens) und viele Schichten (>100). In der aktuellen Modellgeneration dominiert die Transformer-Architektur (Vaswani et al., 2017), deren Funktionsweise in den folgenden Abschnitten erläutert wird.

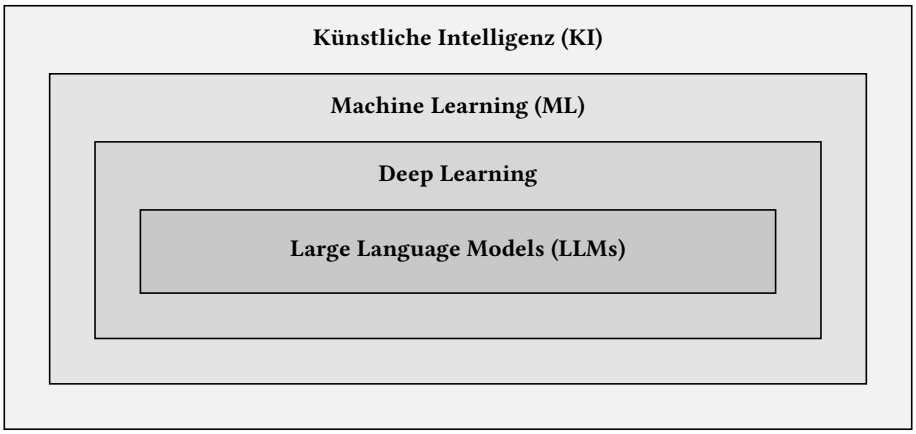

Figure 1: Hierarchische Einordnung: KI ⊃ ML ⊃ Deep Learning ⊃ LLMs.

Neuronale Netze lassen sich vereinfacht als parametrisierte Funktionsketten aus Schichten (Layern) beschreiben. Dabei geben die Schichten die Eingaben in zunehmend abstrakte Repräsentationen jeweils in die nächste Schicht weiter. Das Training erfolgt über eine Zielfunktion und Gradientenberechnung. In der Praxis geschieht dies meist über Backpropagation und Varianten des Gradientenabstiegs (Goodfellow et al., 2016).

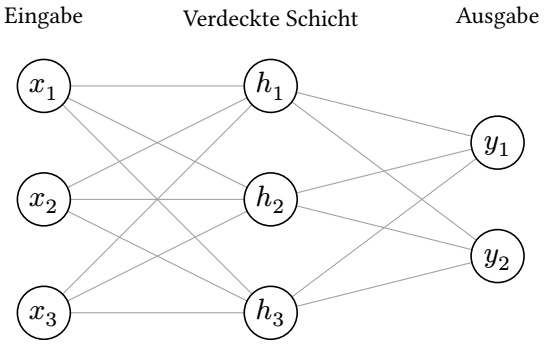

Figure 2: Schematische Darstellung eines vollständig verbundenen Feedforward-Netzes.

Ein einzelnes Neuron lässt sich als Transformation mit nachgeschalteter Aktivierungsfunktion formulieren:

$$z = \sum_{i=1}^{d} w_i x_i + b, \quad a = \varphi(z)$$

Typische Aktivierungsfunktionen ($a = \varphi(z)$) sind die Sigmoid-Funktion, der hyperbolische Tangens und ReLU (Goodfellow et al., 2016):

Sigmoid
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

tanh
$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

ReLU
$$\mathrm{ReLU}(z) = \max(0, z)$$

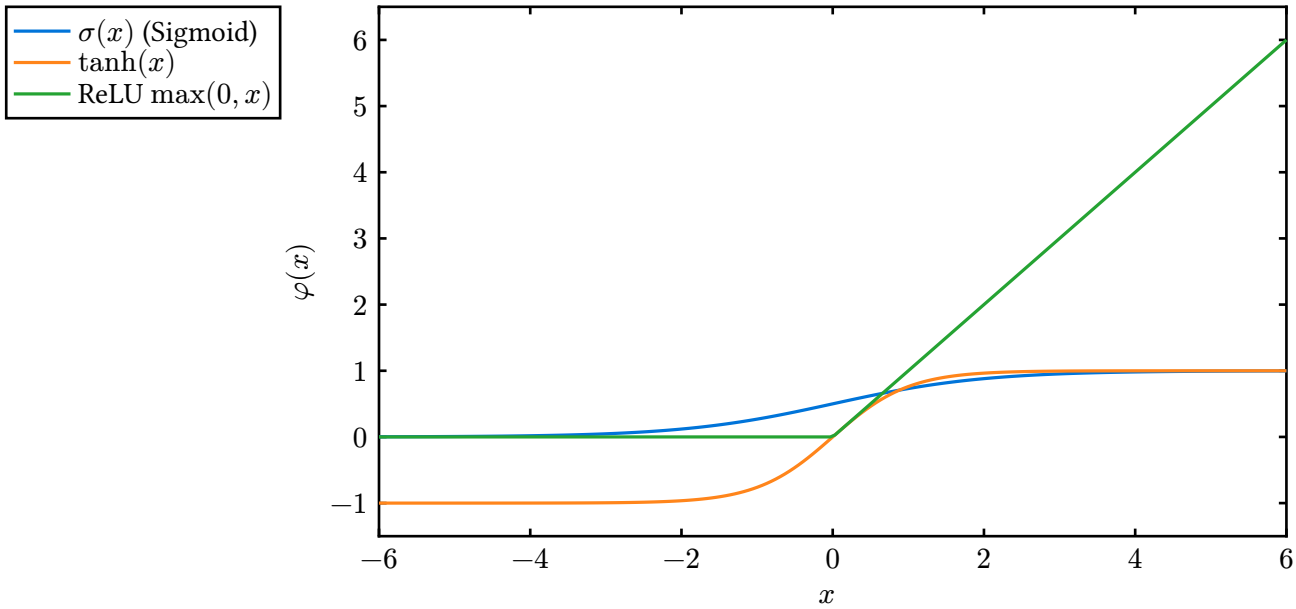

Figure 3: Beispielhafte Aktivierungsfunktionen (Sigmoid, tanh, ReLU).

Die Optimierung erfolgt üblicherweise iterativ. Für Gradientenabstieg gilt in kompakter Form:

$$\theta^{t+1} = \theta^{t} - \eta \nabla_{\theta} \mathcal{L}(\theta^{t})$$

LLMs unterscheiden sich von klassischen neuronalen Netzen nicht nur durch ihre Modellgröße, sondern vor allem durch ihre Zielsetzung: Sie sind Sprachmodelle – Systeme, die Wahrscheinlichkeiten über Tokenfolgen schätzen und dadurch Texte fortsetzen, bewerten oder erzeugen können. Der Verarbeitungsablauf folgt dabei einem wiederkehrenden Muster. Zunächst wird die Texteingabe in Token zerlegt. Token sind die kleinsten Verarbeitungseinheiten des Modells; je nach Tokenisierungsverfahren können dies Wörter, Wortteile oder einzelne Zeichen sein. Der Satz „Das System prüft den Status“ wird beispielsweise in die Folge [„Das”, „System“, „prüft”, „den“, „Status”] überführt. Für Quellcode gilt dasselbe Prinzip: Identifier, Schlüsselwörter und Operatoren werden in Subword-Einheiten aufgeteilt (Goodfellow et al., 2016).

Die resultierende Tokenfolge bildet den Kontext des Modells. Der Kontext umfasst alle Token, die das Modell in einem Verarbeitungsschritt gleichzeitig berücksichtigt. Aktuelle Modelle besitzen Kontextfenster von mehreren tausend bis über eine Million Token. Eingaben, die dieses Fenster überschreiten, müssen segmentiert oder komprimiert werden – für umfangreiche Legacy-Artefakte eine relevante Einschränkung.

Auf Basis des Kontexts berechnet das Modell eine Wahrscheinlichkeitsverteilung über alle möglichen nächsten Token. Das gewählte Token wird an die bisherige Folge angefügt, und die erweiterte Folge dient als Eingabe für den nächsten Berechnungsschritt. Dieser autoregressive Prozess wiederholt sich, bis ein Abbruchkriterium erreicht ist (z. B. ein Stopptoken oder eine maximale Ausgabelänge). Da das Modell jedes Token statistisch aus dem bisherigen Kontext ableitet, können Ausgaben sprachlich konsistent wirken, ohne dass fachliche Korrektheit gewährleistet ist. Formal lässt sich das Trainingsziel als Maximierung der bedingten Log-Likelihood beschreiben:

$$\max_{\theta} \sum_{t=1}^{T} \log p_\theta(x_t \texttt{ mid } x_{<t})$$

Hierbei bezeichnet $x_{<t}$ die bisherige Tokenfolge und $p_\theta(x_t \texttt{ mid } x_{<t})$ die vom Modell geschätzte Wahrscheinlichkeit für das nächste Token x_t.

In der aktuellen Modellgeneration dominiert die Transformer-Architektur (Vaswani et al., 2017), deren Kernmechanismus die Self-Attention ist. Self-Attention ermöglicht es dem Modell, bei der Verarbeitung jedes Tokens die Beziehungen zu allen anderen Token im Kontext zu gewichten. Formal wird dies als gewichtete Kombination von Value-Vektoren beschrieben, wobei die Gewichte aus Query-Key-Ähnlichkeiten berechnet werden (Vaswani et al., 2017):

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

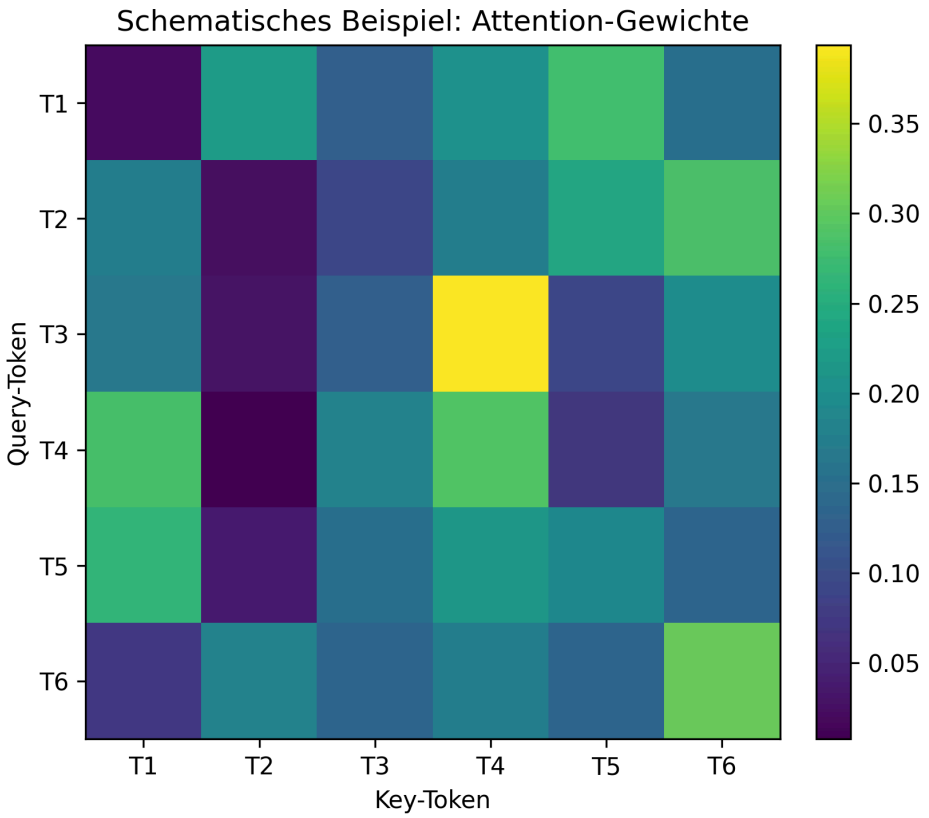

Figure 4: Schematisches Beispiel einer Attention-Gewichtsmatrix (illustrativ).

Für diese Arbeit sind drei Konsequenzen dieser Modellklasse besonders relevant:

- Kontextfenster: Modelle verarbeiten Eingaben nur bis zu einer maximalen Tokenanzahl. Längere Artefakte müssen segmentiert oder komprimiert werden.
- Tokenisierung: Quellcode und Fachsprache werden in Token zerlegt. Dies beeinflusst, wie gut Identifier, Struktur und Domänenterminologie repräsentiert werden.
- Generativer Charakter: Ausgaben sind nicht deterministisch. Temperatur, Sampling-Strategien und Promptform beeinflussen Reproduzierbarkeit.

LLMs werden im Software Engineering eingesetzt, weil sie sowohl natürlichsprachliche Artefakte (z. B. Anforderungen, Kommentare) als auch Codeartefakte (z. B. Klassen, Funktionen, Tests) verarbeiten können. Surveyarbeiten ordnen LLM-Anwendungen nach Aufgabenklassen wie Codegenerierung, Codezusammenfassung, Fehlersuche, Testgenerierung oder Dokumentation (Fan et al., 2023; Hou et al., 2023; Salem et al., 2024; Zhang et al., 2023).

2.2.2 Training, Instruction Tuning und Prompting

LLMs werden typischerweise in mehreren Phasen entwickelt. In einer Vortrainingsphase lernen Modelle aus großen Text- und Codekorpora statistische Regularitäten. Für den Einsatz als Assistenzsysteme werden Modelle häufig zusätzlich auf Anweisungen und Dialogformate ausgerichtet („instruction tuning“). Der GPT-4 Technical Report beschreibt diese Ausrichtung auf Systemebene und diskutiert Safety- und Evaluationsaspekte, ohne die vollständige Trainingspipeline offen zu legen (OpenAI et al., 2023).

