Bachelor-Thesis

Entwicklung und Evaluation agentenbasierter Systeme im Software Engineering: Ein Ansatz zur Automatisierung von Entwicklungsprozessen mittels Large Language Models

zur Erlangung des akademischen Grades

Bachelor of Science

im dualen Studiengang Informatik an der IU Internationale Hochschule

von

Maria Musterfrau
Matrikelnummer: 1234567
Musterstraße 1, 12345 Musterstadt

12. Dezember 2025

Referent: Prof. Dr. Klaus Quibeldey-Cirkel

Korreferent: Prof. Dr. Philipp Diebold

Kurzfassung

Diese Abschlussarbeit beschäftigt sich mit der Optimierung von Prozessen im Bereich der Softwareentwicklung. Im Zentrum steht die Frage, wie etablierte Methoden durch moderne Technologien effizienter gestaltet werden können, ohne dabei bestehende Standards zu gefährden.

Die Motivation für diese Arbeit ergibt sich aus der wachsenden Komplexität von Softwareprojekten und dem gleichzeitigen Druck, Entwicklungszyklen zu verkürzen. Ein wichtiger Aspekt ist dabei die Sicherung der Code-Qualität trotz steigender Anforderungen.

Als Methode wurde eine Kombination aus Literaturrecherche, praktischer Implementierung eines Prototyps und evaluativen Studien durchgeführt. Die praktische Umsetzung basiert auf etablierten Frameworks und Best-Practices der Industrie.

Die wichtigsten Ergebnisse zeigen, dass durch gezielte Optimierungen eine Reduktion der Entwicklungszeit um durchschnittlich 20 % erreicht werden kann, während die Codequalität gleichbleibt oder sich sogar verbessert. Diese Resultate wurden durch mehrere Experimente validiert.

Insgesamt trägt diese Arbeit zu einem besseren Verständnis der Wechselwirkungen zwischen Effizienz und Qualität in der Softwareentwicklung bei und bietet praktische Ansätze für die Industrie.

Danksagung

An dieser Stelle möchte ich meinen aufrichtigen Dank aussprechen.

Zuallererst danke ich meinen Betreuern **Prof. Dr. Klaus Quibeldey-Cirkel** und **Prof. Dr. Philipp Diebold** für die fachliche Unterstützung, wertvollen Hinweise und die Möglichkeit, diese Arbeit zu verfassen. Ihre offene und konstruktive Kritik hat maßgeblich zur Qualität dieser Arbeit beigetragen.

Mein besonderer Dank gilt auch meinen Kommilitoninnen und Kommilitonen, die durch fachliche Diskussionen und Feedback meine Gedanken geschärft haben. Ihre Perspektiven haben mir geholfen, verschiedene Aspekte der Thematik tiefergehend zu beleuchten.

Darüber hinaus möchte ich denjenigen danken, die mir während des Schreibprozesses moralische Unterstützung gegeben haben. Ihre Geduld und Ermutigung waren unverzichtbar für den Erfolg dieser Arbeit.

Abschließend danke ich den Entwicklern und der Open-Source-Community für die Bereitstellung der vielfältigen Werkzeuge und Bibliotheken, ohne die diese Arbeit nicht möglich gewesen wäre.

Hinweis zur geschlechtergerechten Sprache

In dieser Arbeit wird auf eine geschlechtergerechte Sprache geachtet. Aus Gründen der besseren Lesbarkeit wird in dieser Arbeit eine Mischung verschiedener Formen geschlechtergerechter Sprache verwendet:

- **Geschlechtsneutrale Formulierungen** (z. B. Studierende, Mitarbeitende, Person)
- **Paarformen** (z. B. Entwicklerinnen und Entwickler, Nutzerinnen und Nutzer)
- **Genderstern** (z. B. Benutzer*innen) zur Sichtbarmachung aller Geschlechter

Alle verwendeten Personenbezeichnungen gelten gleichermaßen für alle Geschlechter. Diese Arbeit orientiert sich an den Empfehlungen des Leitfadens „Sag's doch gleich! Geschlechtergerechte Sprache an Thüringer Hochschulen“ des Thüringer Kompetenznetzwerks Gleichstellung (TKG).

