The University Of Sheffield.

Searching for Readable, Realistic Test Cases

Phil McMinn

including joint work with
Sheeva Afshan, Gordon Fraser, Muzammil Shahbaz & Mark Stevenson

Automatic Testing has long been concerned with mainly achieving

coverage

Phil McMinn – p.mcminn@sheffield.ac.uk

But typically the generated inputs and the resulting outputs will have to be evaluated by a human in order to check for correctness

How to reduce human-checking effort?

How to reduce human oracle cost?

Reduce the *amount* of work

generate test data that maximises ge

but minimises th

Quantitative approaches

tia, P. McMinn, and S. Yoo.
g for the number of tests generated in search based test data generation
with an application to the oracle cost problem. SBST 2010

reduce size of test cases

A. Leitner, M. Oriol, A. Zeller, I. Ciupa, and B. Meyer. Efficient unit test case minimization.
ASE 2007, pp. 417–420. ACM.

How to reduce human oracle cost?

Reduce the *difficulty* of the work

how easily can the scenario comprising a generated test case be understood so it can be evaluated for correctness?

Typical Automatic Test Case Generation

-4048
-10854
-29141
3140
733
....

Machine-generated test data tends to not fit the operational profile of a program particularly well

Typical Automatic Test Case Generation

!&^@s.sd

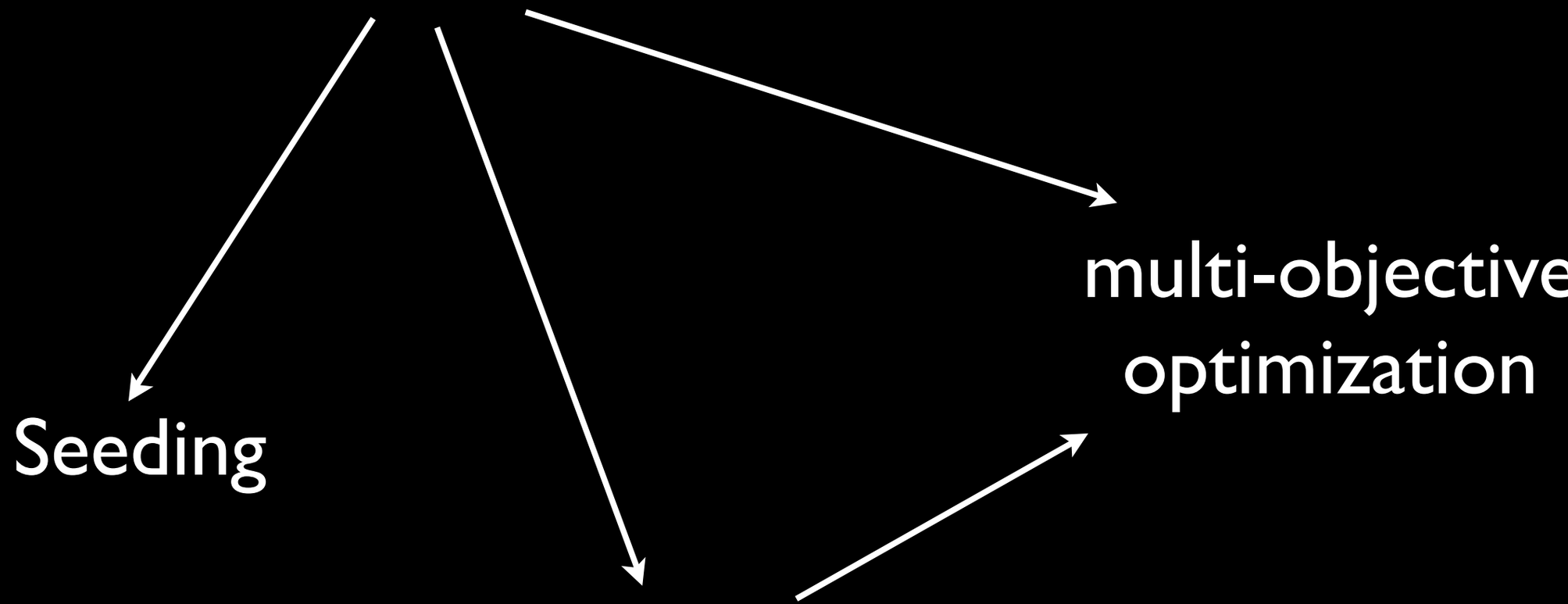
Valid

mark.harman@ucl.ac.uk

Readable, realistic

Tools in the toolbox

Tools in the toolbox



Seeding

Programmer sanity checks

P. McMinn, M. Stevenson and M. Harman.
"Reducing Qualitative Human Oracle Costs associated with Automatically Generated Test Data". Proc. STOV 2010