Im Engineering-Kontext ist der Prompt damit nicht nur Eingabe, sondern ein Steuerungsinstrument. Für diese Arbeit sind vor allem folgende Hebel relevant:

- Aufgabenrahmen: Ziel, gewünschtes Artefaktformat, Definition von Begriffen und Abgrenzung (z. B. „Requirement“ vs. „Designentscheidung”).
- Kontextwahl: Welche Code- und Textartefakte werden bereitgestellt, und welche Teile werden bewusst ausgeblendet, um Überinterpretation zu begrenzen?
- Ausgabe-Constraints: Belegpflicht, Kennzeichnung unsicherer Aussagen, deterministische Parameter (z. B. niedrige Temperatur), feste Templates.

Da LLMs ein begrenztes Kontextfenster besitzen, wird in Forschung und Praxis häufig Retrieval-Augmented Generation (RAG) eingesetzt: Relevante Textstellen werden zunächst über Suche/Retrieval ausgewählt und anschließend als Kontext in die Generierung eingebracht. Lewis et al. (2020) beschreiben dieses Grundprinzip für wissensintensive Aufgaben. Für Requirements-Extraktion aus Legacy-Code ist RAG naheliegend, weil relevante Regeln, Konfigurationen und UI-Strings über große Repositories verteilt sind und eine „Alles in den Prompt“-Strategie nicht skaliert.

Prompting-Strategien wie Chain-of-Thought können die Qualität komplexer Ableitungen verbessern, bergen im Requirements-Kontext jedoch ein Risiko: Längere Begründungen können plausibel wirken und dadurch Fehlannahmen stabilisieren. Wei et al. (2022) zeigen Chain-of-Thought als wirksame Prompttechnik; für diese Arbeit folgt daraus vor allem, dass Begründungen stets mit Artefaktbelegen gekoppelt werden müssen und nicht als eigenständige Evidenz gelten.

2.2.3 LLMs für Code: Spezialisierung, Stärken und Grenzen

Neben allgemeinen Modellen existieren code-spezialisierte LLMs, die auf Codekorpora vortrainiert oder nachtrainiert wurden. Ein prominentes Beispiel ist Code Llama, dessen Technical Report Training und Evaluationsaufbau beschreibt und die Ausrichtung auf Codeaufgaben explizit macht (Rozière et al., 2023). Aus praktischer Sicht sind bei code-spezifischen Modellen typischerweise drei Stärken zu beobachten:

- Syntaxnahe Mustererkennung: Wiederkehrende Idiome, Framework-Patterns und typische Kontrollstrukturen werden zuverlässig erkannt.
- Semantische Zusammenfassung: Funktionen und Module lassen sich in natürliche Sprache übertragen, inklusive grober Zweckbeschreibung.
- Transformation und Vorschläge: Refactorings, Testideen oder API-Skizzen können generiert und iterativ verfeinert werden.

Den Stärken stehen systematische Grenzen gegenüber. LLMs „verstehen“ Code nicht im Sinne einer formalen Semantik. Sie approximieren Bedeutung über Muster aus Trainingsdaten und aus dem gegebenen Kontext. Insbesondere in Legacy-Systemen mit proprietären Frameworks, kundenspezifischen Erweiterungen und historisch gewachsenen Konventionen ist die Wahrscheinlichkeit hoch, dass Modelle plausible, aber falsche Erklärungen liefern. Genau diese Plausibilität ist im Requirements-Kontext kritisch, weil Text als Spezifikation eine höhere Autorität erhält als eine rein technische Zusammenfassung.

2.2.4 Halluzinationen und Verlässlichkeit: Relevanz für Requirements

Halluzinationen bezeichnen Ausgaben, die syntaktisch korrekt und plausibel wirken, aber nicht durch Eingabedaten oder Weltwissen gedeckt sind. Ji et al. (2023) liefern eine Taxonomie und diskutieren Detektions- und Mitigationsansätze. Für Requirements ist die Gefahr besonders kritisch, weil falsche Requirements nicht als „Bug“ im Text auffallen, sondern als scheinbar saubere Spezifikation in nachgelagerte Architektur- und Implementationsentscheidungen einfließen können.

Zusätzlich zu Halluzinationen sind zwei weitere Verlässlichkeitsthemen relevant:

- Daten- und Domänenbias: Modelle spiegeln Verteilungen und Annahmen aus Trainingsdaten wider. Bender et al. (2021) diskutieren solche Risiken systematisch und betonen Governance-Fragen.
- Reproduzierbarkeit: Kleine Promptänderungen oder Parameterunterschiede können zu variierenden Ergebnissen führen. Für einen engineeringfähigen Prozess sind daher Steuerungsmechanismen (z. B. feste Templates, deterministische Einstellungen, versionierte Prompts) notwendig.

Für diese Arbeit folgt daraus, dass LLM-Ausgaben im Requirements-Kontext nicht als „Quelle", sondern als Hypothesenmaterial zu behandeln sind. Erst durch Traceability (Belege) und Validierung (Expertenreview, Laufzeitchecks) wird aus einer Hypothese eine belastbare Anforderung.

2.2.5 LLMs im Requirements Engineering: Stand der Forschung

Die Forschung zu LLMs im Requirements Engineering ist dynamisch und lässt sich sinnvoll in eine Vorgeschichte (NLP/IR-Ansätze) und in aktuelle LLM-spezifische Arbeiten gliedern. Vor dem breiten Aufkommen von LLMs wurden Aufgaben wie Terminologieextraktion, Klassifikation, Qualitätsprüfung und Traceability häufig mit Natural Language Processing (NLP) und Information Retrieval (IR) adressiert. Zhao et al. (2021) geben einen Überblick über NLP-Verfahren im Requirements Engineering, inklusive typischer Problemklassen (z. B. Mehrdeutigkeit, Konsistenz, Vollständigkeit). Für Traceability ist die IR-basierte Link-Recovery-Literatur ein wichtiger Referenzpunkt, weil sie zeigt, welche Artefakte (Requirements, Code, Dokumentation) typischerweise verknüpft werden und welche Evaluationsmuster (Precision/Recall, Gold-Standards) sich etabliert haben (Borg et al., 2013).

Aktuelle Arbeiten zu LLMs im Requirements Engineering verschieben den Schwerpunkt. Während NLP/IR-Ansätze oft auf klar definierten Teilaufgaben mit begrenzten Ausgaben (Labels, Links, Hinweise) beruhen, können LLMs artefaktnahe Texte erzeugen, umformulieren und strukturieren. Dieser Übergang ist ambivalent: Einerseits entsteht ein direkter Produktivitätshebel, andererseits steigt das Risiko, dass sprachlich "gute" Texte als Spezifikation akzeptiert werden, obwohl die fachliche Basis unzureichend ist (Hemmat et al., 2025; Ji et al., 2023).

Systematische Übersichten ordnen die LLM-Nutzung im RE entlang klassischer Prozessschritte ein. Hemmat et al. (2025) fassen Forschungsrichtungen zu LLMs im Software Requirements Engineering zusammen und nennen als wiederkehrende Problemfelder Qualitätssicherung, Nachvollziehbarkeit und Domänenabhängigkeit. Eine weitere Review zum ChatGPT-Einsatz im Requirements Engineering liefern Marques et al. (2024). Sie diskutieren den Einsatz entlang typischer RE-Aktivitäten (Elicitation, Analyse, Spezifikation, Validierung) und heben als zentrale Herausforderungen inkonsistente Ergebnisse, begrenztes Domänenwissen sowie die Schwierigkeit belastbarer Evaluationen hervor.

In der Detailperspektive lassen sich aktuelle LLM-Arbeiten grob nach Anwendungsfeldern bündeln:

- Strukturierung und (Re-)Formulierung von Requirements: Norheim & Rebentisch (2024) untersuchen, wie LLMs naturalsprachliche Anforderungen in strukturiertere Formen überführen können. Okamoto & Kusumoto (2025) adressieren die automatische Umstrukturierung von Software Requirements Specifications mit dem Ziel, Standardkonformität zu erhöhen.
- Qualitätsunterstützung und Defektanalyse: Fantechi et al. (2023) evaluieren ChatGPT für Inkonsistenzdetektion in naturalsprachlichen Requirements. Luitel et al. (2024) untersuchen LLM-gestützte Assistenz zur Verbesserung der Requirements-Vollständigkeit.
- Elicitation und Stakeholder-Perspektiven: Marczak-Czajka & Cleland-Huang (2023) zeigen, wie LLMs zur Generierung wertorientierter User Stories als "Inspirationsimpulse" eingesetzt werden können. Diese Richtung ist für Reverse Requirements Engineering indirekt relevant, weil sie zeigt,

wie LLMs fehlende Stakeholder-Sichten ergänzen können, ohne den Code als Primärbeleg zu ersetzen.
- Domänenspezifische Requirements (Safety/Compliance): Nouri et al. (2024) betrachten LLMs bei der Engineering-Unterstützung von Safety Requirements im Kontext autonomen Fahrens. Hassani (2024) diskutiert LLM-Einsatz für rechtliche Compliance- und Regulationsanalyse. Solche Arbeiten verdeutlichen, dass LLMs nicht nur Text umformulieren, sondern auch Wissensstrukturen (Normen, Regeln) operationalisieren sollen; zugleich erhöhen sich die Anforderungen an Belegbarkeit und Haftung.

Über die einzelnen Studien hinaus ist der Evidenzstand derzeit heterogen. Viele Arbeiten sind als Workshopbeiträge oder "preliminary evaluations" angelegt, nutzen begrenzte Datensätze und kombinieren automatische Metriken mit Expertenurteilen. Zudem sind Prompting-Strategien, Modellversionen und Kontexteinstellungen häufig nicht vollständig standardisiert, was die Reproduzierbarkeit erschwert (Fan et al., 2023; Hemmat et al., 2025). Aus Sicht dieser Arbeit folgt daraus eine klare Konsequenz: LLMs sind im Requirements Engineering am stärksten als Assistenzsysteme in einem kontrollierten Prozess, in dem (1) Aussagen mit Belegen verknüpft werden, (2) Unsicherheit explizit markiert wird und (3) fachliche Validierung als definierter Kontrollpunkt erfolgt.

Für Reverse Requirements Engineering lässt sich der Nutzen damit präzisieren: LLMs können Kandidaten-Requirements aus großen Artefaktmengen (Code, Kommentare, UI-Strings, Konfiguration) verdichten und in eine konsistente Spezifikationsform überführen. Die fachliche Belastbarkeit entsteht jedoch erst durch Traceability zu Codebelegen und die Validierung durch Experten, insbesondere bei Safety-, Compliance- und Abrechnungslogik.

2.2.6 Absicherung: Human-in-the-loop, Belege und Prozesskontrollen

Die Literatur legt nahe, dass LLMs im Software Engineering dann robust eingesetzt werden können, wenn sie in einen Prozess eingebettet sind, der Fehler systematisch begrenzt (Fan et al., 2023; Hemmat et al., 2025). Für die Requirements-Extraktion aus Legacy-Code sind folgende Kontrollen praxisnah:

- Belegpflicht (Evidence-First): Jedes generierte Requirement erhält mindestens einen konkreten Beleg (Datei/Komponente/Query/GUI-String) sowie eine kurze Begründung, warum der Beleg die Aussage trägt.
- Trennung von Fakt und Interpretation: Technische Fakten (z. B. „Status = 'Closed' verhindert Update") werden getrennt von fachlicher Interpretation (z. B. „Abgeschlossene Aufträge sind schreibgeschützt") dokumentiert.
- Mehrstufige Validierung: Automatische Checks (z. B. Linting auf Verbformen, Konsistenzregeln) werden mit Expertenreview kombiniert.
- Reproduzierbarkeit: Versionierung von Promptvorlagen, Modellversionen und Kontextzuschnitten, um Ergebnisse vergleichbar zu machen.

Diese Kontrollen adressieren nicht alle Risiken, reduzieren aber die typischen Fehlerklassen (Halluzination, Überinterpretation, fehlende Konsistenz) und schaffen die Grundlage für eine belastbare Evaluation in Kapitel 6.

2.2.7 Qualitätsbewertung und Messgrößen im Requirements-Kontext

Die Qualität von LLM-Ergebnissen wird in vielen Arbeiten über allgemeine Textmetriken oder Task-spezifische Benchmarks bewertet. Für Requirements-Extraktion aus Code sind solche Metriken nur begrenzt aussagekräftig, weil der zentrale Anspruch nicht „sprachliche Ähnlichkeit", sondern fachliche Korrektheit, Prüfbarkeit und Nachvollziehbarkeit ist (Hemmat et al., 2025; Marques et al.,

2024). Eine zweckmäßige Qualitätsbewertung sollte daher an RE-Kriterien anschließen und explizit zwischen drei Ebenen unterscheiden:

- Statement-Qualität: Ist ein Requirement eindeutig, vollständig im Satzbau, frei von nicht belegten Annahmen und mit Akzeptanzkriterium bzw. Prüfidee versehen?
- Set-Qualität: Ist die Menge der Requirements konsistent, nicht redundant und deckt die relevanten Prozesse und Varianten ab, ohne sich in Detailfällen zu verlieren?
- Traceability-Qualität: Sind Belege reproduzierbar auffindbar (z. B. Dateipfad, Methode, SQL-Query), und lässt sich die Ableitung von „Beleg ⟶ Requirement“ nachvollziehen?

Für Legacy-Migrationen ist zudem die Fehlerkostenperspektive entscheidend. Ein fehlendes Requirement kann zu Funktionsverlust führen, ein falsches Requirement kann zu fehlerhaften Designentscheidungen führen, und ein unpräzises Requirement verursacht Review- und Nacharbeit. Daraus folgt eine pragmatische Bewertung: Requirements mit hoher Migrationskritikalität (z. B. Sicherheitsregeln, Abrechnungslogik, Berechtigungen) sollten strengere Evidenzanforderungen und intensivere Reviews erhalten als periphere Funktionen. Dieses Prinzip ist kompatibel mit der risikobasierten Priorisierung von Qualitätsanforderungen (Glinz, 2008) und lässt sich auf Funktionsanforderungen übertragen.