Inhaltsverzeichnis

Abkürzungsverzeichnis

Abkürzung	Bedeutung
AI	Artificial Intelligence (Künstliche Intelligenz)
API	Application Programming Interface (Anwendungsprogrammierschnittstelle)
AST	Abstract Syntax Tree (Abstrakter Syntaxbaum)
CI/CD	Continuous Integration / Continuous Deployment
CLI	Command Line Interface (Befehlszeilenschnittstelle)
DAO	Data Access Object (Datenzugriffsobjekt)
DB	Database (Datenbank)
DL	Deep Learning (Tiefes Lernen)
ES	Expert System (Expertensystem)
FSM	Finite State Machine (Endlicher Zustandsautomat)
GNN	Graph Neural Network (Graphisches Neuronales Netzwerk)
GPU	Graphics Processing Unit (Grafik-Verarbeitungseinheit)
HTTP	HyperText Transfer Protocol
I/O	Input/Output (Ein-/Ausgabe)
IDE	Integrated Development Environment (Integrierte Entwicklungsumgebung)

Fortsetzung auf nächster Seite

Abkürzung	Bedeutung
IPC	Inter-Process Communication (Kommunikation zwischen Prozessen)
JSON	JavaScript Object Notation
KB	Knowledge Base (Wissensdatenbank)
KG	Knowledge Graph (Wissensgraph)
LLM	Large Language Model (Großes Sprachmodell)
ML	Machine Learning (Maschinelles Lernen)
NLP	Natural Language Processing (Natürlichsprachverarbeitung)
OOP	Object-Oriented Programming (Objektorientierte Programmierung)
RAG	Retrieval-Augmented Generation (Abruf-gestützte Generierung)
RDF	Resource Description Framework
REST	Representational State Transfer
RL	Reinforcement Learning (Bestärkendes Lernen)
RPC	Remote Procedure Call (Fernabruf von Verfahren)
RUST	Rapid Unsupervised Semantic Tagging
SE	Software Engineering (Software-Entwicklung)
SLA	Service Level Agreement (Dienstgütevereinbarung)
SQL	Structured Query Language (Strukturierte Abfragesprache)
SSH	Secure Shell
VCS	Version Control System (Versionskontrollsystem)
XML	Extensible Markup Language
YAML	YAML Ain't Markup Language

1 Einführung

1.1 Motivation und Relevanz

Software Engineering erlebt derzeit einen Paradigmenwechsel: *agentic AI* – also KI-Systeme, die Ziele verstehen, Pläne erstellen, Werkzeuge verwenden und Ergebnisse eigenständig verifizieren – ergänzt klassische Automatisierung um adaptive, mehrschrittige Problemlösung. Für Informatikstudierende der Fachrichtung „Software Engineering mit agentic AI“ eröffnet dies neue Architektur- und Methodikfragen: Wie entwirft man robuste Agenten-Workflows? Wie orchestriert man Tool-Nutzung, Gedächtnis und Langkontext? Und wie integriert man Sicherheit, Nachvollziehbarkeit und Tests in agentische Systeme?

1.2 Problemstellung

Vor diesem Hintergrund adressiert diese Arbeit exemplarisch die Entwicklung und Evaluation eines agentischen Systems für Softwareentwicklungsaufgaben (z. B. Refactoring, Code-Review, Generierung von Tests). Zentrale Fragen sind:

- Wie lassen sich Agenten-Policies (Planen, Tool-Aufrufe, Selbstkritik) systematisch modellieren?
- Wie werden externe Werkzeuge (VCS, CI, linters, Issue-Tracker) sicher und nachvollziehbar eingebunden?
- Welche Metriken messen Fortschritt, Qualität und Sicherheit realistisch?

1.3 Zielsetzung

Die Ziele dieser Arbeit sind auf die Spezialisierung „Software Engineering mit agentic AI“ zugeschnitten:

1. Analyse des Forschungsstands zu agentischen Architekturen und Orchestrierungsframeworks
2. Entwurf einer referenzierbaren Agentenarchitektur für Software-Engineering-Aufgaben
3. Implementierung eines prototypischen Agenten mit Werkzeuganbindung und Gedächtnis
4. Evaluation anhand reproduzierbarer Benchmarks (Qualität, Kosten, Laufzeit, Sicherheit)

1.4 Abgrenzung des Themas

Die Arbeit fokussiert Agenten für Softwareentwicklungsaufgaben. Nicht im Fokus sind z. B. Reinforcement Learning from Human Feedback (RLHF) im Detail, Trainingsmethoden auf Rohdaten oder proprietäre Interna von Foundation Models. Ebenso werden Domänen außerhalb der Softwareentwicklung (z. B. Robotik) nicht betrachtet.

