

This Is Your Thesis Title

Master's Thesis
von
Max Mustermann

Studiengang: Applied Computer Science
Fakultät: Computer Science and Engineering
Esslingen University

This Is Your Thesis Title

Master's Thesis
von
Max Mustermann

Studiengang: Applied Computer Science
Fakultät: Computer Science and Engineering
Esslingen University

Zeitraum: 01.03.2024 - 31.08.2024
Erstprüfer: Prof. Dr. John Doe
Zweitprüfer: Prof. Dr. Jane Doe

Firma: Musterfirma GmbH
Betreuer: Albert Einstein

Vertraulichkeitsklausel

Auf Wunsch der Firma Musterfirma GmbH unterliegt der Inhalt dieser Arbeit - einschließlich der dazugehörigen Daten und Zeichnungen - bis einschließlich „Monat/Jahr“ der Geheimhaltung. Es dürfen keine Kopien oder Abschriften erstellt werden, weder digital noch manuell. Ausnahmen von dieser Regelung bedürfen der schriftlichen Genehmigung der Firma Musterfirma GmbH.

Musterstadt, September 18, 2024 _____
Unterschrift des Autors

Das Verbot der Veröffentlichung dieser Arbeit wird unterstützt durch:

Prof. Dr. John Doe, September 18, 2024 _____
Unterschrift

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Teile der Arbeit, die aus anderen Quellen entnommene Sätze oder Stellen enthalten, sind deutlich mit der Herkunft der Informationen gekennzeichnet. Dies gilt auch für Diagramme, Skizzen, bildliche Darstellungen sowie für Quellen aus dem Internet. Ich erkläre ferner, dass ich diese Arbeit in keinem anderen Prüfungsverfahren als Prüfungsarbeit eingereicht habe und auch in Zukunft nicht einreichen werde.

Musterstadt, September 18, 2024

Unterschrift des Autors

Danksagung

I would like to express my deepest appreciation to my professor, Prof. Dr. John Doe, for his invaluable patience, insightful feedback, and unwavering support throughout this thesis. His guidance has been instrumental in shaping this work. I am also profoundly grateful to my supervisor, Albert Einstein, from Musterfirma GmbH. His generous provision of knowledge and expertise has been crucial to the success of this research. Here you can also thank your family and friends or your pet :)

Thank you all for your significant contributions to this journey.

Abstract

Nenne die Zielsetzung, die Problemstellung und die Forschungsfragen. Wenn deiner Abschlussarbeit bestimmte Hypothesen zugrunde liegen, erwähne diese auch. Zähle die wichtigsten Ergebnisse deiner Forschung auf und erkläre, zu welchem Fazit du gekommen bist. Nenne die relevantesten Eckpunkte aus der fachlichen Diskussion und lege deine Empfehlungen dar. Nimm keinen Bezug auf Literatur und verwende keine Zitate. Verzichte auf jegliche subjektiven Bewertungen oder Rechtfertigungen. Benutze keine Abkürzungen. Zeitform: Präsens, vergangene Ereignisse im Perfekt oder Präteritum

Inhaltsverzeichnis

Abbildungsverzeichnis	I
Tabellenverzeichnis	II
Quellcodeverzeichnis	III
Abkürzungsverzeichnis	IV
List of Todos	V
1 Generelles zu Abschlussarbeiten	1
1.1 Unterschiede zwischen Deutsch- und Englischer Abschlussarbeit	1
1.2 Aufbau einer Abschlussarbeit	1
2 Wie benutzt man diese Vorlage	3
2.1 Konfiguration und Generelles	3
2.2 Nummerierung von Bilder, Tabellen etc.	3
2.3 Overleaf oder andere IDEA?	3
2.4 Abkürzungen verwenden	4
2.5 Quellenangaben verwenden	4
2.6 Bilder einbinden	4
2.7 Tabellen einbinden	6
2.8 Quellcode einbinden	6
2.9 Pseudocode einbinden	7
2.10 Referenz zu Kapiteln, Bildern, Tabellen etc.	7
2.11 ToDos	7
Literaturverzeichnis	9
A Anhang	10
A.1 Script to Search for GitHub Repositories	10
A.2 Script to Process Output Produced by RestTestGen	12

