Szoftver- és rendszerellenőrzés (VIMIMA01)

# Kód alapú tesztelés

## Tesztgenerálás struktúra alapján

Majzik István, Micskei Zoltán

http://www.inf.mit.bme.hu/

1

Utolsó módosítás: 2015.11.10.

# Tartalom

- Tesztelési alapok
- **Kód alapú tesztelés**
  - **Tesztesetek származtatása (forrás)kódból**
- Modell alapú tesztelés
  - Tesztesetek származtatása modellből

## Ötlet

- Adott egy program (bináris vagy forráskód)
- Generáljunk hozzá teszteket!
- Valamilyen cél vagy lefedettség alapján

3

# Tesztek kiválasztása struktúra alapján

```
  int fun1(int a, int b){
    if (a == 0){
1     printf(ERROR_MSG);
2     return -1;
    }
    if (b > a)
3     return b*a + 5;
    else
4    return (a+b) / 2;
  }
```

| a | b | utasítás |
|---|---|---|
| 0 | * | 1, 2 |
| a!=0 | b > a | 3 |
| a!=0 | b < a | 4 |

## Mit hagytunk ki?

**teszteset = bemenet + *elvárt kimenet***

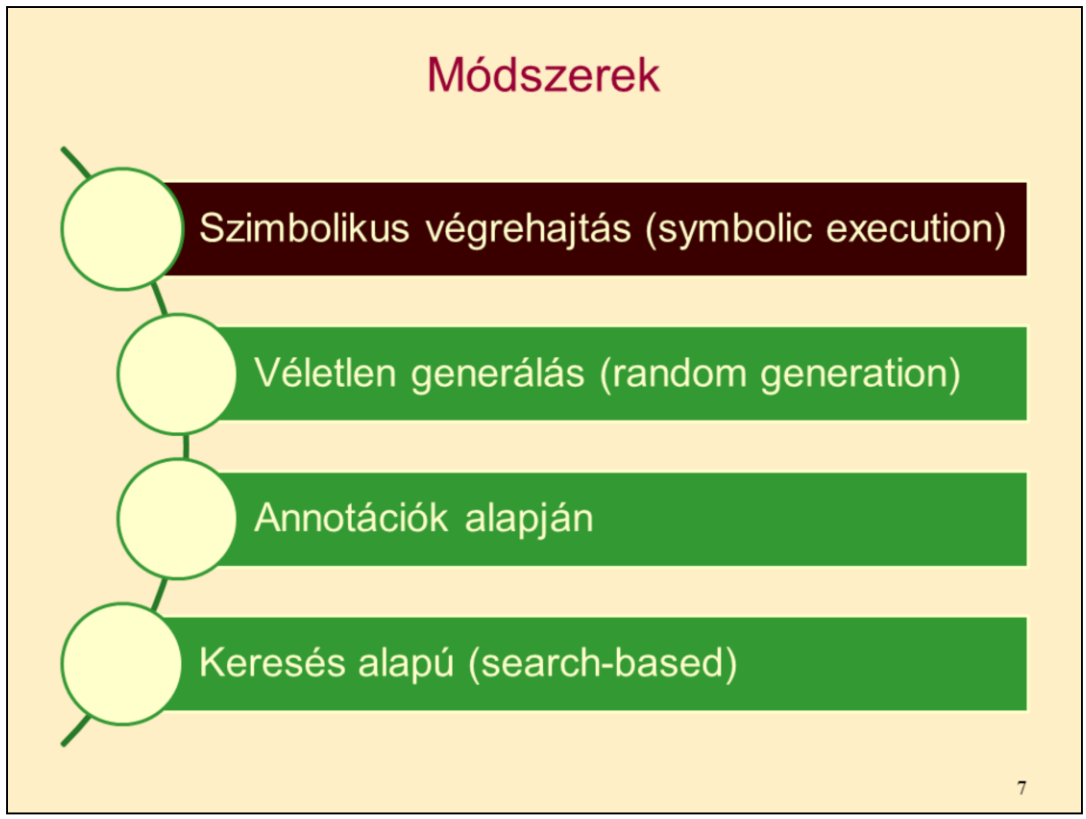
**Mit lehet tenni?**

- Alap, általános hibák (kivételt dob, segfault…)
- Kódban lévő assert() hívások megsértése
- Kézzel határozzuk meg az elvárt eredményt
- Meglévő kimenet felhasználása
  - Regressziós teszt: későbbi változatokkal összehasonlítás
  - Különböző implementációk összehasonlítása

