Teszt-szelekció és priorizálás

Tóth Gábor

Bevezetés

- Inkább szervezési módszerként használatos, több elemi módszert magukban foglalhatnak
- Teszt priorizálás: **először** fontosabb teszteseteket futtatunk
- Teszt-szelekció: **csak** fontosabb tesztesteket futtatunk
- A két módszer kombinációja nagyon hatékony tesztkört eredményez

Fajtái

- **Lefedettség alapú**
- Kockázat alapú
- Priorizálás és szelekció szeleteléssel
- Követelmény alapú

Teszt priorizálás

Teszt priorizálás

- Minél korábban hozzuk elő a hibákat.
- Minél gyorsabban érjünk el egy előre megadott kód lefedettségi szintet.
- Minél gyorsabban alakuljon ki megbízhatóság-érzet a rendszer iránt.
- Minél gyorsabban találjunk meg magas kockázatú hibákat.
- Minél hamarabb találjuk meg a kódban bekövetkezett változások okozta esetleges hibákat.

Teszt priorizálás

- Nem marad ki egyetlen egy teszteset sem a tesztelésből
- Priorizálás jóságának mérése: minél koraibb hibafelfedezés
  - APFD (avg % faults detected) megtalált hibák súlyozott átlaga
  - Az összes teszteset permutáció közül az a legjobb, ahol a lehető leghamarabb a lehető legtöbb hibát megtaláljuk

Teszt priorizálás

Teszteset /hiba	1	2	3	4	5	6	7	8	9	10
A	x				x					
B	x				x	x	x			
C	x	x	x	x	x	x	x			
D					x					
E								x	x	x

APFD = $1 - \frac{TF_1 + TF_2 + \cdots + TF_m}{nm} + \frac{1}{2n}$ (TF_i: i. hiba első felderítésének pozíciója , n: feltárt hibák száma, m: tesztesetek száma)

A-B-C-D-E sorrend 20%-40%-70%-70%-100% APFD: 50

C-E-B-A-D sorrend 70%-100%-100%-100%-100% APFD: 84

Teszt priorizálás: algoritmusok

APFD maximalizálásához megkülönböztetünk:

- Priorizálás nélküli
- Véletlenszerű
- Optimális: exponenciálisan nehéz
- mohó priorizálás: jó alsókorlát

Teszt priorizálás: lefedettség alapúak

- Teljes utasítás lefedettségre optimalizált
  - Nem számít az átfedés, $O(mn)$, m tesztesetek, n utasítások
- Kiegészítő utasítás lefedettségre optimalizált
  - Átfedést nézünk
- Teljes elágazás lefedettségre optimalizált
- Kiegészítő elágazás lefedettségre optimalizált
  - Branch alapú tesztelések jellemzően jobban teljesítenek

Teszt-szelekció

Teszt-szelekció

- Nagyon költséges az összes eset futtatása. Véletlenszerűen kiválasztott részhalmazra nem megbízható.
- Különösen regressziós tesztelésnél
- Részhalmaz kiválasztása olyan módon, hogy a lefedettség minimálisan változzon
- Hasonló a teszt priorizáláshoz

Teszt-szelekció

- Szimulált hűtés (energiafüggvény: választott esetek súlyozott összege)
- Redukciós módszer (teszteset-halmaz és követelmény párok)
  - Egyre több teszteset
- Szeletelés módszer
  - Egy szelet tartalmazni fogja a módosított programrészt
- Adatfolyam módszer (~szeletelés)
  - Definíció-használat párok: ezeket fedjük le
- Tűzfal módszer (hatásanalízisen alapszik): megváltoztatás

JUnit5 lehetőségek priorizáláshoz

Priorizáló annotáció példák:

@TestMethodOrder(MethodOrderer.MethodName.class)
@TestMethodOrder(MethodOrderer.Random.class)
@TestMethodOrder(CustomOrder.class)
@Order(1)

Szelekció annotáció példák:

@MethodSource
@Disabled(”why disabled”)

```java
public class CustomOrder implements MethodOrderer {
    private final Map<String, Integer> priorities;

    public CustomOrder() {
        this.priorities = readPrioritiesFromCSV("src\\example.csv");
    }

    @Override
    public void orderMethods(MethodOrdererContext context) {
        context.getMethodDescriptors().sort(Comparator.comparingInt(this::getPriority));

    }
```