Exposé für eine Bachelorarbeit zum Thema

Konzeption, Implementierung und Evaluation eines Virtual-Tabletop-Plugins für Obsidian.md im Rahmen einer In-App-Plugin-Architektur

Fabian Urbanek

Matrikelnummer: 667074

Hausarbeit

im Modul

Vorbereitungsseminar zur Bachelor-Thesis

im Studiengang Bachelor Wirtschaftsinformatik

der FOM Hochschule für Oekonomie & Management

Fabian Urbanek
Virchowstrasse 23
34121 Kassel

vorgelegt bei

Prof. Dr. Claudius Stern

Kassel, 10. Juli 2025

Zusammenfassung

Das vorliegende Exposé skizziert eine Bachelorarbeit, die die Performance- und Skalierungsgrenzen der Single-Bundle-Plugin-Architektur von Obsidian untersuchen soll. Am Beispiel eines komplexen Virtual Tabletop (VTT)-Plugins wird erforscht, wie sich die Anforderung, alle Plugin-Komponenten in eine einzige JavaScript-Datei zu bündeln, auf die Systemleistung auswirkt.

Die geplante Forschung adressiert eine praktische Herausforderung: Game Master (GM) nutzen Obsidian zur Spielvorbereitung, müssen aber für die eigentliche Spielleitung auf separate VTT-Plattformen wechseln. Ein integriertes VTT-Plugin würde diesen Medienbruch eliminieren, stößt jedoch möglicherweise an technische Grenzen der Plugin-Architektur.

Das Exposé definiert ein experimentelles Forschungsdesign mit fünf Plugin-Varianten unterschiedlicher Komplexität und verschiedenen Optimierungsansätzen. Mittels systematischer Messungen von Startzeit, Speicherverbrauch und Laufzeit-Performance sollen konkrete Schwellenwerte identifiziert werden, ab denen die Single-Bundle-Architektur die Benutzererfahrung beeinträchtigt.

Vorwort

Das in dieser Arbeit gewählte generische Maskulinum bezieht sich zugleich auf die männliche, die weibliche und andere Geschlechteridentitäten. Zur besseren Lesbarkeit wird auf die gleichzeitige Verwendung männlicher und weiblicher Sprachformen verzichtet. Alle Geschlechteridentitäten werden ausdrücklich mit gemeint, soweit die Aussagen dies erfordern.

Inhaltsverzeichnis

Abkürzungsverzeichnis

VTT Virtual Tabletop

GM Game Master

TTRPG Tabletop Role-Playing Game

1 Einleitung

1.1 Tabletop-Rollenspiele und die Rolle des Spielleiters

Ein Tabletop Role-Playing Game (TTRPG) ist eine kooperative Erzählform, bei der mehrere Spieler Charaktere in einer fiktiven Welt verkörpern. Eine zentrale Position nimmt dabei der **Spielleiter** (engl. GM) ein: Er entwirft Schauplätze, verwaltet Nichtspielercharaktere und moderiert Regeln sowie den dramaturgischen Ablauf der Sitzung. All diese Aufgaben erfordern eine umfangreiche Notiz- und Wissensorganisation, etwa Karten, Handouts, Charakterstatistiken und spontane Szenenbeschreibungen.

1.2 Obsidian als solide Basis für Spielleiter

Die Markdown-basierte Notizanwendung **Obsidian** hat sich in der TTRPG-Community als beliebtes Werkzeug zur Spielvorbereitung etabliert. Ihre Stärken liegen in (1) der Dateisystembasierten Ablage, (2) bidirektionalen Links und Graphansicht, (3) einer offenen Plugin-API mit derzeit über 350 Community-Plugins[1]. Blog-Beiträge und Community-Guides zeigen, wie GMs Obsidian zur Weltbeschreibung, Szenenplanung und Regelintegration nutzen[2]. Die Popularität einschlägiger Erweiterungen unterstreicht diesen Bedarf: Das Plugin Fantasy Statblocks zur Darstellung von Monsterwerten wurde inzwischen über 200 000-mal heruntergeladen[3], und der Dice Roller verzeichnet ebenfalls sechsstellige Downloadzahlen[1].