5

# MÓDSZEREK

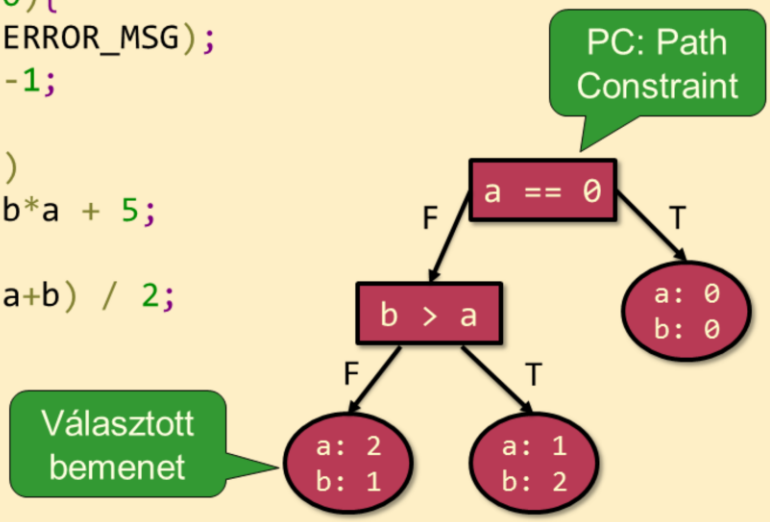
6




A gyakorlatban ezek a technikák sokszor együttesen kerülnek alkalmazásra, nem tisztán szétválaszthatóak.

# Példa: szimbolikus végrehajtás (statikus)

```
  int fun1(int a, int b){
    if (a == 0){
1     printf(ERROR_MSG);
2     return -1;
    }
    if (b > a)
3     return b*a + 5;
    else
4    return (a+b) / 2;
  }
```




8

## Szimbolikus végrehajtás (SE): alapötlet

- Programanalízis technika a ’70-es évekből
- Felhasználás tesztgenerálásra:
  - Szimbolikus változók
  - Utakhoz tartozó kényszerek összegyűjtése
  - Kényszerek megoldása (pl. SMT-megoldó)
  - Megoldások lesznek a tesztek
- 2000-es évek:
  - Elég erős gépek, jó SMT-megoldók
  - Sok ötlet és új eszköz

9

## „Sima” szimbolikus végrehajtás javítása

- SE sok helyen elakadhat
- Ötlet: váltsunk át néha konkrét végrehajtásra
- Megvalósítások:
  - Dynamic Symbolic Execution (DSE)
  - Concolic testing: *conc*erete + symb*olic*

10

Véletlenszerűen választott bemenetekkel futtatás

Bejárt utak és feltételek rögzítése

Ez segít bizonyos kényszereket megoldani

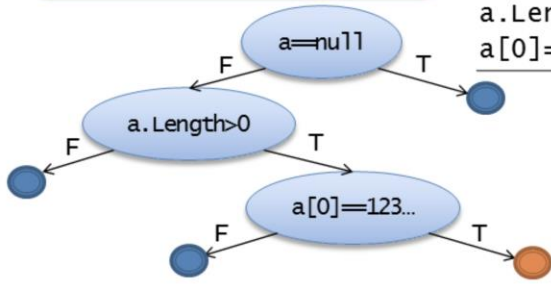
# Dynamic Symbolic Execution


**Code to generate inputs for:**

```
void CoverMe(int[] a)
{
  if (a == null) return;
  if (a.Length > 0)
    if (a[0] == 1234567890)
      throw new Exception("bug");
}
```

| Constraints to solve | Input | Observed constraints |
|---|---|---|
| | null | a==null |
| a!=null | {} | a!=null && |
| a!=null a.Length>0 | | a.Length>0 && a[0]!=1234567890 |
| a!=null && a.Length>0 && a[0]==123456890 | {123..} | a==null && a.Length>0 && a[0]==1234567890 |