Already-existing tests

Gordon Fraser, Andrea Arcuri: The Seed is Strong: Seeding Strategies in Search-Based Software Testing. Proc. ICST 2012

Other sources

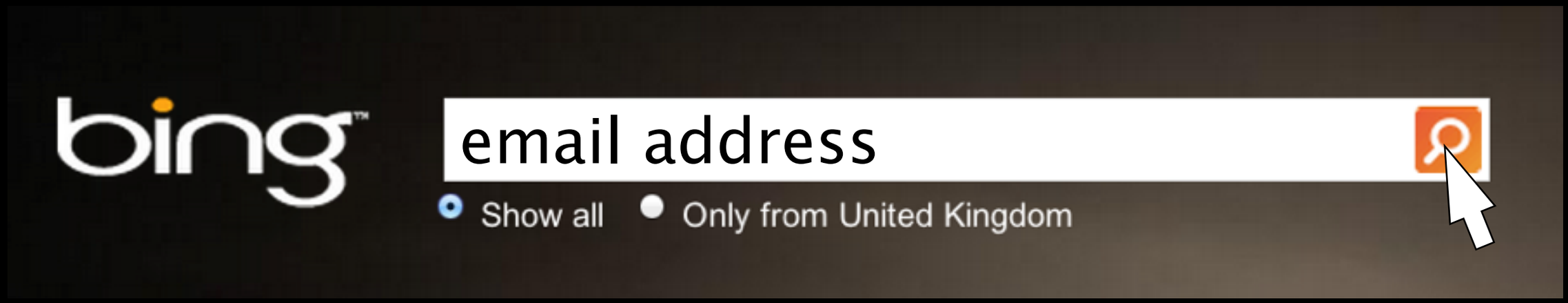
Documentation

Project wikis

Actual usage of the software

Email/IM conversations?

Using the Internet for String examples



WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact Wikipedia

Toolbox

Print/export

Languages
Česky
Dansk

Email address

From Wikipedia, the free encyclopedia

An **email address** identifies an email box to which email messages are delivered. An example format of an email address is `lewis@example.net` which is read as *lewis at example dot net*. Many earlier email systems used different address formats.

Contents [hide]

1 Overview
2 Syntax
  2.1 Local part
  2.2 Domain part
  2.3 Examples
    2.3.1 Valid email addresses
    2.3.2 Invalid email addresses
3 Common local-part semantics
  3.1 Local-part normalization
  3.2 Address tags
4 Validation
  4.1 Identity validation
5 Internationalization
  5.1 Internationalization Examples
  5.2 Internationalization Support
6 See also
7 References

Email address

From Wikipedia, the free encyclopedia

An **email address** identifies an email box to which email messages are delivered. An example format of an email address is `lewis@example.net` which is read as *lewis at example dot net*. Many earlier email systems used different address formats.

Email addresses, such as `jsmith@example.org` have two parts. The part sign is the *local-part* of the address, often the username of the recipient (`jsmi` part after the @ sign is a *domain name* to which the email message will be se (`example.org`).

An **email address** identifies an email box to which email messages are delivered. An example format of an email address is `lewis@example.net` which is read as *lewis at example dot net*. Many earlier email systems used different address formats.

Email addresses, such as `jsmith@example.org` have two parts. The part sign is the *local-part* of the address, often the username of the recipient (`jsmi` part after the @ sign is a *domain name* to which the email message will be se (`example.org`).

P. McMinn, M. Shahbaz and M. Stevenson.
"Search-Based Test Input Generation for String Data Types Using the Results of Web Queries".
Proc. ICST 2012

Phil McMinn – p.mcminn@sheffield.ac.uk

Knowing what to look for

```
class Mailer {

    boolean isValid(String emailAddress) {

        // ...
    }
}
```

Knowing what to look for

```
class Util {

    boolean isEmailAddress(String str) {

        // ...
    }
}
```