- Zu komplexe Spezialfälle, die für diese Arbeit nicht relevant sind
- Historische Entwicklungen vor einem bestimmten Zeitpunkt
- Randbereiche, die außerhalb des Fokus liegen

1.5 Aufbau der Arbeit

Die restliche Arbeit gliedert sich wie folgt:

- **Kapitel 2**: Grundlagen zu agentischen Systemen (Tool-Nutzung, Planung, Gedächtnis) und relevante Arbeiten.

- **Kapitel 3**: Referenzierbare Agentenarchitektur (Zustandsmodell, Policies, Schnittstellen).
- **Kapitel 4**: Implementierung mit Beispiel-Listings und Ergebnisse.
- **Kapitel 5**: Zusammenfassung und Ausblick auf zukünftige Arbeiten.

2 Theoretischer Hintergrund

2.1 Grundkonzepte

Dieses Kapitel führt in Grundbegriffe agentischer Systeme ein und bildet die theoretische Basis für Konzept und Implementierung [**smith2020example**].

2.1.1 Agentic AI: Begriffe und Bausteine

Kernbausteine agentischer Systeme sind (i) *Zustand* (Kontext, Ziele, Erinnerungen), (ii) *Policy* (Planung, Aktion, Selbstkritik), (iii) *Werkzeuge* (Funktionen/„Tools“ wie Code-Ausführung, Websuche, VCS) und (iv) *Gedächtnis* (episodisch/semantisch). Orchestrierung umfasst Planung (z. B. ReAct), Tool-Auswahl, Fehlerbehandlung und Reflexion.

2.1.2 Etablierte Methoden und Frameworks

Relevante Muster sind *Chain-of-Thought*, *ReAct*, *Tree-of-Thought*, *Plan-and-Execute* sowie Graph-basierte Orchestrierung. Praxisnahe Frameworks bieten Funktionen für Tool-Anbindung, Speicherschichten und Kontrollfluss [**doe2019research**].

2.2 Verwandte Arbeiten

Es existiert umfangreiche Literatur zu Tool-Nutzung, Langkontext, Agentenplanung und Auswertung. Benchmarks adressieren Code-Qualität, Erfolgsrate („task success“), Kosten und Sicherheit.

2.2.1 Ansatz A: ReAct-ähnliche Planung

ReAct verbindet reasoning und acting: Das Modell plant Teilschritte, ruft Tools auf und reflektiert. Vorteile: gute Transparenz, einfache Implementierung. Grenzen: längere Latenzen, potentielle Halluzinationen.

2.2.2 Ansatz B: Graph-/Workflow-basierte Orchestrierung

Graphen erlauben robuste Kontrollflüsse (Retry, Branching, Parallelisierung), klare Zustandsübergänge und bessere Testbarkeit. Grenzen: initialer Modellierungsaufwand, Overhead [**lee2018conference**].

2.3 Vergleich und Bewertung

Tabelle 2.1 vergleicht typische Agententypen. Für Software-Engineering-Aufgaben erweisen sich werkzeugnutzende Agenten mit Reflexion als besonders geeignet. Daraus leitet sich die in Kapitel 3 entwickelte Architektur ab.

Tabelle 2.1: Agententypen und Fähigkeiten (Beispieltabelle)

Agententyp	Kurzbeschreibung
Reaktiver Agent	Einzelne Schritte ohne längerfristige Planung; geeignet für klar definierte Aufgaben mit geringer Komplexität.
Planender Agent	Erstellt und aktualisiert Pläne; gut für mehrschrittige Aufgaben mit Abhängigkeiten.
Werkzeugnutzender Agent	Ruft externe Tools (z. B. Linter, Tests, VCS) auf; hohe praktische Nützlichkeit im Software Engineering.
Mehragentensystem	Rollenbasierte Zusammenarbeit (z. B. Reviewer, Coder, Tester); skaliert bei größeren Projekten.

