Die Hausaufgaben müssen von jedem Studierenden einzeln bearbeitet und abgegeben werden. Für die Hausaufgabe sind die aktuellen Informationen vom Blog https://seblog.cs.uni-kassel.de/ss24/generative-ki-in-der-software-technik/ zu berücksichtigen.

Abgabefrist ist der 17.06.2024 – 23:59 Uhr

Abgabe

Wir benutzen für die Abgabe der Hausaufgaben Git. In eurem Repository soll sowohl der aktuelle Stand des Berichtsheftes, als auch die zu den jeweiligen Aufgaben gehörenden Ergebnisse an generiertem Code oder anderen Artefakten persistiert werden.

Wir nutzen **weiterhin** das Repository, was in Hausaufgabe 1 bereits unter folgendem Link angelegt wurde:

https://classroom.github.com/a/XeAtEq1Z

Die Bearbeitung der gestellten Aufgaben muss wie in Vorlesung und Übung besprochen in einem einheitlichen Berichtsheft dokumentiert werden.

- **Handschriftliche (Teil-)Berichte werden nicht gewertet.**
- **Nicht dokumentierte (Teil-)Berichte werden mit 0 Punkten bewertet!**
- **Nicht persistierte Lösungen für Aufgaben wie Code oder andere Artefakte führen zu 0 Punkten für diese Aufgabe.**
- **Das Berichtsheft muss als PDF-Datei (.pdf) abgegeben werden. Jedes Experiment ist in einem eigenen Kapitel anzulegen.**

Bericht

Jedes Experiment soll im Berichtsheft auf einer neuen Seite beginnen, sodass eine klare Unterteilung zwischen den einzelnen Experimenten gegeben ist. Ein Experiment ist schriftlich in die Abschnitte **Vorbereitung**, **Durchführung** und **Auswertung** zu unterteilen (je nach Aufgabentyp kann dies ergänzt werden).

Die Gesamtlänge eines Berichts (**minus der Abbildungen**) sollte in etwa bei 1–2 Din A4 Seiten (*Schriftgröße 11–12*) liegen. Wir empfehlen die Anfertigung des Berichtsheftes mit Latex.

Vorbereitung

Pflichtlektüre

Pflichtlektüre für alle ist, sich einen Überblick über die in PIT verwendeten Mutationsoperatoren zu verschaffen und aus dem Paper „LLMorpheus: Mutation Testing using Large Language Models“ die Kapitel 1–3 und 4.5–4.6 (RQs 3 und 4) zu lesen.

Für Masterstudenten sind zusätzlich die Kapitel 1–3 aus dem Paper „Large Language Models are Few-shot Testers: Exploring LLM-based General Bug Reproduction“ Pflichtlektüre.

Aufgabe 1 – Mutationsanalyse

In dieser Aufgabe sollen mithilfe der KI Mutanten des gegebenen Codes generiert werden. Basierend auf den generierten Mutanten soll im Anschluss der Mutation-Score ermittelt werden. Der gegebene Code ist in Java geschrieben.

Als Szenario für diese Aufgabe nehmen wir an, dass eine Codebasis vorliegt, für die zunächst eine Testsuite per Hand geschrieben wurde, die Branch-Coverage adressiert. Wir möchten nun die Qualität der Testsuite mithilfe von Mutation Testing überprüfen, um eine erweiterte Korrektheit der Codebase dokumentieren zu können. Dies soll mithilfe des berechneten Mutation Scores begründet werden. Gehen Sie in den folgenden Schritten vor.

1. Generieren Sie KI-basiert (mindestens) 20 lauffähige (!) Mutanten des Ausgangscodes. Lassen Sie sich mit dem Mutanten auch eine Erklärung der Änderung (Mutation) ausgeben. Stellen Sie sicher, dass es sich um 20 unterschiedliche Änderungen des Programms handelt.
2. Lassen Sie die vorhandene Testsuite gegen jeden der Mutanten laufen. Überprüfen Sie (gerne auch KI-basiert) eventuell überlebende Mutanten auf Äquivalenz zum Ausgangsprogramm.
3. Geben Sie den erreichten Mutation Score an.

Codebasis

Die zugrundeliegende Javadatei können hier heruntergeladen werden:

`https://github.com/sekassel/gkistss24-files`

Folgende Dateien sind für die Aufgabe relevant:

- `Aufgabenblatt 3/stringutils/StringUtils.java`
- `Aufgabenblatt 3/stringutils/StringUtilsTest.java`

Programmverhalten
Das Programm stellt eine Hilfsklasse zur Verarbeitung von String-Objekten dar.

Bericht

Der durchlaufene Prozess der Codegenerierung soll im Berichtsheft festgehalten werden. Es gilt für diese Aufgabe freie Wahl zwischen ChatGPT und Gemini als Generierungstool. Die Wahl des Tools ist zu dokumentieren.
Dokumentieren Sie in Ihrer Evaluation explizit, inwieweit die im Szenario festgelegten Ziele erreicht werden konnten.