Knowing what to look for

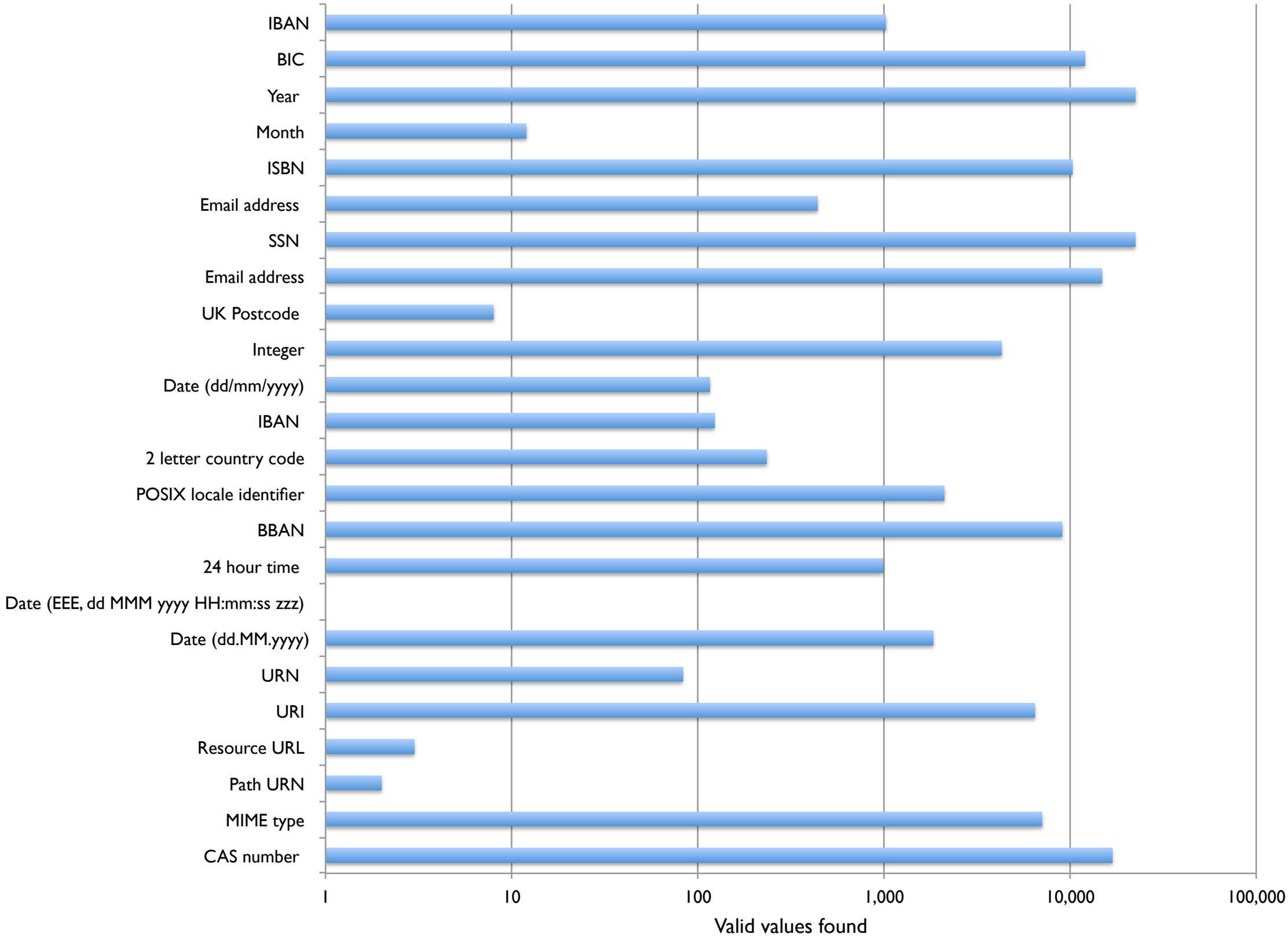
```
class EmailAddress {

    boolean isValid(String str) {

        // ...
    }
}
```

Phil McMinn – p.mcminn@sheffield.ac.uk

String Types

Chemeval	CAS registry numbers
Conzilla	Mime types, Path URNs, Resource URLs, URIs, URNs
Efisto	Dates
GSV05	24 hour times, POSIX locale identifiers, Bank Identifier codes (BICs), International bank account numbers (IBANs), IBAN country codes
LGOL	Dates, Integers, UK postcodes
OpenSymphony	Email addresses, US social security numbers
PuzzleBazar	Email addresses
TMG	International standard book numbers (ISBNs), Month names, Four digit years
WIFE	Bank identifier codes (BICs), International bank account numbers (IBANs)