Abbildungsverzeichnis

2.1	Some Caption	5
-----	------------------------	---

Tabellenverzeichnis

1.1	Unterschiede zwischen Deutsch und Englischer Abschlussarbeit	1
2.1	Some Caption	6

Quellcodeverzeichnis

2.1 Some Caption 6

A.1 Python script so search for GitHub repositories 10





A.2 Python script to search inside RestTestGen output folder for found errors . . 12

Abkürzungsverzeichnis

API Application Programming Interface

JSON JavaScript Object Notation

Todo list

 ToDo: XY muss noch eingebaut werden	8
 Improvement: Du könntest noch etwas genauer über AB schreiben	8
 Waiting for approval	8
 Approved	8

1 Generelles zu Abschlussarbeiten

1.1 Unterschiede zwischen Deutsch- und Englischer Abschlussarbeit

Generell unterscheidet sich das Schreiben auf Englisch nicht groß vom Schreiben auf Deutsch. Allerdings gibt es ein paar Feinheiten, deren du dir bewusst sein solltest, wenn du mit dem Schreiben einer Arbeit auf Englisch beginnst.

Deutsch	Englisch
Nominalstil	Reich an Verben
Sprache oft im Passiv	Sprache im Aktiv
Oft langer, komplexer Satzbau	Kurze, einfache Sätze

Table 1.1: Unterschiede zwischen Deutsch und Englischer Abschlussarbeit

Du solltest dir beim Schreiben deiner Arbeit darüber im Klaren sein, ob du in britischem oder auf amerikanischem Englisch schreibst. Die Unterschiede sind groß und Einheitlichkeit ist sehr wichtig für das Gesamtbild deiner Abschlussarbeit. Dass Nomen im Englischen nicht großgeschrieben werden, solltest du wissen. Aber es gibt noch mehr Unterschiede im akademischen Englisch, so werden zum Beispiel, je nachdem welches Format du nutzt, in Überschriften oft fast alle Wörter ‚capitalized‘.

1.2 Aufbau einer Abschlussarbeit

Generell solltest du dir zuerst die Ziele deiner Arbeit überlegen. Daraus entsteht dann die Zielsetzung. Außerdem solltest du eine Literaturrecherche machen, um deine Arbeit in den Kontext anderer Arbeiten zu setzen und die Problemstellung klar herausarbeiten zu können. Dein Aufbau deiner Thesis kann verschieden sein, jedoch habe ich folgenden Aufbau benutzt:

- Introduction / Einleitung
 - Problem Statement / Problemstellung
 - Objectives / Ziele dieser Arbeit
 - Structure of this Thesis / Aufbau dieser Arbeit
 - Related Work / Verwandte Arbeiten
- Fundamentals / Grundlagen
 - Hier erklärst du grundlegende Technologien oder Begriffe, die für das Verständnis deiner Arbeit relevant sind
- ... Der weitere Aufbau hängt stark davon ab, ob du etwas praktisches machst etc. Oft gliedert man in Design und Implementierung.

- Summary and Outlook / Zusammenfassung und Ausblick
 - Results / Ergebnisse
 - Discussion / Diskussion
 - Conclusion / Fazit
 - Future Work / Ausblick

