CS608
Software Testing

Dr. Stephen Brown

Room Eolas 116

stephen.brown@mu.ie

CS608

Wrapup

(Essentials of Software Testing, Chapter 14)

Course Summary

- Introduction: motivation, exhaustive testing, test heuristics
- BBT: EP, BVA, DT
- WBT: SC, BC, (AP, DU-Pairs, CC, DCC, MCC, MCDC)
- OO: testing in class context (state, inheritance)
- Application: web (desktop, mobile, other systems)
- Test Automation
- Random Testing: oracle, test data, test completion
- Testing in the software process

A Reverse Look at Testing

- We have followed the test development activities in order, each producing outputs for the next activity
- Now we will complement this by considering software testing from the opposite perspective: with a reverse look
- Develop a deeper understanding of ordering of the activities
- Motivating the documentation of the outputs of each step, as a component in the development of high-quality software

A Test Implementation

```
@Test public void test1() {
    TestItem ti=new TestItem();
    AssertEquals( 100, ti.categorise(23, true) );
}
```

- A simple, automated black-box unit test
- Using equivalence partitions (EP)
- To test a method `TestItem.categorise()`
- Three data values: 100, 23, true
- Where do these come from?

Test Cases

ID	TCI Covered	Inputs		Exp. Results
		p1	p2	
test1	a,b,c	23	true	100

- ID (test1) assigned by the tester
- But where do the Input values in the test case (23 and true) come from?
- Where does the Expected Results value (100) come from?
- And where do the TCI Covered identifiers (a, b, and c) come from?
- Again, we examine the previous activity

Test Coverage Items

TCI	Parameter	Equivalence Partition
a	p1	0..100
b	p2	true
c	expected results (return value)	100

- The TCI Covered and data values, shown in the test cases table above, are derived from the test coverage items
- Each test coverage item has a unique identifier (the TCI), and defines an equivalence partition for a parameter

Test Coverage Items

TCI	Parameter	Equivalence Partition
a	p1	0..100
b	p2	true
c	expected results (return value)	100

- The test coverage item identifiers have been assigned by the tester
- The values in the test cases of 23, true, and 100 have been selected by the tester from the equivalence partitions shown
- But where do the three equivalence partitions shown come from
- Again we refer to the previous activity

Test Analysis

Parameter	Equivalence Partition
p1	(*) Integer.MIN_VALUE..-1 0..100 101..Integer.MAX_VALUE
p2	true false

Parameter	Equivalence Partition
Return Value	0 100 101

- Analysis of the software specification identifies the partitions for the input parameters and the output/return value
- The TCI (previous slide) uses the partitions
  - [0..100] for p1
  - [true] for p2
  - [100] for the expected results (return value)
- Where do these parameters, and their associated partitions come from?

Software Specification

public int categorise(int p1, boolean p2)

Determine which group the value p1 is in.

Inputs

p1: the value being analysed

p2: a boolean value (or flag) used to enable/disable the categorisation

Outputs

return value:

0 if p1 is negative

100 is p1 is in the range 0..100 and p2 is true

101 otherwise

Discussion

- For BBT/EP testing, the tester requires in order:
  1. The specification to analyse
     - In the case of white-box testing, the analysis would require the source code as well as the specification
  2. The results of analysing the specification (EPs)
  3. The test coverage items (which define what needs to be tested)
  4. The test cases with test data (which define how to do the tests)
  5. The test implementation (which is executed to perform the tests)

Test Documentation and Level of Detail

- Except for software with high quality requirements, it is unusual to document the outputs of the test design process to the level of detail shown
- Normally, much of the work is done in the testers mind
- It is, however, almost impossible to review a test to ensure it is correct without some documentation of the analysis, test coverage items, test cases, and test data
- Whether the outputs are written down or not, the tester must perform the same work
- While learning how to test, it is excellent practice to write down the output of every activity in the process

Some Further Reading Extracts (see the book)

