M5 - Differential Testing

Guillermo Polito

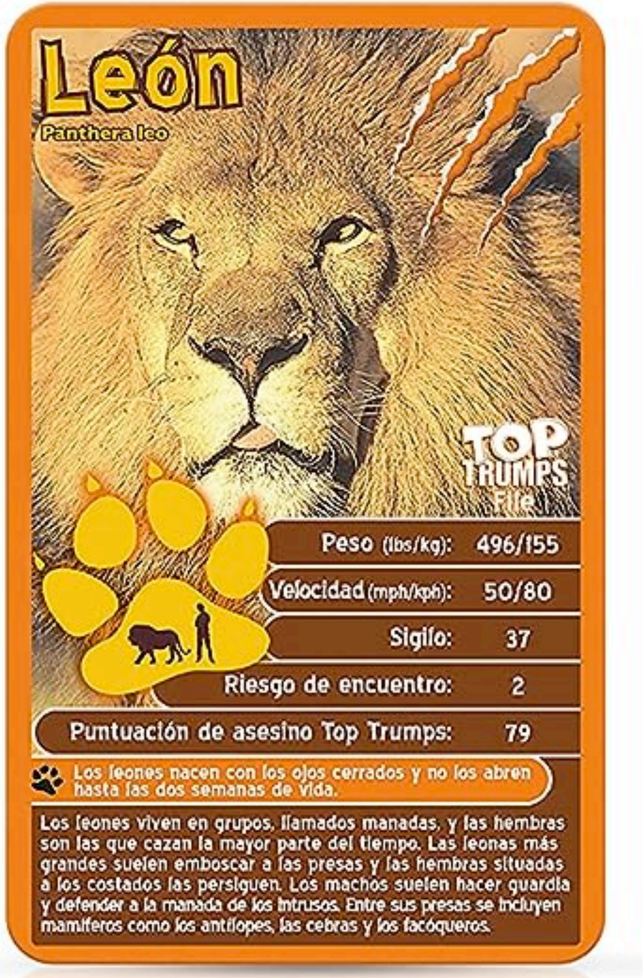
ECI’23 - Universidad de Buenos Aires

Goals

- Introduce differential testing

- A practical approach for the oracle problem: existing software
- Discuss about Implementations vs specifications

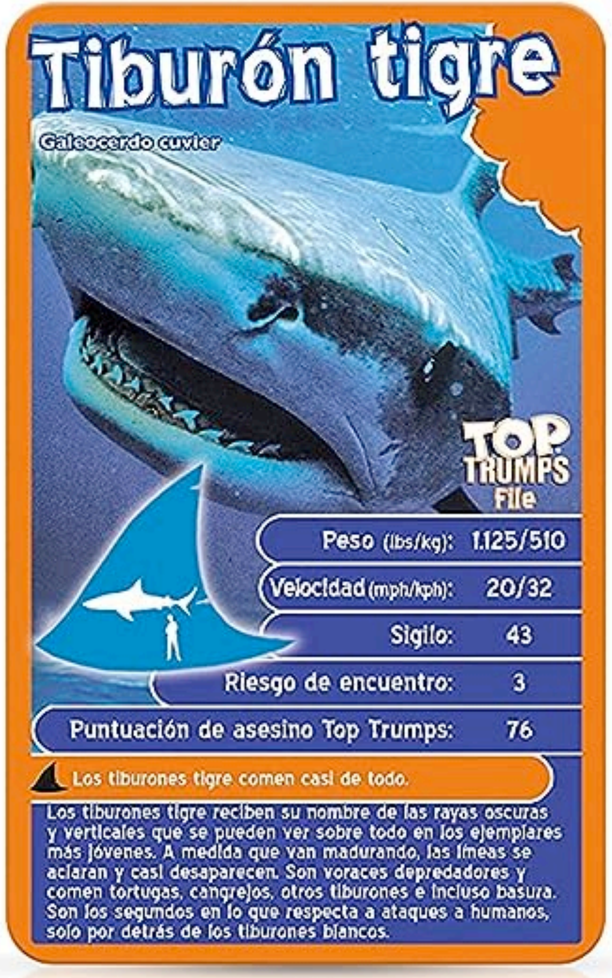
Let’s Talk Benchmarks



- Speed: 80 km/h
- Is the lion ***fast***?

