

Testing und Debugging: Übung 06

Abgabetermin 19.11.2013 13:59

Name

Matrikelnummer

1 Zufallsgesteuerte Testfallgeneratoren.

In dieser Übung wollen wir uns tiefer mit den zufallsgesteuerten generativen Testfallgeneratoren beschäftigen. Als praktisches Beispiel dient die Funktion `create_random_api_calls` aus der aktuellen Programmieraufgabe.

Diese Implementation soll die Vorteile zeigen, die eine geschickte Wahl von default Werten und die Möglichkeit komplexer Argumente für den Testfallgenerator bieten. Weiterhin soll es als Denkanstoß dienen, wenn Sie selbst einen solchen Generator entwickeln müssen.

Hilfreicher Link: <http://docs.python.org/3.2/library/random.html>.

1.1 Verstehen des internen Aufbaus und der Benutzung des Testfallgenerators (40 Punkte)

Gegeben sei folgender Auszug der gespeicherten Ausgabe von `create_random_api_calls()` (vollständige Datei liegt bei) die eine Fehler bei einer der fehlerhaften Implementationen der Queue Klasse auslöst.

API_test_stream

```
# api = 'empty', 'full', 'enqueue', 'dequeue'
# queues = 'Queue6'
# queue_size = None
# max_queue_size = 2000
# stream_length = None
# max_stream_length = 16000
# seed = 13681756363766
Queue6 1983
# chosen stream_length = 5267
empty
full
empty
dequeue
...
```

Aufgaben

1. Warum wird für den Testfallgenerator kein echter Zufallsgenerator, sondern ein Pseudozufallsgenerator verwendet und welchen Vorteil können wir daraus ziehen? (5+5 Punkt)

.

.

.

.

2. Gegeben sei obige Ausgabe des Testfallgenerators. Kann man `create_random_api_calls()` so aufrufen, dass man **reproduzierbar** die exakt selbe Sequenz von API calls erzeugt bekommt, wie sie in der Datei `API_test_stream` gespeichert ist? Begründen Sie ihre Antwort. Geben Sie, wenn möglich, den Aufruf an, der die geforderte Sequenz erzeugt, oder begründen Sie weshalb dies nicht geht. (30 Punkte)

•
•
•
•
•
•
•

1.2 Erst Denken, dann Ausprobieren (60 Punkte)

Ein erfolgreicher Tester haben muss sich von der gedanklichen Einstellung „Stocher ich mal wild drauf los und ruf dabei ein paar API-Funktionen auf; vielleicht stoße ich ja auf eine Bug.“ lösen.

Als Tester sollte man wie folgt vorgehen:

- Lesen und verstehen des Codes und der Spezifikation
- Identifizieren möglicher Schwachstellen im Code.
- Überlegen, wie die Software reagiert, wenn die Schwachstelle getroffen wurde; wie weicht das Ergebnis von der Spezifikation ab.
- Einen Test ausdenken und implementieren, der (hoffentlich) diese Schwachstelle triggert.
- Den Test laufenlassen und das Ergebnis beobachten und mit seinen Erwartungen vergleichen.
- Evtl. Test anpassen und erneut laufen lassen.

Aufgaben

1. Wieso ist es sehr schwer, für **Queue3** mit `create_random_api_calls()` mit den default Einstellungen einen Test zu erzeugen, der den Fehler in **Queue3** zu triggern? (20 Punkte)

-
-
-
-
-

2. *Geben Sie die kürzest mögliche Sequenz von API Aufrufen an, die den Fehler in Queue3 triggert. (Lange Sequenzen desselben API Aufrufs dürfen durch Anzahl \times API Call abgekürzt werden.)* (20 Punkte)

•

3. Geben Sie an, wie man `create_random_api_calls()` aufrufen muss um eine möglichst kurze Sequenz von API Aufrufen zu erhalten, die den Fehler **reproduzierbar** (d.h. bei jedem Aufruf von `create_random_api_calls()` mit ihren Parametern) auslöst. (20 Punkte)

•
•
•
•

2 Programmieraufgabe: Random Testen der 8 fehlerhaften Queue Implementationen (vgl. Übung01). (100 Punkte)

Implementieren Sie einen Random API Tester für die 8 fehlerhaften Queues aus Übung01. Gehen Sie dabei gemäß der Graphik auf Folie 3 von Vorlesung05 vor. Verwenden Sie als Orakel den Vergleich mit einer Referenzimplementierung der Queue (vgl. Musterlösung der Programmieraufgabe von Übung01)

Dateien für die Programmieraufgabe



Grading/initpy
Grading/Grading.py
uebung06.py *

Hinweisdatei für Python: Das Verzeichnis ist ein Modul
 Quellcode der Testklasse, die in `run_test.py` verwendet wird
 In dieser Datei müssen Sie die Funktionen `load_stream(filename)` und `run_stream(stream)` gemäß Aufgabenstellung implementieren.
 In dieser Datei sind die 8 fehlerhaften Implementierungen der Queueklasse
 Der Aufruf (evtl. mehrfach) von `python3 uebung06.py` füllt diese Dateien.