1.3 Motivation für ein integriertes Virtual Tabletop

Während Obsidian die Vorbereitung effizient unterstützt, findet das eigentliche Spiel, häufig über separate VTT-Plattformen, in externen Programmen statt. Ein Plugin, das VTT-Funktionalität direkt in Obsidian bereitstellt, bietet daher mehrere Vorteile:

1. **Nahtloser Workflow:** Spielleiter und Spieler bleiben in ihrer vertrauten Notizumgebung; Kontextwechsel zwischen Vorbereitung und Spiel entfallen.

2. **Synergie mit dem Ökosystem:** Bestehende Plugins (z. B. Automatisierung, Regelreferenzen, Datenbanken) lassen sich kombinieren, ohne Funktionen doppelt implementieren zu müssen.

3. **Ressourcenschonende Entwicklung:** Ein kleines Entwicklerteam kann auf Obsidian als Cross-Plattform-Container (Electron + Markdown-Renderer) zurückgreifen, Deployment, Updater und Basis-UI sind bereits vorhanden.

1.4 Problemstellung

Allerdings verlangt Obsidian, dass jeder Community-Zusatz als **einziges** gebündeltes Skript (`main.js`) ausgeliefert wird[4]. Für ein umfangreiches VTT, mit WebRTC-Netzwerkcode, Grafikengine, Encounter-Logik und Datenbanken, kann dies zu erheblicher Bundle-Größe, langen Startup-Zeiten und UI-Blockaden führen. Vergleichbare Electron-Ökosysteme mussten bereits aufwändige Performance-Optimierungen vornehmen, um großskalige Plugins oder Kernmodule responsiv zu halten[5]–[7]. Eine systematische Untersuchung dieser Grenzen fehlt bislang für Obsidian.

1.5 Zielsetzung der Arbeit

Die vorliegende Bachelorarbeit untersucht daher die Konzeption, Implementierung und Evaluation eines VTT-Plugins für Obsidian.md im Rahmen einer In-App-Plugin-Architektur. Dies umfasst die empirische Messung von

- Startzeit, Hauptspeicher- und CPU-Verbrauch in Abhängigkeit vom Funktionsumfang,
- Wirksamkeit von Gegenmaßnahmen (Lazy Loading, Web Worker, WebAssembly)
- Ableitung von Best-Practices für Entwickler großer Obsidian-Erweiterungen.

Die Arbeit liefert damit sowohl wissenschaftliche Erkenntnisse zur Skalierbarkeit JavaScript-basierter Desktop-Plugins als auch praktische Empfehlungen für die wachsende Obsidian-TTRPG-Community.

2 Diskussion des Arbeitstitels

Der Arbeitstitel lautet:

„Konzeption, Implementierung und Evaluation eines Virtual-Tabletop-Plugins für Obsidian.md im Rahmen einer In-App-Plugin-Architektur"

Ein guter Titel muss inhaltlichen Fokus, methodische Stoßrichtung und Abgrenzung klar erkennen lassen, ohne den Leser mit Detailangaben zu überfrachten. Im Folgenden wird erläutert, warum die gewählte Formulierung diesen Anforderungen gerecht wird.

2.1 Technischer Schwerpunkt und Abgrenzung

Die Begriffe Konzeption, Implementierung und Evaluation strukturieren die Arbeit in drei aufeinander aufbauende Phasen: theoretische Planung, praktische Umsetzung und systematische Bewertung. Die In-App-Plugin-Architektur beschreibt das spezifische Konzept, ein Plugin vollständig innerhalb der Obsidian-Anwendung zu realisieren, wobei sämtlicher Code in eine einzige `main.js`-Datei gebündelt wird[4]. Alternative Architekturen (z. B. Micro-Frontend-Ansätze, Multi-Bundle-Deployments) werden damit implizit ausgeschlossen, was den Untersuchungsrahmen klar eingrenzt.

2.2 Exemplarisches Anwendungsobjekt

Das „VTT-Plugin für Obsidian.md“ definiert den konkreten Anwendungsfall für die Untersuchung. Ein VTT stellt ein ressourcenintensives Anwendungsszenario dar, das typischerweise große Codebasen, Grafik-Engines und Netzwerk-Logik umfasst. Solche Plugins treiben die Obsidian-Architektur nachweislich an ihre Grenzen, wie Erfahrungsberichte zu Lag und hoher CPU-Last belegen[8], [9]. Der explizite Verweis auf „Obsidian.md“ unterstreicht die spezifische Zielplattform und deren Markdown-basierte Architektur. Die Formulierung gewährleistet zudem die Übertragbarkeit der Ergebnisse auf ähnlich komplexe Plugin-Entwicklungen in vergleichbaren Anwendungsumgebungen.