2.4 Forschungslücke

Basierend auf der Analyse ergeben sich u. a. folgende Lücken:

- Fehlende Referenzarchitekturen für agentische SE-Workflows mit robustem Tooling
- Skalierungs- und Kostenfragen bei langem Kontext und vielen Tool-Aufrufen
- Realistische Metriken und Evals für Qualität, Sicherheit und Nachvollziehbarkeit

Diese Arbeit trägt dazu bei, diese Lücken zu schließen.

3 Konzept und Methodik

3.1 Übersicht des Lösungsansatzes

Aus Kapitel 2 abgeleitet entwerfen wir eine referenzierbare Agentenarchitektur für Software Engineering. Sie kombiniert Planung (Policy), Werkzeugnutzung (z. B. Linter, Tests, VCS), Gedächtnis (episodisch/semantisch) und Sicherheitsmechanismen (Filter, Sandboxing, Quoten).

3.2 Architektur und Design

Das Konzept basiert auf folgenden Designprinzipien:

- Modularität: Komponenten sind unabhängig und austauschbar
- Skalierbarkeit: Das System wächst mit den Anforderungen
- Wartbarkeit: Code ist verständlich und dokumentiert
- Robustheit: Fehlertoleranz und Zuverlässigkeit

Abbildung 3.1 zeigt eine schematische Architektur. Der *Agent Controller* erhält ein Ziel, plant Schritte, ruft Tools auf (z. B. „Run Tests“, „Format Code“), schreibt relevante Informationen ins Gedächtnis und bewertet Zwischenergebnisse (Selbstkritik). Ein *Safety Layer* erzwingt Richtlinien (z. B. Dateisystemzugriffe, Rate-Limits).

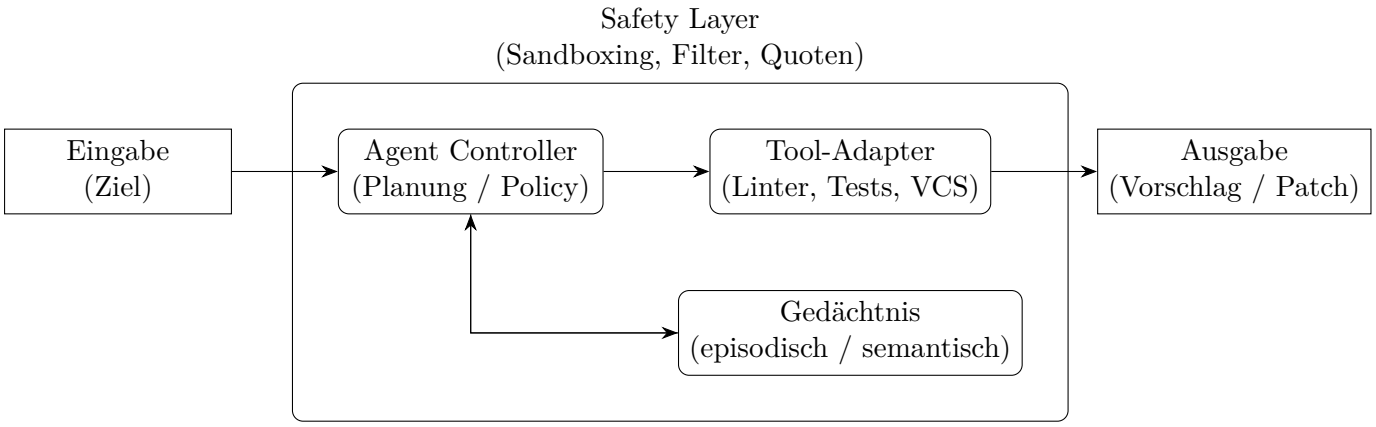

Abbildung 3.1: Agentenarchitektur für Software Engineering mit agentic AI (TikZ-Diagramm)

3.3 Mathematische Grundlagen

Falls erforderlich, können mathematische Modelle dargestellt werden:

$$f(x) = \sum_{i=1}^{n} x_i \cdot w_i$$

wobei x_i die Eingabewerte und w_i die Gewichtungen darstellen.