The lion is fast… against tiger sharks

80 km/h



32 km/h

Comparisons to assess properties

- What properties should we compare?
  - speed? weight? A vs A’? A vs B’?
- How should we measure?
  - Do lions show similar speed everywhere?
  - Do lions show similar speed morning and night?
- What elements should be compare?
  - Lion A vs Lion B? Lion vs Tiger?

How can we use comparisons for testing?

Remember Assertions

SetTest >> testAdd

```
| aSet |
"Context"
aSet := Set new.

“Stimuli"
aSet add: 5.
aSet add: 5.

"Check"
self assert: aSet size equals: 1.
```

in *this context*
when *this happens*
then *this should happen*

Comparisons against known values

SetTest >> testAdd

```
| aSet |
"Context"
aSet := Set new.

“Stimuli"
aSet add: 5.
aSet add: 5.

"Check"
self assert: aSet size equals: 1.
```

in *this context*
when *this happens*
then *this should happen*

Assertions against similar values?

```
SetTest >> testAdd

   | aSet |
   "Context"
   aSet := Set new.

   “Stimuli"
   aSet add: 5.
   aSet add: 5.

   "Check"
   self assert: aSet size equals: ?????.
```

in *this context*
  when *this happens*
    then *this should happen*

Remember Fuzzing Date Parser

```
f := PzRandomFuzzer new.
r := PzBlockRunner on: [ :e | e asDate ].
f run: r times: 20.
```

- Pharo 11
- String>>asDate

```
PASS "DateError: day is after month ends"
PASS "28 April 2006"
PASS "7 September 2029"
PASS "9 March 1995"
FAIL "SubscriptOutOfBounds: 73"
PASS "DateError: day is after month ends"
FAIL "SubscriptOutOfBounds: 0"
PASS "DateError: day is after month ends"
PASS "6 January 2007"
PASS "9 January 1986"
FAIL "SubscriptOutOfBounds: 0"
FAIL "#isAlphaNumeric was sent to nil"
PASS "DateError: day is after month ends"
PASS "1 September 1989"
PASS "DateError: day is after month ends"
PASS "DateError: day may not be zero or negative"
PASS "5 January 0228"
PASS "DateError: day may not be zero or negative"
PASS "7 September 1996"
PASS "2 January 2008"
```

Remember Fuzzing Date Parser

```
f := PzRandomFuzzer new.
r := PzBlockRunner on: [ :e | e asDate ].
f run: r times: 20.
```

- Pharo 11
- String>>asDate

```
PASS "DateError: day is after month ends"
PASS "28 April 2006"
PASS "7 September 2029"
PASS "9 March 1995"
FAIL "SubscriptOutOfBounds: 73"
PASS "DateError: day is after month ends"
FAIL "SubscriptOutOfBounds: 0"
PASS "DateError: day is after month ends"
PASS "6 January 2007"
PASS "9 January 1986"
FAIL "SubscriptOutOfBounds: 0"
FAIL "#isAlphaNumeric was sent to nil"
PASS "DateError: day is after month ends"
PASS "1 September 1989"
PASS "Dat
PASS "Dat
PASS "5 J
PASS "Dat
PASS "7 S
PASS "2 J
```

How do we decide: what is a PASS, what is a FAIL?

Remember Fuzzing Date Parser

```
f := PzRandomFuzzer new.
r := PzBlockRunner on: [ :e | e asDate ].
f run: r times: 20.
```

- Pharo 11
- String>>asDate

```
PASS "DateError: day is after month ends"
PASS "28 April 2006"
PASS "7 September 2029"
PASS "9 March 1995"
FAIL "SubscriptOutOfBounds: 73"
PASS "DateError: day is after month ends"
FAIL "SubscriptOutOfBounds: 0"
PASS "DateError: day is after month ends"
PASS "6 January 2007"
PASS "9 January 1986"
FAIL "SubscriptOutOfBounds: 0"
FAIL "#isAlphaNumeric was sent to nil"
PASS "DateError: day is after month ends"
PASS "1 September 1989"
PASS "Dat
PASS "Dat                                    ve"
PASS "5 J
PASS "Dat                                    ve"
PASS "7 S
PASS "2 J
```

Who decides: what is a PASS, what is a FAIL?

