FH University of Applied Sciences
TECHNIKUM WIEN

Dokumentation & Projekttagebuch

Innovation Lab 1
Jahr 2025/26

Projekt: **2buggAI**
Team: **34**

1. Allgemeine Informationen

Projektname: 2buggAI

Supervisor: Prof(FH) Dr. DI Mehnen

Innovation Lab *1, Wintersemester 2025/26*

Projektteam:
Nikola Cvetkovic, if24b103@technikum-wien.at, Informatik
Krystian Kedzior, if24b272@technikum-wien.at, Informatik

Management-Summary des Projektes

Ziel des Projekts ist die Entwicklung eines Prototyps für KI-unterstütztes Debugging in verschiedenen Programmiersprachen. Der Ansatz nutzt vor allem große Sprachmodelle (LLMs) wie Claude oder ChatGPT zur automatischen Fehlerbeschreibung, -lokalisierung und -behebung bei minimaler Codeänderung. Untersucht werden dabei unterschiedliche Fehlertypen – von syntaktischen und logischen über Designfehler bis hin zu schwer reproduzierbaren Heisenbugs und Schrödinger-Fehlern. Durch die Integration von Debugging-Tools wie Valgrind oder gdb soll das Projekt den Debugging-Prozess effizienter, verständlicher und robuster gestalten.

Rahmenbedingungen und Projektumfeld

Das Projekt wird im Kontext moderner Softwareentwicklung mit Fokus auf Qualität, Performance und Fehlertoleranz umgesetzt. Da der Prototyp für KI-unterstütztes Debugging mehrere Programmiersprachen und bestehende Tools (z. B. Valgrind, gdb) integriert, ist eine modulare und sprachübergreifende Architektur erforderlich. Grundlage bildet der Einsatz großer Sprachmodelle (LLMs) wie Claude, ChatGPT oder Mistral, die über sichere Schnittstellen eingebunden werden. Besondere Aufmerksamkeit gilt dabei Datenschutz- und Sicherheitsaspekten, da bei Debugging-Sessions potenziell sensible Quellcodedaten verarbeitet werden.
Das System wird auf gängigen Entwicklungsplattformen lauffähig sein und offene Schnittstellen für die Integration in bestehende IDEs (z. B. VS Code) vorsehen. Tests erfolgen in einer kontrollierten Entwicklungsumgebung mit simulierten Fehlern unterschiedlicher Typen (syntaktisch, semantisch, logisch).
Der erste lauffähige Prototyp soll bis **Ende des vierten Sprints** bereitstehen, um anschließend für Usability- und Funktions-Tests genutzt zu werden.

Semester-Roadmap

Das Projekt wird in **sechs Sprints** über das Semester hinweg umgesetzt. Das Team besteht aus **zwei Mitgliedern**, die gemeinsam Planung, Entwicklung, Test und Dokumentation des KI-gestützten Debugging-Prototyps übernehmen. Jeder Sprint umfasst rund zwei bis drei Wochen. Ziel ist es, bis zum Ende des sechsten Sprints einen lauffähigen Prototyp sowie eine vollständige Abschlusspräsentation bereitzustellen.

Sprint	Zeitraum	Ziele / Schwerpunkte
1	Projektstart	Anforderungsanalyse, Recherche zu LLM-Tools (Claude, ChatGPT, Mistral) und Debugging-Umgebungen (Valgrind, gdb), Einrichtung der Entwicklungsumgebung
2	Konzeptphase	Architekturentwurf, Definition der Schnittstellen zwischen LLM und Debugging-Tools, Erstellung erster Beispieldaten
3	Prototyping I	Implementierung der Basisfunktionen (Fehlererkennung, Fehlerbeschreibung, Kommunikation mit LLM), erster funktionsfähiger Prototyp
4	Prototyping II	Erweiterung um komplexe Fehlertypen (Heisenbugs, Schrödinger-Bugs), Tests mit unterschiedlichen Programmiersprachen
5	Finalisierung des Prototyps	Usability-Tests, Optimierung der LLM-Prompts, Performance-Analyse, Sicherstellung minimaler Codeänderungen
6	Abschlussphase	Dokumentation, Erstellung und Durchführung der Abschlusspräsentation

Geschätzter Gesamtaufwand: ca. **200 Personenstunden** (2 Personen × 6 Sprints × ~17,5 Stunden/Sprint).
Das Projekt ist so geplant, dass alle wesentlichen Funktionen und Tests bis **Ende des fünften Sprints** abgeschlossen sind.

