Duale Hochschule Baden-Württemberg Mannheim

**Praxisarbeit**

# Konzeption und Implementierung einer Companion App für Arc Raiders.

**Studiengang Informatik**

**Studienrichtung Informationstechnik**


| | |
|---|---|
| Verfasser(in): | Paul Wegfahrt |
| Matrikelnummer: | 2415837 |
| Kurs: | TINF23IT1 |
| Studiengangsleiter: | Prof Dr. Gerhards |
| Wissenschaftlicher Betreuer: | Jürgen Schultheis |
| Bearbeitungszeitraum: | 14.10.2025 – 14.04.2026 |
| Eingereicht am: | 14.04.2026 |

# Ehrenwörtliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit mit dem Titel "*Konzeption und Implementierung einer Companion App für Arc Raiders.*" selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ort, Datum Paul Wegfahrt

# Sperrvermerk

Der Inhalt dieser Arbeit darf weder als Ganzes noch in Auszügen Personen außerhalb des Prüfungsprozesses und des Evaluationsverfahrens zugänglich gemacht werden, sofern keine anders lautende Genehmigung der Ausbildungsstätte vorliegt.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Quelltextverzeichnis

# Abkürzungsverzeichnis

| | |
|---|---|
| **DHBW** | Duale Hochschule Baden-Württemberg |
| **PvPvE** | Player versus Player versus Entities |
| **API** | Application Programming Interface |
| **TTI** | Time to Interactivity |
| **MVP** | Minimum Viable Product |
| **ER** | Entity-Relationship |
| **MCDA** | Multi-Criteria Decision Analysis |
| **DDD** | Domain-Driven Design |

# Abstract

Abstract...

# Zusammenfassung

Zusammenfassung....

# 1 Einleitung

## 1.1 Motivation

Der globale Gaming-Markt verzeichnet ein beispielloses Wachstum: Mit einer Bewertung von 298,98 Milliarden USD im Jahr 2024 wird prognostiziert, dass der Markt bis 2030 auf 600,74 Milliarden USD anwachsen wird [7]. Diese Entwicklung zeigt deutlich, dass Anwendungen in diesem Bereich ein großes Potenzial haben.

Innerhalb dieses dynamischen Marktes hat sich der Extraction Shooter als besonders anspruchsvolles und komplexes Genre etabliert. Spiele wie Escape from Tarkov und Hunt: Showdown definieren dieses Genre durch ihre charakteristischen Merkmale: hochriskante Player versus Player versus Entities (PvPvE)-Gameplay-Mechaniken, komplexe Progressionssysteme mit zahlreichen Quest-Lines, ressourcenbasierte Upgrade-Systeme und die permanente Gefahr des Verlusts aller mitgeführten Items bei einem Spieltod. Arc Raiders, entwickelt von Embark Studios und im Oktober 2025 veröffentlicht, positioniert sich als ambitionierter Vertreter dieses Genres mit dem Ziel, die Komplexität von Tarkov mit einer zugänglicheren Spielerfahrung zu verbinden.

Die inhärente Komplexität von Extraction Shootern stellt Spieler jedoch vor erhebliche Herausforderungen: Multiple Quest-Lines mit unterschiedlichen Zielen und Abhängigkeiten, begrenzter Inventarplatz, knappe Ressourcen und die Notwendigkeit koordinierter Squad-Basierter Strategien erfordern ein hohes Maß an Planung und Informationsmanagement. Companion Apps haben sich in der Gaming-Industrie als effektive Lösung etabliert, um solche Komplexitäten zu bewältigen. Es bestehen zahlreiche Beispiele aus verschiedenen Genres wie League of Legends, Destiny 2 oder eben Extraction-Shootern wie Escape from Tarkov, welche demonstrieren, wie externe Anwendungen das Spielerlebnis durch Statistik-Tracking, Ressourcenmanagement und Team-Koordination signifikant verbessern können [5] [3].

## 1.2 Problemstellung

Spieler von Arc Raiders sehen sich mit mehreren miteinander verknüpften Herausforderungen konfrontiert:

**Informationsasymmetrie und mangelnde Übersicht:** Das Spiel bietet zahlreiche Quest-Lines mit unterschiedlichen Zielen, zeigt jedoch nur die aktiven Quests an. Spieler haben dadurch keinen vollständigen Überblick über verfügbare Quests, deren Abhängigkeiten, den optimalen Pfad zur Erfüllung ihrer Ziele oder benötigter Materialien für die Zukunft. Diese Informationsfragmentierung erschwert strategische Planung und führt zu ineffizienten Entscheidungen.