2.3 Verhältnis zu Forschungsfrage und Methodik

Der Titel impliziert mit „Konzeption“, „Implementierung“ und „Evaluation“ ein dreistufiges Vorgehen: theoretische Planung, praktische Umsetzung und systematische Bewertung. Dies entspricht einem konstruktionsorientierten Forschungsansatz, der Artefakterstellung mit empirischer Analyse verbindet[7]. Die Evaluation wird konkrete Metriken wie Startzeit, RAM-Verbrauch oder Bundle-Größe einbeziehen und deren Auswirkungen auf Performance, Wartbarkeit und Entwicklungsaufwand systematisch dokumentieren.

2.4 Leserführung und Verständlichkeit

Vergleichbare Electron-Plattformen (z. B. Figma oder Slack) mussten bereits umfangreiche Optimierungen vornehmen, um monolithische Bundles performant zu halten[5], [6]. Durch den Bezug auf Obsidian wird ein aktuelles, aber noch wenig erforschtes Feld adressiert. Der Titel ist prägnant genug, um Fachlesern sofort das Thema zu erschließen, und vermeidet überflüssige Kürzel oder Methodenbegriffe. Damit erfüllt er die Anforderungen an einen prägnanten Titel in wissenschaftlichen Arbeiten.

Zusammenfassend beschreibt der Arbeitstitel präzise das Forschungsvorhaben: Die Konzeption, Implementierung und Evaluation eines Virtual-Tabletop-Plugins erfolgt im Rahmen einer In-App-Plugin-Architektur. Der Titel grenzt den Untersuchungsgegenstand klar ab und signalisiert sowohl die praktische als auch die wissenschaftliche Dimension der Arbeit durch die explizite Nennung aller drei Arbeitsschritte.

3 Problemstellung

Obsidian setzt bei Community-Erweiterungen strikt auf ein Single-Bundle-Paradigma: Jede Erweiterung wird als einzige, vorab gebündelte `main.js`-Datei in den Renderer-Prozess geladen[4]. Dieser Entwurfsentscheid minimiert Installationsaufwand, führt aber bei wachsender Plugin-Komplexität zu drei beobachtbaren Problembereichen:

1. **Start-up-Latenz** Großvolumige Bundles verlängern die Initialisierung, da sie vollständig heruntergeladen, geparst und kompiliert werden müssen. Anwenderberichte sprechen von spürbaren Wartezeiten, sobald mehrere schwere Plugins aktiv sind[8].
2. **Ressourcenverbrauch** Ein einziges Skript vereint sämtliche Bibliotheken (Grafik, Netzwerk, Parser) und bleibt dauerhaft im Arbeitsspeicher. Dies skaliert schlecht bei datenintensiven Plugins wie einem VTT, das Karten, Token und Regelwerke gleichzeitig vorhalten muss.
3. **UI-Blockierung** Da Plug-ins im gleichen Thread laufen wie Obsidian UI, können lange JavaScript-Tasks zu spürbarem Lag führen. Ähnliche Effekte sind in anderen Electron-Plattformen dokumentiert und erforderten dort erhebliche Performance-Optimierungen[5]–[7].

Trotz der praktischen Relevanz dieser Punkte existiert bislang keine systematische, empirische Untersuchung, die

- die technischen Herausforderungen eines realitätsnahen, umfangreichen Obsidian-Plugins quantifiziert,
- den Effekt gängiger Gegenmaßnahmen (Lazy Loading, Web Worker, WebAssembly) vergleicht
- belastbare Empfehlungen für Entwickler ableitet.

Forschungsfrage

Aus der beschriebenen Lücke leitet sich die zentrale Forschungsfrage ab:

Welche Herausforderungen ergeben sich bei der Umsetzung eines VTT-Plugins in Obsidian.md, und wie beeinflussen diese Performance, Wartbarkeit und Entwicklungsaufwand?

Die Beantwortung dieser Frage erfordert

1. die Entwicklung mehrerer kontrollierter Varianten des VTT-Plugins mit abgestuften Funktionsumfängen,
2. reproduzierbare Benchmark-Messreihen auf identischer Hardware
3. eine statistische Auswertung, um signifikante Effekte der Optimierungsstrategien nachzuweisen.