Collaboration & Tooling

https://github.com/Dirk1306/2buggAI - GIT, VS Code, Intellij Ultimate

Anmerkungen

-

2. Projekt-Kurzbeschreibung

Das Ziel des Projekts ist die Entwicklung eines **KI-gestützten Debugging-Prototyps**, der Entwicklerinnen beim Erkennen, Verstehen und Beheben von Programmfehlern unterstützt. Der Prototyp soll mit Hilfe **großer Sprachmodelle (LLMs)** wie ChatGPT, Claude oder Mistral den Debugging-Prozess effizienter machen. Dabei steht nicht die reine Code-Korrektur im Vordergrund, sondern die **Fehleranalyse, Lokalisierung und Beschreibung**, um Entwicklerinnen beim Verständnis komplexer Fehlerursachen zu unterstützen.

Das System soll in der Lage sein, verschiedene **Fehlertypen** – darunter syntaktische, semantische, logische oder Designfehler – zu erkennen und zu erklären. Besonders relevant sind schwer reproduzierbare Fehler wie **Heisenbugs** (Fehler, die nur unter bestimmten Umständen auftreten) oder **Schrödinger-Bugs** (Fehler, die erst beim Lesen oder Analysieren des Codes erkannt werden). Dabei wird die KI in eine bestehende Debugging-Umgebung integriert und mit gängigen **Tools wie Valgrind oder gdb** kombiniert, um eine automatisierte Fehleranalyse zu ermöglichen.

Ziele und Deliverables

Am Ende des Projekts soll ein funktionierender **Prototyp** vorliegen, der folgende Funktionen erfüllt:

- Annahme von Quellcode und Fehlerausgaben (Logs, Compiler- oder Laufzeitfehler)
- Automatische **Fehlerbeschreibung** und **Ursachenanalyse** durch ein LLM
- Lokalisierung fehlerhafter Codeabschnitte
- Vorschläge für mögliche Korrekturen mit minimalen Codeänderungen
- Integration mit mindestens einem Debugging-Tool (z. B. Valgrind oder gdb)
- Dokumentation der Debugging-Pipeline und Evaluierung der Ergebnisse

Das Projektteam (2 Personen) setzt diese Funktionen innerhalb von **6 Sprints** um. Im **fünften Sprint** soll der Prototyp vollständig lauffähig sein und im Rahmen einer Präsentation vorgestellt werden.

Größte Herausforderungen

- **Zuverlässigkeit von LLMs:** Sprachmodelle neigen zu ungenauen oder halluzinierten Aussagen. Eine klare Validierung der KI-Antworten ist notwendig.
- **Integration verschiedener Technologien:** Die Kombination von LLMs mit traditionellen Debugging-Tools und mehreren Programmiersprachen erfordert eine flexible Architektur.
- **Fehlerbewertung:** Die Identifikation der tatsächlichen Ursache eines Fehlers bleibt anspruchsvoll und erfordert Tests in verschiedenen Szenarien.

Mehrwert für Anwender*innen

Der größte Nutzen liegt in der **Zeitersparnis** und **Verbesserung der Fehlersuche**. Durch die Nutzung von LLMs können Entwickler*innen Fehler schneller verstehen und beheben, insbesondere in komplexen Projekten oder bei schwer reproduzierbaren Fehlern. Die Lösung fördert zudem den Wissenstransfer, da die KI Erklärungen in natürlicher Sprache liefert – ideal für Lern- und Ausbildungssituationen.

Projektumfang (Scope)

Im Rahmen dieses Projekts wird **ein Prototyp** erstellt, der grundlegende Debugging-Unterstützung in typischen Programmierumgebungen demonstriert.
Nicht-Ziele sind:

- Die vollständige Automatisierung des Debugging-Prozesses
- Eine kommerzielle oder produktionsreife Software
- Unterstützung aller Programmiersprachen oder komplexer Build-Systeme

Stattdessen liegt der Fokus auf der **Machbarkeitsanalyse und Funktionsdemonstration** eines intelligenten, KI-basierten Debugging-Workflows.