2 Wie benutzt man diese Vorlage

2.1 Konfiguration und Generelles

Diese Vorlage ist so aufgebaut, dass du nichts mehr konfigurieren oder einstellen musst. Stattdessen kannst du dich auf das Schreiben deiner Arbeit konzentrieren. Im Hauptverzeichnis liegt die Datei `01_config.tex`. In dieser Datei werden die Grunddaten deiner Arbeit festgelegt. Dazu zählt auch die Sprache. Hier kannst du entscheiden, ob du deine Arbeit auf Deutsch oder Englisch verfassen möchtest. Je nachdem welche Sprache du verwendest, werden alle Inhalte automatisch in der entsprechenden Sprache dargestellt (außer natürlich dein geschriebener Text). Außerdem legst du in dieser Datei dein Name, Titel, Prüfer und alles weitere fest. Dies dient dazu, damit diese an entsprechender Stelle automatisch geladen werden können. Möchtest du also zum Beispiel in deiner Danksagung den Erstprüfer erwähnen, so musst du nur den Befehl `\reviewer` benutzen.

Außerdem empfehle ich, deinen Text sinnvoll in Dateien zu gliedern. Du kannst zum Beispiel jedes Kapitel in eine Datei packen. Ein Beispiel ist hier schon gegeben. Über die Datei `main.tex` werden dann deine Dateien eingebunden (siehe Zeile 154). Um die Compile-Zeit zu steigern, kannst du auch nur einzelne Dateien laden. Dies passiert in Zeile 12 in `main.tex`. Hier gibst du die Dateien an, an welchen du aktuell arbeitest. Der Vorteil hierbei ist, dass Referenzen auf andere Kapitel oder Bilder aus anderen Kapitel dennoch dargestellt werden können, und keine Compile-Fehler auftreten.

Standardmäßig enthält die generierte PDF eine Coverseite. Diese kannst du beim Drucken gut als erste Seite oder als Cover benutzen. Brauchst du diese Seite nicht, kannst du Zeile 26 in `main.tex` auskommentieren oder löschen.

2.2 Nummerierung von Bilder, Tabellen etc.

Bilder, Tabellen etc. werden automatisch nummeriert. Was jedoch kurz für Verwirrung sorgen kann, ist die Art und Weise der Nummerierung. Alle Elemente besitzen die erste Zahl entsprechend des Kapitels, in dem sie sich befinden. Sprich, Abbildung 3.5 befindet sich in Kapitel 3. Die zweite Zahl wird beginnend bei 1 und fortlaufend generiert. Diese Art der Nummerierung wird häufig bei Abschlussarbeiten verwendet.

2.3 Overleaf oder andere IDEA?

Generell empfehle ich dir, jeden Satz in einer neuen Zeile im Editor zu beginnen. Dies erleichtert die Lesbarkeit deines Codes und ermöglicht, falls du mit der Overleaf Review Funktion arbeitest oder mit Git, bessere Kommentare durch deinen Betreuer. Aufgrund der doch langen Compile-Zeit von Overleaf bei größeren Arbeiten empfehle ich die Verwendung von TexLive und Visual Studio Code für Windows (Hier eine Anleitung). Über Git kannst du deine Arbeit sichern und dein Betreuer kann z.B. über Issues Anmerkungen machen. Wenn

du mit Visual Studio Code arbeitest und deine Sätze sehr lang sind, kannst du horizontales Scrollen vermeiden über View → Word Wrap

2.4 Abkürzungen verwenden

In der Datei `02_abbreviations.tex` musst du deine Abkürzungen festlegen. Einige Beispiele sind bereits in der Datei enthalten. Möchtest du eine Abkürzung verwenden, benutze einfach den Befehl `\ac{...}`. Um die Plural-Form zu verwenden, benutze den Befehl `\acp{...}`. LaTeX erkennt automatisch, wenn du die Abkürzung das erste Mal verwendest und schreibt diese aus. Außerdem werden im Abkürzungsverzeichnis nur die Abkürzungen dargestellt, die du auch verwendest. Es ist also nicht schlimm, eine zu lange `02_abbreviations.tex`-Datei zu haben. Das Abkürzungsverzeichnis ist außerdem bereits automatisch alphabetisch sortiert. Ansonsten passiert alles automatisch - außer die beiden Befehle benutzen brauchst du nichts weiter tun. So sieht dann eine verwendete Abkürzung aus: Application Programming Interface (API)

