FAULT LOCALIZATION

Készítette:

Vajda Tibor

Hibalokalizáció

- **Programhibák helyének** behatárolásra/azonosításra a kódban
- Egyfajta debugging
- Egyik **legdrágább** és **legidőigényesebb** tesztelési tevékenység

Manuális

- hatékonysága több tényezőtől függ:
  - fejlesztök **tapasztalatától**,
  - **ítélőképesség**,
  - az adott **program**/rendszer **ismeretétől**,
  - **Kiegészítő információ** (pl. lefedettség) rendelkezésre állásától

Automatikus

- backward slicing is használható például FL technikaként, amennyiben ki tudja **szűrni** azokat a **sorokat**, amelyeknek biztosan **nincs hatása** a **hibás értékre**, és amiket így nem kell megvizsgálni

Folyamat két fő részre

I. A gyanúsnak vélt, **programhibákat** potenciálisan **tartalmazó kód azonosítása**

- ezt a részt lehet és szokás automatizálni
- kifinomultabb módszereknél nagyobb hangsúly ezen a részen
- célja, hogy a gyanús kódot a hibaelőfordulási valószínűség alapján priorizálja

II. A **programozóknak** kell **megvizsgálniuk** az **azonosított kódot**, hogy eldöntsék, valóban tartalmaz-e hibákat

Spectrum-Based Fault Localization (SBFL)

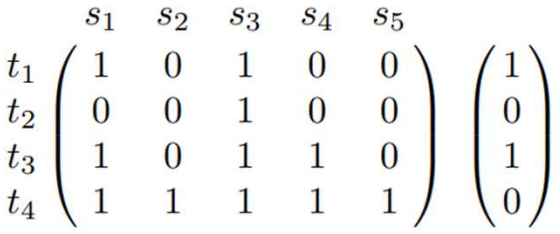
- Egyik legelterjedtebb módszer
- **Programspektrum** itt a programrészekhez az egyes futtatások alapján rendelt információ összességét jelenti
- **Programrész** lehet utasítás, blokk, ág (branch), predikátum, komponens, függvény, stb.
- Spektrum rögzíti a program egyes végrehajtásainak bizonyos aspektusait:
  - az adott **programelem részt vett-e**,
  - illetve **hogyan vett részt** a futási eredmény kialakításában
- Az egyes programfutások **sikeres-/sikertelenségének** mivoltára is szükség van

$$\begin{array}{c} \\ t_1 \\ t_2 \\ t_3 \\ t_4 \end{array} \begin{array}{c} \begin{array}{ccccc} s_1 & s_2 & s_3 & s_4 & s_5 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \end{array} \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

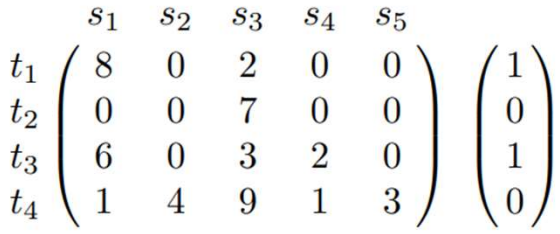
(a) BHS

$$\begin{array}{c} \\ t_1 \\ t_2 \\ t_3 \\ t_4 \end{array} \begin{array}{c} \begin{array}{ccccc} s_1 & s_2 & s_3 & s_4 & s_5 \end{array} \\ \begin{pmatrix} 8 & 0 & 2 & 0 & 0 \\ 0 & 0 & 7 & 0 & 0 \\ 6 & 0 & 3 & 2 & 0 \\ 1 & 4 & 9 & 1 & 3 \end{pmatrix} \end{array} \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

(b) BCS



(a) BHS



(b) BCS

- **Branch Hit Spectrum** (BHS): program egyes ágaihoz jegyezzük végre lettek-e hajtva
- **Branch Count Spectrum** (BCS): végrehajtás ténye, de száma is feljegyzésre kerül
- Számolható például **utasításokra**, **függvényekre**
- Ábrán **lefedettségi mátrix** és **eredményvektor** szerepel (**Programspektrum**)
- Meghatározása után különféle módszerekkel rangsorolás, annak valószínűsége szerint, hogy az adott programelem tartalmazza-e a hibát, vagy sem.