Vorgehensweise

Die Entwicklung folgt einer iterativen Vorgehensweise mit **6 Sprints** (Planung, Prototyping, Testing). In den frühen Phasen werden bestehende Debugging-Tools und LLMs evaluiert, um die optimale Kombination zu bestimmen. Darauf folgt die Implementierung einer modularen Pipeline zur automatischen Fehleranalyse. Abschließend wird der Prototyp getestet und anhand realistischer Szenarien bewertet.

3. Spezifikation der Lösung

3.1 Systemumfeld (Abgrenzung der Lösung)

Die zu entwickelnde Lösung ist ein prototypisches, KI-gestütztes Debugging-System zur Analyse von Programmcode und zugehörigen Debug-Ausgaben.
Das System wird als eigenständige Kommandozeilenanwendung umgesetzt und dient als Analysewerkzeug für Entwicklerinnen und Entwickler.
Zum System gehören:

- Entgegennahme von Quellcode-Dateien
- Ausführung externer Debugging-Werkzeuge
- Aufbereitung von Debug-Ausgaben
- Übergabe der Analyseinformationen an ein Sprachmodell
- Ausgabe strukturierter Fehlerberichte

Nicht zum System gehören:

- Compiler oder Build-Systeme
- Entwicklungsumgebungen (IDEs)
- Betriebssysteme
- Automatische Fehlerkorrektur
- Ausführung produktiver Anwendungen

Das System verarbeitet externe Artefakte (Code, Debug-Ausgaben), ist jedoch nicht selbst für deren Erzeugung verantwortlich.

3.2 Features (Funktionale Anforderungen)

Die funktionalen Anforderungen werden in Form von User Stories spezifiziert und bilden das Product Backlog.

Epic 1: Code- und Fehleranalyse

User Story 1:
Als Entwickler möchte ich Quellcode an das System übergeben können, damit dieser analysiert werden kann.

User Story 2:
Als Entwickler möchte ich, dass das System Debug-Ausgaben verarbeitet, um zusätzliche Informationen über Laufzeitfehler zu erhalten.

Epic 2: Integration von Debugging-Werkzeugen

User Story 3:
Als Entwickler möchte ich, dass das System GDB automatisiert ausführen kann, um Stacktraces und Laufzeitfehler zu erfassen.

User Story 4:
Als Entwickler möchte ich, dass das System Valgrind automatisiert ausführen kann, um Speicherfehler zu erkennen.

Epic 3: KI-gestützte Analyse

User Story 5:
Als Entwickler möchte ich, dass Code und Debug-Ausgaben an ein Sprachmodell übergeben werden, um eine inhaltliche Fehleranalyse zu erhalten.

User Story 6:
Als Entwickler möchte ich eine strukturierte Beschreibung des Fehlers erhalten, damit ich das Problem schneller verstehe.

Epic 4: Ergebnisdarstellung

User Story 7:
Als Entwickler möchte ich, dass mir das Analyseergebnis in strukturierter Form ausgegeben wird, damit es nachvollziehbar und weiterverwendbar ist.

Benutzeroberfläche (Mockups)
Da es sich um eine Kommandozeilenanwendung handelt, besteht die Benutzeroberfläche aus textbasierter Interaktion.
Wesentliche UI-Ansichten:

- Startaufruf mit Parametern (Dateipfad, Toolauswahl)
- Anzeige des Analyseberichts
- Anzeige von Fehlermeldungen bei ungültigen Eingaben

(Visuelle Mockups entfallen zugunsten einer textbasierten Darstellung.)

