

Software Testing

POLET Simon

29-06-2019

Table des matières

1	What is testing ?	3
1.1	Introduction	3
1.2	Equivalence tests	3
1.3	Creating Testable Software	4
1.4	Assertions	4
1.4.1	Definition	4
1.4.2	Using assertions	4
1.4.3	Disable assertions	4
1.5	Kinds of software	4
1.6	Defensive Coding	5
1.7	Fault Injection	5
1.8	Non-Functional Input	6
1.9	Kinds Of testing	6
1.9.1	White Box Testing - Black Box Testing	6
1.9.2	Unit Testing - Integration Testing - System Testing ¹	6
1.9.3	Different kinds of testing ²	6
2	Coverage Testing	7
2.1	Score of the Test	7
2.2	Coverage	7
2.3	Domain Partitioning and not covered elements	8
2.4	Automated White Box Testing	8
3	Random Testing	9
3.1	Driver Script	9
3.2	Input validity	10
3.3	Fuzzing	11
3.4	Random Testing of API's - Fuzzing File Systems	11
3.4.1	First Issue	11
3.4.2	Second Issue	12
3.5	Ways of Generating Random Inputs	13
3.6	Oracles	14
4	Random Testing in Practice	15
4.1	Random Testing in the Bigger picture	15
4.2	How Random Testing Should Work	15
4.3	Tuning Rules and Probabilities	16

1. a.k.a. **Validation Testing**. It's use in a paradigm of black box testing.

2. There is also **Regression Testing**

4.4	Fuzzing Implicit Inputs	16
4.5	Can Random Testing Inspire Confidence?	17
4.6	Tradeoffs in Random Testing	17
5	Testing in Practice	18
5.1	Overwhelmed by a Good Random Tester	18
5.2	Test Case Reduction/Minimization	19
5.3	Reporting Bugs (to an open source project)	19
5.4	Building a Test Suite	20
5.5	Hard Testing Problem	20
6	Summary	21
6.1	Chapter 1	21
6.2	Chapter 2	23
6.3	Chapter 3	24
6.4	Chapter 4	25
6.5	Chapter 5	26
A	Fixed-Size Queue	27
A.1	Ex : Test cases on Fixed-Size Queue	29
B	SplayTree	31
B.1	Ex : Black Box Testing	34
C	Luhn Algorithm	35
C.1	Pseudorandom Generation	35
D	Examples	36
D.1	Ex : Read_all Function	36
D.2	Ex : "Babysitting an Army of Monkeys" - Charlie Miller	37
D.3	Bitwise	38
E	Problem Sets	39
E.1	BlackBox Testing	39
E.2	Coverage	41
E.2.1	Queue Coverage	41
E.2.2	Splay Tree Coverage	44
E.3	Sudoku	48
E.3.1	Sudoku Checker	48
E.3.2	Sudoku Checker Bis	54
E.3.3	Sudoku Randomizer	58
E.3.4	Sudoku Solver	59
E.3.5	Sudoku Solver Bis	66
E.3.6	Sudoku Randomizer	72
E.4	Fuzzer	73

Chapitre 1

What is testing ?

1.1 Introduction

Software testing is a discipline of computer science. The subject she deals with is the study of technics to find and fix problems in a program. Testing a program in the whole have most of the time an insolvent complexity. So, *software testing* test and resolve sub-problems and it helps to become a better tester.

A software under test is tested by different inputs with expected outputs. If you find a bug, you have to check what kind of bug it is.

- Bug in S.U.T
- Bug in acceptability test
- Bug in specification
- Bug in OS, compiler, libraries, hardare

1.2 Equivalence tests

When we are testing our code, we produce a single input and receive a single output. To make our tests relevant, we have to define *equivalences classes of test cases*.

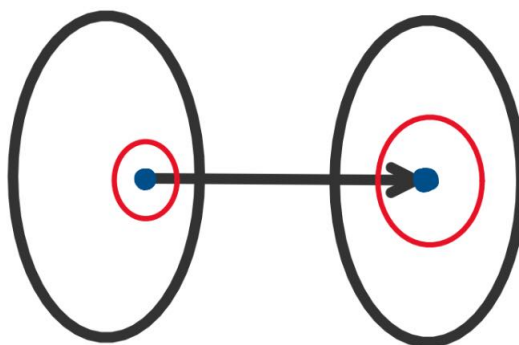


FIGURE 1.1 – Classes d'équivalence d'un input donné vers un output reçu

1.3 Creating Testable Software

Before testing our software, we have to produce it and be sure that it will be testable. Here are some tips to help having a testable software :

- Writing clean code
- Knowing what each module does
- Knowing interactions between modules
- No extra thread
- Avoid global variables
- Avoid pointer soup
- Having unit test for modules
- Adding support for fault injection
- Writing a large number of assertions

Rem : Specification is the most important for the developer because it defines what is acceptable.

1.4 Assertions

1.4.1 Definition

Assertion : Executable check for a property that must be True.

1.4.2 Using assertions

Assertions are used to check the domain of the error. Thanks to that we can blame the correct module. Making our code self checking with assertions lead to more effective testing because we make the code fail closer to the bug. We usually use assertion for checking *document assumptions*, *preconditions*, *postconditions* and *invariants*.

1.4.3 Sous section

Rem : We never use assertions for error handling and we have to avoid making assertions with side effect.

1.4.4 Disable assertions

Advantages	Disavantages
The code is running faster The code keep going	The code could relies to a side effecting assertion

1.5 Kinds of software

Operating System When we are testing something like an operating system kernel, we want to test basically with all possible values of the parameters. It's because we want to have an isolation between the user and the operating system to enforce security.

Graphical User Interface For a GUI, we have to check every single input (clicks, keyboard, events, swipes, ...) which determine a GUI application state.

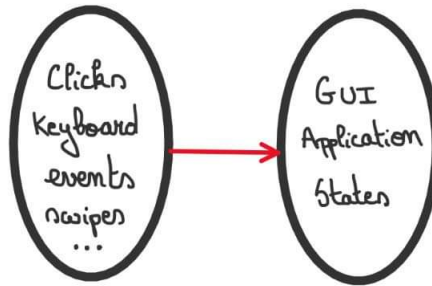


FIGURE 1.2 – GUI Application range and domain

PRINCIPLE : Interfaces that span trust boundaries are special and must be tested in the full range of representable values.

Web Browser A web browser can't control what the OS does. We have to check if it works correctly. Furthermore, a web browser have to check the time at which input arrive.¹

1.6 Defensive Coding

Defensive Coding : It's a form of defensive design intended to ensure the continuing function of a piece of software.

Defensive programming practices are often used where high availability, safety or security is needed.²

1.7 Fault Injection

Fault Injection : Technique for improving the coverage of a test by introducing faults to test code paths, in particular error handling code paths.

1. If the data arrives at a short window of time, the data will be rendered as a web page but if the date comes from the network is scattered across to much time, it wil result a time-out.

2. You can not trust your team mates because you can't even trust yourself.

Ex : In Python, `f = open("tmp/foo", "w")` can have errors that can be difficultly determine because it will almost always succeed. So, we can create `my_open("tmp/foo", "w")` with a implementation almost identical to handle the bug.

1.8 Non-Functional Input

Non-Functional Input : Inputs that have nothing to do with the API provided by the software that we are testing and nothing to do with the API that are used by the software that we are testing.

Ex : Switching between different threads of execution in a multi-threading software under test³. It's difficult to solve because the code is depending of the schedule of these threads and bugs in the software under test can either be concealed or revealed. This is not under the control of the application but of the operating system.

1.9 Kinds Of testing

1.9.1 White Box Testing - Black Box Testing

White Box Testing : Structural tests focus on the implementation.
Black Box Testing : Functional tests focused on the specification.

1.9.2 Unit Testing - Integration Testing - System Testing⁴

Unit Testing : Testing a module in an isolation fashion.
Integration Testing : Taking multiple software modules that have already unit tested in combination with each other.
System Testing : Test if the system as a whole meets his goal.

The goal of unit testing is to find defects in the internal logic in the software under test as early as possible to make them as robust as possible. On the other size, we must verify in integration testing that we have define and specify tightly enough for different groups of people implementing the different modules were able to make compatible assumptions which are necessary for software modules to all work together. For system testing, we are concerned about how our system will be used. We won't test our system for all possible input but rather make sure that it performs acceptably for important use cases.

1.9.3 Different kinds of testing⁵

Differential Testing Testing the same input across different implementations of the software under test and comparing them for equality.

3. It's called context switching.

4. a.k.a. **Validation Testing**. It's use in a paradigm of black box testing.

5. There is also **Regression Testing**

Stress Testing Test at or beyond the normal limits of a software⁶.

Random Testing Using the result of a random number generator to randomly create test inputs and deliver them to the software under test.⁷

6. Usually use to increase the robustness and the reliability of the software under test.

7. Usually used to find corner cases in software systems and the crashme program for Unix Kernel.

Chapitre 2

Coverage Testing

2.1 Score of the Test

When we are testing our code, we want to know how good is our testing. So, we use an automatic tool which tell us where our testing strategy is not doing a good job.

Advantages	Disadvantages
Gives an objective score coverage < 100% \Rightarrow meaningful tasks	Don't find errors of omission 100% coverage \neq all bugs were found

2.2 Coverage

Test Coverage : Measure of the proportion of the program executed during testing.

Test coverage is an automatic way of partitioning the input domain with some observed features of the source code.

Function Coverage Every function in the code is executed during testing.

Statement Coverage Every statement in the code is executed during testing.

Line Coverage Every physical line in the code is executed during testing.

Branch Coverage¹ Every branch in the code is executed in both way during testing.

Loop Coverage Every loop in the code is executed 0 times, once, and more than once during testing.

Modified Condition/Decision Coverage² It's an hybrid which use branch coverage techniques and must takes on every possible input. Furthermore, every condition independently must affect the outcome of a decision.³

1. a.k.a **Decision Coverage**

2. This kind of coverage must be done if we are testing a highly critical system.

3. If we have conditionals that don't independently affect the outcome of a decision, there is a programming mistake which means that somebody didn't understand what was going out.

Path Coverage Every path in the code is executed during testing.

Boundary Value Coverage⁴ Each boundary value in the code must be checked during testing.⁵

Synchronisation Coverage Ensure that during testing the locks actually does something.⁶

2.3 Domain Partitioning and not covered elements

With coverage metrics, we don't cover the whole domain. There are kinds of bugs that don't directly depend of the implementation of the code⁷. There are multiple ways of partitioning the input domain for purpose of testing. No only with an automated coverage, we can also partitioning the domain based on the specification.

When you are covering a code but a part of the code isn't covered, it can means 3 things :

- The code is infeasible⁸
- The code is not worth covering⁹
- The test suite is inadequate

2.4 Automated White Box Testing

Some tools are able to make automated white box test based on the code we produce. For example with path coverage, it will start with random parameters then, will check about constraints to check a paths.¹⁰

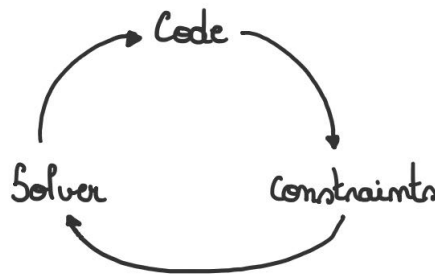


FIGURE 2.1 – Cycle of the automated test

4. Most of the programming errors are boundary errors by one.

5. It's more difficult to test variables that are in interaction because we multiply the number of boundary values to test.

6. We usually use it with **Stress Coverage** to detect bugs and **Interleaving Coverage** which is a concurrency-specific coverage metric when you recall functions which accessed shared data are called in a concurrent fashion that is by multiple threads at the same time.

7. For example, bugs can depend of the device with a full hard disk or something else. It can be resolve for example with **fault injection**

8. It's not necessary bad for example in a checkRep function pieces of code not covered means everything is right. We notify this to the tool with `# pragma : no cover`

9. It's about things that was already tested elsewhere and the test of this would abort the program.

10. For the language C, [Klee](#) is a good automated white box testing tool.

Chapitre 3

Random Testing

3.1 Driver Script

Oracle : Determine whether the output of the software under test is either good or bad.

The *driver script* is the loop making random testing. It works with a random test case generator that takes inputs from a *Pseudo-random number generator* feeded by seed which determines the sequence of random numbers generated. Tests cases are submitted to the software under test¹ and inspected by a test oracle and if the output is not OK, we have to save the test case.

Rem : It's important to have significant domain knowledge to create good random test case generator.