The Date Parser Oracle

```
f := PzRandomFuzzer new.
r := PzBlockRunner on: [ :e | e asDate ].
f run: r times: 20.
```

- Can we use another parser?


```
PASS "DateError: day i
PASS "28 April 2006"
PASS "7 September 202
PASS "9 March 1995"
FAIL "SubscriptOutOfBc
PASS "DateError: day is
FAIL "SubscriptOutOfBounds: 0"
PASS "DateError: day is after month ends"
PASS "6 January 2007"
PASS "9 January 1986"
FAIL "SubscriptOutOfBounds: 0"
FAIL "#isAlphaNumeric was sent to nil"
PASS "DateError: day is after month ends"
PASS "1 September 1989"
PASS "Dat
PASS "Dat
PASS "5 J
PASS "Dat
PASS "7 S
PASS "2 J
```

How do we decide: what is a PASS, what is a FAIL?

The Date Parser Oracle

```
f := PzRandomFuzzer new.
r := PzBlockRunner on: [ :e | e asDate ].
f run: r times: 20.
```

- Can we use another parser?
- (dis)agreement is evidence!
  - Agreement: parsers have same behavior
  - Disagreement: is there a bug?


```
PASS "DateError: day i
PASS "28 April 2006"
PASS "7 September 202
PASS "9 March 1995"
FAIL "SubscriptOutOfBc
PASS "DateError: day is
FAIL "SubscriptOutOfBounds: 0"
PASS "DateError: day is after month ends"
PASS "6 January 2007"
PASS "9 January 1986"
FAIL "SubscriptOutOfBounds: 0"
FAIL "#isAlphaNumeric was sent to nil"
PASS "DateError: day is after month ends"
PASS "1 September 1989"
PASS "Dat
PASS "Dat
PASS "5 J
PASS "Dat
PASS "7 S
PASS "2 J
```

How do we decide: what is a PASS, what is a FAIL?

Date>>fromString vs DateParser

- `DateParser` is a structured parser based on a specified format

```
DateParser readFrom: string readStream pattern: 'dd-mm-yyyy'
```

- Should be more strict than `asDate`

- What kind of comparisons are fair/safe/legal?

A dd-mm-yyyy Grammar

```
ntNumber --> ntDigit, ntNumber | ntDigit.
ntDigit --> ($0 - $9).

ntDate
  --> ntDay, ntSeparator, ntMonth, ntSeparator, ntYear.
ntSeparator --> '-'.
ntDay --> ntNumber.
ntMonth —> ntNumber.
ntYear --> ntNumber.
```

Differential Parser Testing

```
runnerA := PzBlockRunner on: [ :e | e asDate ].
runnerA expectedException: DateError.

runnerB := PzBlockRunner on: [ :e |
  (Date readFrom: e readStream pattern: 'dd-mm-yyyy') ].

diffRunner := PzDifferentialRunner new
  runnerA: runnerA;
  runnerB: runnerB;
  yourself.

f := PzGrammarFuzzer on: PzDateDDMMYYYYGrammar new.
f run: diffRunner times: 100.
```