3.3 Schnittstellen

Benutzerschnittstelle

- Kommandozeilenparameter (Dateipfade, Analyseoptionen)
- Standardausgabe zur Anzeige der Ergebnisse

Schnittstelle zu Debugging-Werkzeugen

- Aufruf externer Prozesse (GDB, Valgrind)
- Einlesen der erzeugten Textausgaben
- Weiterverarbeitung relevanter Informationen

Schnittstelle zum Sprachmodell

- Übergabe strukturierter Analyseinformationen (Code, Debug-Daten)
- Empfang strukturierter Textantworten
- Austauschformat: JSON

3.4 Qualitätseigenschaften und technische Anforderungen

(Nicht-funktionale Anforderungen)

- **Benutzbarkeit:**
  Die Bedienung erfolgt über eine einfache Kommandozeilenschnittstelle mit klaren Parametern.
- **Erweiterbarkeit:**
  Das System ist modular aufgebaut, sodass weitere Debugging-Werkzeuge oder Sprachmodelle ergänzt werden können.
- **Wartbarkeit:**
  Trennung von Tool-Anbindung, Analyse-Logik und KI-Kommunikation.
- **Performance:**
  Die Analyse soll in angemessener Zeit erfolgen und sich an der Laufzeit der Debugging-Werkzeuge orientieren.

- **Portabilität:**
  Einsatz standardisierter Werkzeuge (C++, externe Prozesse).
- **Zuverlässigkeit:**
  Gleiche Eingaben sollen zu reproduzierbaren Ergebnissen führen.

3.5 Sonstige wesentliche Lösungsmerkmale

- Die Lösung ist als Prototyp konzipiert und nicht als produktives Debugging-System.
- Der Fokus liegt auf der Kombination klassischer Debugging-Werkzeuge mit KI-gestützter Analyse.
- Es erfolgt keine automatische Fehlerbehebung, sondern ausschließlich eine Analyse und Beschreibung von Fehlern.
- Das zugrunde liegende Sprachmodell ist austauschbar und nicht fest an eine konkrete Implementierung gebunden.
- Die Lösung dient primär Demonstrations- und Evaluationszwecken.

4. Aufwandschätzung

Für die Umsetzung des Projekts wird der Gesamtaufwand auf Grundlage der geplanten **6 Sprints** und der **Teamgröße von vier Personen (aktuell zwei)** geschätzt.
Als Schätzmethode wurde eine **intuitive Aufwandsschätzung kombiniert mit der Sprint-Planung** (ähnlich der Planning-Poker-Methode) verwendet. Dabei wurde der durchschnittliche Zeitaufwand pro Sprint pro Person mit ca. **15–20 Stunden** angesetzt, um die Konzeption, Entwicklung, Tests und Dokumentation realistisch abzubilden.

Sprint	Inhalt / Schwerpunkt	Geschätzter Aufwand (Personenstunden)
1	Projektstart, Anforderungsanalyse, Setup	60–80
2	Konzeptphase, Architektur, Schnittstellenplanung	30–40
3	Implementierung Basisfunktionen, erster Prototyp	40–50
4	Erweiterung um komplexe Fehlertypen, Integration	40–50
5	Testing, Debugging, Optimierung	30–60
6	Dokumentation, Präsentation, Abschlussarbeiten	20–30
Gesamt	—	**ca. 120–300 Stunden**

Geschätzter Gesamtaufwand: ca. **200 Personenstunden** (2 Personen × 6 Sprints × ~17,5 Stunden/Sprint).

Diese Schätzung berücksichtigt sowohl Entwicklungsarbeit als auch Zeit für Planung, Kommunikation, Testing, Präsentationsvorbereitung und Dokumentation. Der Aufwand wird im Verlauf des Projekts regelmäßig überprüft und bei Bedarf angepasst.

In InnoLab 2 und InnoLab 3: Verwenden Sie die erklärte Delphi Methode, um den Aufwand für diese Semester zu schätzen, schreiben Sie hier die Ergebnisse erklärend rein und verweisen Sie auch auf das verwendete Excel Dokument. >

5. Auslieferung

< In diesem Abschnitt beschreiben Sie den Lieferumfang Ihrer Lösung und alles was man benötigt, um diese an einen Kunden oder ein anderes Softwareteam weiterzugeben (wird in der Praxis auch oft als „Hand-over to Operations“ bezeichnet, wenn die Lösung in die Betriebsphase übergeht).