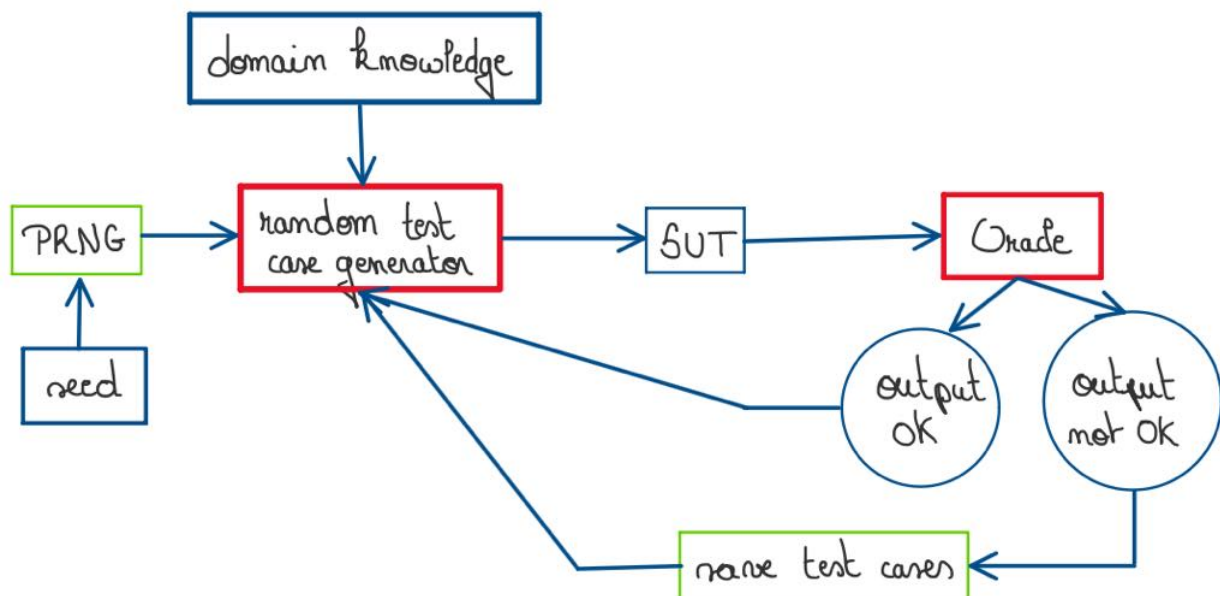


FIGURE 3.1 – Cycle of the driver script

1. The software under test will fail in an obvious way if the programmer included a lot of assertions in the S.U.T.. Those assertions can be annoying as compiler uses because the compiler crashes but are actually very good for random testing and we want them to be there.

3.2 Input validity

A key problem of building a random test case is generating inputs that are valid. That is to say generating inputs that are part of the input domain for the software under test. If we take an invalid input, it's going to be mapped to a part of the output space that correspond to a malformed or crashed output.

Using invalid inputs in random testing is just spinning, the result is boring output because the input is rejected. If the input isn't rejected, it's because the software under test failed to contain sufficient validity checking logic and it results misbehaviours, crashes, ...²

Interfaces that don't spend trust boundaries often lack full *validity checking* because of performance or maintainability reasons.³

Rem : There exists software for which it is impossible to construct a *validity checker*.

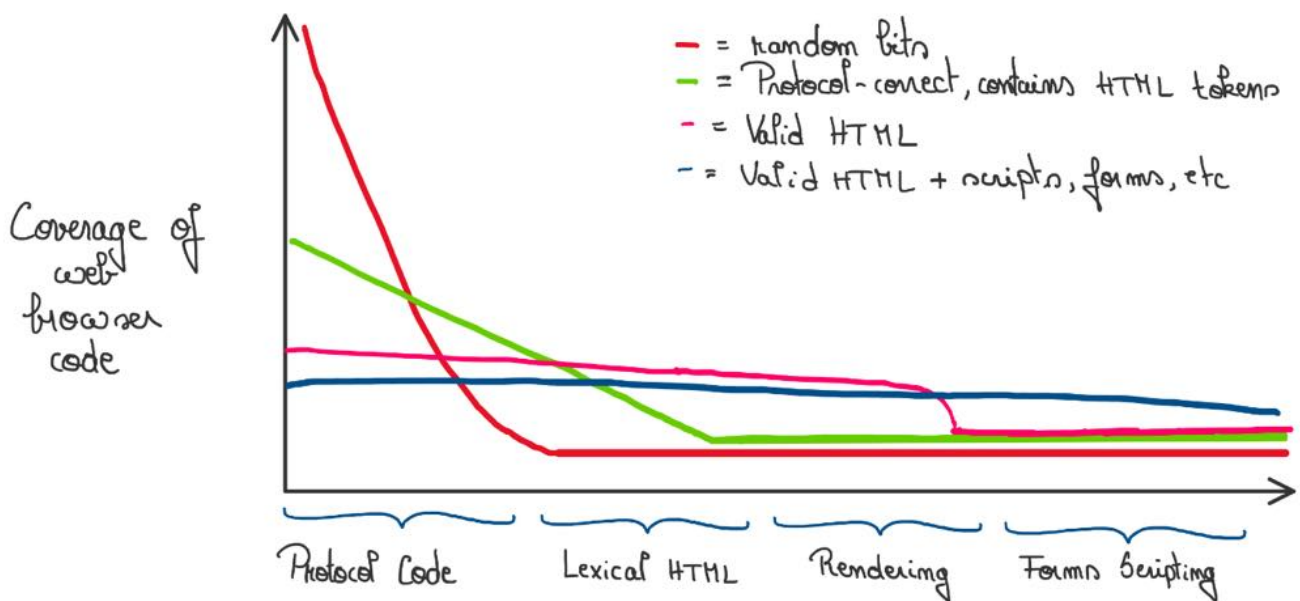


FIGURE 3.2 – Input Validity of a Web Browser Request

2. The problem is that it takes a lot of developer time for nothing.

3. Sometimes we have to trust somebody, otherwise, we can't get a new software written.

3.3 Fuzzing

Fuzzing⁴ : Automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a computer program.⁵

It's used for finding vulnerabilities in applications. The goal is to found bugs in the service that are going to let us mount an attack such as a denial of service or some sort of an exploitable vulnerability that will let us mount an intrusion on that service.⁶

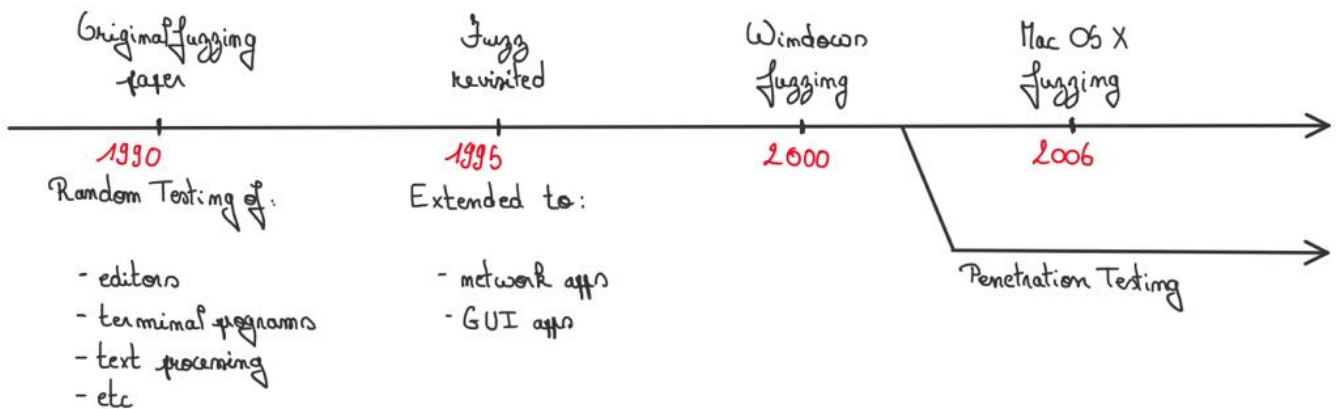


FIGURE 3.3 – Fuzzing Timeline

3.4 Random Testing of API's - Fuzzing File Systems

The major thing creating difficulties in random testing is the structure requires inputs. With API's it's just a collection of functions that can be invoked.⁷ It's more complicated because there are dependencies among API calls. A second issue is that our test are going to become quite large. We have to find how to represent them.

Ex : The file system has to efficiently respond to all sorts of calls that perform file system operations and it's really important that it works well.

3.4.1 First Issue

If we want to do effective random testing of a file system, we need to track dependencies at least in some rudimentary fashion in order to issue a sequence of API calls that's going to do reasonably effective random testing of a file system.

4. It's practically the same thing as random testing.

5. The methodology is basically generating random garbage and not really worrying about the input validity problem.

5. **We can also use our human expertise or human knowledge of the domain interest in order to generate manual test cases that aren't boring.**

6. a.k.a. **Penetration Testing**.

7. They are called with Strings that are sequences of instructions.

3.4.2 Second Issue

Ex : The test cases exist as ephemeral objects in memory. The driver code creates the test cases, passes them to the software under test, looks at the results and keep going until it finishes.

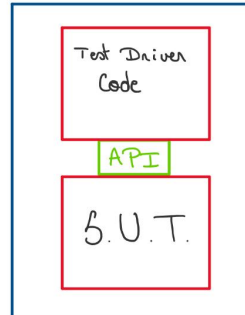


FIGURE 3.4 – Representation of Random Testing in a Python Program

Rem : There are couple of cases in which that's no so good where we'd like to save off that test case for later use in regression testing. Then we can parse files loaded and saved to create test case in memory or print them to disk in order to save test case. It facilitate regression testing and it performs test case reduction.

Testcase Reduction : Piece of technology that's very often combined with *random testing*. It takes a large test case that make the software under test fail and turns into smaller test case that still make the software under test fail.

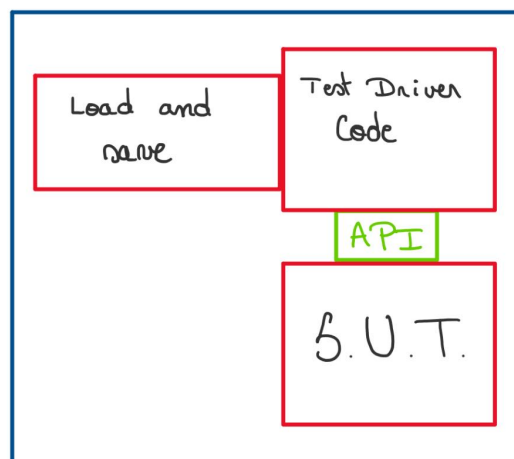


FIGURE 3.5 – Representation of Random Testing in a Python Program

3.5 Ways of Generating Random Inputs

Generative Random Testing Inputs are created from scratch based on the domain knowledge.⁸

Mutation-Based Random Testing Inputs are created by randomly modifying non-randomly created inputs to the software under test. It clusters the domain of input and will start with some known input and randomly modify it in a kind of in the same neighbourhood as the original input.⁹

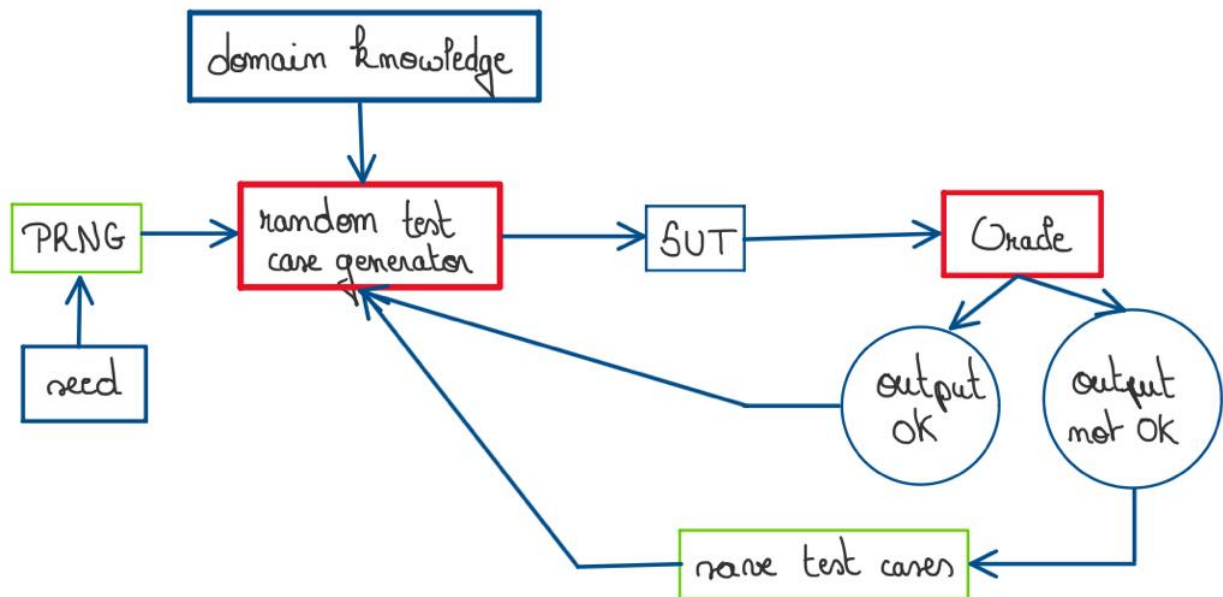


FIGURE 3.6 – Cycle of the driver script

8. It's better in ferreting out weird errors but it takes more time to implement.

9. It's limited to exploring sort of region of the input space that's close to the starting point.

3.6 Oracles

Oracles are extremely important for random testing because if we don't have an automated oracle that tells if a test case did something interesting, we've got nothing.