Queue.py
test_Queue1-
test_Queue8
†

Die mit [*] markierten Dateien müssen **bearbeitet** und **abgegeben** werden.

Die mit [†] markierten Dateien sollten bei der Abgabe, von `create_random_api_calls()` erzeugte Sequenzen erhalten die den jeweiligen Fehler der angegebene Queue auslösen. Diese Dateien müssen mit **abgegeben** werden.

2.1 Aufgabenstellung

Implementieren Sie die Funktion `load_stream(filename)` so, dass nur ein gültiger Datenstrom zurückgegeben wird (vgl. Kommentar im Skelett der Funktion in `uebung06.py`) eventuelle Leerzeilen in der Einzulesenden Datei werden beim Umwandeln in einen Datenstrom entfernt.

In der Funktion `run_stream(stream)` sollen die Funktionen aus der Variablen **API** automatisch getestet werden. Der Test eines API Aufrufs soll dabei wie folgt aussehen:

1. Aufrufen den eingelesenen API Aufrufs und speichern des Rückgabewertes.
2. Aufrufen des dazu passenden Aufrufs in der Referenzimplementation und speichern des Rückgabewertes.
3. Vergleichen der Rückgabewerte via Assertion.
4. Aufrufen von `check_rep()` für die zu testende Queue.

Via eines `try` Blocks werden die Assertions vom Vergleich und von `check_rep()` abgefangen.

Die Funktion führt nacheinander alle API Aufrufe wie eben beschrieben aus und bricht bei einem Fehler (API Rückgabe != Referenzrückgabe) sofort ab.

Die drei möglichen Rückgaben entnehmen Sie wieder dem Kommentar in der Funktion.

Die Funktionen `main_run()` und `load_all_run()` geben jeweils einen Punkt aus, wenn ein API Strom ohne Fehler durchläuft, ein **F** falls ein Fehler auftritt oder ein **X** falls ein ungültiger API Aufruf passiert ist.

Das **X** darf bei korrekter Implementation von `load_stream(stream)` und `run_stream(stream)` niemals ausgelöst werden!

2.2 Vorbereiten der Abgabe

Führen Sie das Programm mittels `python3 uebung06.py` aus, bis bei der Ausgabe von `load_all_run()` jede Queue ein **F** als Ausgabe erhalten hat.

Falls nach mehrfachem Aufruf die Queue3 immer noch kein **F** bekommt, dann Implementieren Sie die Funktion `main_run_Queue3_t` wie im Kommentar am Ende von `uebung06.py` beschrieben.

Ziel ist es, dass Sie in den Dateien `test_Queue1` - `test_Queue8` jeweils einen API Strom gespeichert haben, der den Fehler der jeweiligen Queue auslöst.

2.3 Bewertung

Ihre 8 Ergebnisse werden gegen meine Musterlösung getestet. Diese Läufe müssen den jeweiligen Fehler auslösen und dürfen keine API Fehler enthalten.

Meine 8 Musterergebnisse werden in Ihre Funktionen geladen und auch diese müssen wie beschrieben reagieren.

Ihre Implementation wird gegen 4 fehlerhafte API Ströme getestet, die Entweder in der Lade Funktion abgefangen werden müssen oder aber in der `run_stream(stream)` Funktion mit dem Rückgabewert `None` quittiert werden müssen.

Je richtig behandelte Eingabedatei erhalten sie 5 Punkte, also insgesamt 100 Punkte.

2.4 Abgabe der Programmieraufgabe

1. Falls noch nicht vorhanden, dann erstellen Sie sich auch <https://github.com> einen Account. (Pro Gruppe ist nur ein Account und eine Abgabe nötig!)
2. Schicken Sie bei Ihrer ersten Abgabe den **Namen** des github Accounts (ohne Passwort) an sebastian.stigler@htw-aalen.de mit dem Betreff: *Testing und Debugging Github Account*
3. Gehen Sie in Ihrer virtuellen Maschine in das Aufgabenverzeichnis (wo sich Ihre bearbeitete Aufgabe und die Datei `submit.cfg` befinden).
4. Tippen Sie in der Konsole den Befehl `submit`. Dieser wird beim ersten Ausführen nach den Zugangsdaten Ihres github Accounts fragen. Anschließend werden die bearbeiteten Aufgaben verschlüsselt auf <https://gist.github.com> abgelegt.
5. Diese Datei wird nach dem Abgabetermin automatisch zur Korrektur heruntergeladen.

Viel Erfolg