Alapmetrikák

- BHS esetén ez a **négy** alapmetrika a következő:
- $\mathbf{c_{ef}}$: A programelem hány **hibás** (f - fail) **futás** során **volt végrehajtva** (e - ecexuted)
- $\mathbf{c_{ep}}$: A programelem hány **sikeres** (p - pass) **futás** során **volt végrehajtva** (e)
- $\mathbf{c_{nf}}$: A programelem hány **hibás** (f) **futás** során **nem volt érintve** (n - not executed)
- $\mathbf{c_{np}}$: A programelem hány **sikeres** (p) **futás** során **nem volt érintve** (n)

					score			rank		
	c_{ef}	c_{ep}	c_{nf}	c_{np}	Tarantula	Ochiai	Jaccard	Tarantula	Ochiai	Jaccard
s_1	2	1	0	1	0,67	0,82	0,67	1	1	1
s_2	0	1	2	1	0,00	0,00	0,00	4-5	4-5	4-5
s_3	2	2	0	0	0,50	0,71	0,50	2-3	2	2
s_4	1	1	1	1	0,50	0,50	0,33	2-3	3	3
s_5	0	1	2	1	0,00	0,00	0,00	4-5	4-5	4-5

Gyanússág

- következő lépése a **gyanússág kiszámítása**
- rengeteg **formula** áll rendelkezésünkre

Rangsorolási metrika	**Képlet**
Tarantula	$\frac{\frac{c_{ef}}{c_{ef}+c_{nf}}}{\frac{c_{ef}}{c_{ef}+c_{nf}}+\frac{c_{ep}}{c_{ep}+c_{np}}}$
Ochiai	$\frac{c_{ef}}{\sqrt{(c_{ef}+c_{nf})(c_{ef}+c_{ep})}}$
Jaccard	$\frac{c_{ef}}{c_{ef}+c_{nf}+c_{ep}}$
Zoltar	$\frac{c_{ef}}{c_{ef}+c_{nf}+c_{ep}+10000\cdot\frac{c_{nf}c_{ep}}{c_{ef}}}$
O^p	$c_{ef}-\frac{c_{ep}}{c_{ep}+c_{np}+1}$
Kulczynski2	$\frac{1}{2}\left(\frac{c_{ef}}{c_{ef}+c_{nf}}+\frac{c_{ef}}{c_{ef}+c_{ep}}\right)$
McCon	$\frac{c_{ef}^2-c_{nf}c_{ep}}{(c_{ef}+c_{nf})(c_{ef}+c_{ep})}$

					score			rank		
	c_{ef}	c_{ep}	c_{nf}	c_{np}	Tarantula	Ochiai	Jaccard	Tarantula	Ochiai	Jaccard
s_1	2	1	0	1	0,67	0,82	0,67	1	1	1
s_2	0	1	2	1	0,00	0,00	0,00	4-5	4-5	4-5
s_3	2	2	0	0	0,50	0,71	0,50	2-3	2	2
s_4	1	1	1	1	0,50	0,50	0,33	2-3	3	3
s_5	0	1	2	1	0,00	0,00	0,00	4-5	4-5	4-5

Rangsorolás

- Egyszerűen a kiszámolt formulaértékek alapján **csökkenö sorrendbe** rakjuk a programelemeket
- Ha **jó metrikát** használtunk, akkor a **hibás** programelem a **listánk elején** fog szerepelni

	c_{ef}	c_{ep}	c_{nf}	c_{np}	score			rank		
					Tarantula	Ochiai	Jaccard	Tarantula	Ochiai	Jaccard
s_1	2	1	0	1	0,67	0,82	0,67	1	1	1
s_2	0	1	2	1	0,00	0,00	0,00	4-5	4-5	4-5
s_3	2	2	0	0	0,50	0,71	0,50	2-3	2	2
s_4	1	1	1	1	0,50	0,50	0,33	2-3	3	3
s_5	0	1	2	1	0,00	0,00	0,00	4-5	4-5	4-5

Formulák “jósága”