Weak Oracles Detect whether or not an application crashed. Which means that the software under test violated some rule that the hardware, a programming language or an enhanced execution environment imposed.

Medium Oracles Assertion checks that the programmer has put into the software. It provide a more specific application kind of checking than does the weak oracles.

Strong Oracles *Alternate implementation of the same specification¹⁰, function inverse pair¹¹ and null space transformation.¹²*

10. Ex : Queue checker/Differential testing of compilers/ old version of S.U.T./reference implementation.

11. Ex : Assembler-Disassembler / Encryption-Decrypt / Compression-Decompression / Save-Load / transmit-Receive / encode-decode

12. We take a random test case and we make a change that shouldn't affect how its treated by the software under test.

Chapitre 4

Random Testing in Practice

4.1 Random Testing in the Bigger picture

Random testing is effective because of 3 reasons. First, It's based on weak bug hypothesis rather than particular failures. Every test case is a little experiment and the outcome is a bit of *information*. Second, people tend to make the same mistakes while coding and testing and there is a huge asymmetry between speed of computer and people. Even if random tester is mostly generating stupid test, if it can generate a clever test case one in a million time it still more effective than testing resources by hand.

4.2 How Random Testing Should Work

First, we have to randomly test our *modules independently* in the early development to have solid foundations. Second, when we *introduce a module in a more sophisticate piece of software*, we have to use another kind of random tester such as those that come in at the top level and those that perform system-level fault injection. Third, in the *complete product*, we have to focus on the external interfaces provided.

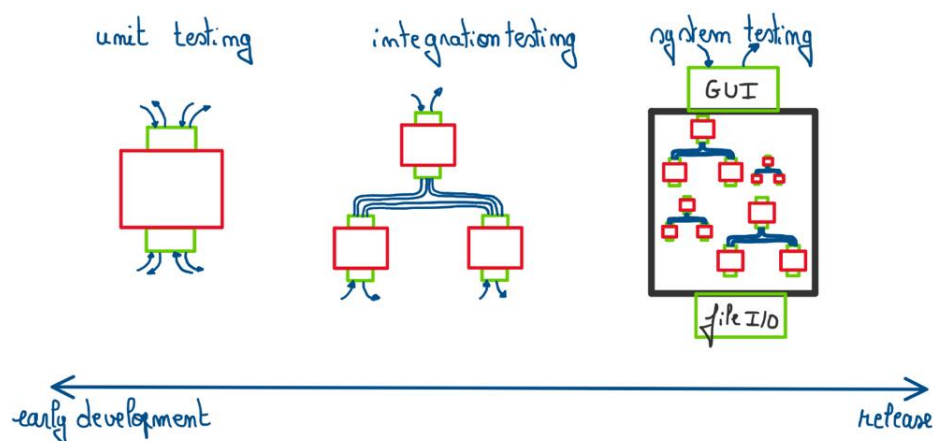


FIGURE 4.1 – Random System Tester

Rem : We have to build the third system-level random tester as early as possible in the development process because it will find some flaws that are not going to flood developers with huge number of bugs. Thanks to that our software will evolve to be more robust.

4.3 Tuning Rules and Probabilities

A question we must ask ourself when we are making random testing is : ***Does this random walk have a reasonable probability of executing all the cases ?***". If we don't have one, we might have to adjust the probabilities.¹

Exemple : Fuzzing the bounded Queue

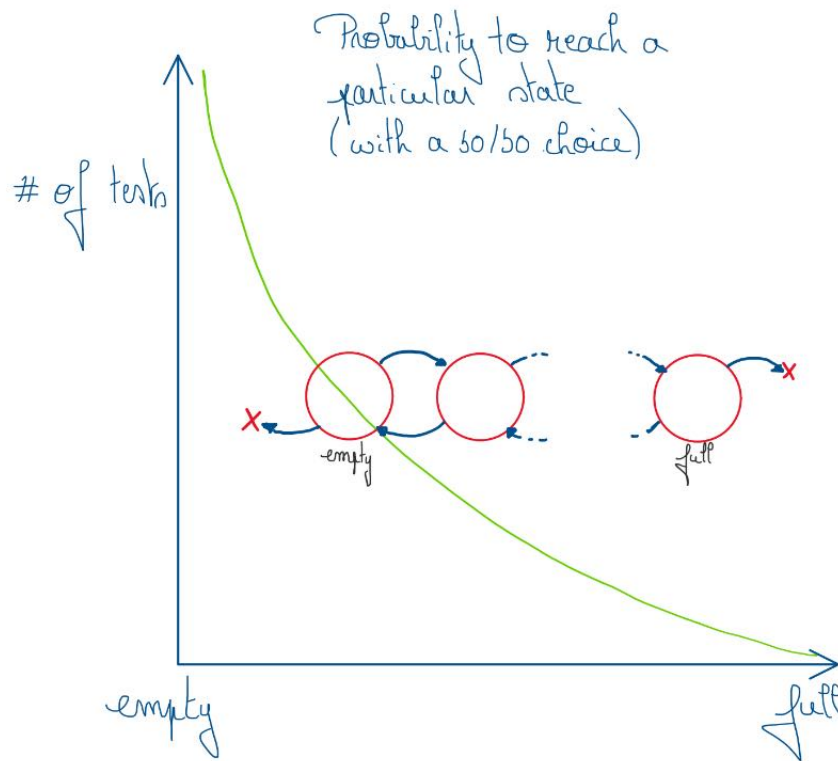


FIGURE 4.2 – Tuning Probabilities

4.4 Fuzzing Implicit Inputs

To fuzz implicit inputs there are different techniques² :

- Perturbing the schedule
 - Generating load by running apps or other applications
 - Generating network activities
 - Using thread stress testing tool
- Inserting delays near synchronisation and access to shared variables
- Using "unfriendly emulators"³

1. In random testing, if we want to do really good job, we need to think about what is that we're testing, how the code is structured and how we're going to execute all the way through it.

2. Ex : The timing at which different threads are on different processors. In that case, the thread scheduler provides a very important form of implicit input to multi-threaded software under test.

3. Emulators especially designed to stress test applications by doing things like invalid cache line, invoking thread schedules in odd ways, ...

4.5 Can Random Testing Inspire Confidence ?

If we have all of the following conditions, random testing can inspire confidence. Otherwise we mustn't only use random testing.



4.6 Tradeoffs in Random Testing

Advantages	Drawbacks
Less test bias and weaker hypothesis about where bugs are	Input validity can be hard
Human cost of testing goes to nearly zero once testing is automated	Oracles can be hard
Often surprise us	No stopping criterion
Every fuzzer finds different bugs	Every fuzzer finds different bugs
	May spend all testing time on boring tests
	May find the same bugs many times
	Can be hard to debug when test case is large and/or make no sense
	May find unimportant bugs

Chapitre 5

Testing in Practice

5.1 Overwhelmed by a Good Random Tester

If we made a good random tester, we possibly can be overwhelmed by too many bugs. In this case, we have to find if all these bugs are caused by the same thing or if they are different kind of bugs.

A first solution is to **pick a bug and report** it and maybe some other failures will go away too. On another side, it's irrelevant which bug it is. Maybe some other failures will go away too. Second, we can use **bug triage**.

Bug Triage : Process by which the severity of different bugs is determined and we start to disambiguate between different bugs in order to basically try to get a handle on which bugs we can report in parallel.¹ **Core Dump** : Recorded state of the working memory of a computer program at a specific time, generally when the program has crashed or otherwise terminated abnormally.

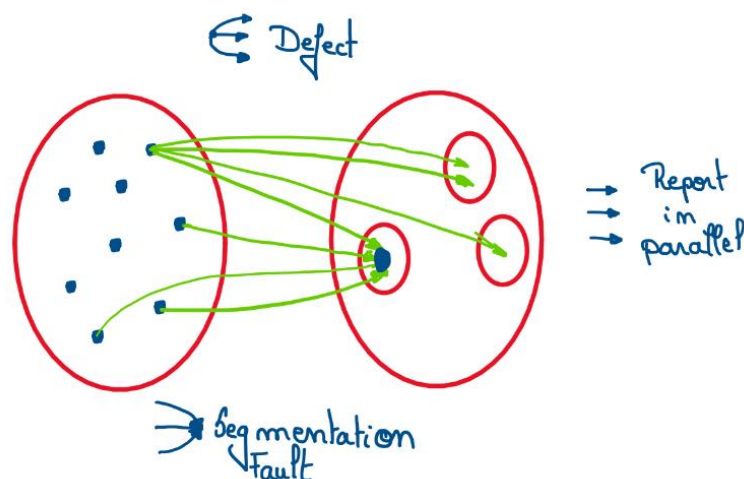


FIGURE 5.1 – Disambiguate by Violation Messages

1. We can disambiguate based on *assertion violation messages*, by *core dump* or *stack trace*, searching over *version history* of the software under test (In a git, use **git bisect** command which do a binary search over revision history in an automated way) or by examine *bug-triggering* test cases.

5.2 Test Case Reduction/Minimization

Looking over a large, randomly-generated test cases is painful, so we have to practice *test case reduction*. We have to eliminate part of the input space¹ and see if the smaller input triggers the test case. If it doesn't we go back to the original test case and try again and if it does we do it again.²

Delta Debugging : Framework that takes a script and takes the test input and automates the process of reduction in a loop which terminates when the delta debugger, which has a bunch of heuristics built in for eliminating parts of the input, can't reduce the input any more.

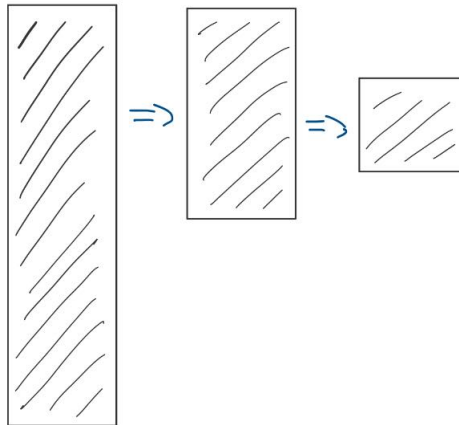


FIGURE 5.2 – Test Case Reduction

5.3 Reporting Bugs (to an open source project)

There are tips to respect when reporting bugs :

- Don't report duplicates
- Respect the conventions
- Include a small, stand-alone test case
- Only report valid test cases
- Indicate what output was expected and what the actual output was
- Give instruction for reproduction
 - Platform details
 - Version S.U.T.

1. Sometimes in a smart way or might be just chop some of it out blindly.

2. We can do that by *manual reduction* or also by *delta debugging*.

5.4 Building a Test Suite

Test Suite : Collection of test which often can be test automatically and that we run periodically.

The goal of test suite is to show that S.U.T. have desired properties ³, that is to say passing over the test⁴.

Regression Test : Any input that cased some previous version to fail.⁵

Rem : It must be a totally separate activity with random testing.

5.5 Hard Testing Problem

There are tips that make testing very hard :

- Lack of / bad specification
- No comparable implementation
- Big S.U.T
- large, highly structured input space
- Non-determinism
- lots of hidden state

3. Ex : Small, feature-specific tests or large, realistic inputs or *regression test*.

4. Especially proving that the S.U.T. don't have severe bugs.

5. A previous bug which occurred in a previous version could come back due to a lack of gotten rid of all the instances of that defect in the source code, an accidental come back to an old version of a file before we fix the bug or it can comes from errors in people's thinking who didn't correct the error in their thinking.

