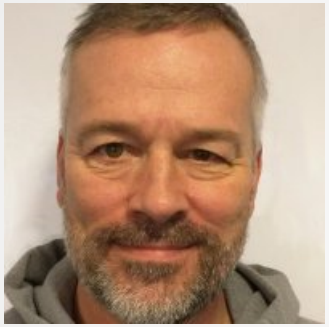
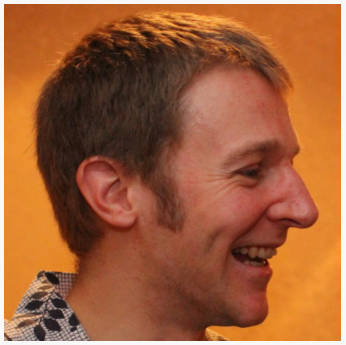
GraphWalker

ein Graphenbasiertes Testgenerierungstool
ältester Commit: 3.5.2014 (Olsson),
aktuellster Commit: 3.12.2017(Olsson) ca 23:00

Entwickler bei github:



Kristian Karl,
Test- und Entwicklungsleiter
Ansprechpartner
https://de.linkedin.com/in/krikar/de
2017/12/04



Michael Holland
https://github.com/maikeru
2017/12/04

Nils Olsson
Qualitätssicherung
https://www.linkedin.com/in/nils-olsson-52203858/
2017/12/04

Arbeitgeber: Spotify (Schweden)

anonymer Github-Nutzer:
nbdnnm – vermutlich auch Tester

Lizenz, Mitwirken, Support

- MIT-Lizens: kostenlos, GW darf ohne Einschränkung:
  - genutzt, kopiert, geändert, verwendet, veröffentlicht, weitergegeben werden
  - weiter lizensiert (sublizensiert) und verkauft werden
- Bedingung: Hinweis zur Lizenz muss in allen Kopien enthalten sein
- Keine Garantie, Entwickler können nicht verantwortlich gemacht werden
- Website (Dokumentation + JAR-Datei): http://graphwalker.github.io/
  Source-Code: https://github.com/GraphWalker
- Ansprechpartner: Kristian Karl (Email)
  Forum: https://groups.google.com/forum/#!forum/graphwalker

GraphWalker & Testen

- GraphWalker erzeugt Abfolge von Java-Klassen
  - Einsatz für Whitebox-Testen möglich
    - Eingaben mittels Funktionsaufrufen in Java simulieren
  - Einsatz auch für Blackbox-Testen
    - Testen von Websiten z.B. über externes Programm Selenium Web Driver
    - Testen von GUIs z.B. über externes Programm Sikuli
- Nur zur Generierung und Eingabe von Testdaten
  - Überprüfung z.B. mit jUnit-Assertions
- GraphWalker benötigt Graphen-Modelle für die Ausführung
  - Erzeugung nur mit yEd (1 Installation pro Person, freie Nutzung, keine weiteren Rechte, keine Garantie oder Haftung)

Installation

- portable Benutzung mittels kompilierter JAR möglich
- für Benutzung mit MAVEN muss Projekt kompiliert werden
  - runterladen per git
  - Maven wird nur in Version 3.1.1 unterstützt
    - Benutzerumgebungsvariable PATH zuweisen
      - auf Windows ab- und anmelden zum aktualisieren
  - Java8 SDK wird unterstützt, Java7 SDK bald nicht mehr
- für Eclipse ist Plugin vorhanden
- es muss auch yEd installiert sein zum Graphen erzeugen

1. Befehle auf Knoten

- Knoten mit Bezeichnung "Start" als Default-Startknoten
  - Maximal 1x
  - Startknoten kann auch bei der Ausführung definiert werden
- IrgendeinKnoten INIT:total=0;
  - beim 1. Besuch wird total=0 gesetzt (nur Steuerstruktur)
- IrgendeinKnoten SHARED:GRUPPE1
  - ermöglicht Sprünge zu anderen Knoten und Modellen mit der selben Gruppe
- IrgendeinKnoten BLOCKED
  - Knoten und alle ein- und ausgehenden Kanten werden ignoriert