**Ressourcenmanagement:** Die Upgrade-Systeme für Workstations erfordern multiple Ressourcentypen über mehrere Stufen hinweg. Bei begrenztem Inventarplatz und knappen Ressourcen fehlen Spielern Werkzeuge zur Kalkulation benötigter Materialien und zur Priorisierung von Upgrades basierend auf ihren individuellen Spielzielen.

**Squad-Koordination:** Arc Raiders basiert auf Squad-orientiertem Gameplay, doch wenn jedes Squad-Mitglied nur seine eigenen Ziele verfolgt, entstehen Konflikte bei der Routenplanung und Ressourcenverteilung. Die fehlende zentrale Übersicht über Squad-Ziele behindert effektive Koordination und optimale Ressourcennutzung.

**Fehlen offizieller Planungstools:** Zum aktuellen Stand (November 2025) existieren keine offiziellen Tools oder Application Programming Interface (API) zur Lösung dieser Probleme. Daten werden von Spielern über diverse Plattformen gesammelt und bereitgestellt.

Diese Problemstellung ist nicht singulär für Arc Raiders, sondern repräsentativ für die Herausforderungen moderner Extraction Shooter mit komplexen Meta-Progression-Systemen. Die Lösung dieser Probleme durch eine dedizierte Companion App könnte somit über Arc Raiders hinaus als Referenzimplementierung für ähnliche Spiele dienen.

# 2 Aufgabenstellung und Entwicklungsmethodik

## 2.1 Operationalisierung der Problemstellung

Die in Abschnitt 1.2 identifizierten Herausforderungen für Spieler von Arc Raiders – Informationsasymmetrie bei Quest-Lines, ineffizientes Ressourcenmanagement und unzureichende Squad-Koordination – werden durch systematische Operationalisierung in konkrete Entwicklungsaufgaben überführt. Operationalisierung bezeichnet dabei die systematische Ableitung von Services und technischen Constraints aus übergeordneten Zielen [6].

Die identifizierten Probleme lassen sich wie folgt operationalisieren:

- **Problem "Informationsasymmetrie und mangelnde Übersicht"**
  → **Ziel** "Vollständiger Überblick über Quest-Lines und Abhängigkeiten"
  → **Service** "Quest Tracking mit Kanban/Flow-Chart-Visualisierung"
  → **Technische Anforderung** "Datenmodell für Quest-Abhängigkeiten, Filterung und Statusverwaltung"
- **Problem "Ressourcenmanagement bei begrenztem Inventar"**
  → **Ziel** "Optimierte Materialnutzung und Upgrade-Priorisierung"
  → **Service** "Material Calculator & Workstation Planner"
  → **Technische Anforderung** "Berechnung für Upgrade-Kosten über multiple Stufen"
- **Problem "Fehlende Squad-Koordination"**
  → **Ziel** "Zentrale Übersicht über Squad-Ziele und effiziente Routenplanung"
  → **Service** "Squad-basierte Routenoptimierung mit interaktiven Karten"
  → **Technische Anforderung** "Darstellung von Zielen und Karten mit Tools zur Routenoptimierung"

Diese Operationalisierung adressiert das in Abschnitt 1.2 beschriebene Fehlen offizieller Planungstools und nutzt die von der Community bereitgestellten Daten als Grundlage. Sie bildet die methodische Basis für die nachfolgend beschriebene Entwicklung einer Companion App, die als Referenzimplementierung für ähnliche Extraction Shooter dienen kann.

## 2.2 Entwicklungsmethodik und Phasenmodell

### 2.2.1 Phase 1: Requirements Engineering

In agilen Entwicklungsumgebungen ist Requirements Engineering integraler Bestandteil zur Sicherstellung, dass sich entwickelnde Bedürfnisse und Erwartungen der Stakeholder während des gesamten Entwicklungsprozesses erfasst werden [1].