Chapitre 6

Summary

6.1 Chapter 1

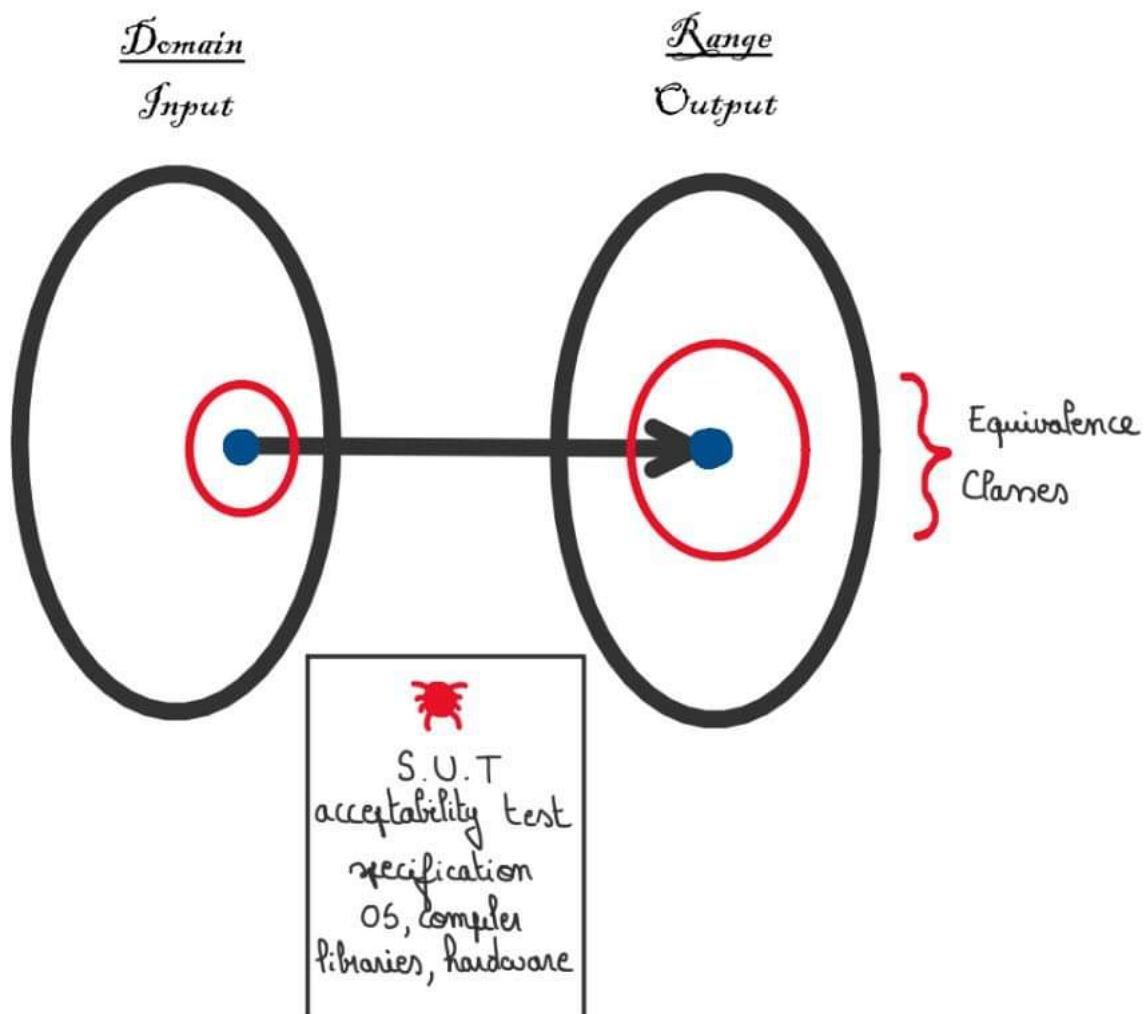


FIGURE 6.1 – Domain and Range of testing

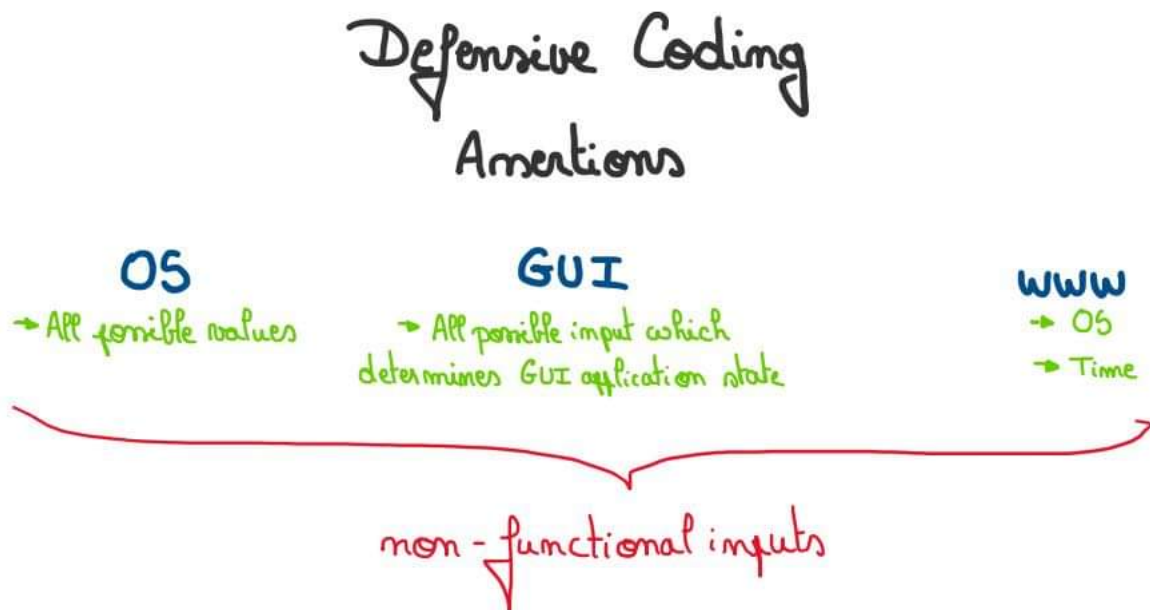


FIGURE 6.2 – Inputs to verify

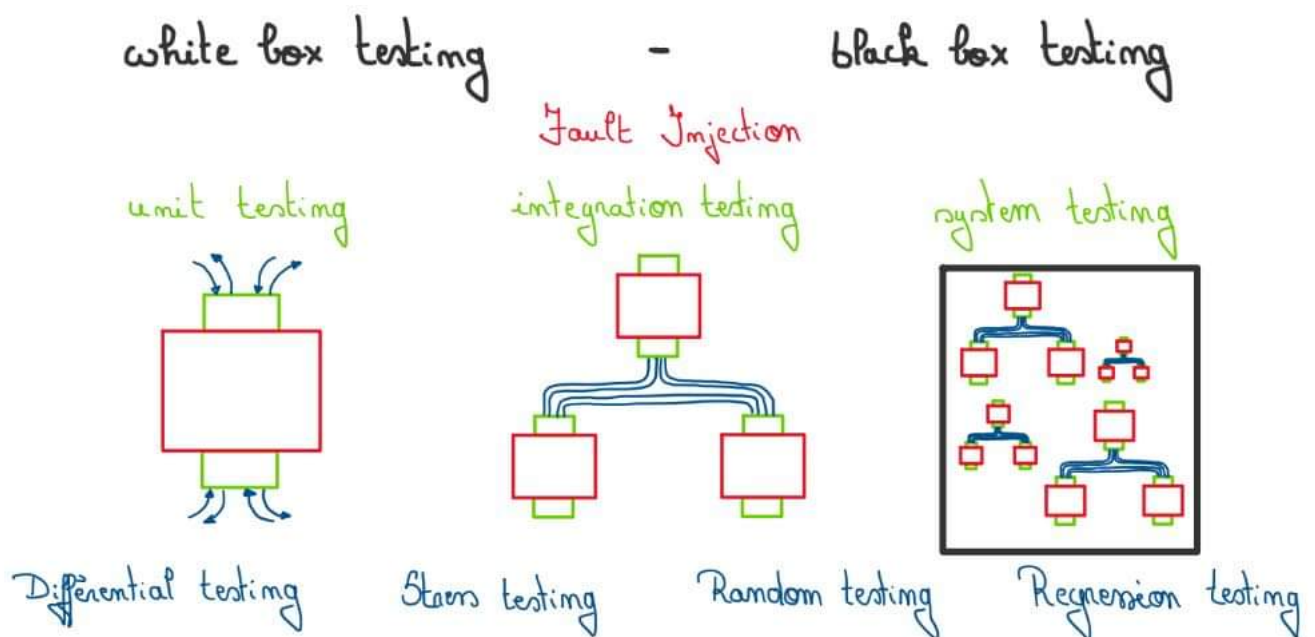


FIGURE 6.3 – Kinds of testing

6.2 Chapter 2

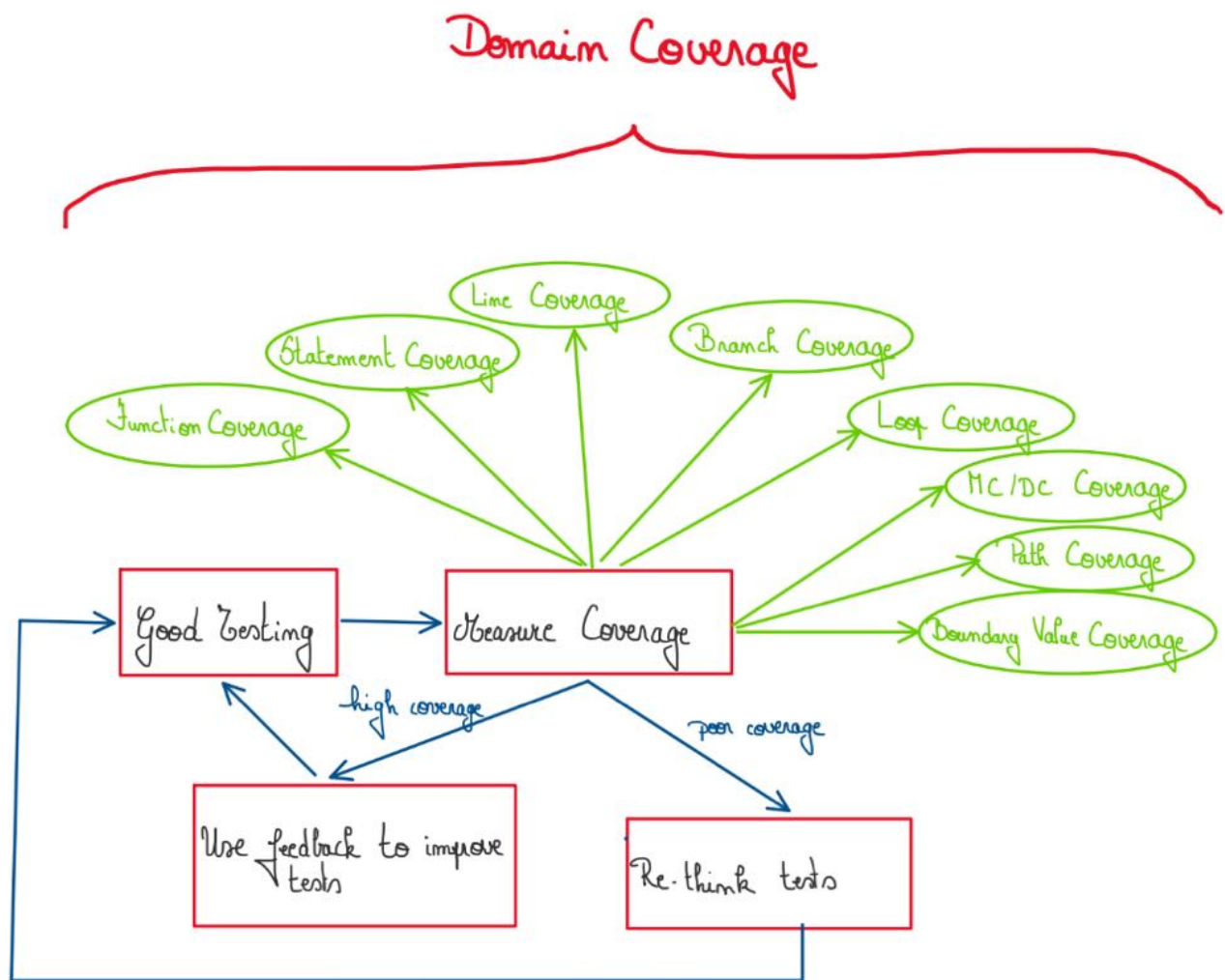


FIGURE 6.4 – Coverage Testing

6.3 Chapter 3

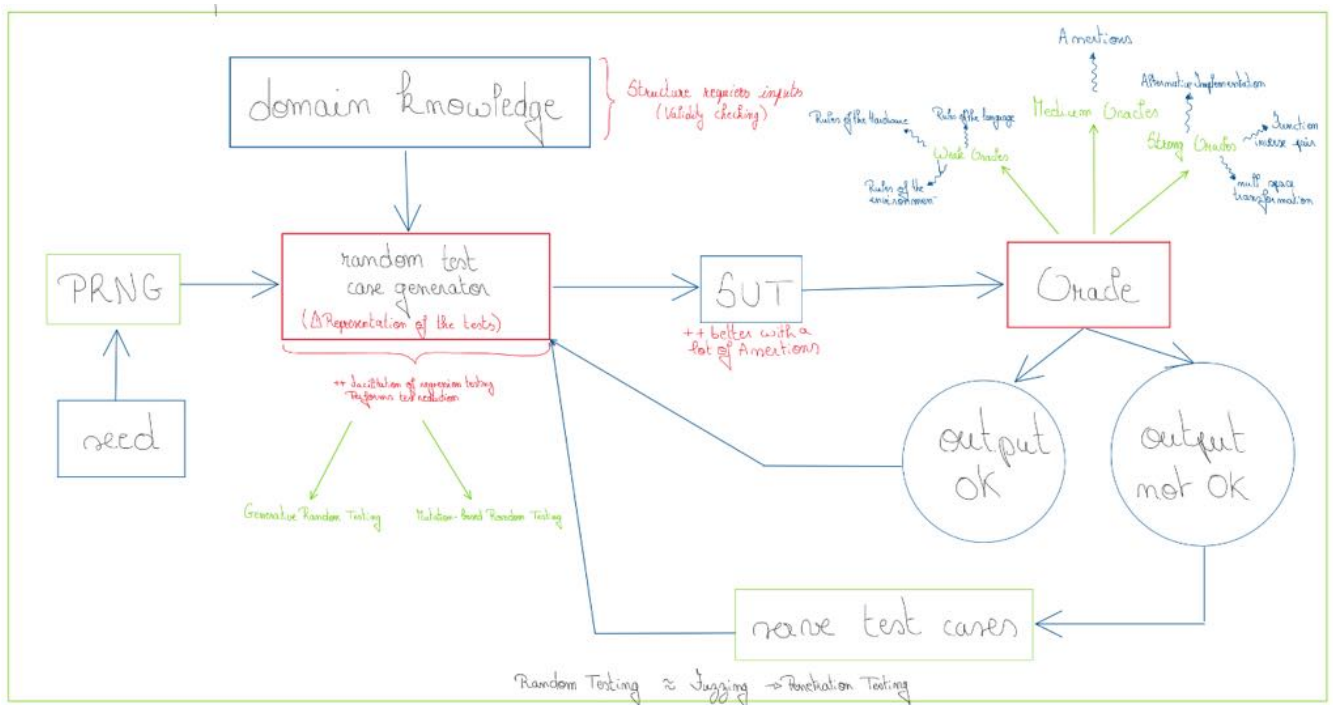


FIGURE 6.5 – Random Testing

6.4 Chapter 4

Does this random walk have a reasonable probability of executing all the cases ?