Phil McMinn – p.mcminn@sheffield.ac.uk

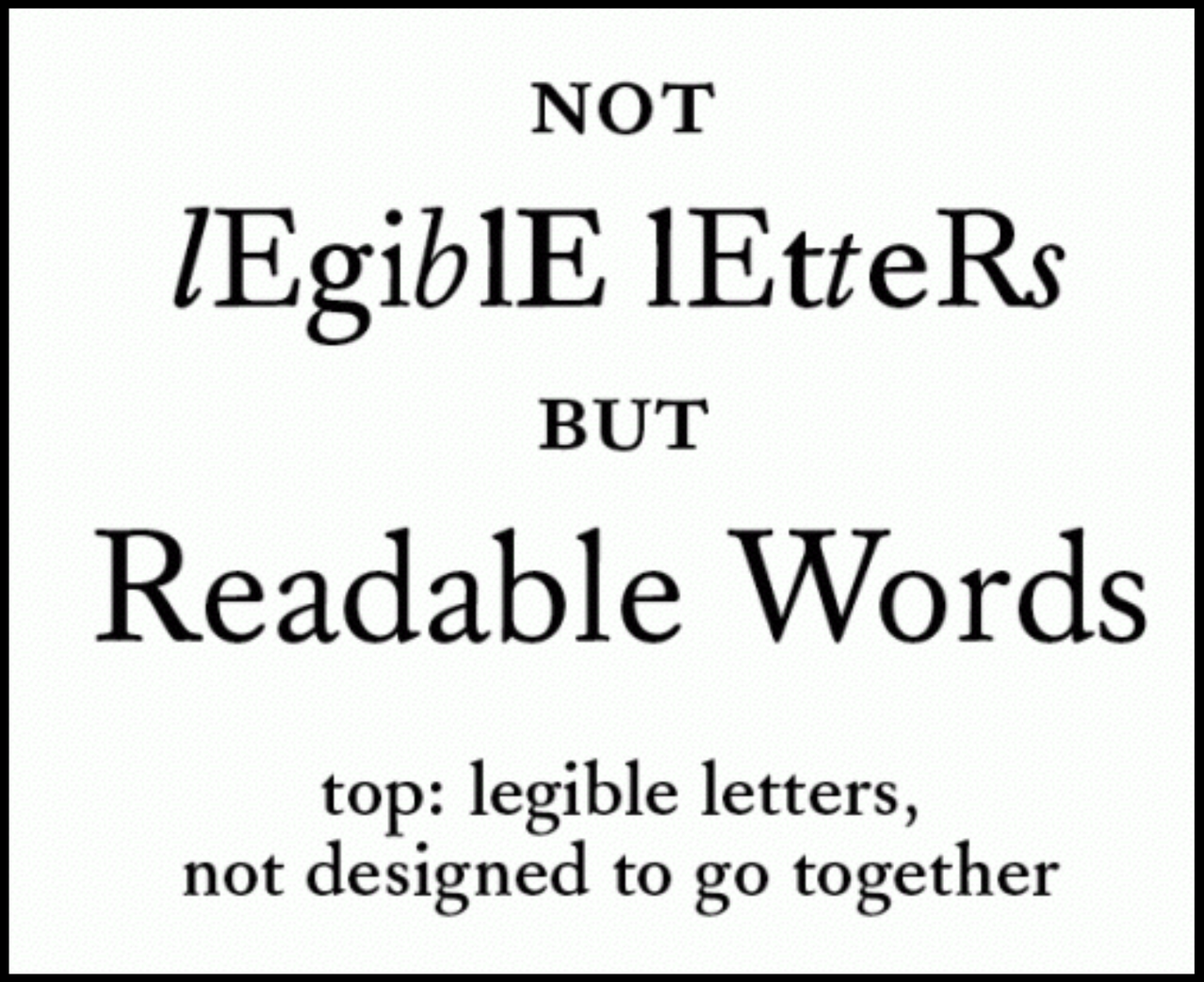


Phil McMinn – p.mcminn@sheffield.ac.uk

Further Objectives

Incorporation of a Language Model for String Generation

S. Afshan, P. McMinn and M. Stevenson. "Evolving Readable String Test Inputs Using a Natural Language Model to Reduce Human Oracle Cost". Proc. ICST 2013



Give better fitness values to strings with character combinations that occur naturally