- Testing in the software process: *Software Engineering*, Sommerville
- Standards on software testing: *IEE/ISO/IEC standard 29119*
- Testing techniques: *Introduction to Software Testing*, Ammann & Offutt
- Testing object-oriented software: *Testing Object Oriented Systems: Models, Patterns, and Tools*, Binder
- Integration testing: *Continuous Integration*, Duvall
- Random testing: *New Strategies for Automated Random Testing*, Ahmad
- Testing for language-specific hazards: *More Java Pitfalls*, Daconta et al
- Program proving: *www.pm.inf.ethz.ch/research/viper.html*
- Testing safety-critical software: *NIST programme for Automated Combinatorial Testing for Software*
- Going off-piste: *How Google Tests Software*, Whittaker et al

Topic: Random Testing

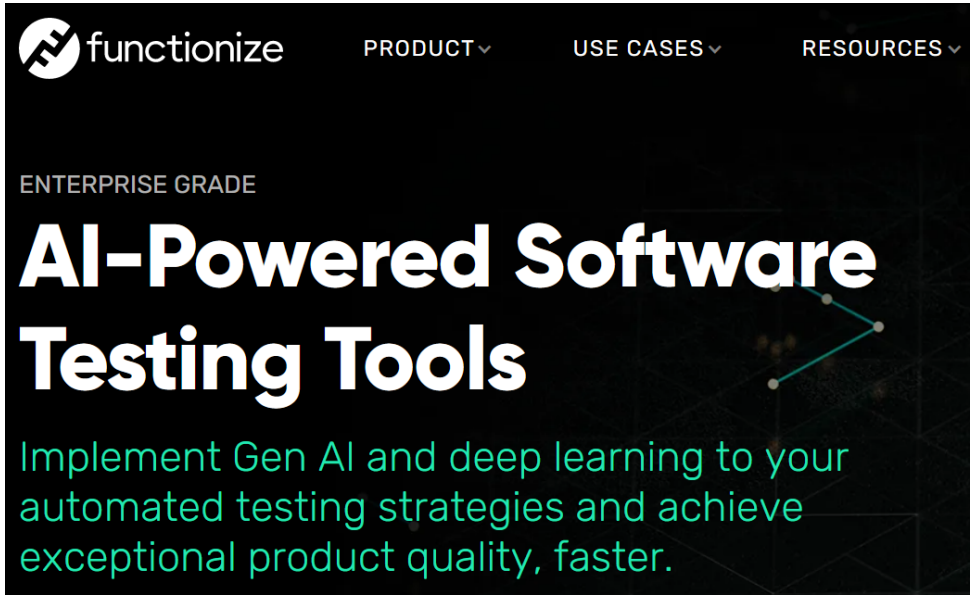
- There is no up-to-date, comprehensive textbook covering the principles of random software testing
- Introduction to the research in *New Strategies for Automated Random Testing* (Ph.D. Thesis, Ahmad)
- *DART: Directed Automated Random Testing* (Godefroid)
- *A Survey on Adaptive Random Testing* (Huang)
  - ART adapts the test input based on the behaviour of previous tests
- Random stability testing for android applications (“Monkey Testing”)
  - *An Empirical Comparison between Monkey Testing and Human Testing* (Mohammed)
  - *Sapienz: multi-objective automated testing for Android applications* (Mao)

Topic: Language-Specific Hazards

- Most programming languages are a compromise between simplicity and features
- This leads to programming mistakes which are language specific as the designers of every language make different decisions about these compromises
- For example, the same statement in different languages may have different effects, and a programmer moving to a new language is likely to make mistakes
- The tester should understand these hazards in order to write language-specific tests
  - *More Java Pitfalls* (Daconta *et al*) – identifies many programming constructs likely to cause faults in the code
- "Anti-Pattern": practices which are not recommended
  - *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis* (Brown, Malveau, McCormick and Mowbray) – a good introduction to the topic for the tester

Topic: AI in testing

- Github Co-pilot
  - Assist in writing tests
  - Provide code snippets
  - e.g. https://www.hatica.io/blog/github-copilot-for-software-test/
- Commercial tools
  - Example:



Testmonitor.com

Will AI Replace Testers?

As these points showcase, it's undeniable that AI is influencing many aspects of our professional lives, including software testing. So questions like, "Will AI replace testers?" are fair.