Well-understood API
+
Small Code
+
Strong Assertions
+
Mature, Tuned Random Tester
+
Good Coverage Result
=
CONFIDENCE

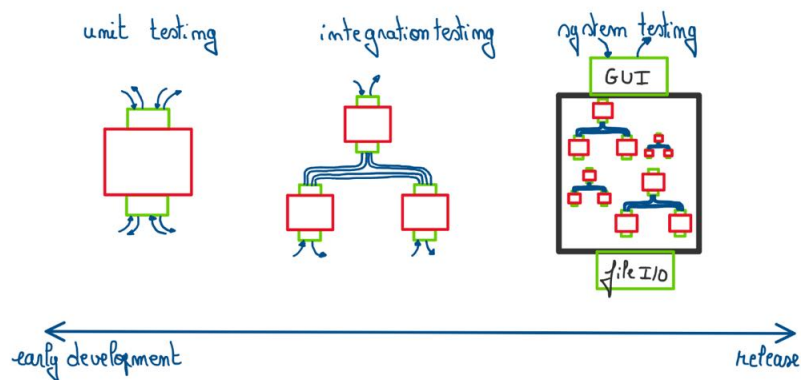


FIGURE 6.6 – Random System Tester

6.5 Chapter 5

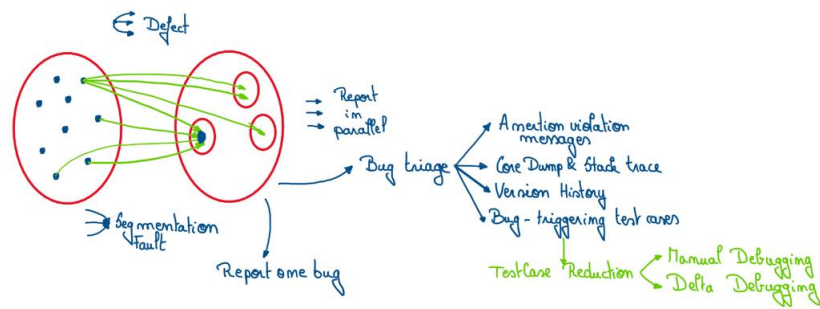


FIGURE 6.7 – Testing in Practice

Annexe A

Fixed-Size Queue

```
1  #-*- coding: utf-8 -*-
2  import array
3  import random
4
5
6  """
7  @overview the Queue class provides a fixed-size FIFO queue of integers.
8  A Queue is mutable.
9  A Queue is a set a values <max, head, tail, size, data> where :
10 @specfield max integer : the maximum size of the Queue
11 @specfield head integer : head of the Queue
12 @specfield tail integer : tail of the Queue
13 @specfield size integer : size of the Queue
14 @specfield data
15 @invariant size >= 0
16 @invariant max >= 0
17 @invariant max >= size
18 """
19 class Queue:
20
21     # FA(c) is a fixed-size FIFO queue of integers
22
23     # IR(c) : size >= 0 &&
24     #         max >= 0 &&
25     #         max >= size &&
26
27     # Constructors
28     """
29     @effects initialize self to a empty of a size_max lenght
30     Queue si size_max >= 0
31     @throws AssertionError sinon
32     """
33     def __init__(self, size_max):
34         assert size_max > 0
35         self.max = size_max
36         self.head = 0
37         self.tail = 0
38         self.size = 0
39         self.data = array.array('i', range(size_max)) # new array of integer
40
41     # Observers
42     """
43     @return returns true if self is empty
```

```

44     """
45     def empty(self):
46         return self.size == 0
47
48     """
49     @return returns true if self is full
50     """
51     def full(self):
52         return self.size == self.max
53
54     # Mutators
55     """
56     @requires x is an integer
57     @modifies self
58     @effects we add x to the end of self
59     @return False if self is full
60         else, return True
61     """
62     def enqueue(self, x):
63         if self.size == self.max:
64             return False # Queue is full
65         # Update meta-data
66         self.data[self.tail] = x
67         self.size += 1
68         self.tail += 1
69         if self.tail == self.max:
70             self.tail = 0 # More efficient to move indice than data in the array
71         return True
72
73     """
74     @modifies self
75     @effects we delete the first element of self
76     @return None if self is empty
77         else, return the first element of self
78     """
79     def dequeue(self):
80         if self.size == 0:
81             return None # Queue is empty
82         # Update meta-data
83         x = self.data[self.head]
84         self.size -= 1
85         self.head += 1
86         if self.head == self.max:
87             self.head = 0 # More efficient to move indice than data in the array
88         return x
89
90     def checkRep(self):
91         assert self.size >= 0 and self.size <= self.max
92         if self.tail > self.head:
93             assert (self.tail - self.head) == self.size
94         if self.tail < self.head:
95             assert (self.head - self.tail) == (self.max - self.size)
96         if self.tail == self.head:
97             assert (self.size == 0) or (self.size == self.max)
98         return

```

./Code/FixedSizeQueue.py

A.1 Ex : Test cases on Fixed-Size Queue

```
1 def test1():
2     q = Queue(3)
3     res = q.empty()
4
5     if not res:
6         print "test1 NOT OK"
7         return
8     res = q.enqueue(10)
9     if not res:
10        print "test1 NOT OK"
11    res = q.enqueue(11)
12    if not res:
13        print "test1 NOT OK"
14    x = q.dequeue()
15    if x != 10:
16        print "test1 NOT OK"
17    if x != 11:
18        print "test1 NOT OK"
19    res = q.empty()
20    if not res:
21        print "test1 NOT OK"
22    print "test1 IS OK"
23
24 def test2():
25     q = Queue(2)
26     res = q.empty()
27     if not res:
28         print "test2 NOT OK"
29         return
30     res = q.enqueue(1)
31     if not res:
32         print "test2 NOT OK"
33         return
34     res = q.enqueue(2)
35     if not res:
36         print "test2 NOT OK"
37         return
38     res = q.enqueue(3)
39     if q.tail != 0:
40         print "test2 NOT OK"
41         return
42     print "test2 IS OK"
43
44 def test3():
45     q = Queue(1)
46     res = q.empty()
47     if not res:
48         print "test3 NOT OK"
49         return
50     x = q.dequeue()
51     if not x is None:
52         print "test3 NOT OK"
53         return
54     res = q.enqueue(1)
55     if not res:
56         print "test3 NOT OK"
```

```
57         return
58     x = q.dequeue()
59     if x != 1 or q.head != 0:
60         print "test3 NOT OK"
61         return
62     print "test3 IS OK"
```

./Code/TestCases_FixedSizeQueue.py

Annexe B

SplayTree

```
1  class Node:
2      def __init__(self, key):
3          self.key = key
4          self.left = self.right = None
5
6      def equals(self, node):
7          return self.key == node.key
8
9  class SplayTree:
10     def __init__(self):
11         self.root = None
12         self.header = Node(None) # For splay()
13
14     def insert(self, key):
15         if (self.root == None):
16             self.root = Node(key)
17             return
18
19         self.splay(key)
20         if self.root.key == key:
21             # If the key is already there in the tree, don't do anything.
22             return
23
24         n = Node(key)
25         if key < self.root.key:
26             n.left = self.root.left
27             n.right = self.root
28             self.root.left = None
29         else:
30             n.right = self.root.right
31             n.left = self.root
32             self.root.right = None
33         self.root = n
34
35     def remove(self, key):
36         self.splay(key)
37         if key != self.root.key:
38             raise "key not found in tree"
39
40         # Now delete the root
41         if self.root.left == None:
42             self.root = self.root.right
43         else:
```

```

44         x = self.root.right
45         self.root = self.root.left
46         self.splay(key)
47         self.root.right = x
48
49     def findMin(self):
50         if self.root == None:
51             return None
52         x = self.root
53         while x.left != None:
54             x = x.left
55         self.splay(x.key)
56         return x.key
57
58     def findMax(self):
59         if self.root == None:
60             return None
61         x = self.root
62         while(x.right != None):
63             x = x.right
64         self.splay(x.key)
65         return x.key
66
67     def find(self, key):
68         if self.root == None:
69             return None
70         self.splay(key)
71         if self.root.key != key:
72             return None
73         return self.root.key
74
75     def isEmpty(self):
76         return self.root == None
77
78     # Error if a key is not in our SplayTree
79     def splay(self, key):
80         l = r = self.header
81         t = self.root
82         self.header.left = self.header.right = None
83         while True:
84             if key < t.key:
85                 if t.left == None:
86                     break
87                 if key < t.left.key:
88                     # Right rotate
89                     y = t.left
90                     t.left = y.right
91                     y.right = t
92                     t = y
93                     if t.left == None:
94                         break
95                 r.left = t
96                 r = t
97                 t = t.left
98             elif key > t.key:
99                 if t.right == None:
100                     break
101                 if key < t.right.key:
102                     # Left rotate

```

```
103         y = t.right
104         t.right = y.left
105         y.left = t
106         t = y
107         if t.right == None:
108             break
109         l.right = t
110         l = t
111         t = t.right
112     else:
113         break
114
115     l.right = t.left
116     r.left = t.right
117     t.left = self.header.right
118     t.right = self.header.left
119     self.root = t
```

./Code/SplayTree.py

B.1 Ex : Black Box Testing

```

1  import unittest
2  from splay import SplayTree
3
4  class TestCase(unittest.TestCase):
5      def setUp(self):
6          self.keys = [0,1,2,3,4,5,6,7,8,9]
7          self.t = SplayTree()
8          for key in self.keys:
9              self.t.insert(key)
10
11     def testInsert(self):
12         for key in self.keys:
13             self.assertEqual(key, self.t.find(key))
14
15     def testRemove(self):
16         for key in self.keys:
17             self.t.remove(key)
18             self.assertEqual(self.t.find(key), None)
19         self.t.remove(-999)
20
21     # Stress testing
22     def testLargeInserts(self):
23         t = SplayTree()
24         nums = 40000
25         gap = 307
26         i = gap
27         while i != 0:
28             t.insert(i)
29             i = (i + gap) % nums
30
31     def testIsEmpty(self):
32         self.assertFalse(self.t.isEmpty())
33         t = SplayTree()
34         self.assertTrue(t.isEmpty())
35
36     def testMinMax(self):
37         self.assertEqual(self.t.findMin(), 0)
38         self.assertEqual(self.t.findMax(), 9)
39
40 if __name__ == "__main__":
41     unittest.main()

```

./Code/TestCases_SplayTree.py

Annexe C

Luhn Algorithm

```
1 def checkLuhn(identifier):
2     sum = identifier[len(identifier) - 1]
3     nDigits = len(identifier)
4     parity = nDigits % 2
5
6     for i in range(nDigits):
7         digit = (int) identifier[i]
8         if i % 2 == parity:
9             digit = digit * 2
10        if digit > 9:
11            digit = digit - 9
12        sum += digit
13    return (sum % 10 == 0)
```

./Code/LuhnAlgorithm.py

C.1 Psëdorandom Generation

```
1 import random
2
3 def generate(pref, l):
4     nrand = l - len(pref) - 1
5     assert nrand > 0
6     n = pref
7
8     for i in range(nrand):
9         n += str(random.randrange(10))
10    n += "0"
11    check = checkLuhn(n)
12    if check != 0:
13        check = 10 - check
14    n = n[:-1] + str(check)
15    return n
```