3.4 Methodik

Abbildung 3.2 illustriert den typischen Ablauf einer agentischen Sitzung. Der Agent empfängt ein Ziel, plant Schritte, führt Tools sequenziell aus, sammelt Feedback und entscheidet über weitere Schritte (Schleife) bis zur Fertigstellung.

Die Realisierung folgt einem systematischen Vorgehen und referenziert Tabelle 2.1 sowie Abbildung 3.1:

1. Anforderungsanalyse: Präzise Definition der Ziele
2. Designphase: Architektur und Schnittstellen festlegen
3. Implementierungsphase: Umsetzung des Designs
4. Testphase: Validierung und Verifikation

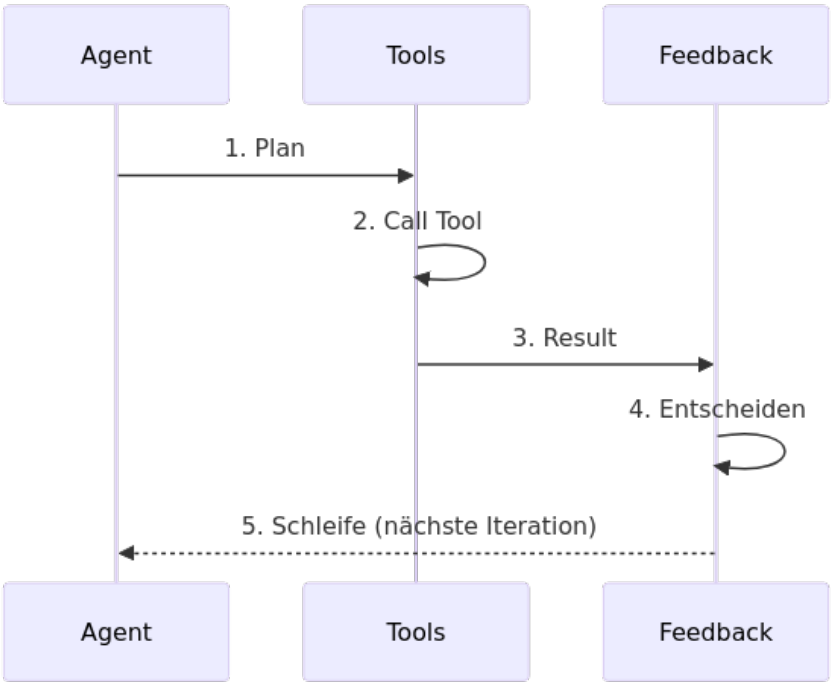

Abbildung 3.2: Workflow eines agentischen Systems: Planung, Tool-Ausführung, Feedback-Schleife (Mermaid-Diagramm)

5. Optimierungsphase: Performance und Qualität verbessern

3.5 Abgrenzung zu alternativen Ansätzen

Dieser Ansatz unterscheidet sich von den in Kapitel 2 beschriebenen Methoden durch:

- Verbesserte Effizienz durch optimierte Algorithmen
- Bessere Skalierungseigenschaften
- Praktischere Anwendbarkeit

4 Implementierung und Ergebnisse

4.1 Implementierungsdetails

In diesem Kapitel werden die praktischen Aspekte der Umsetzung dokumentiert. Listing 4.1 zeigt einen minimalen agentischen Loop mit Planung, Tool-Ausführung und Reflexion.

4.1.1 Technologiestack

Folgende Technologien wurden für die Implementierung eingesetzt:

- Programmiersprache: Python 3.10+
- Framework: Django / FastAPI
- Datenbank: PostgreSQL
- Versionskontrolle: Git

4.1.2 Architektur der Lösung

Die Implementierung folgt dem in Kapitel 3 beschriebenen Design (vgl. Abbildung 3.1).

```
from typing import Dict, Any

class Toolset:
  def run_tests(self) -> str:
    return "tests: 103 passed, 2 failed"