#### 1.1 Anforderungserhebung (Requirements Elicitation)

- **1.1.1 Empirische Datenerhebung aus Spieltests:** Die in Abschnitt 1.1 erwähnte Teilnahme an einem Spieltest von Arc Raiders bildet die empirische Grundlage für die Anforderungserhebung. Durch systematische Beobachtung und Protokollierung werden konkrete Pain Points bei Quest-Management und Ressourcenplanung identifiziert.
- **1.1.2 Stakeholder-Interviews:** Durchführung direkter Gespräche mit Stakeholdern zur Extraktion von Bedürfnissen und Ideen [8]. Strukturierte Interviews mit Arc Raiders-Spielern verschiedener Erfahrungsstufen sowie gemeinsame Brainstorming-Sessions zur Feature-Ideenfindung ermöglichen die Erfassung spezifischer Anforderungen für Squad-basiertes Gameplay. Die in Abschnitt 1.2 identifizierten Problemstellungen werden dabei validiert und präzisiert.
- **1.1.3 Comparative Analysis:** Wie in Abschnitt 1.1 dargelegt, haben sich Companion Apps in der Gaming-Industrie als effektive Lösung etabliert. Die systematische Analyse umfasst primär etablierte Companion Apps für Extraction Shooter. Dies identifiziert Standard-Features und Innovationspotenziale, während die heuristische Evaluation von UI/UX-Patterns für komplexe Informationsdarstellung Best Practices aufzeigt.
- **1.1.4 Ableitung von User Stories:** Transformation der identifizierten Nutzerbedürfnisse in User Stories nach dem Format „Als [Rolle] möchte ich [Funktion], um [Nutzen] zu erreichen“. Beispiele bezogen auf die Problemstellung:
  - „Als Spieler möchte ich alle verfügbaren Quests und deren Abhängigkeiten sehen, um meine Ziele zu planen“,
  - „Als Squad-Leader möchte ich die Quest-Ziele meiner Teammitglieder auf einer Karte visualisieren, um eine effiziente Route für das gesamte Team zu planen“
  - „Als Spieler möchte ich kalkulieren, welche Materialien ich für geplante Workstation-Upgrades, Quests sowie Projekte benötige, um mein begrenztes Inventar optimal zu nutzen“.

### 1.2 Anforderungsanalyse (Requirements Analysis)

- **1.2.1 Kategorisierung und zeitliche Strukturierung:** Gruppierung in funktionale Anforderungen nach Seite (z.B. Dashboard, Quests, Workstations) und Implementierungsphase (z.B. P1-Visualization, P2-Progression) sowie nicht-funktionale Anforderungen (Performance, Usability, Verfügbarkeit). Identifikation von Abhängigkeiten zwischen Anforderungen (z.B. Routenplanung benötigt Maps) zur Planung der Implementierungsreihenfolge.
- **1.2.2 Priorisierung:** Implementierung der höchstpriorisierten Anforderungen zuerst zur Maximierung des Stakeholder-ROI [2]. Anwendung der MoSCoW-Methode (Must, Should, Could, Won't) mit Bewertung nach Business Value (Lösung der Kernprobleme aus Abschnitt 1.2) und technischer Komplexität. Dokumentation in Form eines Kanban Boards sowie Gantt-Diagramms unter Berücksichtigung von Abhängigkeiten zwischen Features.
- **1.2.3 Spezifikation von Akzeptanzkriterien:** Definition messbarer Erfolgskriterien für jede User Story nach SMART-Kriterien (Specific, Measurable, Achievable, Relevant, Time-bound). Beispiele umfassen:
  - „Quest-Suche liefert Ergebnisse in <500ms Time to Interactivity (TTI) für 95% der Anfragen“
  - „Material Calculator berechnet Upgrade-Kosten für beliebige Kombinationen von Workstations korrekt“.

### 1.3 Anforderungsvalidierung

- **1.3.1 Prototyping:** Erstellen von Minimum Viable Product (MVP)-Prototypen zur Visualisierung und Validierung der Anforderungen mit Stakeholdern. Interaktive Mockups und Wireframes ermöglichen frühes Feedback zu UI/UX-Designs und Funktionalitäten, um sicherzustellen, dass die entwickelten Lösungen den identifizierten Bedürfnissen entsprechen.
- **1.3.3 Abgleich mit Problemstellung:** Systematischer Abgleich der definierten Anforderungen mit der ursprünglichen Problemstellung zur Sicherstellung vollständiger Abdeckung aller drei Hauptherausforderungen: Informationsasymmetrie bei Quest-Lines, ineffizientes Ressourcenmanagement und unzureichende Squad-Koordination.