2.5 Quellenangaben verwenden

In der Datei `references.bib` werden deine Quelle festgelegt. Auch hier sind einige Beispiele bereits in der Datei vorhanden. Um eine Quelle in deinem Text zu benutzen, verwende den Befehl `\cite{...}`. Wie bereits bei den Abkürzungen werden alle verwendeten Quellen automatisch in das Literaturverzeichnis geladen. Eine verwendete Quellenangabe sieht dann am Ende so aus im Text [2]. Wenn du möchtest, kannst du den Zitier-Style ändern. Dies geschieht über Zeile 52 und 53 in der Datei `settings.sty`. Jedoch empfehle ich für Abschlussarbeiten an der Fakultät IT die bereits hinterlegte numerische Zitierweise.

Zum organisieren deiner Quellen kann ich dir Zotero empfehlen. Hier kannst du deine ganzen Quellen verwalten, sortieren, mit Tags versehen wie zum Beispiel "noch nicht gelesen", und vieles mehr. Der Vorteil ist, dass du in Zotero die `.bib`-Datei generieren lassen kannst und diese dann ganz einfach in deiner Thesis verwenden kannst. Außerdem bietet Chrome eine Zotero-Erweiterung an. Mit dieser können Paper, Artikel, Websites etc. super schnell und unkompliziert als Quelle gespeichert werden, meist automatisch mit den richtigen Angaben wie Autoren, Datum, etc. Außerdem werden in Zotero die PDFs gespeichert, heißt du musst beim nachlesen von Sachen in deinen Quellen nicht immer erst die PDF wieder im Netz suchen.

2.6 Bilder einbinden

Für Bilder ist bereits der Ordner `/images` hinterlegt. Speichere also alle Bilder dort. Wenn möglich, vor allem bei Grafiken, sollte das PDF Format bevorzugt werden. Dies ermöglicht eine scharfe Darstellung. Um eigene Grafiken zu erstellen, empfehle ich dir app.diagrams.net/. Hier kannst du einfach und unkompliziert alles als PDF exportieren. Falls du eine Grafik aus

einer Quelle benutzt, empfehle ich dir, diese "nachzubauen" und anschließend einen Vermerk anzugeben mit Referent zum Original.

Das [b] beschreibt, wie dein Bild floaten soll. Generell wird empfohlen, auch bei den meisten Papern, dass Bilder entweder auf Seiten oben oder unten floaten sollen. b steht dabei für "bottom", also das Seitenende. t wird benutzt, um das Bild oben floaten zu lassen. h kannst du benutzen, um das Bild an genau dieser Stelle zu generieren. Dies kann auch verwendet werden, falls es dein Erstprüfer erlaubt. Generell gelten diese Floating-Optionen auch für Tabellen, Quellcode oder Pseudocode.

Zum Einbinden von Bildern, benutze den folgenden Code:

```
\begin{figure}[b]
  \centering
  \includegraphics[width=0.85\textwidth]{example.pdf}
  \caption[Some Caption]{Some Caption (based on \cite{test})}
  \label{img:bla}
\end{figure}
```

Über `width=0.85\textwidth` gibst du an, wie breit dein Bild sein soll. Ich empfehle hier, einen Wert von 0.95 nicht zu überschreiten.

Über `\caption` gibst du die Bildunterschrift an. Falls du, wie zuvor erwähnt, auf eine Quelle verweisen möchtest, benutze die geschweiften Klammern. Dies sorgt dafür, dass die Caption in den eckigen Klammern im Abbildungsverzeichnis dargestellt wird und die Caption in den geschweiften Klammern unter dem Bild selber.

Um auf das Bild verweisen zu können (wird in Section 2.10 erklärt), musst du den Befehl `\label` benutzen und das Label definieren. Dies wird anschließend benutzt, um das Bild zu referenzieren. Hier kannst du dir ein Schema überlegen, zum Beispiel für Bilder "img:...".