That's why the TestMonitor team believes in addressing it straightforwardly. Here's our stance:

> *While AI can automate specific testing aspects, it cannot entirely replace human testers. The cognitive skills, creativity, problem-solving abilities, and emotional intelligence that human testers bring to the table are irreplaceable.*

AI Approaches Compared: Rule-Based Testing vs. Learning

There are two main approaches to implementing AI in software testing: rule-based techniques and machine learning techniques.

How to test learning systems

The question that plagues most of us testers in the era of AI is, “Do we need to change our current test approaches to test learning systems?” In short, the simple answer is, “Yes, but only slightly!” Why is that?

2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)

The Impact of Artificial Intelligence on Software Testing

Hussam Hourani
Faculty of Science and IT
Al Zaytoonah University of Jordan
Amman, Jordan
hussam.hourani@gmail.com

Ahmad Hammad
Faculty of Science and IT
Al Zaytoonah University of Jordan
Amman, Jordan
ahmad.hammad94@yahoo.com

Mohammad Lafi
Faculty of Science and IT
Al Zaytoonah University of Jordan
Amman, Jordan
lafi@zuj.edu.jo

***Abstract*—Artificial Intelligence (AI) plays an important role in our life and touch base most of our surrounding applications and systems. A huge amounts of data are created every day from many different sources that need to be monitored and analyzed properly and report results and take actions. A more complex software applications have been built, time is becoming a critical factor to release applications that must be fully tested and comply with Business Requirements. AI plays a key role in Software Testing and can get more accurate results and saves time. This paper discuss the Artificial Intelligence key pillars that can be used in Software Testing. It also open a window on how the future will look like in terms of Artificial Intelligence and the Software Testing. The results show that AI can achieve better results in Software Testing and AI-driven testing will lead the new era of the quality assurance (QA) work in the near future. AI Software Testing will reduce time to market and will increase the efficiency of the organization to produce more sophisticated software and will create smarter automated testing.**

Topic: Program Proving

- The **Viper** toolset, and associated research, supports languages such as: Python, RUST, Java, and OpenCL
  - recommend this as a starting point for the beginner
  - online tutorials and proof tools available:
  - https://www.pm.inf.ethz.ch/research/viper.html
  - https://vercors.ewi.utwente.nl
- The JML language and toolset support the Java language, though not currently the latest version of Java (on windows)
  - http://www.openjml.org
- Spec# was a very promising project by Microsoft to support the C# language, but unfortunately this is discontinued
  - https://research.microsoft.com/en-us/projects/specsharp

Example: condIsNeg()

```
//@ensures \result==(flag && (x<0));

public static boolean condIsNeg(int x, boolean flag) {
   boolean t1=false, t2=false, temp=false;
   if (x<-2047)
      t2 = true;
   else if (x<0)
      t1 = true;
   if (!(x>=-2040)) // should be -2047 for this algorithm
      temp = t2 && flag;
   else if (!(x>=0))
      temp = t1 && flag;
   return temp;
}
```

Example: fail()

Ensure JML is working

```
//@ensures \result==true;
public static boolean fail() {
    return false;
}
```

giveDiscount()

Status giveDiscount(long bonusPoints, boolean goldCustomer)

Inputs

bonusPoints: the number of bonusPoints the customer has accumulated

goldCustomer: true for a Gold Customer

Outputs

return value:

FULLPRICE if bonusPoints≤120 and not a goldCustomer
FULLPRICE if bonusPoints≤80 and a goldCustomer
DISCOUNT if bonusPoints>120
DISCOUNT if bonusPoints>80 and a goldCustomer
ERROR if any inputs are invalid (bonusPoints<1)

Status is defined as follows:

```
enum Status { FULLPRICE, DISCOUNT, ERROR };
```

Example: giveDiscount()

```
// JML Specification

//@ensures (bonusPoints<=0)==>\result==ERROR;

//@ensures ((bonusPoints>0)&&(bonusPoints<=80))==>
  \result==FULLPRICE;

//@ensures ((bonusPoints>80)&&(bonusPoints<=120)&&goldCustomer)==>
  \result==DISCOUNT;

//@ensures ((bonusPoints>80)&&(bonusPoints<=120) &&!goldCustomer)==>
  \result==FULLPRICE;

//@ensures ((bonusPoints>120))==>\result==DISCOUNT;
```