Negated condition




**Done: There is no path left.**

Source: N. Tillmann, P. de Halleux. White Box Test Generation for .NET

# Eszközök

| Név | Platform | Bemenet | Egyéb |
|---|---|---|---|
| KLEE | Linux | C (LLVM bitcode) | |
| Pex | Windows | .NET assembly | VS2015: IntelliTest |
| SAGE | Windows | x86 bináris | MS internal, security hibák |
| Jalangi | - | JavaScript | |
| Symbolic PathFinder | - | Java | |

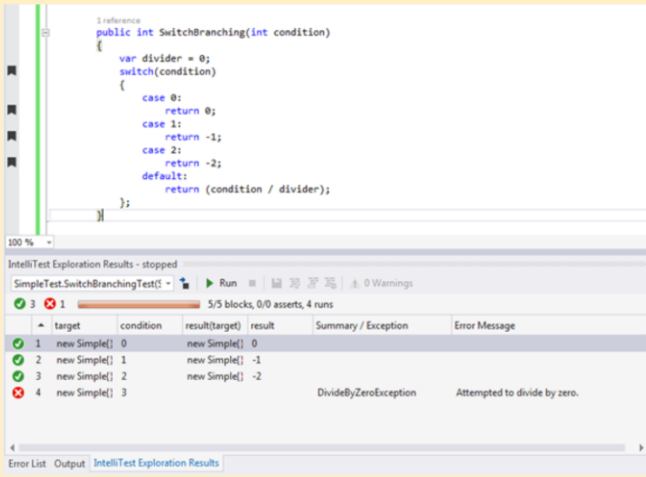
További (jobbára nem fejlesztett) eszközök:
CATG, CREST, CUTE, Euclide, EXE, jCUTE, jFuzz, LCT, Palus, PET…

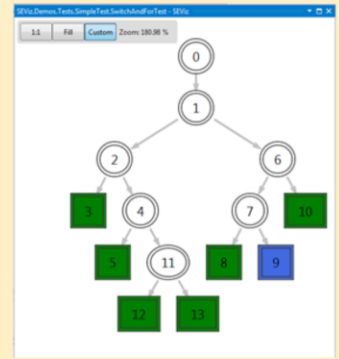
12

Részletes listát lásd:
http://mit.bme.hu/~micskeiz/pages/code_based_test_generation.html

# **DEMO**: Microsoft IntelliTest

Generate unit tests for your code with IntelliTest
https://msdn.microsoft.com/en-us/library/Dn823749.aspx

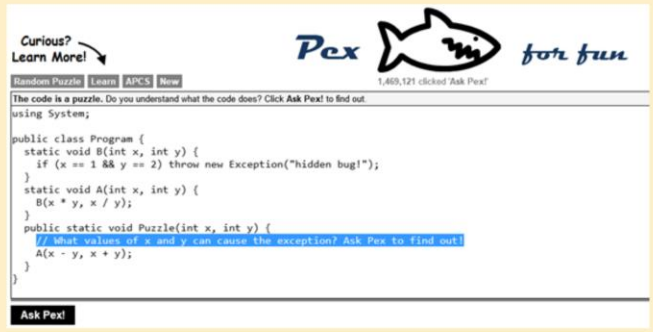



SEViz (Symbolic Execution VisuallZer)
https://github.com/FTSRG/seviz

# Pex for fun / Code Hunt

**http://pexforfun.com**




**http://codehunt.com**

## Kihívások

1. Futási utak exponenciális növekedése
2. Komplex aritmetikai kifejezések
3. Lebegőpontos számítások
4. Összetett struktúrák és objektumok
5. Mutató műveletek
6. Interakció a környezettel
7. Többszálú alkalmazások
8. …

T. Chen et al. „State of the art: Dynamic symbolic execution for automated test generation". Future Generation Computer Systems, 29(7), 2013

# Kihívások (1)

## Futási utak exponenciális növekedése

```
int hardToTest(int x){
	for (int i=0; i<100; i++){
		int j = complexMathCalc(i,x);
		if (j > 0) break;
	}

	return i;
}
```