### 2.2.2 Phase 2: Architektur und Design

#### 2.1 Systemarchitektur

- **2.1.1 Architekturentwurf:** Definition der Systemgrenzen und Komponenten (Frontend, Backend, Datenbank, externe Datenquellen) sowie Auswahl geeigneter Architekturmuster unter Berücksichtigung der Komplexität. Die Dokumentation erfolgt durch Architekturdiagramme nach dem C4-Modell (Context, Container, Component, Code), um verschiedene Abstraktionsebenen abzubilden und Stakeholdern unterschiedliche Detailgrade zu ermöglichen.
- **2.1.2 Technologie-Assessment:** Wie in Abschnitt 1.1 erwähnt, rechtfertigt das enorme Wachstumspotenzial des Gaming-Marktes die Wahl skalierbarer Technologien. Die systematische Evaluation umfasst Frontend-Frameworks (React, Vue, Angular), Backend-Technologien (Next.js API Routes, separate Backend-Lösung), Datenbank-Systeme (PostgreSQL, MongoDB, Firebase) sowie Hosting-Plattformen (Vercel, Netlify, AWS) unter Berücksichtigung von Kosteneffizienz für Hobby-Projekte. Multi-Criteria Decision Analysis (MCDA) ermöglicht die Technologieauswahl basierend auf Kriterien wie Performance, Entwicklerfreundlichkeit, Skalierbarkeit und Kosten.
- **2.1.3 Risikoanalyse:** Identifikation technischer Risiken, insbesondere die in Abschnitt 1.2 erwähnte Abhängigkeit von Community-bereitgestellten Daten statt offizieller API. Bewertung nach Eintrittswahrscheinlichkeit und Impact sowie Definition von Mitigationsstrategien: Backup-Strategien für Datenquellen (Web-Scraping des offiziellen Fandoms gemäß robots.txt, User-Generated Content mit Qualitätssicherung), Caching-Mechanismen zur Reduzierung der Abhängigkeit von externen Quellen sowie Validierung und Qualitätssicherung von Community-Daten.

#### 2.2 Datenmodellierung

- **2.2.1 Konzeptionelle Modellierung:** Erstellung von Modellierungsdiagrammen wie Entity-Relationship (ER)-Digrammen zur Abbildung der Hauptentitäten (Quests, Workstations, Materialien, Spielerprofile) und deren Beziehungen oder Context Diagrams aus dem Domain-Driven Design (DDD) zur Identifikation von Bounded Contexts. Berücksichtigung der in Abschnitt 1.2 beschriebenen Anforderungen an Flexibilität und Erweiterbarkeit für zukünftige Features.
- **2.2.2 Datenquellen-Strategie:** Aufgrund des in Abschnitt 1.2 beschriebenen Fehlens offizieller Tools müssen mehrere Community-Datenquellen miteinander Verglichen und möglicherweise kombiniert werden. Hierbei wird ähnlich wie bei der Technologie-Assessment-Methode eine MCDA angewendet, um die Zuverlässigkeit, Aktualität und

Vollständigkeit der Datenquellen zu bewerten. Strategien zur Datenintegration und -synchronisation werden definiert, um eine konsistente und aktuelle Datenbasis sicherzustellen.

### 2.2.3 Phase 3: Implementierung in Iterationen

Agile Entwicklung betont die iterative Natur mit kontinuierlicher Verfeinerung und Validierung [4].

#### 3.1 Entwicklung

- **3.1.1 Iterative Entwicklung:** Umsetzung der priorisierten User Stories in Iterationen unter Berücksichtigung von Clean-Code Prinzipien. Jede Iteration umfasst Planung, Implementierung, Testing und Review. Kontinuierliche Integration von Feedback aus Reviews zur Anpassung des Backlogs und Verbesserung der Implementierung.
- **3.1.2 Continuous Integration:** Automatisierte Builds bei jedem Commit über GitHub Actions oder ähnliche CI-Tools sowie automatisierte Testausführung der gesamten Test-Suite. Code Quality Checks umfassen Linting (ESLint für JavaScript/TypeScript). Automatisches Deployment auf Staging-Umgebung (z.B. Vercel Preview Deployments) für frühzeitiges Testen.