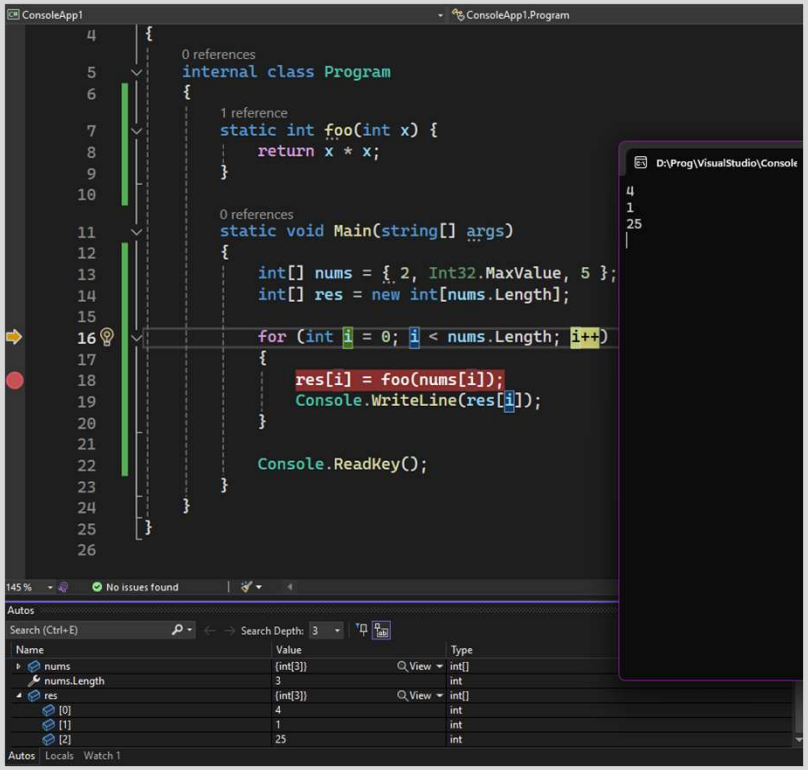
- különböző formulák különböző esetekben eltérően viselkednek

1. legszélesebb körben az **exam score**-t használják metrikaként
   - azoknak a **programelemeknek** a **százalékos arányát** jelenti, amelyeket **feltétlenül meg kell vizsgálni** a **hiba megtalálásához**
2. **Expense** egy exam score variáns olyan programok esetében, melyek **több hibás elemmel** is rendelkeznek
   - **első hiba** megtalálása **előtt ellenőrzött kód százalékos arányát** méri
3. **Top-N** módszer
   - **több egyedi hiba** esetén (amikor összességében sok hibánk van, de **egy-egy mérésnél mindig csak egy** hiba van a szoftverben) a hibás elemeket **hányszor rangsorolja** az adott módszer az **első N megvizsgálandó elem közé**

Hibalokalizáció a gyakorlatban

“One intuitive way to locate bugs when a program execution shows some abnormal behavior is to analyze the corresponding memory dump. Another way is to insert print statements around suspicious code to print out the values of some variables.”

- Department of Computer Science The University of Texas



ConsoleApp1

```python
def avg_data(intervalValue):
    try:...
    except Exception as e:
        raise Exception("avg_data(): "+str(e))


def get_settings():
    try:...
    except Exception as e:
        raise Exception("get_settings(): "+str(e))
```

```python
exceptionRaised=False
while not exceptionRaised:
    try:
        scanInterval,pumpState=get_settings()
        sensorAvg=avg_data(scanInterval)

    except Exception as e:

        htmlPage = """
        <!DOCTYPE html>
        <html>
        <head>
            <title>My MicroPython Web Server</title>
        </head>
        <body>
            <h1>MAIN</h1>
            <h2>"""+str(e)+"""</h2>
            <h2>Restart required</h2>
        </body>
        </html>
        """
```

Források

- Tesztelési módszerek jegyzet
- personal.utdallas.edu/~ewong/fault-localization-survey.pdf
- homes.cs.washington.edu/~mernst/pubs/fault-localization-tr160803.pdf

Köszönöm a figyelmet!