- Ötlet:
  - DFS helyett más bejárási algoritmusok
  - *Summary*: bonyolult függvényt helyettesít

17

# Kihívások (2)

## Komplex aritmetikai kifejezések

```
int hardToTest2(int x){
       if (log(x) > 10)
              return x
       else
              return -x;
}
```

- Legtöbb SMT-megoldó nem boldogul ilyennel
  - De: pl. CORAL-t pont ilyenre fejlesztik
  - Többféle megoldó használata a feltételtől függően

18

# Kihívások (4)

Összetett struktúrák és objektumok

- Struktúrák, rekurzív adatszerkezetek
- Ötlet: **Lazy initialization**
  - Kezdetben minden mező (field) nem inicializált
  - Csak akkor adunk értéket, ha hozzányúlunk
    - Érték: null, referencia egy új objektumra/meglévőre

19

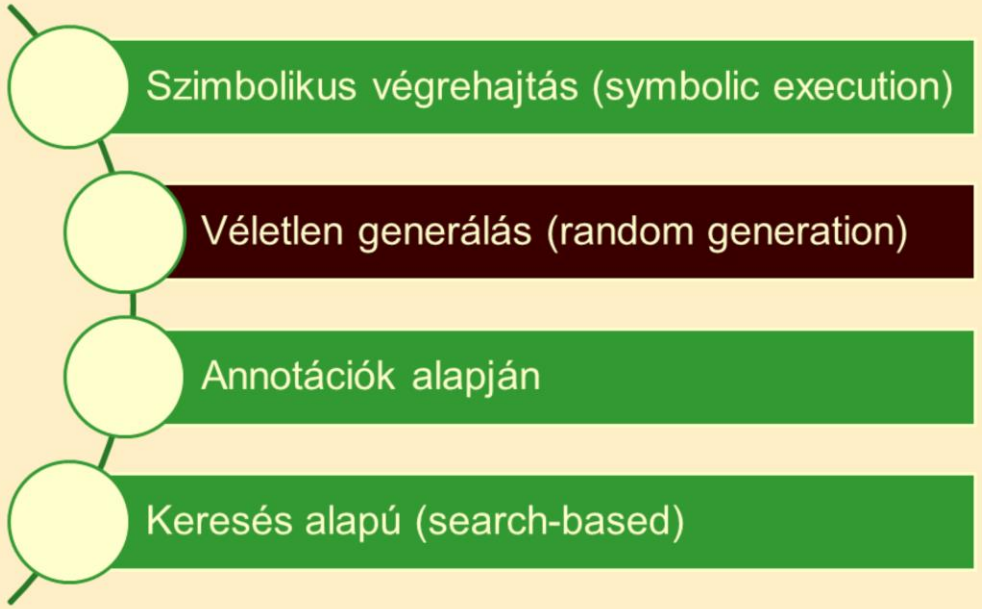
# Kihívások (6)

## Interakció a környezettel

```
int hardToTest3(string s){
  FileStream fs = File.Open(s, FileMode.Open);
  if (fs.Lenth > 1024){
    return 1;
  } else
    return 0;
  }
}
```

- Platform- vagy könyvtárhívás gyakori
- Ötlet:
  - „Environment models” (KLEE): egyszerű C programok
  - speciális Security Manager (Java)

# Módszerek




21

## Véletlen tesztgenerálás

Bemeneti tartományból véletlen választás

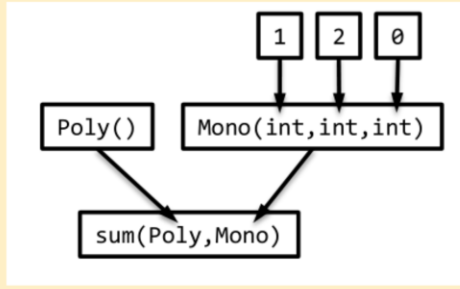
- Előny:
  - Gyors, olcsó
- Ötletek:
  - Ha egy bemenet nem talál hibát, más részt próbálni
  - Különbözőség, távolság, stb. alapján
- Eszköz:


22

Randoop: http://mernst.github.io/randoop/

# Randoop: visszacsatolás vezérelt generálás

- Metódus szekvenciák generálása
- Összetett objektumok:




- Heurisztikák:
  - Kiválasztott eset futtatása
  - Érvénytelen, redundáns eldobása

23

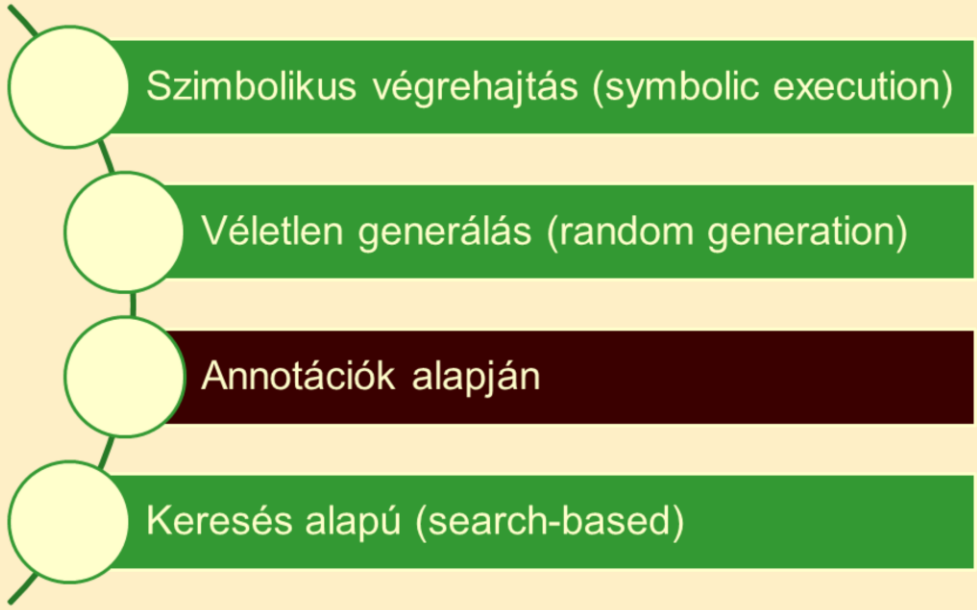
## RT esettanulmányok

- Robusztusság tesztelés
  - Fuzz: véletlenszerű bemenet konzolos programokhoz
    - Unix (1990), Unix (1995), MacOS (2007)
  - NASA: flash fájlrendszer
    - HW hibák szimulálása, összehasonlítás referenciákkal
    - (Modell ellenőrzés nem skálázódott)
- Randoop
  - JDK, .NET könyvtárak: alapvető tulajdonságok ellenőrzése (pl. `o.equals(o) returns true`)
  - JDK 1.5 és 1.6 összehasonlítása
  - Hibák sokat tesztelt komponensekben!

24


- Miller, B. et al.: Fuzz revisited: A re-examination of the reliability of Unix utilities and services. Computer Sciences Technical Report #1268, University of Wisconsin-Madison. (1995)
- A. Groce, G. Holzmann, and R. Joshi. Randomized differential testing as a prelude to formal verification. In IEEE International Conference on Software Engineering (ICSE), pages 621–631, 2007.
- Pacheco C, Lahiri SK, Ernst MD, Ball T. Feedback-directed random test generation. Int. Conf. on Software Engineering, ICSE'07, 2007; 75–84, doi:10.1109/ICSE.2007.37.

# Módszerek




25

# Annotációk felhasználása

Ha vannak a kódban:

- elő- és utófeltételek (pl.: design by contract)
- egyéb annotációk

ezek irányíthatják a tesztgenerálást.

```
/*@ requires amt > 0 && amt <= acc.bal;
    @ assignable bal, acc.bal;
    @ ensures bal == \old(bal) + amt
    @   && acc.bal == \old(acc.bal - amt); @*/
  public void transfer(int amt, Account acc) {
    acc.withdraw(amt);
    deposit(amt);
  }
```