#### 3.2 Review und Retrospektive

- **3.2.1 Review:** Demonstration implementierter Features an Stakeholder zur Einholung von Feedback bezüglich der Lösungen für die Probleme aus Abschnitt 1.2. Backlog-Refinement basierend auf gewonnenen Erkenntnissen sowie Anpassung der Priorisierung bei Bedarf.
- **3.2.2 Retrospektive:** Reflexion des Entwicklungsprozesses mit den Leitfragen: Was lief gut? Was kann verbessert werden? Identifikation von Verbesserungspotenzialen in Prozess, Definition konkreter Aufgaben für den nächsten Zyklus sowie Anpassung der Entwicklungspraktiken basierend auf Lessons Learned.

### 2.2.4 Phase 4: Evaluation und Reflexion

#### 4.1 Quantitative Evaluation

- **4.1.1 Performance-Metriken:** Messung von Page Load Time durch Metriken wie der TTI für alle Hauptseiten. Messung der API-Antwortzeiten für kritische Endpunk-

te (Quest-Suche, Material Calculator, Routenplanung) sowie Überwachung der Server- und Datenbank-Performance (CPU-, Speicher- und Datenbank-Latenz) unter Lastbedingungen.

- **4.1.3 User-Engagement-Metriken:** Sofern Veröffentlichung erfolgt: Daily Active Users (DAU), Feature Usage Statistics zur Identifikation der am häufigsten genutzten Funktionalitäten sowie User Retention Rate zur Bewertung des langfristigen Nutzens.

### 4.2 Qualitative Evaluation

- **4.2.2 User Experience Interviews:** Durchführung von User Experience Interviews mit Spielern zur Bewertung, ob die in Abschnitt 1.2 identifizierten Probleme gelöst wurden. Bewertung der Zielerreichung bezüglich Verbesserung der Quest-Übersicht, Effizienzsteigerung im Ressourcenmanagement sowie Optimierung der Squad-Koordination.
- **4.2.3 Vergleich mit Anforderungen:** Systematischer Vergleich der implementierten Features mit initialen Anforderungen und Akzeptanzkriterien. Identifikation von vollständig erfüllten, teilweise erfüllten und nicht erfüllten Anforderungen mit Begründung der Abweichungen.

### 4.3 Kritische Reflexion

- **4.3.1 Architektur-Bewertung:** Bewertung der gewählten Architektur hinsichtlich Eignung für die Anforderungen, Entwicklungseffizienz, Skalierbarkeit und Wartbarkeit sowie Diskussion von Trade-offs und alternativen Ansätzen. Reflexion, ob die Architekturentscheidungen im Kontext des in Abschnitt 1.1 beschriebenen Marktwachstums zukunftsfähig sind.
- **4.3.2 Technologie-Entscheidungen:** Diskussion des gewählten Technologie-Stacks (React/Next.js, Supabase/PostgreSQL, Vercel) mit Fokus auf Stärken, Schwächen und Lessons Learned. Reflexion der Datenhaltungsstrategie (Community-Daten, Caching, Synchronisation) und der in Abschnitt 1.2 beschriebenen Herausforderung einer fehlenden offiziellen API.
- **4.3.3 Lessons Learned:** Dokumentation gewonnener Erkenntnisse aus dem Entwicklungsprozess: erfolgreiche Praktiken, aufgetretene Herausforderungen und deren Lösungen sowie Empfehlungen für zukünftige Projekte. Reflexion der agilen Methodik und deren Eignung für die Entwicklung einer Gaming Companion App.
- **4.3.4 Generalisierbarkeit:** Bewertung, inwieweit die entwickelte Lösung als Referenzimplementierung für andere Extraction Shooter dienen kann, wie in Abschnitt 1.2 postuliert. Identifikation von spielspezifischen Aspekten und übertragbaren Konzepten.

## 2.3 Methoden und Verfahren

"Welche Methoden und Verfahren werden verwendet?"

### 2.3.1 Verwendete Tools und Technologien

Verwendete Tools und Technologien mit Versionen/Datum (wie z.B. package.json aufbereiten), Ergebnis der Arbeit muss replizierbar sein

# 3 Grundlagen

SState of the Art"

## 3.1 Arc Raiders & Gaming Tools

"Funktionale Anforderungsanalyse sollte erkenntlich sein, jedoch nicht direkt so benannt"

### 3.1.1 Arc Raiders

SSpielbeschreibung"

### 3.1.2 Marktanalyse, vorhandene Tools/Anwendungen

(Parallelen zu Tools aus anderen Spielen wie Tarkov?)