- *Fertige Lösung oder Lösungskomponenten inklusive Source-Code*
- *Systemarchitektur und Datenhaltung*
- *Liste etwaig benötigter Lizenzen und Info über Copyrights (z.B. wenn Dritt-Software / Frameworks o.ä. verwendet wurden).*
- *Etwaige Vorgaben zur Hardware*
- *Beschreibung wie man Ihre Lösung installiert inklusive Liste aller zu installierenden Komponenten, Installationsprozeduren, Migration von Datenbeständen, etc.*

*Die Inhalte dieses Abschnitts zumeist projektspezifisch. Stimmen Sie mit Ihrer Betreuer*in ab, was dieser Abschnitt genau enthalten soll!*

6. Unser Projekt-Tagebuch

Woche 1 – Kick-off mit dem Betreuer

Im ersten Meeting mit unserem Betreuer haben wir das Projektthema und die Erwartungen besprochen. Ziel ist die Entwicklung eines KI-gestützten Debugging-Prototyps mit Fokus auf Fehlerbeschreibung und Lokalisierung. Zudem wurden mögliche Technologien (LLMs wie ChatGPT, Claude, Mistral sowie Debugging-Tools wie Valgrind oder gdb) diskutiert und der grobe Projektumfang festgelegt.

Erkenntnisse:

- Projektziel und Scope definiert
- Erwartungshaltung des Betreuers geklärt
- Erste Aufgabenverteilung im Team

Woche 2 – Team-Meeting und Dokumentation

Im zweiten Meeting haben wir im Team an der Projektdokumentation gearbeitet, die Inhalte wie Management Summary, Roadmap und Sprints erstellt. Wir haben außerdem erste Ideen gesammelt, wie die Debugging-Pipeline des Prototyps aufgebaut werden könnte und welche Tools wir dafür evaluieren wollen.

Erkenntnisse:

- Dokumentstruktur und Aufgabenverteilung festgelegt
- Erste technische Ansätze besprochen

Sprint 1 – Projektstart (29.10.2025)

In Sprint 1 lag der Fokus auf dem **Projektstart und der grundlegenden Vorbereitung**. Das Team führte eine erste **Anforderungsanalyse** durch, definierte die Projektziele und begann mit der Recherche zu den einzusetzenden Technologien.
Ziel war es, eine gemeinsame technische und organisatorische Basis zu schaffen, damit in den folgenden Sprints effizient gearbeitet werden kann.

Tätigkeiten im Team:

- Das Projekt wurde inhaltlich mit dem Betreuer abgestimmt.
- Eine **Projektstruktur** (Ordner, Repository, Versionsverwaltung) wurde eingerichtet.
- Recherchen zu **LLMs** (Claude, ChatGPT, Mistral) und **Debugging-Tools** (Valgrind, gdb) wurden durchgeführt, um die am besten geeignete Ansätze für die Integration zu bewerten.
- Erste **Projektdokumente** (Management Summary, Roadmap, Aufwandsplanung) wurden erstellt.
-

Aufgabenverteilung:

- **Haakon Hu:** Einrichtung der Projektstruktur und GitHub-Repository, Grundaufbau der Dokumentation (18 h)
- **Krystian Piotr Kedzior:** Vergleich und Analyse verschiedener LLMs hinsichtlich Debugging-Fähigkeit und API-Nutzung (17 h)
- **Nikola Cvetkovic:** Recherche von Debugging-Tools wie Valgrind und gdb (16 h)

- **Yasin Hammad:** Zusammenführung der Dokumentation, Erstellung der Sprintübersicht und Anforderungen (17 h)

Erkenntnisse:

- Technologische Basis und Toolauswahl stehen fest.
- GitHub-Workflow und Aufgabenverteilung im Team sind klar definiert.
- Die weitere Arbeit kann ab Sprint 2 inhaltlich starten (Architektur- und Konzeptphase).

Sprint 2 – Architektur und Funktionsdefinition

Im zweiten Sprint wurde die grundlegende Struktur des Systems festgelegt. Es wurde eine vereinfachte Architektur entworfen, bestehend aus einer Kommandozeilenanwendung zur Eingabe von Programmcode, einer Analysekomponente sowie einer KI-Schicht zur Erzeugung strukturierter Fehlerberichte. Zusätzlich wurde vorgesehen, externe Debugging-Werkzeuge wie GDB und Valgrind einzubinden.
Die zentralen Funktionen des Systems wurden definiert, insbesondere die Art der Eingaben (Quellcode, Fehlermeldungen, Debug-Ausgaben) und der Ausgaben (Fehlerbeschreibung, Ursache, betroffene Codebereiche). Der Fokus wurde auf ausgewählte Fehlertypen wie Speicherfehler, einfache Logikfehler und grundlegende Race Conditions gelegt.

