Modular Multi-Stage Agent for Bug Fixing - Analysis of Potentials and Limitations

Abschlussarbeit

zur Erlangung des akademischen Grades

Bachelor of Science (B.Sc.)

an der

Hochschule für Technik und Wirtschaft (HTW) Berlin
Fachbereich 4: Informatik, Kommunikation und Wirtschaft
Studiengang *Internationale Medieninformatik*

1. Gutachter_in: Prof. Dr. Gefei Zhang
2. Gutachter_in: Stephan Lindauer

Eingereicht von Justin Gebert [s0583511]

22.07.2025

Danksagung

[Text der Danksagung]

Abstract

[Summary of the thesis]

Contents

1. Introduction **1**
- 1.1. Background and Motivation . . . 1
- 1.2. Problem Statement . . . 1
- 1.3. Objectives and Research Questions . . . 1

2. Background and Related Work **2**
- 2.1. Software Engineering and Automated Programm Repair . . . 2
- 2.2. History of Automated Program Repair . . . 2
- 2.3. LM-Based Tool Use and CI Context . . . 2
- 2.4. Related Work . . . 2

3. Requirements **3**
- 3.1. Functional Requirements . . . 3
- 3.2. Non-Functional and Safety Requirements . . . 4
- 3.3. Benchmark Setup . . . 4

4. Methodology **5**
- 4.1. Preparation . . . 5
  - 4.1.1. Dataset Selection . . . 5
  - 4.1.2. System Architecture . . . 5
  - 4.1.3. System Components . . . 5
  - 4.1.4. System Configuration . . . 5
- 4.2. Evaluation Stragegy and Metrics . . . 5

5. Implementation **6**
- 5.1. Implementation . . . 6

6. Results **7**

7. Discussion **8**

8. Conclusion **9**
- 8.1. Summary of Findings . . . 9
- 8.2. Lessons Learned . . . 9
- 8.3. Roadmap for Extensions . . . 9

References **10**

A. Appendix **11**
- A.1. Quell-Code . . . 11
- A.2. Tipps zum Schreiben Ihrer Abschlussarbeit . . . 11

List of Figures

List of Tables

3.1. Functional requirements (F0–F8) 3
3.2. Non-Functional (N1–N5) and Safety (S1–S4) requirements 4

Listings

5.1. Ein Beispiel: Hello World (Scala) 6

1. Introduction

1.1. Background and Motivation

Llms and Agents leveraging LLMs have revolutionized the field of software engineering, enabling new paradigms in automated programming and repair. However, the integration of these models into existing software development workflows remains a challenge. This thesis aims to explore the potential of LLMs in enhancing automated program repair (APR) systems, particularly in continuous integration and continuous deployment (CI/CD) environments.

1.2. Problem Statement

modern APR systems requiremt a lot of computational power budget and manual effort [1]

1.3. Objectives and Research Questions

The primary objective of this thesis is to investigate the feasibility and effectiveness of LLMs in APR within the Software Development Lifecycle (CI/CD pipelines) in budget restrained environment. The research questions guiding this investigation include:

2. Background and Related Work

2.1. Software Engineering and Automated Programm Repair

Software engineering is a complex dicipline consisting of constant

2.2. History of Automated Program Repair

2.3. LM-Based Tool Use and CI Context

2.4. Related Work

3. Requirements

3.1. Functional Requirements

ID	Title	Description	Verification
F0	Issue Gathering	Query GitHub for all open issues labeled `BUG` in the repository.	`runAgent` logs list of fetched issue numbers and URLs.
F1	Fetch Code	Fetch the code referenced by the issue into a fresh workspace (via Docker mount).	After F1, workspace/issue/ contains the correct source files.
F2	Apply Patch	Apply an LLM-generated diff patch to the workspace files.	After patch, `git diff` output matches the patch payload.
F3	Build & Test	Run the project's test suite inside a Docker container and capture pass/fail status.	Docker exit code = 0 for pass and a JSON report file exists.
F4	Report Results	Open a PR or post a comment on GitHub with the diff and summary metrics.	A PR or comment appears for each issue, showing diff and summary.
F5	Modular Stages	Implement stages x,y,z as separate modules to generate working patches.	Each stage can be toggled on or off via configuration without breaking the pipeline.
F6	Metrics Collection	Log fix-rate, CI-cycle time, and sandbox events in CSV or JSON format.	A metrics file contains fields: issue, pass/fail, time, incidents.
F7	Per-Issue Trigger	Execute F1–F4 for each fetched issue in sequence and aggregate results.	For N issues, produce N PRs (or "no-fix" comments) and N metric entries.
F8	GitHub Integration	Use the GitHub API (with `GITHUB_TOKEN`) to list issues, create branches, open PRs, and post comments.	All GitHub API calls succeed without manual intervention.
F9	Cost Accounting	The system shall record prompt tokens, completion-tokens and total cost per issue aggregated into a metrics file.	The metrics file contains fields: issue, prompt-tokens, completion-tokens, cost.