Results

- fuzz 100 times
- 70/100 errors!

```
FAIL 74-0-8 FAIL 74-0-8 SubscriptOutOfBounds: 0 PASS-FAIL 74-0-8 DateError
FAIL 3-2-6 PASS 3-2-6 2 March 2006 PASS-FAIL 3-2-6 DateError
PASS 5-515-1 PASS-FAIL 5-515-1 DateError: day is after month ends PASS-FAI
FAIL 63-2-1 PASS 63-2-1 1 February 2063 PASS-FAIL 63-2-1 DateError
FAIL 4220-05-1 PASS 4220-05-1 1 May 4220 PASS-FAIL 4220-05-1 DateError
PASS 8-71-3 PASS-FAIL 8-71-3 DateError: day is after month ends PASS-FAIL 8
FAIL 6-7-34 PASS 6-7-34 7 June 2034 PASS-FAIL 6-7-34 DateError
FAIL 2-2-9 PASS 2-2-9 2 February 2009 PASS-FAIL 2-2-9 DateError
FAIL 29-2-0 FAIL 29-2-0 Error: Month out of bounds: 29. PASS-FAIL 29-2-0 Da
PASS 4-00-94 PASS-FAIL 4-00-94 DateError: day may not be zero or negative
PASS 41150-8-0 PASS-FAIL 41150-8-0 DateError: day may not be zero or negat
FAIL 128-8-6 PASS 128-8-6 6 August 0128 PASS-FAIL 128-8-6 DateError
PASS 6-50054228-3 PASS-FAIL 6-50054228-3 DateError: day is after month end
FAIL 39-4318-675 FAIL 39-4318-675 SubscriptOutOfBounds: 4318 PASS-FAIL 39-4
FAIL 0-7-9 FAIL 0-7-9 SubscriptOutOfBounds: 0 PASS-FAIL 0-7-9 DateError
FAIL 0-069848-2 FAIL 0-069848-2 SubscriptOutOfBounds: 0 PASS-FAIL 0-069848
FAIL 9635-14-56 FAIL 9635-14-56 SubscriptOutOfBounds: 14 PASS-FAIL 9635-14
FAIL 6461-5-8 PASS 6461-5-8 8 May 6461 PASS-FAIL 6461-5-8 DateError
FAIL 07-8-49 PASS 07-8-49 8 July 2049 PASS-FAIL 07-8-49 DateError
FAIL 61-74-51 FAIL 61-74-51 SubscriptOutOfBounds: 74 PASS-FAIL 61-74-51 Dat
FAIL 36-41-8 FAIL 36-41-8 SubscriptOutOfBounds: 41 PASS-FAIL 36-41-8 DateE
FAIL 65-9-2 PASS 65-9-2 2 September 2065 PASS-FAIL 65-9-2 DateError
FAIL 321-3246-2 FAIL 321-3246-2 SubscriptOutOfBounds: 3246 PASS-FAIL 321-3
PASS 6-495-1 PASS-FAIL 6-495-1 DateError: day is after month ends PASS-FAI
FAIL 2-9-74 PASS 2-9-74 9 February 1974 PASS-FAIL 2-9-74 DateError
FAIL 9158-909-0 FAIL 9158-909-0 SubscriptOutOfBounds: 909 PASS-FAIL 9158-90
PASS 7-41203-518 PASS-FAIL 7-41203-518 DateError: day is after month ends
PASS 7-55-8 PASS-FAIL 7-55-8 DateError: day is after month ends PASS-FAIL
FAIL 0-78-9 FAIL 0-78-9 SubscriptOutOfBounds: 0 PASS-FAIL 0-78-9 DateError
FAIL 4-9-7 PASS 4-9-7 9 April 2007 PASS-FAIL 4-9-7 DateError
PASS 6-1242376-1 PASS-FAIL 6-1242376-1 DateError: day is after month ends
PASS 2-169728-6327 PASS-FAIL 2-169728-6327 DateError: day is after month e
FAIL 99133-0-023 FAIL 99133-0-023 SubscriptOutOfBounds: 0 PASS-FAIL 99133-0
FAIL 08-4-64 PASS 08-4-64 4 August 2064 PASS-FAIL 08-4-64 DateError
FAIL 523-55-0 FAIL 523-55-0 SubscriptOutOfBounds: 55 PASS-FAIL 523-55-0 Dat
PASS 5-696-8 PASS-FAIL 5-696-8 DateError: day is after month ends PASS-FAI
FAIL 77-6-8 PASS 77-6-8 8 June 1977 PASS-FAIL 77-6-8 DateError
FAIL 38-946-6 FAIL 38-946-6 SubscriptOutOfBounds: 946 PASS-FAIL 38-946-6 Da
FAIL 2615-5-7 PASS 2615-5-7 7 May 2615 PASS-FAIL 2615-5-7 DateError
```

Where is the bug? Case 1

- 2-5-35106
  - `asDate:` PASS - 5 February 35106
  - `DateParser:` PASS-FAIL - DateError

Where is the bug? Case 2

- 35-683-87
  - `asDate:` FAIL - Out of bounds exception
  - `DateParser:` PASS-FAIL - DateError

Where is the bug? Case 3

- 7-2-34
  - `asDate:` PASS - 7/2/2034
  - `DateParser:` PASS-FAIL - DateError

Building a Differential Runner

```
DifferentialRunner>>value: input

  | resultA resultB |
  resultA := self runnerA value: input.
  resultB := self runnerB value: input.

  resultA first = resultB first ifTrue: [
    ^ self successWith: { input . resultA . resultB } ].
  ^ self failureWith: { input . resultA . resultB}
```

Possible Extensions

- Compare more than one program + consensus algorithms
- Different strategies to compare results
  - Maybe similar status, but different results
  - e.g., two different errors!
- Building grammars automatically out of patterns

```
myGrammar := DateGrammar fromPattern: 'dd-mm-yyyy'
```