## 3.2 Moderne Web-Technologien

### 3.2.1 Typescript

kurz halten Type Safety in großen Projekten Developer Experience

### 3.2.2 React

Komponentenbasiertes UI Komposition over Inheritance

### 3.2.3 Full Stack React Frameworks

Nextjs als React "Backend"Framework, SSR vs. SSG vs. ISR vs. PPR (Partial Prerendering) file based routing

## 3.3 UI/UX Frameworks & Design System

### 3.3.1 Tailwindcss

Utility First CSS Framework, inline v4 mit "Just in Time"Compiler und global.css

### 3.3.2 Moderne Komponenten-Bibliotheken

Shadcn/using unterschied zu "klassischen"Bibliotheken wie MUI bootstrap

### 3.3.3 React-Flow

Visualisierung von Graphen/Netzwerken

## 3.4 State Management & Data Fetching

### 3.4.1 State Management/State Stores

Bibliotheken wie zustand client side

### 3.4.2 react-query / tanstack-query als Daten-Fetching Library

Server State Management Caching Strategien Optimistic Updates

## 3.5 Datenbank und Backend Design

### 3.5.1 Supabase

PostgreSQL-basiert Real-time Capabilities Row Level Security edge functions (Serverless Functions)

### 3.5.2 Orm - Drizzle

ORM's Type-Safety code-first approach

## 3.6 Deployment & DevOps

### 3.6.1 Git basierter Workflow

### 3.6.2 Vercel

Deployment von Nextjs Applikationen weitere features CI/CD Pipelines

## 3.7 Testing Frameworks & Strategien

### 3.7.1 Vitest

Unit und Integration Tests fast, modern, built for TS

### 3.7.2 Cypress

End-to-End Testing real browser testing

# 4 Durchführung

an T2000 orientieren? SZeige, wie du von der Problemstellung zu Requirements kommst Traceability Matrix: Verknüpfe Anforderungen → Issues → Tests → Code"

## 4.1 Anforderungsanalyse

- Zielgruppenanalyse
- Funktionale Anforderungen
- Nicht-funktionale Anforderungen

## 4.2 Vorbereitung

- Modellierung
- Technologieentscheidungen (kurz)

## 4.3 Implementierung

- Architektur
- Beispiel Datenbankmodell, edge function
- Beispiel API Endpunkte
- Beispiel Frontend Komponenten

## 4.4 Testen

- Teststrategie/-umgebung
- Beispiel Test
- Ergebnisse

# 5 Ergebnisse und Diskussion

Metriken definieren: Wie misst du Erfolg? (Performance, Usability, Code-Qualität)

## 5.1 Objektivierung der Ergebnisse

(Tabelle, Aufgabe, erledigt?, verifiziert?)

## 5.2 Diskussion?

Stellungsnahme zu den Ergebnisse, aufzeigen von Alternativen

## 5.3 Marktanalyse, vorhandene Tools/Anwendungen

(Parallelen zu Tools aus anderen Spielen wie Tarkov?)

# 6 Fazit und Ausblick

Was kann man damit anfangen

Wie kann man es erweitern

# Literatur


[1] A. Abukhalaf et al. „Automated requirements engineering framework for agile model-driven development“. In: *Frontiers in Computer Science* 7 (2025). DOI: 10.3389/fcomp.2025.1537100.

[2] Scott W. Ambler. *Agile Requirements Modeling: Strategies for Agile Teams*. Agile Modeling. 2023. URL: http://agilemodeling.com/essays/agileRequirements.htm (besucht am 27. 01. 2025).

[3] Matthew Cochran. „Best Companion Apps In Video Games“. In: (2023).

[4] G. Ebirim et al. „Advancements and innovations in requirements elicitation“. In: *World Journal of Advanced Research and Reviews* 22.01 (2024), S. 1209–1220.

[5] Olga Giersza. „Companion Apps are Taking Video Games to the Next Level“. In: (2024).

[6] Axel van Lamsweerde. „Requirements Engineering in the Year 00: A Research Perspective“. In: *Proceedings of the 22nd International Conference on Software Engineering (ICSE'00)*. IEEE Press, 2000, S. 5–19. DOI: 10.1145/337180.337184.

[7] Grand View Research. *Video Game Market Size, Share And Growth Report*. 2024.

[8] Visure Solutions. *Requirements Gathering Techniques in Agile Software Engineering*. 2025. URL: https://visuresolutions.com/alm-guide/requirements-gathering-techniques-for-agile (besucht am 27. 01. 2025).