Table 3.1.: *Functional requirements (F0–F8)*

3.2. Non-Functional and Safety Requirements

ID	Title	Description	Verification
N1	Performance Budget	Total wall-clock time per issue run $\leq$ X minutes (including Docker startup).	Average CI-cycle time across issues <= X minutes.
N2	Resource Limits	Docker container limited to <= X GB RAM and <= X CPU cores.	Launched with `-memory=Xg -cpus=X`; verify via container stats.
N3	Reproducibility	Runs are deterministic given identical repo state and config.	Multiple runs on the same issue yield identical metrics.
N4	Configurability	User can specify issue labels, timeouts, resource caps, and stage toggles via YAML or JSON.	Changing the config file alters agent behavior accordingly.
N5	Scheduling Config	Workflow can be scheduled via cron or manually triggered (GitHub Actions workflow_dispatch).	Adjusting the schedule in Actions YAML takes effect.
N6	Cost Budget	cost per issue run <= X	Average cost across issues <= X.
S1	Filesystem Isolation	Prevent reads/writes outside the workspace directory (no escapes).	Attempts to access paths outside workspace/ are blocked and logged.
S2	Network Whitelist	Block all outbound network traffic except to configured LLM API endpoints.	Non-LLM outbound connections are refused by Docker network policy.
S3	Rollback on Failure	On test or policy failure, auto-reclone a fresh workspace copy before retry.	After failure, workspace resets to its pre-run state.
S4	Command Whitelisting	Only allow predefined shell commands (e.g., git, pytest, npm test); block others.	Forbidden commands (e.g., rm -rf /) are denied.

Table 3.2.: *Non-Functional (N1–N5) and Safety (S1–S4) requirements*

3.3. Benchmark Setup

- QuixBugs problems (Python).
- Prepare a base Docker image (e.g., `python:3.10`) with pytest and JSON-report support.
- Ensure each workspace clone is clean (no leftover artifacts).
- Record benchmark IDs and Docker image tags in a configuration file for reproducibility.

4. Methodology

4.1. Preparation

4.1.1. Dataset Selection

4.1.2. System Architecture

4.1.3. System Components

4.1.4. System Configuration

4.2. Evaluation Stragegy and Metrics

- Metrics: - Execution Time - Execution Time in CI/CD - Repair Success Rate - nubmer of Attempts - Security issues - Number of Vulnerabilities - Cost per issue

5. Implementation

[Beschreibung der Implementierung[1]auf Basis des Entwurfs und der Methodologie / der geplanten Vorgehensweise zur Problemlösung im Kontext der Anforderungen. Hier ist Raum für Listings, wie z.B. das nun Folgende:

```
object HelloWorld {
def main(args: Array[String]): Unit = {
  println("Hello, world!")
}
}
```

Listing 5.1: *Ein Beispiel: Hello World (Scala)*

5.1. Implementation

Here we break down the implementation of the system into its core components, following the design and methodology outlined in the previous sections. The full code is attached in the **??** appendix.

[1]Beachten Sie bei der Implementierung und deren Dokumentation bitte Clean Code Empfehlungen (vgl. hierzu z.B. [**martin2008**]).

6. Results

[Beschreibung der Ergebnisse aus allen voran gegangenen Kapiteln sowie der zuvor generierten Ergebnisartefakte mit Bewertung, wie diese einzuordnen sind]

7. Discussion

7.1.

8. Conclusion

8.1. Summary of Findings

8.2. Lessons Learned

8.3. Roadmap for Extensions

References

[1] Meghana Puvvadi et al. “Coding Agents: A Comprehensive Survey of Automated Bug Fixing Systems and Benchmarks”. In: *2025 IEEE 14th International Conference on Communication Systems and Network Technologies (CSNT)*. Mar. 2025, pp. 680–686. DOI: 10.1109/CSNT64827.2025.10968728. (Visited on 04/27/2025).

A. Appendix

A.1. Quell-Code

A.2. Tipps zum Schreiben Ihrer Abschlussarbeit

- Achten Sie auf eine neutrale, fachliche Sprache. Keine „Ich“-Form.
- Zitieren Sie zitierfähige und -würdige Quellen (z.B. wissenschaftliche Artikel und Fachbücher; nach Möglichkeit keine Blogs und keinesfalls Wikipedia[1]).
- Zitieren Sie korrekt und homogen.
- Verwenden Sie keine Fußnoten für die Literaturangaben.
- Recherchieren Sie ausführlich den Stand der Wissenschaft und Technik.
- Achten Sie auf die Qualität der Ausarbeitung (z.B. auf Rechtschreibung).
- Informieren Sie sich ggf. vorab darüber, wie man wissenschaftlich arbeitet bzw. schreibt:
  - Mittels Fachliteratur[2], oder
  - Beim Lernzentrum[3].
- Nutzen Sie LaTeX[4].

[1] Wikipedia selbst empfiehlt, von der Zitation von Wikipedia-Inhalten im akademischen Umfeld Abstand zu nehmen [**wikipedia2019**].

[2] Z.B. [**balzert2011**], [**franck2013**]

[3] Weitere Informationen zum Schreibcoaching finden sich hier: `https://www.htw-berlin.de/studium/lernzentrum/studierende/schreibcoaching/`; letzter Zugriff: 13 VI 19.

[4] Kein Support bei Installation, Nutzung und Anpassung allfälliger LaTeX-Templates!

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt durch meine Unterschrift, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe, mich auch keiner anderen als der angegebenen Literatur oder sonstiger Hilfsmittel bedient habe. Die Arbeit hat in dieser oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

_____

Datum, Ort, Unterschrift