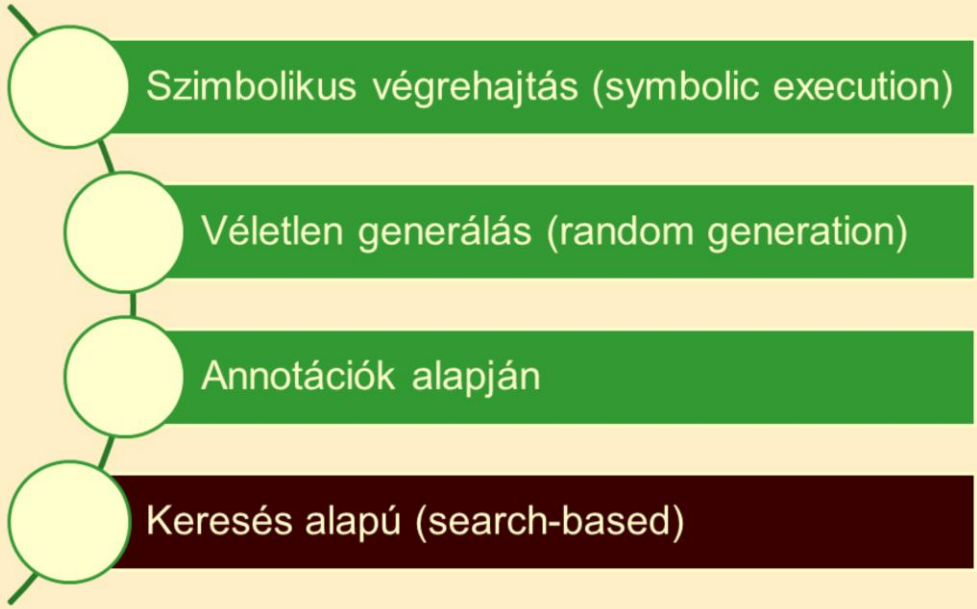
# Eszközök

- AutoTest:
  - Eiffel kód, Design by Contract
  - Bemenet: „object pool”, véletlen generálás
    - Ötlet: előfeltételeket kielégítő bemenetek hozzávétele
  - Elvárt kimenet: szerződés

AutoTest: Bertrand Meyer et al., "Program that Test Themselves", IEEE Computer 42:9, 2009.

- JET:
  - Java kód, JML (Java Modeling Language) annotáció
  - Utófeltételek megsértését nézi (véletlen tesztek)

# Módszerek




28

## Keresés alapú technikák

**Search based Software Engineering (SBSE)**

- Metaheurisztikus algoritmusok
  - genetikus alg., szimulált lehűlés, hegymászó…
- Szükséges:
  - Keresési tér: program + lehetséges bemenetek
  - Célfüggvény: tesztelési cél (pl. adott elágazás lefedése)

29

- McMinn P. Search-based software test data generation: a survey. Software Testing, Verification and Reliability 2004;

14(2):105–156, doi:10.1002/stvr.294.

- Harman M, Mansouri SA, Zhang Y. Search based software engineering: A comprehensive analysis and review of

trends techniques and applications. Technical Report TR-09-03, Dept. of Computer Science, King's College London

2009.

## Eszközök

EVOSUITE

- „Whole test suite generation”
  - Összes teszt cél figyelembe vétele egyszerre
  - Többféle metrika alapján (is) keres
    - Pl. nagy lefedettség + kis tesztkészlet
- Tesztek minimalizálása, olvashatósága
- „Sandbox” használata

30

„As we learned through directories cluttered with randomly named files after experiments, randomly disappearing project files, and an unfortunate episode where one of the authors lost the entire contents of his home directory, applying test generation tools to unknown software can lead to interactions with the environment in unpredictable ways.” -- G. Fraser and A. Arcuri, “A Large Scale Evaluation of Automated Unit Test Generation Using EvoSuite,” *ACM TOSEM*, vol. 24, iss. 2, p. 8, 2014.

# KIÉRTÉKELÉSEK

31

## Mennyire használhatóak ezek a módszerek?

- SF100 benchmark (Java)
  - 100 projekt SourceForge-ról
  - EvouSuite 48%-os elágazás lefedettséget ér el
    - De nagy a szórás!