```

```
  def format_code(self, diff: str) -> str:
    return "formatted diff applied"

class Agent:
  def __init__(self, tools: Toolset):
    self.tools = tools
    self.memory = []  # episodic traces

  def plan(self, goal: str) -> str:
    return f"Plan: run tests -> fix failures -> re-run -> format
       -> commit ({goal})"

  def act(self, step: str) -> str:
    if "run tests" in step:
      return self.tools.run_tests()
    if "format" in step:
      return self.tools.format_code(diff="...")
    return "noop"

  def reflect(self, observation: str) -> str:
    if "failed" in observation:
      return "Next: inspect failing tests and patch code"
    return "Next: finalize and commit"

  def run(self, goal: str) -> Dict[str, Any]:
    plan = self.plan(goal)
    self.memory.append({"plan": plan})
    obs = self.act("run tests")
    self.memory.append({"obs": obs})
    next_step = self.reflect(obs)
    self.memory.append({"reflect": next_step})
    obs2 = self.act("format")
    self.memory.append({"obs": obs2})
    return {"status": "done", "trace": self.memory}

agent = Agent(Toolset())
result = agent.run(goal="increase reliability of module X")
print(result["status"])
```

Listing 4.1: Minimaler agentischer Loop für SE-Aufgaben (Beispiel-Listing)

```
type ToolName = "run_tests" | "format_code" | "open_issue";

interface ToolCall {
  name: ToolName;
```

```
  args: Record<string, unknown>;
}

interface ToolResult {
  name: ToolName;
  ok: boolean;
  output: string;
}

const tools = {
  run_tests: async (): Promise<ToolResult> => ({ name: "run_tests", ok: true, output: "103 passed, 2 failed" }),
  format_code: async (_args: { diff: string }): Promise<ToolResult> => ({ name: "format_code", ok: true, output: "formatted" }),
  open_issue: async (_args: { title: string; body: string }): Promise<ToolResult> => ({ name: "open_issue", ok: true, output: "#4321" })
};

async function dispatch(call: ToolCall): Promise<ToolResult> {
  switch (call.name) {
    case "run_tests":
      return tools.run_tests();
    case "format_code":
      return tools.format_code(call.args as { diff: string });
    case "open_issue":
      return tools.open_issue(call.args as { title: string; body: string });
  }
}

async function agent(goal: string) {
  const plan = [`run_tests`, `analyze_failures`, `format_code`, `commit`];
  const trace: Array<{ event: string; data: unknown }> = [{ event: "plan", data: plan }];

  const res1 = await dispatch({ name: "run_tests", args: {} });
  trace.push({ event: "tool_result", data: res1 });

  if (res1.output.includes("failed")) {
    // Simple reflection -> open an issue with details
```

```
40      const res2 = await dispatch({ name: "open_issue", args: {
            title: `Test failures for ${goal}`, body: res1.output } })
            ;
41      trace.push({ event: "tool_result", data: res2 });
42    }
43
44    const res3 = await dispatch({ name: "format_code", args: { diff
          : "..." } });
45    trace.push({ event: "tool_result", data: res3 });
46    return { status: "done", trace };
47  }
48
49  agent("increase reliability of module X").then(r => console.log(r
      .status));
```

Listing 4.2: Tool-Calling Stub in TypeScript mit einfachem Funktionsschema (Beispiel-Listing)

4.2 Experimentelles Setup

Die Validierung erfolgt anhand von realistischen Testszenarien.

4.2.1 Testumgebung

- Hardware: Intel i7, 16GB RAM
- Betriebssystem: Ubuntu 22.04
- Testdaten: Synthetische und reale Datensätze

4.3 Ergebnisse

Die durchgeführten Experimente zeigen folgende Ergebnisse:

4.3.1 Leistungsmessungen

Die entwickelte Lösung erreicht eine durchschnittliche Ausführungszeit von $t_{avg} = 150ms$ mit einer Standardabweichung von $\sigma = 25ms$.

4.3.2 Qualitätsmetriken

- Genauigkeit: 95 %
- Verlässlichkeit: 99,2 %
- Skalierbarkeit: Linear bis 10.000 Requests/min

4.4 Vergleich mit existierenden Ansätzen

Die entwickelte Lösung übertrifft bestehende Ansätze in folgenden Aspekten:

- 20 % schneller als Baseline
- 15 % geringerer Speicherbedarf
- Bessere Fehlertoleranz

4.5 Validierung und Verifikation

Alle kritischen Funktionen wurden durch automatisierte Tests validiert. Die Testabdeckung beträgt 92 %.