Aufgabenverteilung:

Krystian:
- Konzept für die KI-Integration
- Definition von Eingabe- und Ausgabeformaten

Nikola:
- Analyse von GDB und Valgrind
- Planung der technischen Einbindung der Debugging-Tools

Sprint 3 – Konzeptuelle Vorbereitung

In diesem Sprint lag der Schwerpunkt auf der Vorbereitung der nächsten Entwicklungsphase. Aufgrund parallel stattfindender Prüfungen wurde bewusst auf eine Implementierung verzichtet und stattdessen die geplanten technischen Schritte konkretisiert. Ziel war es, eine klare Grundlage für die Umsetzung im folgenden Sprint zu schaffen.

Aufgabenverteilung:

Krystian:
- Planung der Struktur der KI-Anfragen
- Konzeption des Datenformats für die spätere Analyse

Nikola:
- Planung der Einbindung von GDB und Valgrind
- Analyse relevanter Debug-Ausgaben

Sprint 4 – Integration und Tests

Im vierten Sprint wurde der Prototyp funktional erweitert und getestet. GDB und Valgrind wurden in den Ablauf integriert, sodass Debug-Informationen automatisiert erzeugt und weiterverarbeitet werden konnten. Anhand mehrerer fehlerhaften Programme wurde überprüft, ob der Datenfluss von der Codeeingabe bis zur KI-Ausgabe funktioniert.

Der Fokus lag dabei auf der Stabilität des Ablaufs und der sinnvollen Strukturierung der Analyseinformationen.

Aufgabenverteilung:

Krystian:
- Strukturierung der KI-Anfragen und Antworten
- Unterstützung bei der Auswertung der Testergebnisse

Nikola:
- Technische Integration von GDB und Valgrind
- Durchführung der Debug-Tests

LLM-Wechsel

Zusätzlich wurde zwischen Sprint 4 und Sprint 5 ein Wechsel des verwendeten Sprachmodells vorgenommen. Während in den vorherigen Sprints ein Prototyp auf Basis von Claude verwendet wurde, erfolgte in dieser Phase die Umstellung auf die OpenAI-API. Die bestehende Struktur wurde entsprechend angepasst, sodass die Kommunikation mit dem neuen Modell weiterhin in das bestehende System eingebettet werden konnte.

Sprint 5 – Stabilisierung und Dokumentation

Im letzten Sprint wurde der Prototyp konsolidiert. Kleinere Fehler im Ablauf wurden behoben und die Ergebnisse der Testläufe ausgewertet. Zusätzlich wurde der Aufbau des Systems dokumentiert, um die Funktionsweise nachvollziehbar darzustellen.
Am Ende dieses Sprints stand ein funktionsfähiger Demonstrator, der den Ansatz eines KI-gestützten Debugging-Systems praktisch umsetzt.

Aufgabenverteilung:

Krystian:
- Verbesserung der KI-Berichte
- Auswertung der Analyseergebnisse

Nikola:
- Stabilisierung der Debugging-Komponente
- Fehlerbehebung bei der Tool-Ausgabe

Sprint 6 – Abschlussphase und Präsentationsvorbereitung

Im sechsten Sprint lag der Schwerpunkt auf dem Projektabschluss. Der bestehende Prototyp wurde nochmals überprüft und kleinere Unstimmigkeiten im Ablauf korrigiert. Zudem wurde das Gesamtsystem als Demonstrator vorbereitet, um den vollständigen Analyseprozess von der Codeeingabe bis zur KI-Ausgabe zeigen zu können.
Ein weiterer Fokus lag auf der Zusammenführung aller Ergebnisse sowie auf der Vorbereitung der Präsentation und der schriftlichen Abgabe. Dabei wurden sowohl die technische Umsetzung als auch die gewonnenen Erkenntnisse zusammengefasst und reflektiert.