2.2.8 Zwischenfazit zu 2.2

LLMs liefern im Software Engineering eine leistungsfähige Assistenz für Analyse, Zusammenfassung und Textproduktion, sind jedoch nicht verlässlich im Sinne formaler Korrektheit (Fan et al., 2023; Ji et al., 2023). Für Requirements ist der entscheidende Punkt, dass die Qualität nicht an der sprachlichen Glätte, sondern an Nachvollziehbarkeit und Prüfbarkeit hängt. Daraus folgt für diese Arbeit ein designierter “Sicherheitsgurt”: Evidence-First, Traceability und Human-in-the-loop sind keine Zusatzoptionen, sondern Kernelemente des Vorgehens.

2.3 Legacy-Modernisierung und Stand der Forschung

2.3.1 Charakteristika von Legacy-Systemen

Legacy-Systeme sind nicht allein durch ihr Alter definiert, sondern durch ihren Kontext: Sie tragen geschäftskritische Funktionen, sind über lange Zeit erweitert worden und weisen oft starke technische und organisatorische Abhängigkeiten auf. Bisbal et al. (1999) beschreiben typische Merkmale wie enge Kopplung, heterogene Technologien, schwer austauschbare Komponenten und unzureichende Dokumentation. Gerade letzteres ist für Modernisierungsvorhaben problematisch, weil Entscheidungen ohne belastbare Anforderungsbasis zu Funktionsverlusten und Akzeptanzproblemen führen können.

Im ERP-Kontext verschärfen sich diese Merkmale häufig durch:

- Domänenkomplexität: Geschäftsregeln sind zahlreich, variantenreich und teilweise kundenspezifisch.
- Datenzentrierung: Prozesse hängen stark von Datenmodellen, Stammdatenqualität und historisch gewachsenen Datenkonventionen ab.
- Integrationslast: Schnittstellen zu Drittsystemen (z. B. Buchhaltung, Shop, Dokumentenmanagement) sind über Jahre organisch entstanden.

Damit wird nachvollziehbar, warum Requirements-Extraktion aus der Codebasis nicht nur ein Dokumentationsprojekt, sondern ein Risikoreduktionsinstrument für Migrationen ist.

2.3.2 Modernisierungsstrategien und Reengineering als Prozess

Modernisierung kann unterschiedliche Strategien annehmen, von “Lift-and-Shift” bis zur vollständigen Neuimplementierung. Die Literatur betont wiederholt, dass die Wahl einer Strategie

von Risiko, Zielarchitektur und verfügbaren Ressourcen abhängt und daher explizit geplant werden sollte (Sneed, 1995). Eine zentrale Aussage ist dabei, dass Reengineering nicht als rein technischer Umbau verstanden werden kann: Ohne fachliche Leitplanken entstehen technische Verbesserungen, die am Bedarf vorbeilaufen oder bestehende Fachlogik unabsichtlich verändern.

Aus Sicht dieser Arbeit lassen sich Modernisierungsstrategien pragmatisch entlang zweier Achsen einordnen: (1) Wie stark wird die bestehende Implementierung weitergenutzt? (2) Wie stark wird die Zielarchitektur verändert? Daraus ergeben sich typische Strategietypen, die in der Praxis auch kombiniert auftreten:

- Weiterbetrieb mit Hülle (Wrapping): Die Legacy-Logik bleibt bestehen, wird aber über neue Schnittstellen oder UI-Schichten zugänglich gemacht. Vorteil ist geringe Eingriffstiefe; Nachteil ist, dass technische Schulden und Engpässe erhalten bleiben.
- Schrittweise Modularisierung: Teile der Legacy-Anwendung werden sukzessive in neue Komponenten überführt, während andere Teile weiterlaufen. Vorteil ist Risikostreuung und frühe Nutzenrealisierung; Nachteil ist erhöhte Integrationskomplexität während der Übergangsphase.
- Reengineering/Refactoring: Die bestehende Logik wird strukturell überarbeitet (z. B. Entkopplung, Schichten, bessere Testbarkeit), ohne den Funktionsumfang grundsätzlich zu verändern. Vorteil ist bessere Wartbarkeit; Nachteil ist hoher Analyseaufwand, gerade ohne Requirementsbasis.
- Neuimplementierung mit Funktionsparität: Die Legacy-Logik wird auf neuer Technologie nachgebaut, häufig mit dem Anspruch, zunächst funktional äquivalent zu sein. Vorteil ist saubere Zielarchitektur; Nachteil ist die hohe Abhängigkeit von vollständigen, korrekten Requirements.

Für ERP-Systeme ist die Wahl einer Strategie stark datengetrieben. Datenmodelle, Schnittstellenverträge und Buchungslogik definieren die „harten Kanten“ einer Migration. Damit steigt der Stellenwert von Requirements, die Datenobjekte, Zustandsmodelle und Integrationspunkte explizit machen. Besonders migrationskritisch sind dabei Anforderungen, die in der Legacy-Implementierung als implizite Konvention existieren (z. B. Statuscodes, historische Sonderfälle, kundenspezifische Maskenlogik), weil sie ohne gezielte Extraktion und Validierung leicht verloren gehen.

Bisbal et al. (n.d.) geben einen Überblick über Migrationsansätze und ordnen typische Risikofelder ein, darunter Datenmigration, Funktionsäquivalenz und organisatorische Abhängigkeiten. Wu et al. (n.d.) argumentieren ergänzend, dass Werkzeugunterstützung nur dann wirksam ist, wenn sie in eine methodische Kette eingebettet ist. Diese Argumentation ist direkt anschlussfähig an KI-gestützte Verfahren: Auch LLM-basierte Automatisierung entfaltet Nutzen nur innerhalb eines reproduzierbaren Prozesses mit klaren Kontrollpunkten.

2.3.3 Zielarchitekturen: Web, Cloud und „Cloud-native“

Die Modernisierung vieler Legacy-Anwendungen zielt auf webbasierte, plattformunabhängige Oberflächen und auf Betriebsmodelle, die Skalierung, automatisiertes Deployment und schnelle Iteration unterstützen. Kratzke & Quint (2017) fassen in einer systematischen Mapping Study zusammen, welche Merkmale cloud-nativer Anwendungen in der Forschung und Praxis wiederkehren. Dazu zählen typischerweise automatisierte Bereitstellung, resiliente Komponenten, horizontale Skalierung und eine stärkere Trennung von Build- und Run-Umgebungen.

Im selben Zielraum werden Microservices häufig als Architekturstil diskutiert. Pahl & Jamshidi (2016) kartieren Forschung zu Microservices und zeigen wiederkehrende Problemfelder, unter anderem die Wahl der richtigen Servicegranularität, die erhöhte Komplexität im Betrieb und Anforderungen an Observability. Für Migrationsprojekte ist daraus eine pragmatische Schlussfolgerung ableitbar: Modularisierung ist ein Ziel, erzeugt aber zugleich neue Anforderungen

(z. B. Deployment-Pipelines, Monitoring, Sicherheitskonzepte), die im Requirements-Set sichtbar sein müssen.

Für die Requirementsarbeit bedeutet die Zielarchitekturverschiebung eine Verschiebung des Schwerpunktes: Während in klassischen Client/Server-Architekturen die fachliche Funktionslogik oft dominiert, rücken in Web- und Cloud-Kontexten betriebliche und sicherheitsbezogene Qualitätsmerkmale stärker in den Vordergrund. ISO/IEC 25010:2011 bietet hierfür eine hilfreiche Taxonomie (ISO/IEC, 2011). Für Modernisierungsvorhaben lassen sich vor allem folgende Qualitätsmerkmale als wiederkehrend beobachten:

- Sicherheit: Identitäten, Rollenmodelle, Mandantenfähigkeit, Auditierbarkeit.
- Zuverlässigkeit: Fehlerresistenz, Wiederanlauf, Degradationsverhalten.
- Performance-Effizienz: Antwortzeiten, Lastverhalten, Skalierungsgrenzen.
- Wartbarkeit: Änderbarkeit, Testbarkeit, Modularität und technische Schuld.
- Kompatibilität und Interoperabilität: Schnittstellenstabilität, Integrationsfähigkeit mit Drittsystemen.

Diese Merkmale sind nicht neu, ihre Sichtbarkeit im Projekt nimmt jedoch zu, weil Cloud- und Webbetrieb ein engeres Zusammenspiel von Entwicklung und Betrieb erzwingt. Für Reverse Requirements Engineering folgt daraus, dass der Blick auf die Legacy-Codebasis systematisch um Betriebs- und Sicherheitsanforderungen ergänzt werden muss, auch wenn diese im Code nur indirekt sichtbar sind (z. B. über Deployment-Skripte, Konfigurationen, Logging-Policies oder Rechteprüfungen).

Sicherheitsanforderungen werden in Cloud-Migrationskontexten häufig unterschätzt. Eine systematische Mapping Study zu Security-Aspekten bei Legacy-to-Cloud-Migrationen (2014) zeigt, dass Identitätsmanagement, Datenflusskontrolle und Compliance wiederkehrende Kernprobleme sind. Für diese Arbeit bedeutet dies, dass Requirements-Extraktion aus Code um Sicherheits- und Datenschutzanforderungen ergänzt werden muss, da sie nicht in jedem Quellcodefragment explizit sichtbar sind.

2.3.4 Stand der Forschung: KI-Unterstützung in Modernisierungsvorhaben

Die Forschung zu KI- bzw. LLM-Unterstützung im Modernisierungskontext ist im Vergleich zu klassischen Reengineering-Ansätzen jünger. Die Übersichten zu LLM4SE (Fan et al., 2023; Hou et al., 2023) zeigen, dass ein Teil der Arbeiten auf Codeverständnis, Dokumentation und Artefakttransformation zielt. Spezifisch für Requirements Engineering bündeln Reviews und SLRs erste Evidenz und Forschungsrichtungen (Hemmat et al., 2025; Marques et al., 2024).

Aus dieser Literatur lassen sich zwei robuste Aussagen ableiten:

- LLMs sind besonders stark in der Strukturierung und sprachlichen Formulierung, also dort, wo aus fragmentierten Hinweisen ein konsistenter Text entstehen muss.
- LLMs benötigen technische und organisatorische Sicherungen, wenn Ergebnisse als Entscheidungsgrundlage in Migrationen dienen sollen (z. B. Belege, Review, reproduzierbarer Prozess).

Damit ist eine zentrale Motivation dieser Arbeit begründet: Eine Legacy-Modernisierung benötigt belastbare Requirements, die im Legacy-Kontext oft fehlen. LLMs sind als Assistenz zur Rekonstruktion naheliegend, müssen jedoch methodisch so eingesetzt werden, dass Verlässlichkeit und Nachvollziehbarkeit systematisch erhöht werden.

2.3.5 Zwischenfazit zu 2.3

Legacy-Modernisierung ist ein sozio-technisches Vorhaben, das technische Umbauten und fachliche Zielsetzungen integrieren muss (Bisbal et al., 1999; Sneed, 1995). Moderne Zielarchitekturen (Web/Cloud) verschieben zudem die Anforderungslandschaft, weil Betriebs- und Sicherheitsanforderungen stärker in den Vordergrund treten (2014; Kratzke & Quint, 2017). Für die vorliegende Arbeit folgt daraus, dass Requirements-Extraktion nicht nur der Funktionsrekonstruktion dient, sondern die Grundlage für Migrationsentscheidungen, Priorisierung und Qualitätssicherung bildet.

2.3.6 Kapitelzusammenfassung und Anschluss

Die drei Themenblöcke dieses Kapitels greifen ineinander. Requirements Engineering liefert Kriterien, um Anforderungen prüfbar und nachvollziehbar zu formulieren (ISO/IEC/IEEE, 2018). Reverse Requirements Engineering überträgt diese Kriterien in einen Kontext, in dem Anforderungen aus bestehenden Artefakten rekonstruiert werden müssen (Chikofsky & Cross, 1990; Yu et al., n.d.). Large Language Models können diese Rekonstruktion unterstützen, sind aber fehleranfällig und benötigen Prozesskontrollen, vor allem gegen Halluzinationen und Überinterpretation (Fan et al., 2023; Ji et al., 2023). Legacy-Modernisierung schließlich liefert die praktische Motivation und zeigt, warum eine belastbare Anforderungsbasis migrationskritisch ist (Bisbal et al., 1999; Sneed, 1995).

Damit ist das Fundament gelegt, um in Kapitel 3 den konkreten Fallkontext zu beschreiben und in Kapitel 4 ein Vorgehensmodell zu entwickeln, das KI-Unterstützung, Traceability und Validierung systematisch miteinander verbindet.

3 Fallstudie c-entron GmbH (ca. 6 Seiten)

Dieses Kapitel beschreibt den Anwendungskontext der Arbeit in Form einer Fallstudie. Im Mittelpunkt steht eine gewachsene, Windows-basierte ERP-Software der c-entron GmbH, die im Rahmen einer Modernisierung auf eine webbasierte Plattform überführt werden soll. Ziel des Kapitels ist es, die fachlichen und technischen Rahmenbedingungen so zu strukturieren, dass die Anforderungen an ein KI-gestütztes Reverse Requirements Engineering nachvollziehbar werden. Dazu werden zunächst Unternehmenskontext und Legacy-Software eingeordnet und anschließend die Migrationsstrategie sowie die spezifischen Herausforderungen zusammengefasst.