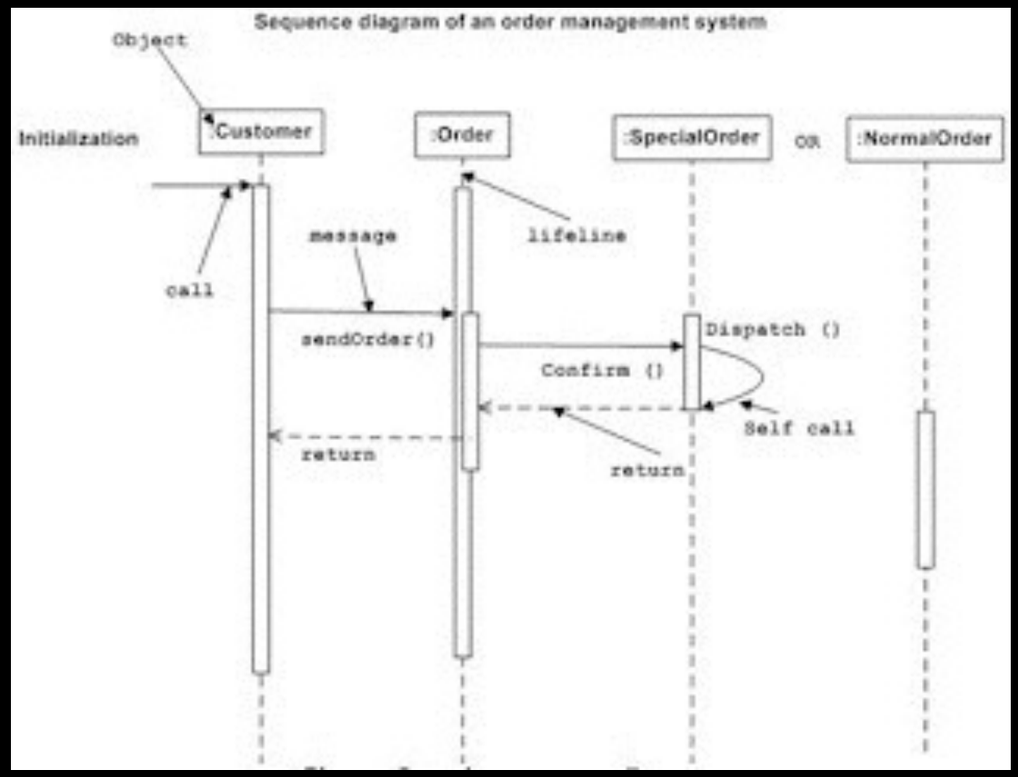
Crowdsourced evaluation

- Candidates evaluated test cases faster and more accurately

Further Objectives

Common call sequence patterns

Gordon Fraser, Andreas Zeller.
Exploiting Common Object Usage in Test Case Generation. Proc. ICST 2011



Give better fitness values to test cases with method call sub-sequences that occur in practice

Other Ideas for Qualitative Reduction

- Readable/understandable outputs
- Generate inputs that more obviously fall into a certain partition of the input/output space such as valid or invalid

Fault-finding capability

Open question:

Does lowering human oracle cost hinder fault finding capability?

Comments to improve test case understanding

Current

```
public void testExe4()  throws Throwable  {

    Triangle triangle0 = new Triangle();
    int int0 = 249;
    int int1 = 911;
    int int2 = 911;

    int int3 = triangle0.exe(int0, int1, int2);

    assertEquals(3, int3);
}
```

Comments to improve test case understanding

Ideal

```
public void testIsoceles() {
    Triangle triangle = new Triangle();
    int a = 249;
    int b = 911;
    int c = 911;

    // test for an isoceles triangle
    int type = triangle.exe(a, b, c);

    assertEquals(3, type);
}
```

Comments to improve test case understanding

Possible

(using Symbolic Execution)

```
public void testExe4()  throws Throwable  {

    Triangle triangle = new Triangle();
    int a = 249;
    int b = 911;
    int c = 911;

    // simplified path constraints: c < (a + b) and a < b and b == c

    // This test is similar to the test 'testExe0' except a < b

    // This is the only test that exercises the following condition(s): a < b and b == c

    int type = triangle.exe(a, b, c);

    assertEquals(3, int3);
}
```

Summary

- Generating tests for coverage should not be the only goal
- In order for our tools to be taken seriously we need to generate readable, realistic tests
- Ways to incorporate these using search-based and symbolic execution approaches