Hier ist ein Beispiel, wie ein Bild mit der Floating-Option h aussieht:

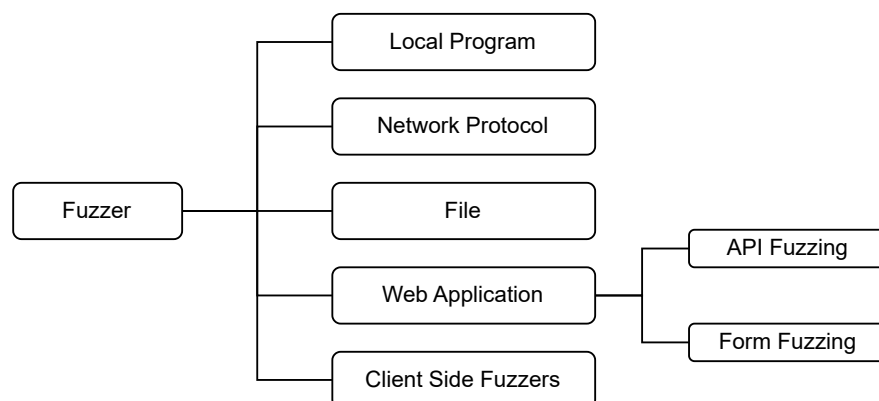


Figure 2.1: Some Caption (based on [2])

2.7 Tabellen einbinden

Das Einbinden von Tabellen funktioniert prinzipiell genau gleich wie mit Bilder. Es gibt die Floatin-Option, eine Caption sowie ein Label. Auch hier kannst du optional für die Caption eckige Klammern benutzen. Hier ist ein Beispiel für eine Tabelle. Schau dir einfach den Code an, dieser ist selbsterklärend.

Fuzzer	Test 2
APIFuzzer	failed
OFFAT	failed
openapi-fuzzer	failed
RESTler	failed
RestTestGen	passed

Table 2.1: Some Caption

2.8 Quellcode einbinden

Für das Einbinden von Quellcode wird das Verzeichnis `\code` benutzt. Hier legst du einfach deine Quellcodedaten ab, welche du darstellen möchtest. Quellcode bindest du über den folgenden Befehl ein:

```
\lstinputlisting[float=b, caption=Some Caption, label=lst:bla]{redos.ts}
```

Auch hier können die bereits beschriebenen Float-Optionen verwendet werden. Aus Gründen der Übersichtlichkeit verzichtet meine Vorlage auf farbliches Hervorheben im Quellcode. Generell solltest du nur die relevanten Zeilen im Quellcode darstellen, welche absolut relevant sind. Die vollständige Datei kannst du dann zum Beispiel im Anhang einfügen. Benutze hierfür die beispielhafte Datei `/content/99_appendix.tex`

```
1 @Get('/redos')
2 executeReDoS(@Query('email') email: string): string {
3     console.log("Redos path called")
4     email = "this_is_some_very_very_long_malformed_string"
5     const regex = /^([0-9a-zA-Z]([-\.\w]*[0-9a-zA-Z])*)@{1}([0-9a-zA-Z](-\w)*[0-9a-zA-Z]
6         [\.\.]+[a-zA-Z]{2,9})$/;
7     if (regex.test(email)) {
8         return 'Valid email';
9     } else {
10         return 'Invalid email';
11     }
```

Quellcode 2.1: Some Caption

2.9 Pseudocode einbinden

Es kann vorkommen, dass du Pseudocode oder Algorithmen zum Beispiel im Rahmen eines Designs einbinden möchtest. Hier siehst du ein Beispiel, wie ein Algorithmus eingebunden ist. Schau dir einfach den Code dazu an, dieser ist selbsterklärend. Auch hier gibt es wieder die bekannten Elemente wie die Floating-Option, die Caption und das Label.