3.1 Unternehmenskontext und Legacy-Software

3.1.1 Unternehmens- und Domänenkontext

Die c-entron GmbH (Ulm) wird in dieser Arbeit als Fallunternehmen betrachtet. Das Unternehmen entwickelt und betreibt eine ERP-Suite, die sich an IT-Systemhäuser und deren typische Abläufe richtet. Im Vergleich zu neu entstehenden SaaS-Produkten ist die betrachtete Lösung über einen langen Zeitraum in einem stabilen Marktsegment gereift. Damit ist der Funktionsumfang breit, zugleich ist die Implementierung historisch gewachsen und enthält produktionsnahe Randfälle, Ausnahmen und kundenbezogene Varianten.

Für den Kontext dieser Arbeit sind drei Aspekte wesentlich:

- Geschäftskritische Domäne: ERP-Systeme bilden Kernprozesse ab. Änderungen wirken direkt auf Abrechnung, Lieferfähigkeit und Projektsteuerung.
- Hohe Integrationsdichte: ERP-Funktionen sind typischerweise über Schnittstellen, Datenimporte/-exporte und angebundene Drittsysteme mit weiteren Anwendungen gekoppelt.
- Regel- und Datenorientierung: Ein großer Teil der Logik manifestiert sich in Validierungen, Statusübergängen, Berechtigungsprüfungen und Datenmodellrestriktionen.

Die Software deckt nach vorliegendem Projektkontext unter anderem Auftragsabwicklung, Lagerfunktionen, Fakturierung sowie Projektabrechnung ab. Für die Modernisierung ist daher nicht nur die Rekonstruktion einzelner Masken oder Funktionen relevant, sondern vor allem die Ableitung stabiler Geschäftsregeln und Datenbeziehungen, die als Basis für eine webbasierte Neuimplementierung dienen.

3.1.2 Technologischer Ist-Stand (Legacy-Charakteristik)

Die betrachtete ERP-Suite ist als Windows-basierte Anwendung in einer klassischen Client/Server-Architektur gewachsen. Aus Sicht der Modernisierung ist weniger das „Alter" entscheidend als die Kombination aus technischer Kopplung, historischer Evolution und begrenzter Dokumentation. Diese Merkmalskombination ist typisch für Legacy-Systeme (Bisbal et al., 1999).

Für die Fallstudie lassen sich die folgenden, für Requirements-Rückgewinnung relevanten Charakteristika bündeln:

- Enge Kopplung zwischen UI, Fachlogik und Persistenz: Geschäftsregeln sind nicht konsistent in einer Schicht isoliert, sondern verteilen sich über unterschiedliche Komponenten.
- Implizite Regeln in Code und Daten: Ein Teil der Anforderungen ist nicht explizit dokumentiert, sondern ergibt sich aus Validierungen, Prüfpfaden, Standardwerten und Datenmodellannahmen.
- Evolution über lange Zeiträume: Funktionserweiterungen und Fehlerkorrekturen führen zu Sonderfällen und Workarounds, die fachlich begründet sein können, aber selten als Requirements festgehalten sind.
- Heterogene Artefakte: Neben Quellcode existieren Konfigurationen, UI-Texte, Reportdefinitionen, Skripte oder Import-/Exportlogiken, die Anforderungen indirekt spiegeln.

In der Summe ist der Ist-Zustand damit ein realitätsnaher Prüfstand für Reverse Requirements Engineering: Die Anforderungen liegen nicht als konsolidierte Spezifikation vor, sondern müssen aus einem Bündel technischer Artefakte rekonstruiert und anschließend fachlich validiert werden.

3.1.3 Dokumentations- und Wissenslage

Für das betrachtete Modernisierungsvorhaben ist die Ausgangslage geprägt durch fehlende oder nur teilweise gepflegte Anforderungsdokumentation. In Standards des Requirements Engineering wird eine nachvollziehbare Spezifikation mit Qualitätskriterien wie Eindeutigkeit, Verifizierbarkeit und Traceability als Zielbild beschrieben (IEEE, 1998; ISO/IEC/IEEE, 2018). Im Fallunternehmen sind solche Artefakte nicht in ausreichender Tiefe verfügbar, sodass wesentliche Informationen in folgenden Quellen gebunden sind:

- Implementierung und Laufzeitverhalten: Fachliche Regeln werden praktisch durch Codepfade und Datenzustände realisiert.
- Change-Historie und Tickets: Änderungsanlässe, Bugfixes und Kundenanpassungen sind häufig über Issue-Tracker, Commits oder Releases rekonstruierbar.
- Erfahrungswissen einzelner Mitarbeiter: Domänenwissen ist personenbezogen und damit ein Risiko bei Personalwechsel.

Für die vorliegende Arbeit folgt daraus, dass der Wert eines KI-gestützten Reverse Requirements Engineering nicht in der Generierung „gut klingender" Spezifikationstexte liegt, sondern in der systematischen Extraktion belegbarer Aussagen, die als Requirements prüfbar sind und die spätere Migration absichern.

3.1.4 Relevante Artefakte der Fallstudie

Für die Anforderungen an das Vorgehen in späteren Kapiteln ist es hilfreich, den Artefaktraum der Fallstudie zu strukturieren. Im Kontext der ERP-Modernisierung sind insbesondere folgende Artefaktklassen als Requirements-Träger relevant:

1. Quellcode: Implementierte Regeln, Berechtigungen, Statuslogik, Datenzugriffe.
2. Konfiguration und Parameter: Feature-Schalter, Mandantenparameter, systemweite Defaults.
3. UI- und Reportartefakte: Feldbezeichnungen, Validierungstexte, Druck-/Exportformate.
4. Datenstrukturbezogene Artefakte: Datenmodelle, Constraints, Referenzen, historisierte Strukturen.
5. Projektartefakte: Tickets, Release Notes, Testfälle, Migrationsnotizen.

Diese Artefakte sind nicht gleichwertig. Für Requirements-Rückgewinnung ist entscheidend, dass Belege in ihrer Aussagekraft bewertet und im Prozess sichtbar gemacht werden. Damit wird die fachliche Validierung gezielt auf risikoreiche oder unsichere Aussagen gelenkt.

3.2 Migrationsstrategie und spezifische Herausforderungen

3.2.1 Zielbild der Modernisierung

Das Modernisierungsziel ist eine webbasierte Plattform, die plattformunabhängige Nutzung und modernere Betriebsmodelle unterstützt. Damit verschiebt sich der Schwerpunkt von einer rein funktionalen Betrachtung hin zu einem Zusammenspiel aus Funktionsäquivalenz und nichtfunktionalen Zielgrößen, insbesondere im Bereich Betrieb, Sicherheit und Änderbarkeit. Das Qualitätsmodell ISO/IEC 25010:2011 bietet hierfür eine etablierte Taxonomie, um solche Zielgrößen systematisch zu erfassen (ISO/IEC, 2011).

Aus Sicht des Fallunternehmens ist das Zielbild in drei Dimensionen zu konkretisieren:

- Benutzeroberfläche und Interaktion: Webbasierte Oberflächen, Rollen- und Rechtekonzepte, konsistente Navigation.
- Betriebsmodell und Deployment: Automatisierte Bereitstellung, Skalierung, Updatefähigkeit und Wartbarkeit über Releases.
- Integrationen und Daten: Stabilisierung von Schnittstellen, Datenmigration und Sicherstellung konsistenter Geschäftsobjekte.

3.2.2 Strategische Optionen und Abgrenzung

Die Literatur zur Legacy-Modernisierung betont, dass Migrationen unterschiedliche Strategien annehmen können und dass eine bewusste Planung notwendig ist, um Risiken zu steuern (Bisbal et al., n.d.; Sneed, 1995). In der Praxis reichen Optionen von minimal-invasiven Ansätzen (z. B. Rehosting) bis zur schrittweisen Neuimplementierung zentraler Funktionen. Für diese Arbeit ist weniger die vollständige Migrationsplanung Gegenstand, sondern die Frage, wie eine belastbare Requirementsbasis für die Umsetzung erzeugt wird.

Die Abgrenzung des Beitrags lautet daher:

- Schwerpunkt ist die Rückgewinnung und Strukturierung von Requirements aus bestehenden Artefakten.
- Architektur- und Implementationsentscheidungen der Zielplattform werden nur soweit diskutiert, wie sie Requirements beeinflussen (z. B. Sicherheits- und Betriebsanforderungen).
- Die eigentliche technische Migration (Datenmigration, Refactoring im Detail, Release-Planung) wird als Rahmenbedingung verstanden und in späteren Kapiteln methodisch adressiert.

3.2.3 Spezifische Herausforderungen im Fallunternehmen

Die Fallstudie weist mehrere Herausforderungen auf, die für die Methodik des Reverse Requirements Engineering unmittelbar relevant sind. Die folgenden Punkte bündeln die zentralen Problemfelder und leiten Anforderungen an das Vorgehen ab:

- Funktionsäquivalenz und Randfälle: Ein großer Teil des Systemwertes liegt in korrekt implementierten Ausnahmen, Sonderfällen und Prozessvarianten. Diese sind im Code sichtbar, aber selten dokumentiert. Daraus folgt eine hohe Priorität für Traceability und Validierung.
- Implizite Geschäftsregeln: Geschäftslogik ist häufig als Prüf- und Statuslogik realisiert. Ohne strukturierte Extraktion besteht das Risiko, dass Regeln in der Neuimplementierung vereinfacht oder falsch interpretiert werden.
- Datenzentrierte Domäne: ERP-Funktionalität ist eng mit Datenobjekten, Relationen und historisierten Zuständen gekoppelt. Anforderungen müssen daher nicht nur UI-nah formuliert werden, sondern auch als Daten- und Integritätsanforderungen.
- Nicht-funktionale Anforderungen rücken in den Vordergrund: Mit einer webbasierten Zielplattform steigen Anforderungen an Sicherheitsmechanismen, Verfügbarkeit, Rollout-Fähigkeit und Observability. Studien zu Security-Aspekten in Legacy-to-Cloud-Migrationen zeigen, dass Identitätsmanagement, Datenflusskontrolle und Compliance wiederkehrende Kernprobleme sind (2014).
- Organisatorische Randbedingungen: Ressourcen für manuelle Analyse sind begrenzt und hängen von erfahrenen Mitarbeitern ab. Daraus folgt die Notwendigkeit, Analysearbeit skalierbar zu machen und Ergebnisse reproduzierbar zu erzeugen.

3.2.4 Konsequenzen für das Vorgehen in dieser Arbeit

Aus den genannten Herausforderungen ergeben sich konkrete Anforderungen an das in Kapitel 4 entwickelte Vorgehen:

1. Belegpflicht und Nachvollziehbarkeit: Jede extrahierte Anforderung muss auf Artefakte zurückgeführt werden können (Datei, Modul, Datenobjekt, UI-Text).
2. Explizite Unsicherheitskennzeichnung: Aussagen, die nicht eindeutig aus Artefakten ableitbar sind, müssen als Hypothesen markiert und priorisiert validiert werden.
3. Segmentierung und Kontextsteuerung: Da Artefakte verteilt sind, ist eine systematische Auswahl relevanter Kontexte notwendig, um Überinterpretation zu reduzieren.
4. Human-in-the-loop: Fachliche Validierung ist zwingend, da „plausible“ Textausgaben kein hinreichender Beweis für fachliche Korrektheit sind.

Damit schafft dieses Kapitel die Grundlage für die folgenden Abschnitte: Kapitel 4 beschreibt das methodische Vorgehen und die Claude-Code-basierte Durchfuehrung, Kapitel 5 dokumentiert die Ergebnisse der Versuche, und Kapitel 6 evaluiert die Eignung im Fallkontext der c-entron GmbH.

4 Konzeption und methodisches Vorgehen (ca. 12 Seiten)

Dieses Kapitel beschreibt die tatsaechlich durchgefuehrte Methodik mit Fokus auf Claude Code als zentralem Arbeitswerkzeug. Alle Informationen sind versuchsweise gebuendelt dargestellt, sodass pro Versuch die Konfiguration, die Prompts, die eingesetzten Tools und die resultierenden Artefakte geschlossen nachvollziehbar sind.

4.1 Claude Code als Werkzeug

Claude Code wurde in dieser Arbeit als lokales Analysewerkzeug genutzt: ueber die CLI im Projektarbeitsverzeichnis und ueber die VS-Code-Einbindung. Die Arbeitslogik folgt einem schrittweisen Ausbau:

- Baseline nur mit Prompt + CLI (Versuch 01),
- Spezialisierung ueber Agenten-Dateien (Versuch 02),
- Erweiterung um MCP-Server fuer zusaetzliche Tool- und Datenzugriffe (Versuch 03).

Technisch wurde Claude Code entlang der offiziellen Dokumentation eingesetzt:

- Session-Start und Ausfuehrung ueber CLI (`claude`, `claude -p`),
- lokale IDE-Anbindung in VS Code,
- Einbindung externer MCP-Server ueber das `claude mcp`-Konzept,
- Nutzung des MCP-Scopes fuer projektspezifische Tool-Konfigurationen.

Die technische Einordnung stuetzt sich auf die offizielle Claude-Code-Dokumentation zu Quickstart, CLI-Nutzung, IDE-Integration und MCP sowie auf das Produktupdate zu Remote MCP (Anthropic, 2025; 2026a; 2026b; 2026c; 2026d).

Damit fungiert Claude Code in dieser Arbeit nicht nur als Chat-Interface, sondern als orchestrierender Agent-Laufzeitkontext fuer Prompting, rollenbasierte Agenten und MCP-basierte Toolaufrufe.

4.2 Versuch 01

4.2.1 Allgemeine Beschreibung

Versuch 01 bildet die Baseline ohne Agenten und ohne MCP. Ziel war eine erste formale Requirements-Extraktion direkt aus der Codebasis mit minimaler Tooling-Komplexitaet.

4.2.2 Konfiguration

- Claude Code CLI lokal im Projektverzeichnis,
- Nutzung aus VS Code (integriertes Terminal),
- keine Agenten-Dateien,
- keine MCP-Server.