Status giveDiscount(long bonusPoints, boolean goldCustomer)

Inputs

bonusPoints: the number of bonusPoints the customer has accumulated
goldCustomer: true for a Gold Customer

Outputs

return value:
FULLPRICE if bonusPoints≤120 and not a goldCustomer
FULLPRICE if bonusPoints≤80 and a goldCustomer
DISCOUNT if bonusPoints>120
DISCOUNT if bonusPoints>80 and a goldCustomer
ERROR if any inputs are invalid (bonusPoints<1)

Status is defined as follows:

```
enum Status { FULLPRICE, DISCOUNT, ERROR };
```

Example: Round Robin Method

```
int findNextSetWrap(int s, boolean[] x) throws Exception
{
    // Error checks here removed…
    int temp=s;
    for (int i = 1; i<x.length; i++) {
        temp++;
        if (temp==x.length) temp = 0; // should be 0
        if (x[temp]) return temp;
    }
}
```

Round Robin Method: Specification

```
//@   ensures (x.length==0) ==> (\result==-1);
//@   ensures (\result>=-1) && (\result<x.length) && (\result!=s);
//@   ensures (\forall int i; (0<=i) && (i<x.length); !x[i]) ==> (\result==-1);
//@   ensures (\result>=0)==>x[\result];
//@   ensures (\result>s)==>(\forall int j; (s<j) && (j<\result); !x[j]);
//@   ensures (\result >= 0 && \result<=s)==>(\forall int j; 0<=j && j<\result; !x[j]) && (\forall int k; (s<k) && (k<x.length); !x[k]);

int findNextSetWrap(int s, boolean[] x) throws Exception {
   // Error checks here removed…
   int temp=s;
   for (int i = 1; i<x.length; i++) {
      temp++;
      if (temp==x.length) temp = 0; // should be 0
      if (x[temp]) return temp;
   }
}
```

Round Robin Methods: Loop Invariants

```
//@ loop_invariant (i>=1) && (i<=1+x.length);
//@ loop_invariant temp == (i<=x.length-s ? s+i-1 : s+i-1-x.length);
//@ loop_invariant (i<=x.length&&temp>=s)==>(\forall int j; (s<j) && (j<=temp); !x[j]);
//@ loop_invariant (i>1 && temp<=s)==>(\forall int j; 0<=j && j<=temp; !x[j]);
//@ loop_invariant (i>1 && temp<=s)==> (\forall int k; (s<k) && (k<x.length); !x[k]);
//@ decreases x.length +1 - i;
for (int i = 1; i<x.length; i++) {
   temp++;
   if (temp==x.length) temp = 0; // should be 0
   if (x[temp]) return temp;
}
```

Research Topics

- Many conferences (list in book) – identifying research fields:
  - Search-Based Software Testing
  - Mutation Analysis
  - Regression Test Reduction
  - Model-Based Testing
  - Automated Software Verification
  - New Technologies: AI, etc.
  - Software Process and Tools

Assessment

- Continuous Assessment: 20%
- Written Exam (on campus): 80%
- See Library EXPERT web page for previous exam papers
- 2023 and 2024 are best examples (use IEEE terminology)
- Previous years papers still relevant, but may reflect different styles
- Pre-2022 papers include some non-IEEE terminology

The Exam

- 4 questions do 3
- Similar to previous years (see library website)
- Differences:
  - IEEE notation:
    - Old papers refer to *test cases* where we refer to *test coverage items* this year
    - Old papers refer to *tests & test data* where we refer to *test cases* this year
  - No CFG's or ALL PATHS testing or CC/DC/DCC/MCC/MCDC/D-U Pairs
  - Note: 2020 and 2021 were open book exams

Quick Review of Last Year's Paper

CS608 SUMMER 2024

Today's Optional Lab

- Lab 11 – Program Proving
- Experimental, new lab – read instructions in zipfile carefully
- Your task:
  - Try inserting various faults into xxx.java
  - And see JML prover finds them