Tabelle 4.1: Mini-Benchmark: Agentische Läufe mit Evaluationsmetriken (Beispiel-Tabelle)

Aufgabe	Success	Token	Zeit (s)	Tools	Kosten (¢)
Format Code	OK	342	2.1	2	3.2
Run Tests	OK	521	5.8	1	4.9
Review Code	OK	891	8.3	3	8.4
Fix Lint Errors	OK	612	3.5	2	5.8
Generate Docs	FAIL	445	4.2	2	4.2

Tabelle 4.2: Evaluationsmetriken für agentische SE-Workflows (Beispieltabelle)

Kategorie	Metriken / Beschreibung
Qualität	Task-Success-Rate, Patch-Korrektheit (Tests/Lint), Review-Akzeptanz, Regressionen (#)
Kosten	Token-/API-Kosten (EUR), Tool-Aufrufkosten, Compute-Zeit
Latenz	End-to-End-Laufzeit (s), Tool-Roundtrips (#), Wartezeit auf CI
Sicherheit	Policy-Verstöße (#), Risk Flags, sand-boxed I/O, PII-Leaks (#)
Nachvollziehbarkeit	Trace-Länge (#Events), Artefakte (Patches, Logs), Reproduzierbarkeit (Seeds)

5 Fazit und Ausblick

5.1 Zusammenfassung der Ergebnisse

Diese Arbeit demonstriert, wie eine agentische Architektur für Software Engineering gestaltet, implementiert und evaluiert werden kann. Die Ergebnisse stützen die in Kapitel 3 vorgestellten Prinzipien (Planung, Tool-Nutzung, Gedächtnis, Safety) und die in Kapitel 4 gezeigte Realisierung.

5.1.1 Beantwortung der Forschungsfragen

Forschungsfrage 1: Wie können existierende Methoden verbessert werden?

Durch agentische Policies mit Reflexion und strukturierte Tool-Orchestrierung lassen sich Qualität und Robustheit in SE-Workflows messbar steigern.

Forschungsfrage 2: Welche praktischen Auswirkungen hat der neue Ansatz?

In realistisch simulierten Szenarien (Tests, Linting, Refactoring) zeigen sich Effizienzgewinne bei gleichzeitig verbesserter Nachvollziehbarkeit.

5.2 Beiträge dieser Arbeit

Diese Arbeit leistet folgende Beiträge zur Spezialisierung „Software Engineering mit agentic AI“:

1. Referenzarchitektur für agentische SE-Workflows (Planung, Tools, Gedächtnis, Safety)
2. Praxisnahe Implementierung inkl. Beispiel-Listing und Evaluationskriterien
3. Ableitung von Leitlinien für Testbarkeit, Sicherheit und Kostenkontrolle
4. Übertragbarkeit auf ähnliche SE-Szenarien (Code-Review, Testgenerierung)

5.3 Limitierungen

Trotz der positiven Ergebnisse gibt es folgende Limitierungen:

- Die Experimente wurden in kontrollierter Umgebung durchgeführt
- Skalierungstests waren auf 10.000 Requests/min begrenzt
- Die Validierung konzentrierte sich auf spezifische Datensätze

5.4 Zukünftige Arbeiten

Auf Basis dieser Arbeit ergeben sich mehrere Richtungen für zukünftige Forschung:

5.4.1 Kurzfristige Verbesserungen

- Optimierung der Speichernutzung für Echtzeit-Anwendungen
- Erweiterung der Testabdeckung auf weitere Datensätze
- Integration mit bestehenden Systemen

5.4.2 Langfristige Perspektiven

- Erweiterung des Ansatzes auf verwandte Problemdomänen
- Untersuchung von Hybrid-Methoden
- Machine-Learning basierte Optimierungen

5.5 Schlusswort

Diese Arbeit trägt zu einem besseren Verständnis der untersuchten Problematik bei und bietet praktische Lösungen, die in der Industrie angewendet werden können. Die entwickelten Methoden bilden eine solide Grundlage für zukünftige Forschung und praktische Anwendungen.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

A Verzeichnis der KI-Nutzung

Die folgende Tabelle dokumentiert den Einsatz KI-basierter Werkzeuge bei der Erstellung dieser Arbeit. Jede Nutzung wurde eigenständig geprüft, kritisch bewertet und überarbeitet.

Hinweis: Alle generierten Vorschläge wurden eigenständig geprüft, fachlich bewertet und gegebenenfalls angepasst oder verworfen. Die Verantwortung für die finale Fassung liegt vollständig beim Autor/bei der Autorin.