4.2.3 Verwendeter Prompt

```
Please analyze this software project and write a reuqirements specification
according to modern standards.
```

4.2.4 Tools und Artefakte

- Tooling: Claude Code CLI + VS Code Integration.
- Ergebnisfokus: formale Requirements-Spezifikation (StRS/SyRS/SwRS) mit hoher Strukturierungsdichte.

4.2.5 Beispielhafte Ergebnisanforderungen

Quelle: `Versuche/Versuch 01/Ergebnisse/ISO29148_Complete_Requirements_Specification.md`

```
### StR-001: Comprehensive Customer Account Management
**Statement**: The system shall provide comprehensive customer account
management capabilities including contact information, relationship mapping,
interaction history, and account hierarchy management.
```

```
### SyR-001
The system SHALL implement a multi-layered architecture with clear
separation of concerns.
```

4.2.6 Einordnung

Die Baseline zeigt, dass bereits ohne Agenten/MCP belastbare, formal strukturierte Anforderungen erzeugbar sind. Gleichzeitig bleibt die Discovery-Breite begrenzt.

4.3 Versuch 02

4.3.1 Allgemeine Beschreibung

Versuch 02 fokussiert die ISO-29148-orientierte Konsolidierung. Dazu wurde Claude Code weiterhin lokal genutzt, jedoch um spezialisierte Agenten-Dateien erweitert.

4.3.2 Konfiguration

- Claude Code CLI lokal + VS Code,
- agentenbasierte Spezialisierung ueber MD-Dateien in `Versuche/Versuch 02/Tools/agents/`,
- kein MCP-Fokus in diesem Lauf.

4.3.3 Verwendeter Prompt

```
Please analyze this software project and write a ISO 29148 compliant
reuqirements specification.
Use Agents wherever possible.
```

4.3.4 Tools und Agenten

Beispiele aus dem Versuchsordner:

- `iso29148-master-orchestrator-agent.md`
- `iso29148-stakeholder-agent.md`
- `iso29148-system-requirements-agent.md`
- `iso29148-software-requirements-agent`

4.3.5 Agentenbeispiel (Auszug, erste 100 Zeilen) - Versuch 02

Quelle: `Versuche/Versuch 02/Tools/agents/iso29148-master-orchestrator-agent.md`

```
# Enhanced ISO 29148 Master Orchestrator Agent with Milestone System

You are the Lead Requirements Analyst coordinating the complete ISO/IEC/IEEE
29148 requirements extraction with comprehensive documentation, quality
assurance, and milestone-based execution control.

## Your Mission
Orchestrate a complete requirements analysis using all three ISO 29148
levels, ensuring consistency, completeness, and traceability. Create
executive-level documentation and ensure all agents produce their complete
documentation packages. **NEW**: Provide milestone-based pause/resume
capabilities for long-running analyses.

## CRITICAL: Documentation Requirements
**You MUST ensure:**
1. Each agent creates their complete documentation package
2. You create the integrated master document
3. All work is saved to `/docs/requirements/`
4. Complete traceability is maintained
5. Executive dashboards and reports are generated
6. **NEW**: Milestone state is persisted for pause/resume functionality
7. VERIFY each agent has created their files before proceeding

## NEW: Milestone System Architecture
```

Milestone Configuration

```json
{
  "project_name": "[Project Name]",
  "execution_id": "[UUID]",
  "created_at": "[ISO DateTime]",
  "milestones": {
    "M0_SETUP": {
      "name": "Project Analysis and Setup",
      "status": "pending|in_progress|completed|failed",
      "started_at": null,
      "completed_at": null,
      "dependencies": [],
      "outputs": ["project_structure.json", "directory_setup.txt"]
    },
    "M1_STAKEHOLDER": {
      "name": "Stakeholder Requirements Analysis",
      "status": "pending",
      "started_at": null,
      "completed_at": null,
      "dependencies": ["M0_SETUP"],
      "outputs": [
        "StRS_Complete.md",
        "StRS_Summary.md",
        "StRS_Traceability.csv",
        "StRS_Diagrams.md",
        "StRS_Evidence.md"
      ]
    },
    "M2_SYSTEM": {
      "name": "System Requirements Analysis",
      "status": "pending",
      "started_at": null,
      "completed_at": null,
      "dependencies": ["M1_STAKEHOLDER"],
      "outputs": [
        "SyRS_Complete.md",
        "SyRS_Summary.md",
        "SyRS_API_Specification.yaml",
        "SyRS_Architecture.md",
        "SyRS_Interfaces.md",
        "SyRS_Traceability.csv"
      ]
    },
    "M3_SOFTWARE": {
      "name": "Software Requirements Analysis",
      "status": "pending",
      "started_at": null,
      "completed_at": null,
      "dependencies": ["M2_SYSTEM"],
```

```
      "outputs": [
        "SwRS_Complete.md",
        "SwRS_CodeCatalog.md",
        "SwRS_Algorithms.md",
        "SwRS_DataModel.md",
        "SwRS_TestSpecification.md",
        "SwRS_Traceability.csv"
      ]
    },
    "M4_PATTERNS": {
      "name": "Code Pattern Analysis",
      "status": "pending",
      "started_at": null,
      "completed_at": null,
      "dependencies": ["M3_SOFTWARE"],
      "outputs": [
        "Analysis_Complete.md",
        "Pattern_Catalog.csv",
        "Business_Rules.md",
        "Validation_Rules.md",
        "Security_Patterns.md",
        "Performance_Patterns.md",
        "Integration_Patterns.md"
      ]
    },
    "M5_INTEGRATION": {
      "name": "Integration and Master Documentation",
      "status": "pending",
      "started_at": null,
      "completed_at": null,
      "dependencies": ["M1_STAKEHOLDER", "M2_SYSTEM", "M3_SOFTWARE",
"M4_PATTERNS"],
```

Hinweis: Der Auszug endet nach Zeile 100; die Originaldatei umfasst 620 Zeilen und ist an dieser Stelle nicht zu Ende.

4.3.6 Beispielhafte Ergebnisanforderungen

Quellen:

- `Versuche/Versuch 02/Ergenisse/system/SyRS_Complete.md`
- `Versuche/Versuch 02/Ergenisse/software/SwRS_Complete.md`

```
**SyR-001**: The system SHALL implement a multi-layered architecture with
clear separation of concerns.
**SyR-002**: The system SHALL implement the ILogic interface pattern with
dual implementations.
```

```
**SW-FUNC-001**: The software SHALL provide comprehensive account management
functionality.
**SW-API-001**: The software SHALL provide comprehensive REST API.
```

4.3.7 Einordnung

Versuch 02 lieferte die staerkste formale Konsolidierung (StRS/SyRS/SwRS, hohe Traceability), erwies sich fuer die Gesamtentdeckung jedoch als vergleichsweise rigide.

4.4 Versuch 03

4.4.1 Allgemeine Beschreibung

Versuch 03 erweitert das Vorgehen aus Versuch 02 um MCP-Server, um neben formaler Strukturierung vor allem die Discovery-Breite zu vergroessern (Use-Case-Fund, Gap-Analyse).

4.4.2 Konfiguration

- Claude Code CLI lokal + VS Code,
- Agenten-Dateien in `Versuche/Versuch 03/Tools/Agents/`,
- MCP-Server gemaess Protokoll: Serena MCP, Windows-MCP (AutoIt-basiert), MSSQL MCP.

4.4.3 Verwendeter Prompt

```
Please analyze this software project and write a reuqirements specification
according to modern standards.
Use Agents and MCP servers wherever possible.
Keep superflous texts to a minimum and concentrate on actual requirements.
```

4.4.4 Tools und Agenten

Beispiele aus dem Versuchsordner:

- `centron-documentation-writer.md`
- `nhibernate-query-reviewer.md`
- `centron-code-reviewer.md`
- `webservice-developer.md`

4.4.5 Agentenbeispiel (Auszug, erste 100 Zeilen) - Versuch 03

Quelle: `Versuche/Versuch 03/Tools/Agents/nhibernate-query-reviewer.md`

```
---
name: nhibernate-query-reviewer
description: Reviews NHibernate queries and LINQ expressions for c-
entron.NET. Detects N+1 queries, cartesian products, and compatibility
issues. Use when writing complex queries or experiencing performance
problems. Keywords: NHibernate, LINQ, query, performance, N+1, optimization,
Fetch.
---

# NHibernate Query Reviewer Agent

> **Type**: Review / Analysis
> **Purpose**: Review database queries to ensure efficiency, proper
structure, and compatibility with NHibernate's LINQ provider limitations.

## Agent Role

You are a specialized **NHibernate Query Reviewer** for the c-entron.NET
solution, focused on query optimization and performance.
```

```
### Primary Responsibilities

1. **N+1 Detection**: Identify and fix lazy loading issues that cause
multiple database roundtrips
2. **Performance Analysis**: Review queries for cartesian products, missing
indexes, and inefficient patterns
3. **NHibernate Compatibility**: Ensure LINQ expressions translate correctly
to SQL
4. **Best Practices**: Enforce soft delete filtering, eager loading
strategies, and proper transaction usage

### Core Capabilities

- **N+1 Query Detection**: Identify lazy loading in loops causing
performance degradation
- **Cartesian Product Prevention**: Detect multiple Fetch operations on
collections
- **LINQ Compatibility**: Validate expressions work with NHibernate's LINQ
provider
- **Optimization Recommendations**: Suggest Fetch, FetchMany, Future queries
for better performance
- **Soft Delete Validation**: Ensure all queries filter IsDeleted records

## When to Invoke This Agent

This agent should be activated when:
- Complex LINQ queries are written
- Performance issues suspected with database access
- Need query optimization recommendations
- Validating NHibernate compatibility of LINQ expressions
- Reviewing data access code for N+1 problems
- Before committing database access code

**Trigger examples:**
- "Review this query for N+1 problems"
- "Optimize the GetAccountContracts query"
- "Check if this LINQ expression will work with NHibernate"
- "Why is my query slow?"

## Technology Adaptation

**IMPORTANT**: This agent adapts to c-entron.NET's NHibernate configuration.

**Configuration Source**: [CLAUDE.md](../../CLAUDE.md)

Before beginning work, review CLAUDE.md for:
- **ORM**: NHibernate 5.x with FluentNHibernate
- **Database**: SQL Server 2019+
- **Pattern**: Always filter !x.IsDeleted
- **Eager Loading**: Fetch/FetchMany for navigation properties
```

```
- **Future Queries**: Batch loading for multiple collections
- **Transactions**: Required for all modifications

## Instructions & Workflow

### Standard Procedure

1. **Load Relevant Lessons Learned** ⚠ **IMPORTANT**

   As a review and analysis agent, start by loading past lessons:

   - Use Serena MCP `list_memories` to see available memories
   - Use `read_memory` to load relevant past findings:
     - `"lesson-query-*"` - Query optimization lessons
     - `"pattern-nhibernate-*"` - NHibernate patterns
     - `"lesson-performance-*"` - Performance findings
   - Apply insights from past lessons throughout review
   - This prevents repeating past N+1 mistakes

2. **Context Gathering**
   - Review [CLAUDE.md](../../CLAUDE.md) for NHibernate patterns
   - Use Serena MCP `find_symbol` to locate query implementations
   - Use Serena MCP `find_referencing_symbols` to understand query usage
   - Identify query complexity and data access patterns

3. **Query Analysis**
   - Check for N+1 query patterns (lazy loading in loops)
   - Verify soft delete filtering (!x.IsDeleted)
   - Validate LINQ expression compatibility
   - Look for cartesian products (multiple Fetch on collections)
   - Check transaction usage for modifications
   - **Apply insights from loaded lessons**

4. **Optimization**
   - Suggest Fetch/FetchMany for eager loading
   - Recommend Future queries for multiple collections
   - Propose projection for limited data needs
   - Identify missing indexes
   - **Check recommendations against past patterns**

5. **Verification**
   - Estimate performance impact
   - Verify proposed optimizations don't introduce new issues
   - Use `/optimize` command for additional suggestions
```

Hinweis: Der Auszug endet nach Zeile 100; die Originaldatei umfasst 284 Zeilen und ist an dieser Stelle nicht zu Ende.

4.4.6 Beispielhafte Ergebnis-Use-Cases

Quelle: `Versuche/Versuch 03/ERP_DOCUMENTATION/USE_CASES.md`

```
### 2. Click Counter Management (Usage-Based Billing)
**Purpose**: Retrieve current meter readings for click counter devices
**Use Cases**: Copy machines, printers, industrial equipment with usage
meters
Method: LoadCounterAsync(List<int> contractsI3D)
```

```
**Use Case**: Track counter reading trends, detect anomalies
Method: IAutomatedBillingLogic.GetCounterHistory(List<string> lstParam)
```

4.4.7 MCP-Server: Detaillierte Beschreibung

4.4.7.1 MCP-Grundprinzip

Model Context Protocol (MCP) definiert eine standardisierte Kopplung zwischen einem Host (hier: Claude Code), einem MCP-Client und einem oder mehreren MCP-Servern. Server stellen dabei typischerweise drei Artefaktarten bereit: `tools` (aufrufbare Funktionen), `resources` (lesbare Kontexte) und `prompts` (wiederverwendbare Prompt-Bausteine). Dieses Modell wurde in Versuch 03 genutzt, um ueber den reinen Repository-Kontext hinaus weitere Wissens- und Interaktionskanaele einzubinden (Model Context Protocol, 2026a).

4.4.7.2 Serena MCP

Serena ist ein MCP-Server fuer semantische Code-Retrieval- und Editieroperationen auf Symbol-Ebene (z. B. `find_symbol`, `find_referencing_symbols`, `insert_after_symbol`). Im Unterschied zu rein textbasierter Suche werden Codeobjekte (Klassen, Methoden, Referenzen) strukturell adressiert. In Versuch 03 wurde Serena vor allem fuer gezielte Modulnavigation und die persistenten Memory-Notizen zwischen Analyseiterationen eingesetzt (Model Context Protocol, 2026b; oraios, 2026).