Aufgabe 2 – Mutationstesten

In dieser Aufgabe sollen mithilfe der KI Mutanten des gegebenen Codes generiert werden (wieder mind. 20 wie bei Aufgabe 1). Basierend auf den generierten Mutanten soll ermittelt werden, welche die gegebene Testsuite bereits eliminiert und welche nicht. Ziel ist hier, möglichst Mutanten zu generieren, die überleben. Diese überlebenden Mutanten sollen (falls nicht äquivalent zum Ausgangsprogramm) als Grundlage genutzt werden, um die Testsuite zu erweitern, also Tests zu ergänzen, die diese Mutanten töten. Der gegebene Code ist in Java geschrieben.

Als Szenario für diese Aufgabe nehmen wir an, dass eine Codebasis vorliegt, die nur wenig dokumentiert wurde und deren Testsuite wir erweitern müssen. Da wir den Nutzen des vorliegenden Codes nicht ganz verstehen, beschließen wir, mittels eines LLMs zunächst Mutanten des Codes zu erzeugen. Die überlebenden Mutanten nutzen wir dann, um die Testsuite (ebenfalls mit dem LLM) zu erweitern und unsere Aufgabe damit zu erfüllen.

Codebasis

Die zugrundeliegende Javadatei können hier heruntergeladen werden:

`https://github.com/sekassel/gkistss24-files`

Folgende Dateien sind für die Aufgabe relevant:

- `Aufgabenblatt 3/sekasselmaps/MapService.java`
- `Aufgabenblatt 3/sekasselmaps/MapServiceTest.java`
- `Aufgabenblatt 3/sekasselmaps/model/City.java`
- `Aufgabenblatt 3/sekasselmaps/model/Location.java`
- `Aufgabenblatt 3/sekasselmaps/model/Order.java`
- `Aufgabenblatt 3/sekasselmaps/model/Street.java`

Programmverhalten
Das Programm sucht einen möglichen Pfad von einer Stadt zu einer anderen.

Bericht

Der durchlaufene Prozess der Codegenerierung soll im Berichtsheft festgehalten werden. Es gilt für diese Aufgabe freie Wahl zwischen ChatGPT und Gemini als Generierungstool. Die Wahl des Tools ist zu dokumentieren.
Dokumentieren Sie in Ihrer Evaluation explizit, inwieweit die im Szenario festgelegten Ziele erreicht werden konnten (eventuelle weitere Qualitätseigenschaften der Tests können informell diskutiert werden).

Aufgabe 3 – Käfersucher (Für Masterstudenten)

In dieser Aufgabe soll mithilfe der KI ein Bug-Issue eines Repositorys gelöst werden. Der gegebene Code ist in Java geschrieben.

Als Szenario für diese Aufgabe nehmen wir an, dass wir dem besagten Issue zugewiesen wurden und nach einiger Zeit des Herumprobierens immer noch nicht wissen, warum der festgestellte Bug auftritt. In unserer Verzweiflung möchten wir uns an diese neu-moderne KI wenden, um uns das Leben einfacher zu machen. Das Ziel ist es, das Problem mit möglichst wenig Eigenaufwand zu finden, erklären zu können, einen Test zu haben, der das Issue aufgedeckt hätte (um bei zukünftigen Codeänderungen auf der sicheren Seite zu sein), das Problem zu reparieren und damit das Issue schließen zu können.

Codebasis

Die zugrundeliegende Javadatei können hier heruntergeladen werden:

`https://github.com/sekassel/gkistss24-files`

Folgende Dateien sind für die Aufgabe relevant:

- `Aufgabenblatt 3/bugcatcher/issue.md`
- `Aufgabenblatt 3/bugcatcher/NumberUtils.java`
- `Aufgabenblatt 3/bugcatcher/NumberUtilsTest.java`
- `Aufgabenblatt 3/bugcatcher/solution.patch`

Bermerkung
Die Datei solution.patch enthält das Git-Diff der existierenden Lösung. Diese ist lediglich als Referenz gedacht, um die Qualität der generierten Lösung besser bewerten zu können. Des Weiteren müssen `junit:4.13.1` und `apache.commons.commons-lang3:3.0` in das Projekt eingebunden werden, um die Dateien compilieren zu können. Hier ist ein Beispiel für Gradle dargestellt

```
implementation group: 'junit', name: 'junit', version: '4.13.1'
implementation group: 'org.apache.commons', name: 'commons-lang3', version: '3.0'
```

Bericht

Der durchlaufene Prozess der Codegenerierung soll im Berichtsheft festgehalten werden. Es gilt für diese Aufgabe freie Wahl zwischen ChatGPT und Gemini als Generierungstool. Die Wahl des Tools ist zu dokumentieren.
Dokumentieren Sie in Ihrer Evaluation explizit, inwieweit die im Szenario festgelegten Ziele erreicht werden konnten (eventuelle weitere Qualitätseigenschaften der Tests können informell diskutiert werden).

Anhang

Es folgt eine Auflistung hilfreicher Links und Quellen zu den Themen dieser Veranstaltung. Die Links sind als Hilfestellung zur zur Bearbeitung der Aufgaben angedacht. Alle Materialien sind ebenfalls auf dem Discord Server einsehbar.

Quellen

- Übersicht über PITs Mutationsoperatoren: https://pitest.org/quickstart/mutators/
- Das Paper „LLMorpheus: Mutation Testing using Large Language Models“: https://github.com/sekassel/gkistss24-files
- Das Paper „Large Language Models are Few-shot Testers: Exploring LLM-based General Bug Reproduction“: https://github.com/sekassel/gkistss24-files