./Code/LuhnNumberGeneration.py

Annexe D

Examples

D.1 Ex : Read_all Function

```
1  #include <stdlib.h>
2  #include <sys/types.h>
3  #include <sys/uio.h>
4  #include <unistd.h>
5  #include <assert.h>
6  #include <sys/time.h>
7  #include <fcntl.h>
8  #include <sys/stat.h>
9  #include <string.h>
10 #include <stdio.h>
11
12 ssize_t read_fi(int fildes, void *buf, size_t nbyte)
13 {
14     nbyte = (rand() % nbyte) + 1;
15     return read(fildes, buf, nbyte);
16 }
17
18 ssize_t read_all(int fildes, void *buf, size_t nbyte)
19 {
20     assert(fildes >= 0);
21     assert(buf);
22     assert(nbyte >= 0);
23
24     size_t left = nbyte;
25     while(1){
26         int res = read_fi(fildes, buf, left);
27         printf("%d\n", res);
28         if(res < 1)
29             return res;
30         buf += res;
31         left -= res;
32         assert(left >= 0);
33         if(left == 0)
34             return nbyte;
35     }
36 }
37
38 int main(void)
39 {
40     srand(time(NULL));
```

```

41
42     int fd = open("SplayTree.py", O_RDONLY);
43     assert(fd >= 0);
44
45     struct stat buf;
46     int res = fstat(fd, &buf);
47     assert(res == 0);
48
49     off_t len = buf.st_size;
50     char *definitive = (char *) malloc(len);
51     assert(definitive);
52
53     res = read(fd, definitive, len);
54     assert(res == len);
55
56     int i;
57     char *test = (char *) malloc(len);
58     for(i = 0; i < 100; i++){
59         resr = lseek(fd, 0, SEEK_SET);
60         assert(res == 0);
61         int j;
62         for(j = 0; j < len; j++){
63             test[j] = rand();
64         }
65         res = read_all(fd, test, len);
66         assert(res == len);
67         assert(strncmp(test, definitive, len) == 0);
68     }
69     return;
70 }

```

./Code/Read_All.c

D.2 Ex : "Babysitting an Army of Monkeys" - Charlie Miller

```

1  # Load file into buffer
2
3  numwrites = random.randrange(math.ceil((float(len(buf)) / FuzzFactor))) + 1
4  for j in range(numwrites):
5      rbyte = random.randrange(256)
6      rn = random.randrange(len(buf))
7      buf[rn] = "%c"%(rbyte)
8
9  # Save buffer
10 # Run process
11 # Look at exit code
12 # If it doesn't die, kill it
13 # Start over

```

./Code/BabysittingAnArmyOfMonkeys.py

1

-
1. [Babysitting an Army of Monkeys 1](#)
 1. [Babysitting an Army of Monkeys 2](#)

D.3 Bitwise

```
1 import random
2
3
4 def high_common_bits(a, b):
5     mask = 0x8000000000000000
6     output = 0
7
8     for i in reversed(range(64)):
9         if (a&mask) == (b&mask):
10             output |= a&mask
11         else:
12             output |= mask
13         return output
14     mask >>= 1
15     return output
16
17 def low_common_bits(a, b):
18     mask = 1
19     output = 0
20
21     for i in range(64):
22         if (a&mask) == (b&mask):
23             output |= mask
24         return output
25     mask <<= 1
26     return output
27
28 def test(a, b):
29     print("a = " + str(a) + " b = " + str(b))
30     print(high_common_bits(a, b))
31     print(low_common_bits(a, b))
32
33
34 for i in range(10000):
35     a = random.getrandbits(64)
36     b = a
37     for j in range(random.randrange(63)):
38         b ^= 1 << random.randrange(0,63)
39     print(high_common_bits(a, b) + low_common_bits(a, b))
```

./Code/Bitwise.py

Annexe E

Problem Sets

E.1 BlackBox Testing

```
1  # CORRECT SPECIFICATION:
2  #
3  # the Queue class provides a fixed-size FIFO queue of integers
4  #
5  # the constructor takes a single parameter: an integer > 0 that
6  # is the maximum number of elements the queue can hold.
7  #
8  # empty() returns True if and only if the queue currently
9  # holds no elements, and False otherwise.
10 #
11 # full() returns True if and only if the queue cannot hold
12 # any more elements, and False otherwise.
13 #
14 # enqueue(i) attempts to put the integer i into the queue; it returns
15 # True if successful and False if the queue is full.
16 #
17 # dequeue() removes an integer from the queue and returns it,
18 # or else returns None if the queue is empty.
19 #
20 # Example:
21 # q = Queue(1)
22 # is_empty = q.empty()
23 # succeeded = q.enqueue(10)
24 # is_full = q.full()
25 # value = q.dequeue()
26 #
27 # 1. Should create a Queue q that can only hold 1 element
28 # 2. Should then check whether q is empty, which should return True
29 # 3. Should attempt to put 10 into the queue, and return True
30 # 4. Should check whether q is now full, which should return True
31 # 5. Should attempt to dequeue and put the result into value, which
32 #    should be 10
33 #
34 # Your test function should run assertion checks and throw an
35 # AssertionError for each of the 5 incorrect Queues. Pressing
36 # submit will tell you how many you successfully catch so far.
37
38 from queue_test import *
39
40 def test():
```

```
41 # Queue1 silently holds only 2 byte unsigned integers , than wraps around
42 q = Queue(1)
43 succeeded = q.enqueue(65537) # % 2**16 + 1
44 assert succeeded
45 value = q.dequeue()
46 assert value == 65537
47
48 #Queue2 silently fails to hold more than 15 elements
49 q = Queue(17) # If greather than 15, put to 15
50 for i in range(16):
51     succeeded = q.enqueue(i)
52     assert succeeded
53
54 #Queue3 implements empty() by checking if dequeue() succeeds.
55 #This changes the state of the queue unintentionally.
56 q = Queue(2)
57 succeeded = q.enqueue(10)
58 assert succeeded
59 assert not q.empty() # dequeue the element
60 value = q.dequeue()
61 assert value == 10
62
63 #Queue4 dequeue() of an empty queue return False instead of None
64 q = Queue(2)
65 value = q.dequeue() # not faithful to specification
66 assert value == None
67
68 #Queue5 holds one less item than intended
69 q = Queue(2) # size_max -= 1
70 for i in range(2):
71     succeeded = q.enqueue(1)
72     assert succeeded
73
74 test()
```

./ProblemSets/ProblemSet1/ProblemSet1.py

E.2 Coverage

E.2.1 Queue Coverage

```
1  # TASK:
2  #
3  # Achieve full statement coverage on the Queue class.
4  # You will need to:
5  # 1) Write your test code in the test function.
6  # 2) Press submit. The grader will tell you if you
7  #    fail to cover any specific part of the code.
8  # 3) Update your test function until you cover the
9  #    entire code base.
10 #
11 # You can also run your code through a code coverage
12 # tool on your local machine if you prefer. This is
13 # not necessary, however.
14 # If you have any questions, please don't hesitate
15 # to ask in the forums!
16
17 import array
18
19 class Queue:
20     def __init__(self, size_max):
21         assert size_max > 0
22         self.max = size_max
23         self.head = 0
24         self.tail = 0
25         self.size = 0
26         self.data = array.array('i', range(size_max))
27
28     def empty(self):
29         return self.size == 0
30
31     def full(self):
32         return self.size == self.max
33
34     def enqueue(self, x):
35         if self.size == self.max:
36             return False
37         self.data[self.tail] = x
38         self.size += 1
39         self.tail += 1
40         if self.tail == self.max:
41             self.tail = 0
42         return True
43
44     def dequeue(self):
45         if self.size == 0:
46             return None
47         x = self.data[self.head]
48         self.size -= 1
49         self.head += 1
50         if self.head == self.max:
51             self.head = 0
52         return x
53
54     def checkRep(self):
```

```
55         assert self.tail >= 0
56         assert self.tail < self.max
57         assert self.head >= 0
58         assert self.head < self.max
59         if self.tail > self.head:
60             assert (self.tail-self.head) == self.size
61         if self.tail < self.head:
62             assert (self.head-self.tail) == (self.max-self.size)
63         if self.head == self.tail:
64             assert (self.size==0) or (self.size==self.max)
65
66 # Test code that achieves 100% coverage of the
67 # Queue class.
68
69 def test():
70     q = Queue(2)
71     assert q
72     q.checkRep()
73
74     empty = q.empty()
75     assert empty
76     q.checkRep()
77
78     full = q.full()
79     assert not full
80     q.checkRep()
81
82     result = q.dequeue()
83     assert result == None
84     q.checkRep()
85
86     result = q.enqueue(10)
87     assert result == True
88     q.checkRep()
89
90     result = q.enqueue(20)
91     assert result == True
92     q.checkRep()
93
94     empty = q.empty()
95     assert not empty
96     q.checkRep()
97
98     full = q.full()
99     assert full
100    q.checkRep()
101
102    result = q.enqueue(30)
103    assert result == False
104    q.checkRep()
105
106    result = q.dequeue()
107    assert result == 10
108    q.checkRep()
109
110    result = q.dequeue()
111    assert result == 20
112    q.checkRep()
113
```

```
114 test()
```

```
./ProblemSets/ProblemSet2/PS2.1/ProblemSet2_1.py
```

Coverage report: 100%

<i>Module</i> ↓	<i>statements</i>	<i>missing</i>	<i>excluded</i>	<i>coverage</i>
ProblemSet2_1.py	77	0	0	100%
Total	77	0	0	100%

coverage.py v4.5.4, created at 2019-08-05 12:33

FIGURE E.1 – Coverage

E.2.2 Splay Tree Coverage

```

1  # TASK:
2  #
3  # Achieve full statement coverage on the Queue class.
4  # You will need to:
5  # 1) Write your test code in the test function.
6  # 2) Press submit. The grader will tell you if you
7  #    fail to cover any specific part of the code.
8  # 3) Update your test function until you cover the
9  #    entire code base.
10 #
11 # You can also run your code through a code coverage
12 # tool on your local machine if you prefer. This is
13 # not necessary, however.
14 # If you have any questions, please don't hesitate
15 # to ask in the forums!
16
17 # Test code that achieves 100% coverage of the SplayTree class.
18
19 class Node:
20     def __init__(self, key):
21         self.key = key
22         self.left = self.right = None
23
24     def equals(self, node):
25         return self.key == node.key
26
27 class SplayTree:
28     def __init__(self):
29         self.root = None
30         self.header = Node(None) #For splay()
31
32     def insert(self, key):
33         if (self.root == None):
34             self.root = Node(key)
35             return
36
37         self.splay(key)
38         if self.root.key == key:
39             # If the key is already there in the tree, don't do anything.
40             return
41
42         n = Node(key)
43         if key < self.root.key:
44             n.left = self.root.left
45             n.right = self.root
46             self.root.left = None
47         else:
48             n.right = self.root.right
49             n.left = self.root
50             self.root.right = None
51         self.root = n
52
53     def remove(self, key):
54         self.splay(key)
55         if self.root is None or key != self.root.key:
56             return
57

```

```

58     # Now delete the root.
59     if self.root.left == None:
60         self.root = self.root.right
61     else:
62         x = self.root.right
63         self.root = self.root.left
64         self.splay(key)
65         self.root.right = x
66
67     def findMin(self):
68         if self.root == None:
69             return None
70         x = self.root
71         while x.left != None:
72             x = x.left
73         self.splay(x.key)
74         return x.key
75
76     def findMax(self):
77         if self.root == None:
78             return None
79         x = self.root
80         while (x.right != None):
81             x = x.right
82         self.splay(x.key)
83         return x.key
84
85     def find(self, key):
86         if self.root == None:
87             return None
88         self.splay(key)
89         if self.root.key != key:
90             return None
91         return self.root.key
92
93     def isEmpty(self):
94         return self.root == None
95
96     def splay(self, key):
97         l = r = self.header
98         t = self.root
99         if t is None:
100             return
101         self.header.left = self.header.right = None
102         while True:
103             if key < t.key:
104                 if t.left == None:
105                     break
106                 if key < t.left.key:
107                     y = t.left
108                     t.left = y.right
109                     y.right = t
110                     t = y
111                 if t.left == None:
112                     break
113                 r.left = t
114                 r = t
115                 t = t.left
116             elif key > t.key:

```

```

117         if t.right == None:
118             break
119         if key > t.right.key:
120             y = t.right
121             t.right = y.left
122             y.left = t
123             t = y
124             if t.right == None:
125                 break
126             l.right = t
127             l = t
128             t = t.right
129         else:
130             break
131         l.right = t.left
132         r.left = t.right
133         t.left = self.header.right
134         t.right = self.header.left
135         self.root = t
136
137     def test():
138         n1 = Node(18)
139         n2 = Node(18)
140         assert n1.equals(n2)
141
142         s = SplayTree()
143         current_min = None
144         current_max = None
145         s.splay(18)
146
147         empty = s.isEmpty()
148         assert empty == True
149         _min = s.findMin()
150         assert _min == None
151         _max = s.findMax()
152         assert _max == None
153
154         found = s.find(10)
155         assert found == None
156
157         s.insert(100)
158         current_min = 100
159         current_max = 100
160
161         for i in range(10,20):
162             empty = s.isEmpty()
163             assert empty == False
164
165             s.insert(i)
166             s.insert(i)
167
168             if not current_min or i < current_min:
169                 current_min = i
170
171             found = s.find(i)
172             assert found == i
173
174             _min = s.findMin()
175             assert _min == current_min