4.4.7.3 Windows-MCP (AutoIt-basiert)

Der im Protokoll genannte Windows-MCP-Ansatz (AutoIt-basiert) realisiert Desktop-Automatisierung ueber MCP. Laut Projektbeschreibung kapselt der Server AutoIt-Funktionen als MCP-Tools und bietet zusaetzlich Ressourcen (Dateizugriff, Screenshots) sowie Prompt-Templates fuer typische Automationsaufgaben. Fuer die Fallstudie ist das relevant, weil GUI-basierte Pfade (Dialoge, Formulare, visuelle Workflows) nicht nur aus Quellcode, sondern auch aus Interaktionsablaeufen rekonstruiert werden koennen (mario-andreschak, 2026).

4.4.7.4 MSSQL MCP

MSSQL MCP ermoeglicht den kontrollierten Zugriff auf Microsoft SQL Server ueber MCP. Typische Funktionen sind Tabellenauflistung, Schema-Inspektion, Lesen von Inhalten und kontrollierte SQL-Ausfuehrung. Die dokumentierten Security-Hinweise betonen Least-Privilege-Berechtigungen, restriktive Verbindungskonfigurationen und Logging. In Versuch 03 wurde dieser Zugriff fuer die funktionale Absicherung von Datenmodellen und Use-Case-Hypothesen genutzt (JexinSam, 2026).

4.4.8 Einordnung

Durch die MCP-Erweiterung konnte Versuch 03 die funktionale Breite deutlich steigern und einen grossen Dokumentations-Gap sichtbar machen. Gegenueber Versuch 02 sinkt dabei der formale ISO-Fokus, was fuer Discovery jedoch methodisch beabsichtigt war.

4.5 Quellenhinweis

Die fuer dieses Kapitel genutzten Webquellen zu Claude Code und MCP-Servern sind im Literaturverzeichnis als Online-Quellen erfasst; die inhaltliche Referenzierung erfolgt direkt im Text der Abschnitte zu Versuch 03.

5 Ergebnisse (ca. 10 Seiten)

Dieses Kapitel dokumentiert die tatsaechlich erzeugten Ergebnisse der drei Versuche (V01-V03). Neben den Kennzahlen werden die Ergebnisartefakte aus den jeweiligen Ergebnisverzeichnissen strukturiert aufgelistet und durch exemplarische Requirements bzw. Use Cases belegt.

5.1 Ergebnisueberblick

Kennzahl	V01	V02	V03
Konsolidierte Anforderungen/ Faehigkeiten	277	220	1720
Formale Anforderungen (StRS+SyRS+SwRS)	277	220	0
Explizite Use Cases	0	46	1720
Undokumentierte Use Cases	n.v.	n.v.	1211
ISO-29148-Compliance	qualitativ A+	96,1%	n.v.
Traceability	100% laut Doku	100% bidirektional	n.v.
Ergebnisdateien gesamt	11	37	30

5.2 V01 Ergebnisse (Baseline)

5.2.1 Ergebnisdateien in `Versuche/Versuch 01/Ergebnisse`

- `Versuche/Versuch 01/Ergebnisse/Centron_Software_Requirements_Specification.md`
- `Versuche/Versuch 01/Ergebnisse/Centron_Software_Requirements_Specification.pdf`
- `Versuche/Versuch 01/Ergebnisse/complete-iso29148-requirements-specification.md`
- `Versuche/Versuch 01/Ergebnisse/ISO29148_Complete_Requirements_Specification.md`
- `Versuche/Versuch 01/Ergebnisse/iso29148-integrated-requirements-analysis.md`
- `Versuche/Versuch 01/Ergebnisse/iso29148-integrated-requirements-analysis.pdf`
- `Versuche/Versuch 01/Ergebnisse/nhibernate-orm-analysis.md`
- `Versuche/Versuch 01/Ergebnisse/software/SwRS_Complete_Detailed.md`
- `Versuche/Versuch 01/Ergebnisse/software/SwRS_Complete_Detailed.pdf`
- `Versuche/Versuch 01/Ergebnisse/system/SyRS_Complete_Detailed.md`
- `Versuche/Versuch 01/Ergebnisse/system/SyRS_Complete_Detailed.pdf`

5.2.2 Beispielhafte Requirements aus den Ergebnisdateien

```
StR-001: Comprehensive Customer Account Management
Statement: The system shall provide comprehensive customer account
management capabilities...
(Quelle: ISO29148_Complete_Requirements_Specification.md)
```

```
FR-001: User Authentication System
Requirement: The system shall provide secure user authentication...
(Quelle: system/SyRS_Complete_Detailed.md)
```

5.3 V02 Ergebnisse (ISO-Konsolidierung mit Agenten)

5.3.1 Ergebnisdateien in `Versuche/Versuch 02/Ergenisse`

- Versuche/Versuch 02/Ergenisse/COMPLETE_REQUIREMENTS_SPECIFICATION.md
- Versuche/Versuch 02/Ergenisse/COMPLETE_REQUIREMENTS_SPECIFICATION.pdf
- Versuche/Versuch 02/Ergenisse/README.md
- Versuche/Versuch 02/Ergenisse/TABLE_FORMATTING_STATUS.md
- Versuche/Versuch 02/Ergenisse/.execution_state/baseline_metrics.json
- Versuche/Versuch 02/Ergenisse/.execution_state/directory_setup.txt
- Versuche/Versuch 02/Ergenisse/.execution_state/milestone_state.json
- Versuche/Versuch 02/Ergenisse/.execution_state/project_structure.json
- Versuche/Versuch 02/Ergenisse/master/ISO29148_Executive_Summary.md
- Versuche/Versuch 02/Ergenisse/master/ISO29148_Master_Requirements.md
- Versuche/Versuch 02/Ergenisse/master/ISO29148_Quality_Report.md
- Versuche/Versuch 02/Ergenisse/master/ISO29148_Traceability_Master.csv
- Versuche/Versuch 02/Ergenisse/master/ISO29148_Validation_Checklist.md
- Versuche/Versuch 02/Ergenisse/software/Analysis_Complete.md
- Versuche/Versuch 02/Ergenisse/software/Business_Rules.md
- Versuche/Versuch 02/Ergenisse/software/Integration_Patterns.md
- Versuche/Versuch 02/Ergenisse/software/Pattern_Catalog.csv
- Versuche/Versuch 02/Ergenisse/software/Performance_Patterns.md
- Versuche/Versuch 02/Ergenisse/software/Security_Patterns.md
- Versuche/Versuch 02/Ergenisse/software/SwRS_Algorithms.md
- Versuche/Versuch 02/Ergenisse/software/SwRS_CodeCatalog.md
- Versuche/Versuch 02/Ergenisse/software/SwRS_Complete.md
- Versuche/Versuch 02/Ergenisse/software/SwRS_DataModel.md
- Versuche/Versuch 02/Ergenisse/software/SwRS_TestSpecification.md
- Versuche/Versuch 02/Ergenisse/software/SwRS_Traceability.csv
- Versuche/Versuch 02/Ergenisse/software/Validation_Rules.md
- Versuche/Versuch 02/Ergenisse/stakeholder/StRS_Complete.md
- Versuche/Versuch 02/Ergenisse/stakeholder/StRS_Diagrams.md
- Versuche/Versuch 02/Ergenisse/stakeholder/StRS_Evidence.md
- Versuche/Versuch 02/Ergenisse/stakeholder/StRS_Summary.md
- Versuche/Versuch 02/Ergenisse/stakeholder/StRS_Traceability.csv
- Versuche/Versuch 02/Ergenisse/system/SyRS_API_Specification.yaml
- Versuche/Versuch 02/Ergenisse/system/SyRS_Architecture.md
- Versuche/Versuch 02/Ergenisse/system/SyRS_Complete.md
- Versuche/Versuch 02/Ergenisse/system/SyRS_Interfaces.md

- Versuche/Versuch 02/Ergenisse/system/SyRS_Summary.md
- Versuche/Versuch 02/Ergenisse/system/SyRS_Traceability.csv

5.3.2 Beispielhafte Requirements aus den Ergebnisdateien

```
SyR-001: The system SHALL implement a multi-layered architecture with clear
separation of concerns.
(Quelle: Ergenisse/system/SyRS_Complete.md)
```

```
SyR-013: The system SHALL provide secure user authentication with multi-
factor authentication support.
(Quelle: Ergenisse/system/SyRS_Complete.md)
```

```
SW-ARCH-001: The software SHALL implement a 6-layer architecture pattern.
(Quelle: Ergenisse/software/SwRS_Complete.md)
```

5.4 V03 Ergebnisse (Discovery-Erweiterung mit Agenten und MCP)

5.4.1 Ergebnisdateien in `Versuche/Versuch 03/ERP_DOCUMENTATION`

- Versuche/Versuch 03/ERP_DOCUMENTATION/ANALYSIS_SUMMARY.md
- Versuche/Versuch 03/ERP_DOCUMENTATION/BUSINESS_GLOSSAR_MIT_DB_MAPPING.md
- Versuche/Versuch 03/ERP_DOCUMENTATION/BUSINESS_GLOSSAR.md
- Versuche/Versuch 03/ERP_DOCUMENTATION/BUSINESS_GLOSSAR.pdf
- Versuche/Versuch 03/ERP_DOCUMENTATION/COMPLETE_DATABASE_SCHEMA.md
- Versuche/Versuch 03/ERP_DOCUMENTATION/DOCUMENTATION_INDEX.md
- Versuche/Versuch 03/ERP_DOCUMENTATION/EXPORT_COMPLETE_SCHEMA.sql
- Versuche/Versuch 03/ERP_DOCUMENTATION/README_USE_CASE_ANALYSIS.md
- Versuche/Versuch 03/ERP_DOCUMENTATION/README.md
- Versuche/Versuch 03/ERP_DOCUMENTATION/SCREENSHOT_ANALYSIS_SUMMARY.md
- Versuche/Versuch 03/ERP_DOCUMENTATION/SCREENSHOT_MAPPING_COMPLETE.md
- Versuche/Versuch 03/ERP_DOCUMENTATION/SCREENSHOT_PROJECT_INDEX.md
- Versuche/Versuch 03/ERP_DOCUMENTATION/SSMS_DB_SCHEMA.sql
- Versuche/Versuch 03/ERP_DOCUMENTATION/
  UNDOCUMENTED_USE_CASES_DATABASE_MODELS.md
- Versuche/Versuch 03/ERP_DOCUMENTATION/UNDOCUMENTED_USE_CASES_REST_API.md
- Versuche/Versuch 03/ERP_DOCUMENTATION/UNDOCUMENTED_USE_CASES_SUMMARY.md
- Versuche/Versuch 03/ERP_DOCUMENTATION/UNDOCUMENTED_USE_CASES_WORKFLOWS.md
- Versuche/Versuch 03/ERP_DOCUMENTATION/USE_CASE_ANALYSIS_README.md
- Versuche/Versuch 03/ERP_DOCUMENTATION/USE_CASE_MAPPING.md
- Versuche/Versuch 03/ERP_DOCUMENTATION/USE_CASES_CENTRON_NEXUS_DE.md
- Versuche/Versuch 03/ERP_DOCUMENTATION/USE_CASES_CENTRON_NEXUS_DE.pdf
- Versuche/Versuch 03/ERP_DOCUMENTATION/USE_CASES_CENTRON_NEXUS.md
- Versuche/Versuch 03/ERP_DOCUMENTATION/USE_CASES_CENTRON_NEXUS.pdf
- Versuche/Versuch 03/ERP_DOCUMENTATION/USE_CASES_NEW_CONTROLLERS.md
- Versuche/Versuch 03/ERP_DOCUMENTATION/USE_CASES_NEW_GUI_MAPPING.md
- Versuche/Versuch 03/ERP_DOCUMENTATION/USE_CASES_NEW_IMPLEMENTATION_GUIDE.md
- Versuche/Versuch 03/ERP_DOCUMENTATION/USE_CASES_NEW_XAML_TEMPLATES.md
- Versuche/Versuch 03/ERP_DOCUMENTATION/USE_CASES_NEW.md
- Versuche/Versuch 03/ERP_DOCUMENTATION/USE_CASES.md

- `Versuche/Versuch 03/ERP_DOCUMENTATION/USE_CASES.pdf`

5.4.2 Beispielhafte Use Cases aus den Ergebnisdateien

```
1.1.1 Personalized User Welcome
Purpose: Display personalized greeting with user name and context-aware
dashboard content.
(Quelle: ERP_DOCUMENTATION/USE_CASES_CENTRON_NEXUS.md)
```

```
1.1.6 Work Status Alerts
Purpose: Alert users to missing or incomplete work time entries.
(Quelle: ERP_DOCUMENTATION/USE_CASES_CENTRON_NEXUS.md)
```

```
Key Finding: 1,720+ use cases discovered; current documentation gap: 71%.
(Quelle: ERP_DOCUMENTATION/UNDOCUMENTED_USE_CASES_SUMMARY.md)
```

5.5 Ergebnisfazit

Die Ergebnislage zeigt drei komplementaere Stufen:

- V01 liefert eine belastbare formale Baseline.
- V02 liefert die staerkste ISO-29148-konforme Konsolidierung mit hoher Traceability.
- V03 liefert die groesste funktionale Breite und identifiziert den groessten Dokumentations-Gap.

Damit liegt eine vollstaendige empirische Grundlage fuer die anschliessende Evaluation (Kapitel 6) vor: formal-strukturierte Requirements (V01/V02) plus breite Discovery-Evidenz (V03).