Beitrag der Arbeit

Die Untersuchung liefert

- quantitative Schwellenwerte, ab denen Single-Bundle-Plugins die Nutzererfahrung merklich beeinträchtigen,
- eine komparative Bewertung praktischer Optimierungsansätze
- ein funktionsfähiges VTT-Plugin, das im Gegensatz zu cloudbasierten Lösungen vollständig lokal läuft und Nutzern volle Kontrolle über ihre Spielinhalte garantiert.

Damit schließt die Arbeit eine Forschungslücke an der Schnittstelle von Desktop-Webtechnologie und Game-Preparation-Workflows und unterstützt gleichzeitig kleine Entwicklerteams dabei, performante High-Feature-Plugins für Obsidian zu realisieren.

4 Methodisches Vorgehen

Die Forschungsfrage erfordert ein **experimentelles Design**, das reproduzierbar, messgenau und statistisch auswertbar ist. Die gewählte Vorgehensweise orientiert sich an Best-Practices für Software-Benchmarking[10], [11] und am Grundkonzept kontrollierter Variantentests[12].

4.1 Versuchsaufbau

Hardware- und Software-Konstanten Alle Messreihen erfolgen auf einem identischen Testsystem mit ausreichender Leistung für reproduzierbare Ergebnisse. Die genaue Hardware-Spezifikation sowie die verwendeten Software-Versionen werden in der finalen Arbeit dokumentiert.

Plugin-Varianten (Feature-Staffel) Das VTT-Plugin wird in mehreren Entwicklungsstufen untersucht, beginnend mit einem minimalen Prototyp bis hin zur vollständigen Funktionalität. Die genaue Anzahl und Abgrenzung der Varianten wird während der Implementierungsphase festgelegt.

Zusätzlich werden verschiedene Optimierungsansätze evaluiert, die sich aus der Literaturrecherche und praktischen Experimenten ergeben. Mögliche Ansätze umfassen Code-Splitting, asynchrones Laden von Komponenten oder die Auslagerung rechenintensiver Operationen.

4.2 Metriken und Messwerkzeuge

- **Performance-Metriken** Startzeit, Speicherverbrauch und CPU-Auslastung werden mit geeigneten Benchmarking-Tools erfasst. Die konkrete Toolauswahl erfolgt basierend auf Verfügbarkeit und Eignung für Electron-Anwendungen.
- **Wartbarkeits-Indikatoren** Code-Komplexität, Modularität und Bundle-Größe werden analysiert, um Rückschlüsse auf die langfristige Wartbarkeit zu ziehen.

- **Entwicklungsaufwand** Dokumentation der benötigten Zeit für Implementierung und Debugging verschiedener Features sowie auftretender Hindernisse.

4.3 Datenerhebung

Die Messungen werden automatisiert durchgeführt, um Reproduzierbarkeit und Konsistenz zu gewährleisten. Die erhobenen Daten werden strukturiert gespeichert und anschließend statistisch ausgewertet. Die genauen Analyse-Methoden richten sich nach der Art der erhobenen Daten und werden in der finalen Arbeit detailliert beschrieben.

4.4 Validitätssicherung

- **Interne Validität** Störfaktoren wie Hintergrundprozesse oder Caching-Effekte werden durch geeignete Maßnahmen minimiert.
- **Externe Validität** Die Implementierung orientiert sich an realistischen Anforderungen eines Virtual Tabletops, sodass die Ergebnisse auf ähnliche Plugin-Projekte übertragbar sind.
- **Replizierbarkeit** Sämtlicher Code, Messdaten und Dokumentation werden öffentlich zugänglich gemacht, um Nachvollziehbarkeit zu gewährleisten.

Dieses methodische Setup ermöglicht eine quantitative Beantwortung der Forschungsfrage und liefert belastbare Daten zu den technischen Herausforderungen sowie deren Einfluss auf Performance, Wartbarkeit und Entwicklungsaufwand in der Single-Bundle-Architektur.

5 Risikobetrachtung

Eine frühzeitige Analyse möglicher Risiken trägt wesentlich zur Machbarkeit und Qualität der Bachelorarbeit bei. Die folgenden Abschnitte strukturieren die wesentlichen Gefährdungen in **technische**, **methodische** und **organisatorische** Risiken und skizzieren jeweils geeignete Gegenmaßnahmen.