```



```

176
177     _max = s.findMax()
178     assert _max == current_max
179
180     for i in range(10,20):
181         empty = s.isEmpty()
182         assert empty == False
183
184         s.remove(i)
185         s.remove(i)
186
187         found = s.find(i)
188         assert found == None
189
190     s.insert(373)
191     s.remove(373)
192
193 test()

```

./ProblemSets/ProblemSet2/PS2.2/ProblemSet2_2.py

Coverage report: 100%

<i>Module ↓</i>	<i>statements</i>	<i>missing</i>	<i>excluded</i>	<i>coverage</i>
ProblemSet2_2.py	142	0	0	100%
Total	142	0	0	100%

coverage.py v4.5.4, created at 2019-08-05 13:00

FIGURE E.2 – Coverage

E.3 Sudoku

E.3.1 Sudoku Checker

```

1  # SPECIFICATION:
2  #
3  # check_sudoku() determines whether its argument is a valid Sudoku
4  # grid. It can handle grids that are completely filled in, and also
5  # grids that hold some empty cells where the player has not yet
6  # written numbers.
7  #
8  # First, your code must do some sanity checking to make sure that its
9  # argument:
10 #
11 # - is a 9x9 list of lists
12 #
13 # - contains, in each of its 81 elements, an integer in the range 0..9
14 #
15 # If either of these properties does not hold, check_sudoku must
16 # return None.
17 #
18 # If the sanity checks pass, your code should return True if all of
19 # the following hold, and False otherwise:
20 #
21 # - each number in the range 1..9 occurs only once in each row
22 #
23 # - each number in the range 1..9 occurs only once in each column
24 #
25 # - each number the range 1..9 occurs only once in each of the nine
26 #   3x3 sub-grids, or "boxes", that make up the board
27 #
28 # This diagram (which depicts a valid Sudoku grid) illustrates how the
29 # grid is divided into sub-grids:
30 #
31 # 5 3 4 | 6 7 8 | 9 1 2
32 # 6 7 2 | 1 9 5 | 3 4 8
33 # 1 9 8 | 3 4 2 | 5 6 7
34 # -----
35 # 8 5 9 | 7 6 1 | 4 2 3
36 # 4 2 6 | 8 5 3 | 7 9 1
37 # 7 1 3 | 9 2 4 | 8 5 6
38 # -----
39 # 9 6 1 | 5 3 7 | 0 0 0
40 # 2 8 7 | 4 1 9 | 0 0 0
41 # 3 4 5 | 2 8 6 | 0 0 0
42 #
43 # Please keep in mind that a valid grid (i.e., one for which your
44 # function returns True) may contain 0 multiple times in a row,
45 # column, or sub-grid. Here we are using 0 to represent an element of
46 # the Sudoku grid that the player has not yet filled in.
47
48 # Matrix for equivalence tests
49
50 # Not a matrix
51 not_matrix = """[5,3,4,6,7,8,9,1,2],
52                 [6,7,2,1,9,5,3,4,8],
53                 [1,9,8,3,4,2,5,6,7],
54                 [8,5,9,7,6,1,4,2,3],

```

```

55         [4,2,6,8,5,3,7,9,1],
56         [7,1,3,9,2,4,8,5,6],
57         [9,6,1,5,3,7,2,8,4],
58         [2,8,7,4,1,9,6,3,5],
59         [3,4,5,2,8,6,1,7,9]]"""
60
61 # Matrix of not numbers
62 not_number_matrix = [[ "5", "3", "4", "6", "7", "8", "9", "1", "2" ],
63                      [ "6", "7", "2", "1", "9", "5", "3", "4", "8" ],
64                      [ "1", "9", "8", "3", "4", "2", "5", "6", "7" ],
65                      [ "8", "5", "9", "7", "6", "1", "4", "2", "3" ],
66                      [ "4", "2", "6", "8", "5", "3", "7", "9", "1" ],
67                      [ "7", "1", "3", "9", "2", "4", "8", "5", "6" ],
68                      [ "9", "6", "1", "5", "3", "7", "2", "8", "4" ],
69                      [ "2", "8", "7", "4", "1", "9", "6", "3", "5" ],
70                      [ "3", "4", "5", "2", "8", "6", "1", "7", "9" ]]
71
72 # check_sudoku should return None
73 # One element lack
74 ill_formed = [[5,3,4,6,7,8,9,1,2],
75              [6,7,2,1,9,5,3,4,8],
76              [1,9,8,3,4,2,5,6,7],
77              [8,5,9,7,6,1,4,2,3],
78              [4,2,6,8,5,3,7,9], # <—
79              [7,1,3,9,2,4,8,5,6],
80              [9,6,1,5,3,7,2,8,4],
81              [2,8,7,4,1,9,6,3,5],
82              [3,4,5,2,8,6,1,7,9]]
83
84 # One row lack
85 row_lack = [[5,3,4,6,7,8,9,1,2],
86            [6,7,2,1,9,5,3,4,8],
87            [1,9,8,3,4,2,5,6,7],
88            [8,5,9,7,6,1,4,2,3],
89            [4,2,6,8,5,3,7,9,1],
90            [7,1,3,9,2,4,8,5,6],
91            [9,6,1,5,3,7,2,8,4],
92            [2,8,7,4,1,9,6,3,5]]
93
94 # One column lack
95 column_lack = [[5,3,4,6,7,8,9,1],
96               [6,7,2,1,9,5,3,4],
97               [1,9,8,3,4,2,5,6],
98               [8,5,9,7,6,1,4,2],
99               [4,2,6,8,5,3,7,9],
100              [7,1,3,9,2,4,8,5],
101              [9,6,1,5,3,7,2,8],
102              [2,8,7,4,1,9,6,3],
103              [3,4,5,2,8,6,1,7]]
104
105 # check_sudoku should return True
106 # Valid full sudoku
107 valid = [[5,3,4,6,7,8,9,1,2],
108         [6,7,2,1,9,5,3,4,8],
109         [1,9,8,3,4,2,5,6,7],
110         [8,5,9,7,6,1,4,2,3],
111         [4,2,6,8,5,3,7,9,1],
112         [7,1,3,9,2,4,8,5,6],
113         [9,6,1,5,3,7,2,8,4],

```

```

114         [2,8,7,4,1,9,6,3,5],
115         [3,4,5,2,8,6,1,7,9]]
116
117 # check_sudoku should return False
118 # Invalid but well-formed
119 invalid = [[5,3,4,6,7,8,9,1,2],
120            [6,7,2,1,9,5,3,4,8],
121            [1,9,8,3,8,2,5,6,7],
122            [8,5,9,7,6,1,4,2,3],
123            [4,2,6,8,5,3,7,9,1],
124            [7,1,3,9,2,4,8,5,6],
125            [9,6,1,5,3,7,2,8,4],
126            [2,8,7,4,1,9,6,3,5],
127            [3,4,5,2,8,6,1,7,9]]
128
129 # Duplicate on collumn but not row
130 invalid_column = [[5,3,4,6,7,8,9,1,2],
131                  [6,7,2,1,9,5,3,4,8],
132                  [1,9,0,3,8,2,5,6,7],
133                  [8,5,9,7,6,1,4,2,3],
134                  [4,2,6,8,5,3,7,9,1],
135                  [7,1,3,9,2,4,8,5,6],
136                  [9,6,1,5,3,7,2,8,4],
137                  [2,8,7,4,1,9,6,3,5],
138                  [3,4,5,2,8,6,1,7,9]]
139
140 # Duplicate on row but not column
141 invalid_row = [[5,3,4,6,7,8,9,1,2],
142               [6,7,2,1,9,5,3,4,8],
143               [1,9,0,3,8,2,5,6,7],
144               [8,5,9,7,6,1,4,2,3],
145               [4,2,6,8,5,3,7,9,1],
146               [7,1,3,9,2,4,8,5,6],
147               [9,6,1,5,3,7,2,8,4],
148               [2,8,7,4,1,9,6,3,5],
149               [3,4,5,2,0,6,1,7,9]]
150
151 # Duplicate in a block
152 invalid_block = [[0,0,0,0,0,0,0,0,0],
153                 [0,0,0,0,0,0,0,0,0],
154                 [0,0,0,0,0,0,0,0,0],
155                 [0,0,0,0,0,0,0,0,0],
156                 [0,0,0,0,0,0,0,0,0],
157                 [0,0,0,0,0,0,0,0,0],
158                 [0,0,0,1,0,0,0,0,0],
159                 [0,0,0,0,1,0,0,0,0],
160                 [0,0,0,0,0,0,0,0,0]]
161
162 # check_sudoku should return True
163 # valid semi-empty sudoku
164 easy = [[2,9,0,0,0,0,0,7,0],
165         [3,0,6,0,0,8,4,0,0],
166         [8,0,0,0,4,0,0,0,2],
167         [0,2,0,0,3,1,0,0,7],
168         [0,0,0,0,8,0,0,0,0],
169         [1,0,0,9,5,0,0,6,0],
170         [7,0,0,0,9,0,0,0,1],
171         [0,0,1,2,0,0,3,0,6],
172         [0,3,0,0,0,0,0,5,9]]

```

```

173
174 # valid empty sudoku
175 empty = [[0,0,0,0,0,0,0,0,0],
176           [0,0,0,0,0,0,0,0,0],
177           [0,0,0,0,0,0,0,0,0],
178           [0,0,0,0,0,0,0,0,0],
179           [0,0,0,0,0,0,0,0,0],
180           [0,0,0,0,0,0,0,0,0],
181           [0,0,0,0,0,0,0,0,0],
182           [0,0,0,0,0,0,0,0,0],
183           [0,0,0,0,0,0,0,0,0]]
184
185 # check_sudoku should return True
186 hard = [[1,0,0,0,0,7,0,9,0],
187          [0,3,0,0,2,0,0,0,8],
188          [0,0,9,6,0,0,5,0,0],
189          [0,0,5,3,0,0,9,0,0],
190          [0,1,0,0,8,0,0,0,2],
191          [6,0,0,0,0,4,0,0,0],
192          [3,0,0,0,0,0,0,1,0],
193          [0,4,0,0,0,0,0,0,7],
194          [0,0,7,0,0,0,3,0,0]]
195
196 # Unsolvable Valid Matrix
197 unsolvable = [[5,3,4,6,7,8,9,1,2],
198               [6,7,2,1,9,5,3,4,8],
199               [1,9,0,3,8,2,5,6,7],
200               [8,5,9,7,6,1,4,2,3],
201               [4,2,6,8,5,3,7,9,1],
202               [7,1,3,9,2,4,8,5,6],
203               [9,6,1,5,3,7,2,8,4],
204               [2,8,7,4,1,9,6,3,5],
205               [3,4,5,2,8,6,1,7,9]]
206
207
208 def matrix_trans(grid):
209
210     # Initialisation of the transposed matrix
211     grid_range = len(grid[0])
212     gridTrans = []
213
214     for i in range(grid_range):
215         gridTrans.append([])
216
217     # Reverse line and columns
218     for row in grid:
219         for element in row:
220             gridTrans[row.index(element)].append(element)
221
222     return gridTrans
223
224 def row_validity(row):
225
226     # Creation of temporary variable for modification
227     row_tmp = [x for x in row if x != 0]
228     if len(row_tmp) != len(set(row_tmp)): # Checks if no duplicates
229         return False
230
231     for element in row_tmp:

```

```

232         row_tmp = row_tmp[1:]
233         if element in row_tmp:
234             return False
235
236     return True
237
238 def matrix_validity(grid):
239
240     # Creation of the transposed grid
241     gridTrans = matrix_trans(grid)
242
243     for row in grid:
244         if row_validity(row) == False: # Checks rows validity
245             return False
246
247     for row in gridTrans:
248         if row_validity(row) == False : # Checks columns validity
249             return False
250
251     return True
252
253
254 def check_sudoku(grid):
255
256     sub_list = []
257
258     # Checking the sanity check
259     if not isinstance(grid, (list)): # Checks that grid is a list
260         return None
261
262     if len(grid) != 9: # Checks the number of columns
263         return None
264
265     for row in grid:
266         if not isinstance(row, (list)) or len(row) != 9: # Checks that row is a
267                                                         # list and the number
268                                                         # of lines
269
270             for element in row:
271                 if element not in [0,1,2,3,4,5,6,7,8,9]: # Checks that elements
272                                                         # are in the domain
273                     return None
274
275     if not matrix_validity(grid):
276         return False
277
278     # Boundary delimitation of the subsets
279     for i in [(0,3),(3,6),(6,9)]:
280         for j in [(0,3),(3,6),(6,9)]:
281             # Creation of subsets bases on boundaries
282             sub_grid = [row[i[0]:i[1]] for row in grid[j[0]:j[1]]]
283             # List of elements in the sub-grid
284             for row in sub_grid:
285                 for element in row:
286                     if element != 0:
287                         sub_list.append(element)
288
289             if len(sub_list) != len(set(sub_list)):
290                 return False