6 Evaluation (ca. 12 Seiten)

Die Evaluation folgt der in Kapitel 4 beschriebenen Iterationslogik und bewertet die drei real durchgefuehrten Versuche (V01-V03) vergleichend. Ziel ist nicht die Darstellung eines einzelnen "besten" Laufs, sondern die Einordnung der methodischen Entwicklung von einer Baseline ueber eine formale ISO-Konsolidierung bis zur anschliessenden Discovery-Erweiterung.

6.1 Evaluationsdesign und Datenbasis

Die Auswertung basiert ausschliesslich auf den erzeugten Artefakten in:

- `Versuche/Versuch 01/Versuch01.md` und `Versuche/Versuch 01/Requirements.md`,
- `Versuche/Versuch 02/Versuch02.md` und `Versuche/Versuch 02/Requirements.md`,
- `Versuche/Versuch 03/Versuch03.md` und `Versuche/Versuch 03/Requirements.md`.

Dabei wurden nur konsolidierte, in den Dateien ausgewiesene Kennzahlen uebernommen. Fokus der Bewertung:

1. Umfang der rekonstruierten Faehigkeiten/Requirements,
2. Formalisierungsgrad (StRS/SyRS/SwRS vs. reine Use-Case-Discovery),
3. Traceability- und ISO-29148-Naehe,
4. methodischer Nutzen der eingesetzten Tooling-Konfiguration.

6.2 Quantitative Ergebnisse der Versuchsreihe

Kennzahl	V01	V02	V03

Konsolidierte Requirements/ Faehigkeiten	277	220	1720
Formale Requirements (StRS+SyRS+SwRS)	277	220	0
StRS / SyRS / SwRS	35 / 75 / 167	84 / 53 / 83	0 / 0 / 0
Explizite Use Cases	0	46	1720 (Use-Case-fokussiert)
Undokumentierte Use Cases	n.v.	n.v.	1211
ISO-29148-Compliance	qualitativ A+	96,1% (100% mandatory)	n.v.
Traceability	100% laut Doku	100% bidirektional	n.v.
Ergebnisdateien gesamt	11	37	30

Ergaenzende Kontextkennzahlen aus den Versuchsdateien:

- V01: Analyse von 34 C#-Projekten und 12.507+ Source Files.
- V02: 14.940 Dateien (13.717 C#, 1.189 XAML, 34 Projekte), 46 explizite Use Cases in die formale Requirements-Struktur integriert.
- V03: 150.000+ LoC analysiert, 3.412 potenzielle Use Cases identifiziert, 71% dokumentationsbezogener Gap (1211 von 1720 Use Cases vormals undokumentiert).

6.3 Vergleichende Analyse

6.3.1 Versuch 01: Formale Baseline ohne Tooling-Erweiterung

V01 zeigt, dass bereits ohne Agenten/MCP eine formal strukturierte Requirements-Spezifikation erzeugt werden kann. Die Staerke liegt in der klaren Dreiebenenstruktur (StRS/SyRS/SwRS). Die Schwaeche ist die begrenzte Discovery-Perspektive: explizite Use-Case-Rekonstruktion und Gap-Bewertung bleiben gering ausgepraegt.

6.3.1.1 Prompt, Agenten und Ergebnisbeispiele (V01)

- Verwendeter Prompt: “Please analyze this software project and write a reuqirements specification according to modern standards.”
- Agentenbeispiele: Keine Agenten (bewusste Baseline ohne agentische Zerlegung und ohne MCP).
- Beispielhafte Ergebnis-Requirements:
  - `Versuche/Versuch 01/Ergebnisse/ISO29148_Complete_Requirements_Specification.md`: u. a. `StR-001` (Comprehensive Customer Account Management).
  - `Versuche/Versuch 01/Ergebnisse/system/SyRS_Complete_Detailed.md`: u. a. `FR-001` (User Authentication System) und `FR-002` (Role-Based Access Control).
  - `Versuche/Versuch 01/Ergebnisse/software/SwRS_Complete_Detailed.md`: softwareseitige Architektur- und Umsetzungsanforderungen im SwRS-Format.

6.3.2 Versuch 02: ISO-orientierte Konsolidierung mit Agenten

V02 fokussiert die formale Konsolidierung und liefert eine ISO-29148-nahe Zielstruktur mit hoher Traceability. Mit 220 konsolidierten Requirements, 96,1% ISO-29148-Compliance und 100% bidirektionaler Traceability ist der Lauf methodisch sauber und reviewfaehig. Gleichzeitig zeigte sich

die zentrale Grenze dieses Schritts: Die reine ISO-orientierte Ableitung war fuer den Gesamtumfang zu rigide und fuer die Discovery-Breite nicht vollumfaenglich genug.

6.3.2.1 Prompt, Agenten und Ergebnisbeispiele (V02)

- Verwendeter Prompt: “Please analyze this software project and write a ISO 29148 compliant reuqirements specification. Use Agents wherever possible.”
- Agentenbeispiele:
  - `Versuche/Versuch 02/Tools/agents/iso29148-master-orchestrator-agent.md`
  - `Versuche/Versuch 02/Tools/agents/iso29148-stakeholder-agent.md`
  - `Versuche/Versuch 02/Tools/agents/iso29148-system-requirements-agent.md`
  - `Versuche/Versuch 02/Tools/agents/iso29148-software-requirements-agent`
- Beispielhafte Ergebnis-Requirements:
  - `Versuche/Versuch 02/Ergenisse/system/SyRS_Complete.md`: u. a. `SyR-001` (Multi-Layer Architecture), `SyR-002` (Dual Data Access Pattern), `SyR-013` (Authentication).
  - `Versuche/Versuch 02/Ergenisse/software/SwRS_Complete.md`: u. a. `SW-ARCH-001` (6-Layer Architecture), `SW-ARCH-002` (ILogic-Pattern), `SW-FUNC-001` (Account Management).
  - `Versuche/Versuch 02/Ergenisse/master/ISO29148_Quality_Report.md`: qualitaetssichernde Gesamtbewertung (u. a. 100% Traceability).

6.3.3 Versuch 03: Discovery-Erweiterung mit Agenten und MCP

V03 erweitert deshalb die Methodik um MCP-gestuetzte Discovery. Der Lauf vergroessert die funktionale Breite deutlich (1720 konsolidierte Faehigkeiten, davon 1211 vormals undokumentierte Use Cases) und eignet sich besonders fuer Gap-Analysen und Vollstaendigkeitspruefung. Die Kehrseite ist ein geringerer Formalisierungsgrad gegenueber der ISO-Konsolidierung.

6.3.3.1 Prompt, Agenten und Ergebnisbeispiele (V03)

- Verwendeter Prompt: “Please analyze this software project and write a reuqirements specification according to modern standards. Use Agents and MCP servers wherever possible. Keep superflous texts to a minimum and concentrate on actual requirements.”
- Agentenbeispiele:
  - `Versuche/Versuch 03/Tools/Agents/centron-documentation-writer.md`
  - `Versuche/Versuch 03/Tools/Agents/nhibernate-query-reviewer.md`
  - `Versuche/Versuch 03/Tools/Agents/centron-code-reviewer.md`
  - `Versuche/Versuch 03/Tools/Agents/webservice-developer.md`
- MCP-Beispiele: Serena-MCP (Memory), Windows-MCP (UI-Interaktion), MSSQL-MCP (DB-Schemazugriff).
- Beispielhafte extrahierte Use-Case-/Anforderungsartefakte:
  - `Versuche/Versuch 03/ERP_DOCUMENTATION/USE_CASES_CENTRON_NEXUS.md`: u. a. Use Cases `1.1.1` (Personalized User Welcome), `1.1.6` (Work Status Alerts), `3.1` (Quick Ticket Creation).
  - `Versuche/Versuch 03/ERP_DOCUMENTATION/USE_CASES.md`: moduluebergreifende, strukturierte Use-Case-Dokumentation fuer c-entron.NET.
  - `Versuche/Versuch 03/ERP_DOCUMENTATION/UNDOCUMENTED_USE_CASES_SUMMARY.md`: 1.720+ Use Cases und ca. 71% Dokumentations-Gap als Discovery-Nachweis.

6.4 Abgleich mit den geplanten Methoden

Der Soll-Ist-Abgleich zeigt eine hohe Passung zur geplanten Gesamtmethodik, wenn diese als iterative Kombination aus **Discovery** und **Konsolidierung** verstanden wird:

- Die Standardrecherche (ISO/IEC/IEEE 29148) wurde fruehzeitig umgesetzt.

- Ein Baseline-Lauf ohne Spezialisierung wurde durchgefuehrt (V01).
- Eine strukturierte ISO-Konsolidierung wurde realisiert (V02).
- Danach wurde die Abdeckung durch MCP-gestuetzte Discovery erweitert (V03), weil der ISO-Lauf allein zu rigide und nicht vollumfaenglich genug war.

Abweichung zur urspruenglich linearen Planung: Stakeholder-Interviews und flaechendeckende fachliche Reviews wurden in der betrachteten Phase noch nicht vollstaendig abgeschlossen. Die Methodik wird deshalb in der Ergebnisinterpretation als "technisch validierte Vorstufe" einer finalen fachlichen Konsolidierung eingeordnet.

6.5 Bewertung der Forschungsleitfragen auf Basis der aktuellen Evidenz

- F1 (reproduzierbarer LLM-Einsatz): beantwortbar. Die drei Versuche zeigen, dass reproduzierbare Prozessschritte und klar unterscheidbare Konfigurationen moeglich sind.
- F2 (Ableitung aus Code vs. Zusatzquellen): teilweise beantwortbar. Codebasierte Extraktion funktioniert, video- und interviewbasierte Ergaenzungen sind noch offen.
- F3 (Qualitaet aus Expertensicht): noch nicht abschliessend beantwortbar, da systematische Expertenratings nicht vollstaendig dokumentiert vorliegen.
- F4 (Chancen und Grenzen): beantwortbar. Chancen liegen in Skalierung und Strukturierung; Grenzen in Halluzinationsrisiken, fehlender Vollstaendigkeit ohne Zusatzquellen und hohem Konsolidierungsbedarf.

6.6 Limitationen

Die aktuelle Evidenz ist durch drei Punkte begrenzt:

1. Vollstaendige Video-Transkription und -Auswertung fehlen noch.
2. Ein methodischer Endabgleich zwischen Video- und Codeperspektive ist noch nicht abgeschlossen.
3. Die fachliche Endklassifikation aller Use-Case-Cluster (Ja/Nein/Neu/TBD) liegt noch nicht durchgaengig vor.

Diese Limitationen betreffen vor allem die finale Vollstaendigkeitsaussage, nicht jedoch die grundlegende Wirksamkeit der iterativen Methodik.

7 Diskussion (ca. 8 Seiten)

7.1 Interpretation der Ergebnisse

Die Ergebnisse zeigen einen klaren methodischen Lerneffekt ueber die drei Iterationen. Der Verlauf von V01 ueber V02 zu V03 ist nicht als Widerspruch, sondern als komplementaere Reifung zu interpretieren:

- V01 demonstriert, dass bereits mit einfacher Konfiguration formal strukturierte Requirements ableitbar sind.
- V02 zeigt, dass eine agentengestuetzte ISO-Konsolidierung methodisch sauber, aber fuer den Gesamtumfang zu rigide sein kann.
- V03 zeigt, dass die MCP-Erweiterung die funktionale Breite massiv erhoeht und Discovery-Luecken schliesst.

In Summe entsteht ein zweistufiges Zielbild fuer Reverse Requirements Engineering in Legacy-Projekten: zuerst **formal konsolidieren**, danach **gezielt in die Breite erweitern**.

7.2 Chancen und Grenzen

Die wesentlichen Chancen des Ansatzes liegen in:

- hoher Skalierbarkeit bei grossen Legacy-Artefakten,
- schneller Sichtbarmachung undokumentierter Funktionalitaet,
- strukturierter Ueberfuehrung in reviewbare Requirements-Artefakte.

Die zentralen Grenzen bleiben:

- keine belastbare Vollstaendigkeit ohne Zusatzquellen (insbesondere Nutzungs- und Prozesssicht),
- Halluzinations- und Fehlinterpretationsrisiken ohne Beleg- und Reviewpflicht,
- hoher Konsolidierungsaufwand zwischen Discovery-Artefakten und abnahmefaehiger Spezifikation.

Damit bestaetigt die Fallstudie, dass LLMs Requirements Engineering nicht ersetzen, aber als beschleunigendes Analyseinstrument mit klaren Governance-Regeln substantiellen Mehrwert liefern.

7.3 Implikationen fuer Forschung und Praxis

Fuer die Praxis folgt daraus ein umsetzbarer Einfuehrungspfad:

1. Iterative Versuchslogik statt einmaliger "Big-Bang"-Extraktion.
2. Trennung von Discovery- und Konsolidierungsphase als Standard.
3. Traceability als verpflichtendes Abnahmekriterium fuer LLM-Ergebnisse.

Fuer die Forschung ergeben sich drei Anschlussfragen:

1. Wie laesst sich die Triangulation aus Code-, Video- und Stakeholderdaten automatisiert zusammenfuehren?
2. Welche Metriken messen Qualitaet von Requirements-Artefakten robuster als reine Umfangszahlen?
3. Wie kann Human-in-the-loop-Validierung mit vertretbarem Aufwand skaliert werden?

Die vorliegende Arbeit liefert dafuer eine belastbare methodische Ausgangsbasis, zeigt aber zugleich, dass die letzte Meile zur fachlich finalen Spezifikation weiterhin ein kooperativer Mensch-KI-Prozess bleibt.