5.1 Technische Risiken

R1, Inkonsistente Testumgebung Unterschiedliche Hardware- oder Software-Konfigurationen könnten zu nicht vergleichbaren Messergebnissen führen.
Mitigation: Standardisierte Testumgebung und dokumentierte Systemvoraussetzungen für alle Messungen.

R2, Kompatibilitätsprobleme Die gewählten Technologien oder Optimierungsansätze könnten sich als inkompatibel mit der Obsidian-Architektur erweisen.
Mitigation: Frühzeitige Prototypen und alternative Implementierungsstrategien als Fallback.

R3, Technische Komplexität Die Implementierung eines vollständigen VTTs könnte technisch anspruchsvoller sein als erwartet.
Mitigation: Modularer Aufbau mit klar definierten Minimalanforderungen für jede Entwicklungsstufe.

5.2 Methodische Risiken

R4, Messungenauigkeiten Externe Faktoren könnten die Performance-Messungen beeinflussen und zu ungenauen Ergebnissen führen.
Mitigation: Mehrfache Messungen und statistische Auswertung zur Identifikation von Ausreißern.

R5, Unklare Ursache-Wirkungs-Beziehungen Bei gleichzeitiger Anwendung mehrerer Optimierungen könnte die Zuordnung von Effekten schwierig werden.

Mitigation: Schrittweise Implementierung und isolierte Bewertung einzelner Maßnahmen.

5.3 Organisatorische Risiken

R6, Zeitmanagement Die Entwicklung und Evaluation könnte mehr Zeit in Anspruch nehmen als eingeplant.
Mitigation: Zeitpuffer einplanen und klare Prioritäten für Kernfunktionalitäten setzen.

R7, Datenverlust Technische Probleme könnten zum Verlust von Implementierungsständen oder Messdaten führen.
Mitigation: Regelmäßige Backups und Versionskontrolle mit mehreren Sicherungsebenen.

R8, Abstimmungsschwierigkeiten Terminliche Engpässe bei der Betreuung könnten den Fortschritt behindern.
Mitigation: Frühzeitige Terminplanung und regelmäßige Kommunikation mit dem Betreuer.

Durch diese Maßnahmen werden die größten Unsicherheiten adressiert; verbleibende Restrisiken gelten als akzeptabel und werden während der Projektdurchführung kontinuierlich überwacht.

6 Persönliche Motivation

Ich interessiere mich sehr für sowohl App-Entwicklung als auch kollaborative Storytelling-Spielsysteme wie Dungeons & Dragons und mit diesem Projekt kann ich beides gezielt vereinen.

Obsidian nutze ich bereits seit mehreren Jahren für persönliche Zwecke und unter anderem auch zur Spielvorbereitung. Die Idee, ein VTT direkt zu integrieren, entstand aus praktischem Bedarf: Vorbereitung, Regelreferenz und Spielsession sollen in einer durchgängigen Umgebung vereint werden. Als Entwickler reizt mich besonders die Herausforderung, eine umfangreiche Anwendung unter den Rahmenbedingungen der Single-Bundle-Plugin-Architektur performant umzusetzen.

Die technischen Beschränkungen der Obsidian-Plugin-Architektur stellen eine interessante Herausforderung dar, die kreative Lösungsansätze erfordert. Diese Rahmenbedingungen bieten die Möglichkeit, verschiedene Optimierungsstrategien zu erforschen und deren Wirksamkeit empirisch zu belegen. Gleichzeitig entsteht ein praktischer Nutzen für die Community durch ein lokal lauffähiges Tool.

Die Bachelorarbeit ermöglicht es mir, ein anspruchsvolles Softwareprojekt systematisch zu konzipieren, umzusetzen und wissenschaftlich zu evaluieren. Dabei kann ich theoretisches Wissen aus dem Studium mit praktischer Entwicklungsarbeit verbinden und meine Fähigkeiten in der empirischen Forschung weiterentwickeln.

7 Vorläufige Gliederung mit Mengenschätzung

Die folgende Grobstruktur stellt einen ersten Entwurf dar und wird im Verlauf der Bearbeitung weiter konkretisiert. Die Seitenangaben sind als grobe Richtwerte bei einem Gesamtumfang von ca. 40-60 Seiten zu verstehen.