Algorithm 1 Handling basic authentication via configuration file

```
1: function GETAUTHDATA(config)
2:   ...
3:   return userID, password
4: end function
5: function GENERATEBASICAUTHHEADER(userID, password)
6:   credentials  $\leftarrow$  userID + ':' + password
7:   encodedCredentials  $\leftarrow$  BASE64ENCODE(credentials)
8:   authHeader  $\leftarrow$  'Authorization: Basic ' + encodedCredentials
9:   return authHeader
10: end function
11: configFilePath  $\leftarrow$  'path/to/config/file'
12: userID, password  $\leftarrow$  GETAUTHDATA(configFilePath)
13: authHeader  $\leftarrow$  GENERATEBASICAUTHHEADER(userID, password)
14: ...
```

2.10 Referenz zu Kapiteln, Bildern, Tabellen etc.

Du solltest darauf achten, dass jedes Bild, jede Tabelle, jeder Quellcode etc. auch im Text erwähnt wird. Dabei kannst du mit Hilfe des `\autoref{...}` Befehls schnell und unkompliziert auf das Element verweisen. Innerhalb der geschweiften Klammern schreibst du das Label, das du definiert hast. Alle referenzierten Inhalte sind außerdem anklickbar. Hier ein Beispiel:

Zu diesem Zweck fügen wir einen neuen Pfad `redos` hinzu, der mit der HTTP-Methode GET aufgerufen wird, wie in Quellcode 2.1 dargestellt.

2.11 ToDos

In diese Vorlage ist die Verwendung von ToDos integriert. Dies kann dir dabei helfen, deinen aktuellen Fortschritt besser zu verfolgen oder auch eine bessere Kommunikation mit deinem Betreuer ermöglichen. Die folgenden ToDos stehen dabei zur Verfügung:

```
\todo{XY muss noch eingebaut werden}
\improvement{Du könntest noch etwas genauer über AB schreiben}
\approve
\approved
```

ToDo: XY muss noch eingebaut werden

Improvement: Du könntest noch etwas genauer über AB schreiben

Waiting for approval

Approved

Falls du dieses Feature nicht brauchst, kommentiere Zeile 26 in `01_config.tex` ein und lösche Zeile 27. Wenn du die ToDos verwendest und am Ende deine finale PDF generieren möchtest, brauchst du nicht alle ToDos manuell entfernen. Hierzu kannst du auch einfach Zeile 26 in `01_config.tex` einkommentieren und Zeile 27 löschen.

Literaturverzeichnis

- [1] OpenAI. *ChatGPT*. 2024. URL: <https://chatgpt.com/> (visited on 07/30/2024).
- [2] Takanen, Ari et al. *Fuzzing for Software Security Testing and Quality Assurance*. Volume Second edition. Artech House Information Security and Privacy Series. Artech House, 2018. ISBN: 978-1-60807-850-9. URL: <http://www.redi-bw.de/db/ebsco.php/search.ebscohost.com/login.aspx%3fdirect%3dtrue%26db%3dnlebk%26AN%3d1825934%26site%3dehost-live>.