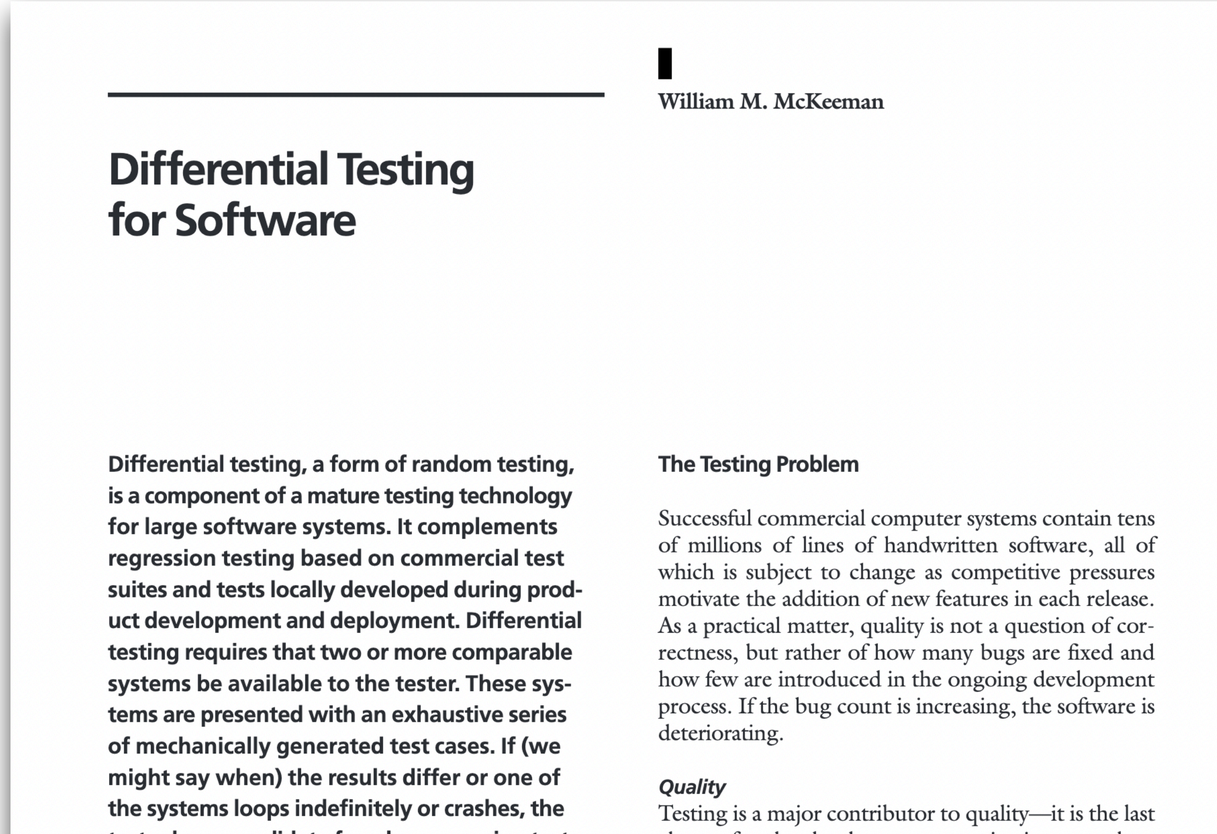
Takeaways

- Two programs following the same *specification* can test each other
- Maybe neither holds the ground truth

- Not only useful for parsing. Think also:
  - compilers
  - applications migrated between different technologies
  - rewritten applications

Material

- Differential Testing for Software. DIGITAL TECHNICAL JOURNAL, 1998. W. M. McKeeman.



William M. McKeeman

Differential Testing for Software

Differential testing, a form of random testing, is a component of a mature testing technology for large software systems. It complements regression testing based on commercial test suites and tests locally developed during product development and deployment. Differential testing requires that two or more comparable systems be available to the tester. These systems are presented with an exhaustive series of mechanically generated test cases. If (we might say when) the results differ or one of the systems loops indefinitely or crashes, the

The Testing Problem

Successful commercial computer systems contain tens of millions of lines of handwritten software, all of which is subject to change as competitive pressures motivate the addition of new features in each release. As a practical matter, quality is not a question of correctness, but rather of how many bugs are fixed and how few are introduced in the ongoing development process. If the bug count is increasing, the software is deteriorating.

Quality

Testing is a major contributor to quality—it is the last