2. Befehle auf Kanten

- Kanten nur in eine Richtung
- IrgendeineKante weight=0.2
  - Kante wird gewichtet, Wichtung wird global berechnet
  - Wert zwischen 0.0 und 1.0
- IrgendeineKante/total++;
  - Alle Funktionen nach / werden für Steuerfluss ausgeführt; Abschluss mit;
- IrgendeineKante[total <= 10 && vocals <5]
  - [logischer Ausdruck] ist Bedingung, dass Kante ausgeführt werden kann
- IrgendeineKante BLOCKED
  - Kante wird ignoriert

3. Generatoren

- random(IrgendeineStoppbedingung)
  - wählt zufällig Kanten aus
- weighted_random(IrgendeineStoppbedingung)
  - wählt Kanten nach Gewicht
- quick_random(IrgendeineStoppbedingung)
  - versucht doppelte Kanten zu vermeiden
  - wählt Kanten nach Dijkstra-Algorithmus
- a_star(Stoppbedingung mit Knoten/Kante)
  - sucht kürzesten Pfad zu Knoten/Kante

4. Stoppbedingungen

- edge_coverage(100)
  - stoppt nach 100%iger Kantenüberdeckung
  - analog Knotenüberdeckung möglich (vertex_c...)
- reached_vertex(zuErreichenderKnoten)
  - stoppt wenn Knoten erreicht (analog mit Kanten)
- time_duration(1)
  - stoppt nach 1s Ausführungszeit
  - nur ganzzahlige Sekunden möglich
- length(10)
  - nach 10 Knoten-Kanten Paaren wird gestoppt
- never
  - läuft bis Benutzer abbricht (in Konsole mit Strg+C)

5. Erfolgreicher Test?

- Stoppbedingungen können boolesch kombiniert werden
- Wenn Generator alle Stoppbedingungen erfüllt hat endet Test erfolgreich
- Wenn nicht alle Stoppbedingungen erfüllt werden konnten, weil keine weiteren Kanten möglich waren endet Test nicht erfolgreich
- Test kann auch bei Assertions nicht erfolgreich enden (wenn z.B. durch jUnit aktiv)
- Test auch ohne Programmcode möglich, um Graph zu prüfen
  - zu implementierende Schnittstellen können automatisch erzeugt werden mit "mvn graphwaler:generate-sources"

Test am Beispiel Running Example

- StdIn wurde modifiziert, um automatische Testeingaben durchzuführen
- Hauptklasse wurde erweitert um Programm auf Aufruf auszuführen
- Mavenprojekt wurde auf Grundlage von graphwalker-maven-archetype erstellt
  - Download von http://central.maven.org/maven2/
  - nicht in Standard-Maven-Installation vorhanden
  - Maven benötigt spezielle Ordnerstrukturen
  - Eclipse ohne Plugin schwer zu bedienen
    - Interface wird nicht im source-Package generiert
    - nach Speichern der Implementierung manchmal Fehler
      - dann Interface verschieben, damit Fehler verschwindet
      - Editierung in Notepad++ » keine Fehler

Testcodes

- Offline-Test mit JAR
  - java -jar .\graphwalker-cli-3.4.2.jar offline --model .\runningExampleRandom.graphml "random(reached_edge(e_restart))"
  - java -jar .\graphwalker-cli-3.4.2.jar offline --model .\runningExampleRandom.graphml "random(vertex_coverage(100))"
- Test per Powershell und mvn
  - mvn graphwalker:generate-sources
  - mvn graphwalker:test

Inhalt

- S1/2 Copyright
- S3 Was ist GraphWalker
- S4 Installation
- S5/6 Funktionen Knoten/Kanten
- S7/8 Generatoren Stoppfunktionen
- S9 erfolgreicher Test
- S10/11 Beispiel

Quellen:
http://graphwalker.github.io/
https://github.com/GraphWalker
https://github.com/GraphWalker/graphwalker-project/network
https://groups.google.com/forum/#!forum/graphwalker
Sowie an den Bildern angegeben
Zeitpunkt: 2017-12-04