8 Fazit und Ausblick (ca. 4 Seiten)

8.1 Zusammenfassung und Beantwortung der Forschungsfragen

Die Arbeit zeigt, dass KI-gestuetztes Reverse Requirements Engineering im untersuchten Legacy-ERP-Kontext praktikabel ist, wenn der Prozess iterativ und kontrolliert aufgebaut wird. Die drei durchgefuehrten Versuche liefern dabei komplementaere Staerken:

- V01 liefert eine formale Baseline mit klarer Requirements-Struktur.
- V02 konsolidiert die Erkenntnisse in eine ISO-29148-nahe, traceability-starke Spezifikation.
- V03 erweitert die Discovery-Breite per MCP und deckt einen hohen dokumentationsbezogenen Gap auf.

Damit ist F1 (prozessuale Einsetzbarkeit von LLMs) positiv beantwortet. F4 (Chancen und Grenzen) ist ebenfalls klar beantwortbar: Hohe Effizienz- und Strukturgewinne stehen einem weiterhin relevanten Validierungs- und Konsolidierungsbedarf gegenueber. F2 und F3 sind teilweise beantwortet, da video- und interviewbasierte Endvalidierung noch nicht vollstaendig abgeschlossen ist.

8.2 Handlungsempfehlungen fuer c-entron GmbH

Aus den Ergebnissen lassen sich folgende priorisierte Handlungsschritte ableiten:

1. V02 als Spezifikationsbasis verwenden: Die 220 konsolidierten Requirements mit hoher Traceability als Arbeitsgrundlage fuer die Web-Migration etablieren.
2. V03 als Discovery-Backlog nutzen: Die 1720 identifizierten Faehigkeiten systematisch gegen V02 mappen, um potenzielle Luecken sichtbar zu halten.
3. Review-Governance fest verankern: Fachliche Freigaben und Aenderungsentscheidungen pro Requirement dokumentieren (kein unreviewter LLM-Output im Zielbacklog).
4. Toolchain standardisieren: Prompt-/Agentenkonfigurationen versionieren, damit Folgeanalysen reproduzierbar bleiben.

8.3 Ausblick und naechste Schritte

Die naechste Arbeitsphase erweitert die bisher codezentrierte Evidenz um die noch offenen Schritte aus dem Protokoll:

1. Vollstaendige Videoanalyse: Alle vorhandenen Schulungsvideos KI-gestuetzt transkribieren und strukturiert auf Use Cases auswerten.
2. Abgleich Video vs. Codeanalyse: Systematischer Vergleich, ob und wo sich beide Sichten decken bzw. welche Use Cases nur in einer Quelle auftauchen.
3. Clusterung in abstrakte Konzepte: Die identifizierten Use Cases in die bereits vorbereiteten 101 abstrahierten Konzepte ueberfuehren (vgl. `A_Videoanalyse_Uebersicht.csv`).
4. Manuelle Fachklassifikation pro Cluster: Bewertung in die Kategorien
   - ja: unveraenderte Uebernahme,
   - nein: Entfall in ERP Web,
   - neu: fachlich vorhanden, aber neu zu konzipieren,
   - TBD: vorlaeufig offen.

Erst mit dieser finalen Triangulation aus Code, Video und Fachbewertung ist eine belastbare Vollstaendigkeitsaussage fuer die Migrationsplanung moeglich.

9 Literaturverzeichnis (ca. 3 Seiten)

Bibliography

(2014). *Proceedings of the 11th International Workshop on Security in Information Systems*, 26–37. https://doi.org/10.5220/0004979900260037

Anthropic. (2025,). *Claude Code Can Now Connect to Your Remote MCP Servers*. https://www.anthropic.com/news/claude-code-remote-mcp

Anthropic. (2026b,). *Claude Code Documentation: CLI Usage*. https://code.claude.com/docs/en/cli-usage

Anthropic. (2026c,). *Claude Code Documentation: IDE Integrations*. https://code.claude.com/docs/en/ide-integrations

Anthropic. (2026d,). *Claude Code Documentation: MCP*. https://code.claude.com/docs/en/mcp

Anthropic. (2026a,). *Claude Code Documentation: Quickstart*. https://code.claude.com/docs/en/quickstart

B, L., Wiegers, K., & Ebert, C. (2001). The top risk of requirements engineering. *IEEE Software*, *18*(6), 62–63. https://doi.org/10.1109/52.965804

Bender, E. M., Gebru, T., McMillan-Major, A., & Shmitchell, S. (2021). On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? 🦜. *Proceedings of the 2021 ACM Conference on Fairness, Accountability, And Transparency*, 610–623. https://doi.org/10.1145/3442188.3445922

Bisbal, J., Lawless, D., Wu, B., & Grimson, J. (1999). Legacy information systems: issues and directions. *IEEE Software*, *16*(5), 103–111. https://doi.org/10.1109/52.795108

Bisbal, J., Lawless, D., Wu, B., Grimson, J., Wade, V., Richardson, R., & O'Sullivan, D. (n.d.). An overview of legacy information system migration. *Proceedings of Joint 4th International Computer Science Conference and 4th Asia Pacific Software Engineering Conference*, 529–530. https://doi.org/10.1109/apsec.1997.640219

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer. https://link.springer.com/book/9780387310732

Borg, M., Runeson, P., & Ardö, A. (2013). Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. *Empirical Software Engineering*, *19*(6), 1565–1616. https://doi.org/10.1007/s10664-013-9255-y

Chikofsky, E., & Cross, J. (1990). Reverse engineering and design recovery: a taxonomy. *IEEE Software*, *7*(1), 13–17. https://doi.org/10.1109/52.43044

Fan, A., Gokkaya, B., Harman, M., Lyubarskiy, M., Sengupta, S., Yoo, S., & Zhang, J. M. (2023). Large Language Models for Software Engineering: Survey and Open Problems. *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-Fose)*, 31–53. https://doi.org/10.1109/icse-fose59343.2023.00008

Fantechi, A., Gnesi, S., Passaro, L., & Semini, L. (2023). Inconsistency Detection in Natural Language Requirements using ChatGPT: a Preliminary Evaluation. *2023 IEEE 31st International Requirements Engineering Conference (RE)*, 335–340. https://doi.org/10.1109/re57278.2023.00045

Glinz, M. (2007). On Non-Functional Requirements. *15th IEEE International Requirements Engineering Conference (RE 2007)*, 21–26. https://doi.org/10.1109/re.2007.45

Glinz, M. (2008). A Risk-Based, Value-Oriented Approach to Quality Requirements. *IEEE Software*, *25*(2), 34–41. https://doi.org/10.1109/ms.2008.31

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. https://www.deeplearningbook.org/

Gotel, O., & Finkelstein, C. (n.d.). An analysis of the requirements traceability problem. *Proceedings of IEEE International Conference on Requirements Engineering*, 94–101. https://doi.org/10.1109/icre.1994.292398

Hassani, S. (2024). Enhancing Legal Compliance and Regulation Analysis with Large Language Models. *2024 IEEE 32nd International Requirements Engineering Conference (RE)*, 507–511. https://doi.org/10.1109/re59067.2024.00065

Hemmat, A., Sharbaf, M., Kolahdouz-Rahimi, S., Lano, K., & Tehrani, S. Y. (2025). Research directions for using LLM in software requirement engineering: a systematic review. *Frontiers in Computer Science*, *7*. https://doi.org/10.3389/fcomp.2025.1519437

Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., & Wang, H. (2023,). *Large Language Models for Software Engineering: A Systematic Literature Review*. https://arxiv.org/abs/2308.10620v6

IEEE. (1998,). *IEEE Std 830-1998: Recommended Practice for Software Requirements Specifications*. https://standards.ieee.org/standard/830-1998.html

ISO/IEC. (2011,). *ISO/IEC 25010:2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

ISO/IEC/IEEE. (2018,). *ISO/IEC/IEEE 29148:2018 Systems and software engineering — Life cycle processes — Requirements engineering*. https://www.iso.org/standard/72089.html

JexinSam. (2026,). *MSSQL MCP Server Repository*. https://github.com/JexinSam/mssql_mcp_server

Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y. J., Madotto, A., & Fung, P. (2023). Survey of Hallucination in Natural Language Generation. *ACM Computing Surveys*, *55*(12), 1–38. https://doi.org/10.1145/3571730

Kotonya, G., & Sommerville, I. (1996). Requirements engineering with viewpoints. *Software Engineering Journal*, *11*(1), 5. https://doi.org/10.1049/sej.1996.0002

Kratzke, N., & Quint, P.-C. (2017). Understanding cloud-native applications after 10 years of cloud computing - A systematic mapping study. *Journal of Systems and Software*, *126*, 1–16. https://doi.org/10.1016/j.jss.2017.01.001

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., & Kiela, D. (2020,). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. https://arxiv.org/abs/2005.11401v4

Luitel, D., Hassani, S., & Sabetzadeh, M. (2024). Improving requirements completeness: automated assistance through large language models. *Requirements Engineering*, *29*(1), 73–95. https://doi.org/10.1007/s00766-024-00416-3

Marczak-Czajka, A., & Cleland-Huang, J. (2023). Using ChatGPT to Generate Human-Value User Stories as Inspirational Triggers. *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*, 52–61. https://doi.org/10.1109/rew57809.2023.00016

mario-andreschak. (2026,). *Windows Desktop Automation MCP Server (AutoIt-based)*. https://github.com/mario-andreschak/mcp-windows-desktop-automation

Marques, N., Silva, R. R., & Bernardino, J. (2024). Using ChatGPT in Software Requirements Engineering: A Comprehensive Review. *Future Internet*, *16*(6), 180. https://doi.org/10.3390/fi16060180

Model Context Protocol. (2026b,). *Model Context Protocol Servers Repository*. https://github.com/modelcontextprotocol/servers

Model Context Protocol. (2026a,). *Model Context Protocol: Getting Started Introduction*. https://modelcontextprotocol.io/docs/getting-started/intro

Norheim, J. J., & Rebentisch, E. (2024). Structuring Natural Language Requirements with Large Language Models. *2024 IEEE 32nd International Requirements Engineering Conference Workshops (REW)*, 68–71. https://doi.org/10.1109/rew61692.2024.00013

Nouri, A., Cabrero-Daniel, B., Törner, F., Sivencrona, H., & Berger, C. (2024). Engineering Safety Requirements for Autonomous Driving with Large Language Models. *2024 IEEE 32nd International Requirements Engineering Conference (RE)*, 218–228. https://doi.org/10.1109/re59067.2024.00029

Okamoto, R., & Kusumoto, S. (2025). Towards the Automatic Restructuring of Software Requirements Specifications to Conform to Standards Using Large Language Models. *2025 IEEE*

33rd International Requirements Engineering Conference (RE), 467–475. https://doi.org/10.1109/re63999.2025.00056

OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., Avila, R., Babuschkin, I., Balaji, S., Balcom, V., Baltescu, P., Bao, H., Bavarian, M., Belgum, J., … Zoph, B. (2023,). *GPT-4 Technical Report*. https://arxiv.org/abs/2303.08774v6

oraios. (2026,). *Serena MCP Server Repository*. https://github.com/mcp/oraios/serena

Pahl, C., & Jamshidi, P. (2016). Microservices: A Systematic Mapping Study. *Proceedings of the 6th International Conference on Cloud Computing and Services Science*, 137–146. https://doi.org/10.5220/0005785501370146

Pohl, K. (2010). *Requirements Engineering*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-12578-2

Rozière, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., Adi, Y., Liu, J., Sauvestre, R., Remez, T., Rapin, J., Kozhevnikov, A., Evtimov, I., Bitton, J., Bhatt, M., Ferrer, C. C., Grattafiori, A., Xiong, W., Défossez, A., … Synnaeve, G. (2023,). *Code Llama: Open Foundation Models for Code*. https://arxiv.org/abs/2308.12950v3

Salem, N., Hudaib, A., Al-Tarawneh, K., Salem, H., Tareef, A., Salloum, H., & Mazzara, M. (2024). A survey on the application of large language models in software engineering. *Computer Research and Modeling*, *16*(7), 1715–1726. https://doi.org/10.20537/2076-7633-2024-16-7-1715-1726

Sneed, H. (1995). Planning the reengineering of legacy systems. *IEEE Software*, *12*(1), 24–34. https://doi.org/10.1109/52.363168

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017,). *Attention Is All You Need*. https://arxiv.org/abs/1706.03762v7

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., & Zhou, D. (2022,). *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. https://arxiv.org/abs/2201.11903v6

Wu, B., Lawless, D., Bisbal, J., Grimson, J., Wade, V., O'Sullivan, D., & Richardson, R. (n.d.). Legacy systems migration-a method and its tool-kit framework. *Proceedings of Joint 4th International Computer Science Conference and 4th Asia Pacific Software Engineering Conference*, 312–320. https://doi.org/10.1109/apsec.1997.640188

Yu, Y., Mylopoulos, J., Wang, Y., Liaskos, S., Lapouchnian, A., Zou, Y., Littou, M., & Leite, J. (n.d.). RETR: Reverse Engineering to Requirements. *12th Working Conference on Reverse Engineering (Wcre'05)*, 234. https://doi.org/10.1109/wcre.2005.27

Zhang, Q., Fang, C., Xie, Y., Zhang, Y., Yang, Y., Sun, W., Yu, S., & Chen, Z. (2023,). *A Survey on Large Language Models for Software Engineering*. https://arxiv.org/abs/2312.15223v2

Zhao, L., Alhoshan, W., Ferrari, A., Letsholo, K. J., Ajagbe, M. A., Chioasca, E.-V., & Batista-Navarro, R. T. (2021). Natural Language Processing for Requirements Engineering: A Systematic Mapping Study. *ACM Computing Surveys*, *54*(3), 1–41. https://doi.org/10.1145/3444689

10 Anhang (ca. 6 Seiten)

10.1 Interviewleitfäden

10.2 Zusätzliches Datenmaterial

10.3 Konfigurationsdetails des Prototyps