1. **Einleitung** (ca. 5 Seiten)
    - Hinführung zum Thema und Motivation
    - Darstellung der Problemstellung
    - Zielsetzung und Forschungsfrage
    - Aufbau der Arbeit
2. **Theoretische Grundlagen** (ca. 5-10 Seiten)
    - Technische Rahmenbedingungen (Obsidian, Electron)
    - Konzeptuelle Grundlagen (VTTs, Plugin-Architekturen)
    - Performance-Analyse und Benchmarking-Methoden
    - Stand der Forschung und verwandte Arbeiten
3. **Konzeption und Implementierung** (ca. 15-20 Seiten)
    - Anforderungsanalyse und Systemdesign
    - Entwicklung verschiedener Lösungsansätze
    - Implementierung der Testumgebung
    - Dokumentation der Messverfahren
4. **Evaluation und Ergebnisse** (ca. 5-10 Seiten)
    - Durchführung der Performance-Messungen
    - Auswertung und Interpretation der Daten
    - Vergleich verschiedener Optimierungsstrategien
    - Diskussion der Ergebnisse

5. **Fazit und Ausblick** (ca. 2-5 Seiten)
    - Zusammenfassung der wesentlichen Erkenntnisse
    - Beantwortung der Forschungsfrage
    - Limitationen und kritische Reflexion
    - Ausblick auf zukünftige Entwicklungen

8 Erste Literaturrecherche

Die Thematik der Leistungsoptimierung in Electron-basierten Plugin-Architekturen ist aktuell und weitgehend praxisgetrieben. Entsprechend stützen sich die zu erwartenden Grundlagen vor allem auf technische Dokumentationen, Entwicklerblogs und ausgewählte wissenschaftliche Beiträge zu Performance-Messung, WebAssembly und VTTs. Bei einer ersten Recherche wurden, neben den bereits im Exposé zitierten Quellen, insbesondere die folgenden Titel als nützlich erachtet:

- Obsidian MD, Build a Plugin, Developer Docs, 2024.
- R. Chen, How We Built the Figma Plugin System and Also Sleep Well at Night, Figma Engineering Blog, 2019.
- M. Christian & J. Rodgers, Rebuilding Slack on the Desktop, Slack Engineering Blog, 2019.
- Microsoft, Visual Studio Code, Our Approach to Extensibility, 2015.
- Electron Maintainers, Electron Developer Guide: Performance, 2023.
- E. Wallace, WebAssembly cut Figma's load time by 3×, Figma Blog, 2017.
- Foundry VTT Devs, Foundry VTT Developer Wiki, Canvas Architecture, 2023.
- J. Williams, Speeding Up VS Code (Extensions) in 2022, Tech Blog, 2022.
- D. Prechter, hyperfine, Command-Line Benchmark Tool, GitHub Repo, 2024.
- D. C. Montgomery, Design and Analysis of Experiments, 9. Aufl., Wiley, 2017.

9 Vorläufiger Zeitplan

Der folgende Zeitplan zeigt die geplante 12-wöchige Bearbeitungszeit der Bachelorarbeit. Die Phasen können sich je nach Erkenntnisfortschritt und auftretenden Herausforderungen zeitlich verschieben oder überlappen.

9.1 Projektphasen im Überblick

Tabelle 9.1: Vorläufiger Zeitplan für die Bachelorarbeit (12 Wochen)

Phase	**Arbeitspaket**	**Wochen**											
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Literatur	Literaturrecherche Theoretische Grundlagen												
Konzeption	Anforderungsanalyse Systemdesign												
Implementierung	Prototyp entwickeln Messumgebung												
Evaluation	Performance-Tests Optimierung												
Dokumentation	Datenauswertung Dokumentation												
Finalisierung	Überarbeitung												
Abgabe													**X**

9.2 Meilensteine und Pufferzonen

Der Zeitplan berücksichtigt folgende kritische Aspekte:

- **Überlappende Phasen**: Dokumentation beginnt bereits während der Konzeptionsphase, um kontinuierlich Erkenntnisse festzuhalten
- **Iterative Entwicklung**: Prototyping und Testing laufen teilweise parallel, um frühzeitig Feedback zu erhalten