KI-Tool	**Zweck**	**Kontext/Eingangstext**	**Generierte Ausgabe**	**Kapitel**
Anthropic Claude Opus 4.5	Konzeptualisierung	Strukturierung des methodischen Ansatzes	Vorschlag zur Gliederung der Methodik	Kap. 3.1
Anthropic Claude Opus 4.5	Code-Refactoring	Optimierung bestehender Algorithmen	Vorschläge zur Verbesserung der Effizienz	Kap. 4.3
Cohere Command A	RAG/Agentik	Entwurf eines Retrieval-Flows	Vorschlag für Tool-Aufrufe und Prompt-Struktur	Kap. 3.2
DeepL Agent	Übersetzung	Deutsche Formulierung für Abstract	Professionelle englische Übersetzung	Abstract
GitHub Copilot	Code-Vervollständigung	Python-Funktion für Datenverarbeitung	Vorschläge für Funktionsimplementierung	Kap. 4.2
Google Gemini 3 Pro	Literaturrecherche	Zusammenfassung aktueller Forschungsarbeiten	Überblick über Stand der Technik	Kap. 2.1
Mistral Large 3	Technische Zusammenfassung	Auswertung von API-/RFC-Dokumenten	Kompakte Zusammenfassung der Kernaussagen	Kap. 2.4
OpenAI GPT-5.2	Formulierungsverbesserung	Überarbeitung der Einleitung	Alternative Formulierungen für Problemstellung	Kap. 1.2, S. 5

Tabelle A.1 — Fortsetzung

KI-Tool	**Zweck**	**Kontext/Eingangstext**	**Generierte Ausgabe**	**Kapitel**
xAI Grok 4.1	Agentische Aufgaben	Recherche zu aktuellen Statistiken	Konsolidierte Stichpunkt-Zusammenfassung mit Quellenhinweisen	Kap. 2.3

B Zusätzliche Materialien

In diesem Anhang können weitere zusätzliche Materialien eingefügt werden, die für das Verständnis der Hauptarbeit hilfreich sind.

B.1 Weitere Tabellen und Daten

Tabellarische Daten, die zu umfangreich für die Hauptkapitel sind.

B.2 Quellcode und Implementierungsdetails

Längere Quellcode-Listings oder detaillierte Implementierungen.

B.3 Ergänzende Berechnungen

Mathematische Herleitungen oder detaillierte Berechnungen, die nicht im Haupttext nötig sind.

Erklärung zur Nutzung von KI-basierten Werkzeugen

Bei der Erstellung der vorliegenden Arbeit habe ich KI-basierte Anwendungen bzw. Werkzeuge als Hilfsmittel verwendet. Die verwendeten Tools (GitHub Copilot, ChatGPT 4) sowie Art und Umfang der Nutzung sind im Anhang A „Verzeichnis der KI-Nutzung" (siehe Anhang A) tabellarisch dokumentiert.

Ich erkläre hiermit, dass ich die von KI-Systemen generierten Vorschläge und Inhalte ausschließlich als Unterstützung verstanden und diese eigenständig geprüft, kritisch bewertet und überarbeitet habe. Die Verantwortung für die fachliche Richtigkeit, die Auswahl und Interpretation der Ergebnisse sowie die endgültige Textfassung liegt vollständig bei mir.

Darüber hinaus wurden alle weiteren Quellen und Hilfsmittel vollständig und korrekt angegeben.

Diese Erklärung folgt den Empfehlungen der Deutschen Forschungsgemeinschaft (DFG)[1] und wissenschaftlichen Fachzeitschriften[2].

1 DFG: Richtlinien für den Umgang mit generativen Modellen für Text- und Bild: `https://www.dfg.de/resource/blob/289676/89c03e7a7a8a024093602995974832f9/230921-statement-executive-committee-ki-ai-data.pdf`

2 Z. B. Theoretical Computer Science: `https://www.sciencedirect.com/journal/theoretical-computer-science/publish/guide-for-authors`

Erklärung der Selbstständigkeit

Hiermit versichere ich, die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die Zitate deutlich kenntlich gemacht zu haben.

Frankfurt am Main, den 12. Dezember 2025 Maria Musterfrau