G. Fraser and A. Arcuri, "Sound Empirical Evidence in Software Testing," ICSE 2013

- Beágyazott rendszer (C)
  - CREST és KLEE futtatása az ABB egy projektjén
  - Kb. 60%-os elágazás lefedettséget érnek el
    - De sok esettel nem boldogulnak!

X. Qu, B. Robinson: A Case Study of Concolic Testing Tools and Their Limitations, ESEM 2011

32

# Tényleg jók ezek a technikák?

- Tesztelőt mennyire segíti
  - 49 résztvevő tesztelt (manuális, automatikus)
  - Nagy kód lefedettséget elérő generált tesztek nem találtak több hibát, mint a kézzel előállított tesztek.

G. Fraser et al., "Does Automated White-Box Test Generation Really Help Software Testers?," ISSTA 2013

- Valós hibák megtalálása
  - Defetcts4J: 357 valós hiba 5 projektből
  - Eszközök: EvoSuite, Randoop, Agitar
  - Csak a hibák 55%-át találták meg

S. Shamshiri et al., „Do automatically generated unit tests find real faults? An empirical study of effectiveness and challenges." ASE 2015

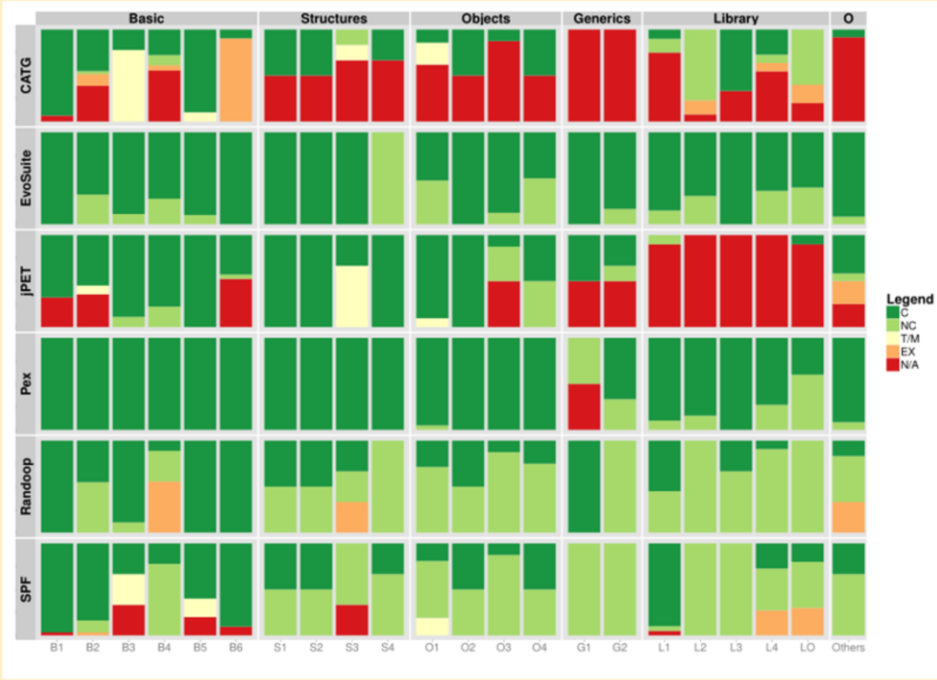
33

## Tesztgeneráló eszközök összehasonlítása

- Programrészekből álló bemenet az eszközöknek
  - Fontos nyelvi elemek lefedése
  - Problémás esetek (ld. kihívások)
- 300 darab Java/.NET kódrészlet
  - 6 eszközön végrehajtva
- Tapasztalat:
  - Egyszerű dolgokon is elakadnak

L. Cseppentő, Z. Micskei: „Evaluating Symbolic Execution-based Test Tools," ICST'15

# Tesztgeneráló eszközök összehasonlítása (2)




35

## Olvasnivaló

1. S. Anand et al. (2013) "*An orchestrated survey of methodologies for automated software test case generation*," Journal of Systems and Software, 86:8, pp. 1978–2001. DOI: j.jss.2013.02.061

## Összefoglalás

- Tesztbemenetek generálása struktúra alapján
- Többféle módszer
- Kihívások:
  - Skálázódás
  - Orákulum előállítása
- Aktív kutatási terület!

37