A Anhang

A.1 Script to Search for GitHub Repositories

Quellcode A.1 is used to search on GitHub for repositories matching the keywords provided by the user. In comparison to a manual search, this script offers several advantages. Firstly, the output includes all repositories found, and secondly, the output produces a link to the repository, which can then be used for further processing. We developed this script by making use of the capabilities of ChatGPT 3.5, a sophisticated large language model [1].

```
1 import requests
2 import argparse
3 import subprocess
4
5 def build_search_url(base_url, search_terms):
6     url = base_url + "+" + "+".join(search_terms)
7     return url
8
9 def search_github(search_url, search_in_code=False):
10     if search_in_code:
11         search_url += "&in:file"
12
13     # Initialize variables for pagination
14     all_results = []
15     page = 1
16     per_page = 100 # Max results per page (GitHub API limit)
17
18     while True:
19         # Append pagination parameters to the search URL
20         paginated_url = f"{search_url}&page={page}&per_page={per_page}"
21         response = requests.get(paginated_url)
22         response_json = response.json()
23
24         # Append current page results to the list of all results
25         all_results.extend(response_json["items"])
26
27         # Check if there are more pages to fetch
28         if len(response_json["items"]) < per_page:
29             break # No more items to fetch
30
31         # Increment page number for next request
32         page += 1
33
34     return all_results
35
36 def extract_repo_urls(search_results):
37     repo_urls = []
38     for item in search_results:
39         repo_urls.append(item["html_url"])
40     return repo_urls
41
42 def check_for_api_spec(url):
43     subprocess.run(["python", "git_search_file.py", url])
44
45 def main():
46     base_url = "https://api.github.com/search/repositories?q="
47
```

```
48     # Parse command-line arguments
49     parser = argparse.ArgumentParser(description="Search GitHub repositories.")
50     parser.add_argument("search_terms", nargs="+", help="Search terms")
51     parser.add_argument("--code", action="store_true", help="Search in code")
52     args = parser.parse_args()
53
54     search_url = build_search_url(base_url, args.search_terms)
55     search_results = search_github(search_url, args.code)
56     repo_urls = extract_repo_urls(search_results)
57
58     print("Found Repositories:")
59     for url in repo_urls:
60         print(url)
61
62 if __name__ == "__main__":
63     main()
```

Quellcode A.1: Python script so search for GitHub repositories

A.2 Script to Process Output Produced by RestTestGen

Since RestTestGen generates a JavaScript Object Notation (JSON) file for every request sent, we need to search through these files to identify any errors found by RestTestGen. To automate this process, we developed a Python script shown in Quellcode A.2. The script takes two parameters: the directory containing the JSON files and the search term to look for within these files. To search for errors in the files, we use the search term "FAIL". We developed this script by making use of the capabilities of ChatGPT 3.5, a sophisticated large language model [1].

```
1 import os
2 import sys
3
4 def search_files(directory, search_string):
5     filenames = []
6     line_numbers = []
7     lines = []
8
9     for root, dirs, files in os.walk(directory):
10         for file in files:
11             filename = os.path.join(root, file)
12             with open(filename, 'r') as f:
13                 for line_number, line in enumerate(f, start=1):
14                     if search_string in line:
15                         filenames.append(filename)
16                         line_numbers.append(line_number)
17                         lines.append(line.strip())
18
19     return filenames, line_numbers, lines
20
21 def main():
22     # Check if the user provided a directory and search string arguments
23     if len(sys.argv) != 3:
24         print("Usage: {} <directory> <search_string>".format(sys.argv[0]))
25         sys.exit(1)
26
27     search_dir = sys.argv[1]
28     search_string = sys.argv[2]
29
30     # Check if the directory exists
31     if not os.path.isdir(search_dir):
32         print("Directory '{}' does not exist.".format(search_dir))
33         sys.exit(1)
34
35     print("Searching for '{}' in files within directory: {}".format(search_string,
36                             search_dir))
37     filenames, line_numbers, lines = search_files(search_dir, search_string)
38     print("")
39
40     if not filenames:
41         print("No occurrences of '{}' found in files within directory: {}".format(
42             search_string, search_dir))
43         sys.exit(0)
44
45     max_filename_length = max(len(os.path.basename(filename)) for filename in
46                               filenames)
```



```
44     print("Filename{}Line Numbers    Matched Line".format(' ' * (max_filename_length -
45         7)))
46     print("-" * (max_filename_length + 36))
47     for filename, line_number, line in zip(filenamees, line_numbers, lines):
48         print("{:<{}} {:>12} {}".format(os.path.basename(filename),
49             max_filename_length, line_number, line))
50 if __name__ == "__main__":
51     main()
```

Quellcode A.2: Python script to search inside RestTestGen output folder for found errors