- **Pufferzeit**: Die letzten drei Wochen vor Abgabe sind primär für Überarbeitung und unvorhergesehene Anpassungen reserviert
- **Flexibilität**: Bei unerwarteten technischen Herausforderungen können Implementierungsphasen verlängert und Evaluationsphasen entsprechend angepasst werden

9.3 Risikofaktoren im Zeitplan

Potenzielle Verzögerungen könnten entstehen durch:

- Technische Komplexität der Plugin-Architektur höher als erwartet
- Schwierigkeiten bei der Performance-Messung in der Obsidian-Umgebung
- Verfügbarkeit und Stabilität der verwendeten Entwicklungstools
- Umfang der erforderlichen Literaturrecherche größer als geplant

Literatur

[1] Obsidian MD. „Community Plugin Directory.“ Downloadzahlen abgerufen am 29.06.2025. (2025), https://obsidian.md/plugins besucht am 29. 06. 2025.

[2] M. Shea. „Using Obsidian for TTRPG Prep.“ (2023), https://slyflourish.com/obsidian_prep.html besucht am 29. 06. 2025.

[3] L. Hartung. „Fantasy Statblocks – Community Plugin.“ > 200 000 Downloads laut Plugin Hub. (2024), https://github.com/lukas-h/fantasy-statblocks-obsidian besucht am 29. 06. 2025.

[4] Obsidian MD. „Build a Plugin.“ (2024), https://docs.obsidian.md/Plugins/Getting%20started/Build%20a%20plugin besucht am 29. 06. 2025.

[5] R. Chen. „How We Built the Figma Plugin System and Also Sleep Well at Night.“ (2019), https://www.figma.com/blog/how-we-built-the-figma-plugin-system/ besucht am 29. 06. 2025.

[6] M. Christian und J. Rodgers. „Rebuilding Slack on the Desktop.“ (2019), https://slack.engineering/rebuilding-slack-on-the-desktop/ besucht am 29. 06. 2025.

[7] J. Williams. „Speeding Up VS Code (Extensions) in 2022.“ (2022), https://jason-williams.co.uk/posts/speeding-up-vscode-extensions-in-2022/ besucht am 29. 06. 2025.

[8] TfTHacker. „Call for plugin performance optimization (Forum-Beitrag).“ (2022), https://forum.obsidian.md/t/call-for-plugin-performance-optimization-especially-for-plugin-startup/32321 besucht am 29. 06. 2025.

[9] lhr0909. „Can plugins use Web Worker? (Diskussion im Obsidian-Forum).“ Beitrag vom 29. April 2024, abgerufen am 28.06.2025. (2024), https://forum.obsidian.md/t/can-plugins-use-web-worker/81040.

[10] D. Prechter, *hyperfine — command-line benchmark tool*, 2024. https://github.com/sharkdp/hyperfine besucht am 29. 06. 2025.

[11] Chrome Developers. „Measure Performance with the Chrome DevTools Performance Panel.“ (2024), https://developer.chrome.com/docs/devtools/performance/ besucht am 29. 06. 2025.

[12] V. R. Basili, R. W. Selby und D. H. Hutchens, „Experimentation in Software Engineering,“ *IEEE Transactions on Software Engineering*, Jg. SE-12, Nr. 7, S. 733–743, Juli 1986. DOI: 10.1109/TSE.1986.6312975.

Ehrenwörtliche Erklärung

Hiermit versichere ich, dass die vorliegende Arbeit von mir selbstständig und ohne unerlaubte Hilfe angefertigt worden ist, insbesondere dass ich alle Stellen, die wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen sind, durch Zitate als solche gekennzeichnet habe. Ich versichere auch, dass die von mir eingereichte schriftliche Version mit der digitalen Version übereinstimmt. Weiterhin erkläre ich, dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde / Prüfungsstelle vorgelegen hat. Ich erkläre mich damit einverstanden/nicht einverstanden, dass die Arbeit der Öffentlichkeit zugänglich gemacht wird. Ich erkläre mich damit einverstanden, dass die Digitalversion dieser Arbeit zwecks Plagiatsprüfung auf die Server externer Anbieter hochgeladen werden darf. Die Plagiatsprüfung stellt keine Zurverfügungstellung für die Öffentlichkeit dar.

Fabian

Kassel, 10.07.2025
(Ort, Datum)

F. Urbanek
(Unterschrift)