```

```

291         # Reset of the sub-list
292         sub_list = []
293
294     return True
295
296 matrix = [not_matrix, not_number_matrix, ill_formed, row_lack, column_lack, \
297           valid, invalid, invalid_column, invalid_row, invalid_block, easy, \
298           empty, hard, unsolvable]
299
300 for m in matrix:
301     if matrix.index(m) in [0,1,2,3,4]:
302         assert check_sudoku(m) == None
303     elif matrix.index(m) in [6,7,8,9]:
304         assert check_sudoku(m) == False
305     elif matrix.index(m) in [5,10,11,12]:
306         assert check_sudoku(m)

```

./ProblemSets/ProblemSet3/PS3.1/SudokuChecker.py

Coverage report: 100%

Module ↓	statements	missing	excluded	coverage
SudokuChecker.py	54	0	0	100%
Total	54	0	0	100%

coverage.py v4.5.4, created at 2019-08-05 14:04

FIGURE E.3 – Coverage

E.3.2 Sudoku Checker Bis

```

1  # SPECIFICATION:
2  #
3  # check_sudoku() determines whether its argument is a valid Sudoku
4  # grid. It can handle grids that are completely filled in, and also
5  # grids that hold some empty cells where the player has not yet
6  # written numbers.
7  #
8  # First, your code must do some sanity checking to make sure that its
9  # argument:
10 #
11 # - is a 9x9 list of lists
12 #
13 # - contains, in each of its 81 elements, an integer in the range 0..9
14 #
15 # If either of these properties does not hold, check_sudoku must
16 # return None.
17 #
18 # If the sanity checks pass, your code should return True if all of
19 # the following hold, and False otherwise:
20 #
21 # - each number in the range 1..9 occurs only once in each row
22 #
23 # - each number in the range 1..9 occurs only once in each column
24 #
25 # - each number the range 1..9 occurs only once in each of the nine
26 #   3x3 sub-grids, or "boxes", that make up the board
27 #
28 # This diagram (which depicts a valid Sudoku grid) illustrates how the
29 # grid is divided into sub-grids:
30 #
31 # 5 3 4 | 6 7 8 | 9 1 2
32 # 6 7 2 | 1 9 5 | 3 4 8
33 # 1 9 8 | 3 4 2 | 5 6 7
34 # -----
35 # 8 5 9 | 7 6 1 | 4 2 3
36 # 4 2 6 | 8 5 3 | 7 9 1
37 # 7 1 3 | 9 2 4 | 8 5 6
38 # -----
39 # 9 6 1 | 5 3 7 | 0 0 0
40 # 2 8 7 | 4 1 9 | 0 0 0
41 # 3 4 5 | 2 8 6 | 0 0 0
42 #
43 # Please keep in mind that a valid grid (i.e., one for which your
44 # function returns True) may contain 0 multiple times in a row,
45 # column, or sub-grid. Here we are using 0 to represent an element of
46 # the Sudoku grid that the player has not yet filled in.
47 #
48 # check_sudoku should return None
49 ill_formed = [[5,3,4,6,7,8,9,1,2],
50               [6,7,2,1,9,5,3,4,8],
51               [1,9,8,3,4,2,5,6,7],
52               [8,5,9,7,6,1,4,2,3],
53               [4,2,6,8,5,3,7,9], # <—
54               [7,1,3,9,2,4,8,5,6],
55               [9,6,1,5,3,7,2,8,4],
56               [2,8,7,4,1,9,6,3,5],
57               [3,4,5,2,8,6,1,7,9]]

```



```

58
59 # check_sudoku should return True
60 valid = [[5,3,4,6,7,8,9,1,2],
61          [6,7,2,1,9,5,3,4,8],
62          [1,9,8,3,4,2,5,6,7],
63          [8,5,9,7,6,1,4,2,3],
64          [4,2,6,8,5,3,7,9,1],
65          [7,1,3,9,2,4,8,5,6],
66          [9,6,1,5,3,7,2,8,4],
67          [2,8,7,4,1,9,6,3,5],
68          [3,4,5,2,8,6,1,7,9]]
69
70 # check_sudoku should return False
71 invalid = [[5,3,4,6,7,8,9,1,2],
72            [6,7,2,1,9,5,3,4,8],
73            [1,9,8,3,8,2,5,6,7],
74            [8,5,9,7,6,1,4,2,3],
75            [4,2,6,8,5,3,7,9,1],
76            [7,1,3,9,2,4,8,5,6],
77            [9,6,1,5,3,7,2,8,4],
78            [2,8,7,4,1,9,6,3,5],
79            [3,4,5,2,8,6,1,7,9]]
80
81 # check_sudoku should return True
82 easy = [[2,9,0,0,0,0,0,7,0],
83         [3,0,6,0,0,8,4,0,0],
84         [8,0,0,0,4,0,0,0,2],
85         [0,2,0,0,3,1,0,0,7],
86         [0,0,0,0,8,0,0,0,0],
87         [1,0,0,9,5,0,0,6,0],
88         [7,0,0,0,9,0,0,0,1],
89         [0,0,1,2,0,0,3,0,6],
90         [0,3,0,0,0,0,0,5,9]]
91
92 # check_sudoku should return True
93 hard = [[1,0,0,0,0,7,0,9,0],
94         [0,3,0,0,2,0,0,0,8],
95         [0,0,9,6,0,0,5,0,0],
96         [0,0,5,3,0,0,9,0,0],
97         [0,1,0,0,8,0,0,0,2],
98         [6,0,0,0,0,4,0,0,0],
99         [3,0,0,0,0,0,0,1,0],
100        [0,4,0,0,0,0,0,0,7],
101        [0,0,7,0,0,0,3,0,0]]
102
103
104 def check_sudoku(grid):
105
106     # Initialisation of variables
107     row_tmp = []
108     gridTrans = [[],[],[],[],[],[],[],[],[]]
109     sub_gridTrans = [[],[],[]]
110     sub_list = []
111
112     # Checking the sanity check
113     if not isinstance(grid, (list)): # Checks that grid is a list
114         return None
115
116     if len(grid) != 9: # Checks the number of columns

```

```

117         return None
118
119     for row in grid:
120         if not isinstance(row, (list)) or len(row) != 9: # Checks that row is a
121             return None                                     # list and the number
122                                                         # of lines
123
124         for element in row:
125             if element not in [0,1,2,3,4,5,6,7,8,9]: # Checks that elements
126                 return None                             # are in the domain
127
128             if element != 0:
129                 row_tmp.append(element) # Creation of the list for check row
130                                         # validity
131
132             # Creation of the transposed grid
133             gridTrans[row.index(element)].append(element)
134
135             # Checking the validity of the grid
136             if len(row_tmp) != len(set(row_tmp)): # Checks that no element in rows
137                 return False                       # are redundant
138
139             # Reset the variables for the next iteration of the loop
140             row_tmp = []
141
142     # Creation of the transposed grid
143     for row in gridTrans:
144         for element in row:
145
146             if element != 0:
147                 row_tmp.append(element) # Creation of the list for check column
148                                         # validity
149
150             if len(row_tmp) != len(set(row_tmp)): # Checks that no element in
151                 return False                       # columns are redundant
152
153     row_tmp = []
154
155     # Boundary delimitation of the subsets
156     for i in [(0,3),(3,6),(6,9)]:
157         for j in [(0,3),(3,6),(6,9)]:
158             # Creation of subsets bases on boundaries
159             sub_grid = [row[i[0]:i[1]] for row in grid[j[0]:j[1]]]
160             # List of elements in the sub-grid
161             for row in sub_grid:
162                 for element in row:
163                     if element != 0:
164                         sub_list.append(element)
165
166             if len(sub_list) != len(set(sub_list)):
167                 return False
168
169             # Reset of the sub-list
170             sub_list = 0
171
172     return True
173
174
175 print(check_sudoku(ill_formed)) # —> None

```

```
176 print(check_sudoku(valid))      # → True
177 print(check_sudoku(invalid))    # → False
178 print(check_sudoku(easy))        # → True
179 print(check_sudoku(hard))        # → True
```

./Code/SudokuChecker.py

E.3.3 Sudoku Randomizer

```

1  import random
2
3  from SudokuChecker import check_sudoku
4
5  def random_sudoku():
6      sudoku = []
7      bias = -0.2
8
9      for i in range(9):
10         sudoku.append([])
11
12     for row in sudoku:
13         for i in range(9):
14             if (random.random() < 0.5 + bias):
15                 row.append(random.randint(1,9))
16                 bias -= 0.35
17             else:
18                 row.append(0)
19                 bias += 0.05
20
21     bias = -0.2
22
23     return sudoku
24
25 for i in range(10000):
26     print(check_sudoku(random_sudoku()))

```

./ProblemSets/ProblemSet3/PS3.1/SudokuRandomizer.py

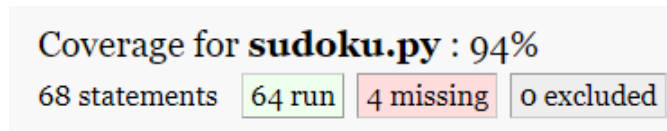


FIGURE E.4 – Coverage

```

204     # Checking the sanity check
205     if not isinstance(grid, (list)): # Checks that grid is a list
206         return None
207
208     if len(grid) != 9: # Checks the number of columns
209         return None
210
211     for row in grid:
212         if not isinstance(row, (list)) or len(row) != 9: # Checks that row is a
213             return None                                     # list and the number
214                                                         # of lines
215
216     for element in row:
217         if element not in [0,1,2,3,4,5,6,7,8,9]: # Checks that elements
218             return None                             # are in the domain

```

FIGURE E.5 – Coverage

E.3.4 Sudoku Solver

```

1  import time
2
3
4  # CHALLENGE PROBLEM:
5  #
6  # Use your check_sudoku function as the basis for solve_sudoku(): a
7  # function that takes a partially-completed Sudoku grid and replaces
8  # each 0 cell with an integer in the range 1..9 in such a way that the
9  # final grid is valid.
10 #
11 # There are many ways to cleverly solve a partially-completed Sudoku
12 # puzzle, but a brute-force recursive solution with backtracking is a
13 # perfectly good option. The solver should return None for broken
14 # input, False for inputs that have no valid solutions, and a valid
15 # 9x9 Sudoku grid containing no 0 elements otherwise. In general, a
16 # partially-completed Sudoku grid does not have a unique solution. You
17 # should just return some member of the set of solutions.
18 #
19 # A solve_sudoku() in this style can be implemented in about 16 lines
20 # without making any particular effort to write concise code.
21
22 # Matrix for equivalence tests
23
24 # Not a matrix
25 not_matrix = """[5,3,4,6,7,8,9,1,2],
26                 [6,7,2,1,9,5,3,4,8],
27                 [1,9,8,3,4,2,5,6,7],
28                 [8,5,9,7,6,1,4,2,3],
29                 [4,2,6,8,5,3,7,9,1],
30                 [7,1,3,9,2,4,8,5,6],
31                 [9,6,1,5,3,7,2,8,4],
32                 [2,8,7,4,1,9,6,3,5],
33                 [3,4,5,2,8,6,1,7,9]"""
34
35 # Matrix of not numbers
36 not_number_matrix = [[ "5", "3", "4", "6", "7", "8", "9", "1", "2" ],
37                      [ "6", "7", "2", "1", "9", "5", "3", "4", "8" ],
38                      [ "1", "9", "8", "3", "4", "2", "5", "6", "7" ],
39                      [ "8", "5", "9", "7", "6", "1", "4", "2", "3" ],
40                      [ "4", "2", "6", "8", "5", "3", "7", "9", "1" ],
41                      [ "7", "1", "3", "9", "2", "4", "8", "5", "6" ],
42                      [ "9", "6", "1", "5", "3", "7", "2", "8", "4" ],
43                      [ "2", "8", "7", "4", "1", "9", "6", "3", "5" ],
44                      [ "3", "4", "5", "2", "8", "6", "1", "7", "9" ]]
45
46 # check_sudoku should return None
47 # One element lack
48 ill_formed = [[5,3,4,6,7,8,9,1,2],
49              [6,7,2,1,9,5,3,4,8],
50              [1,9,8,3,4,2,5,6,7],
51              [8,5,9,7,6,1,4,2,3],
52              [4,2,6,8,5,3,7,9], # <—
53              [7,1,3,9,2,4,8,5,6],
54              [9,6,1,5,3,7,2,8,4],
55              [2,8,7,4,1,9,6,3,5],
56              [3,4,5,2,8,6,1,7,9]]
57

```

```

58 # One row lack
59 row_lack = [[5,3,4,6,7,8,9,1,2],
60             [6,7,2,1,9,5,3,4,8],
61             [1,9,8,3,4,2,5,6,7],
62             [8,5,9,7,6,1,4,2,3],
63             [4,2,6,8,5,3,7,9,1],
64             [7,1,3,9,2,4,8,5,6],
65             [9,6,1,5,3,7,2,8,4],
66             [2,8,7,4,1,9,6,3,5]]
67
68 # One collumn lack
69 column_lack = [[5,3,4,6,7,8,9,1],
70                [6,7,2,1,9,5,3,4],
71                [1,9,8,3,4,2,5,6],
72                [8,5,9,7,6,1,4,2],
73                [4,2,6,8,5,3,7,9],
74                [7,1,3,9,2,4,8,5],
75                [9,6,1,5,3,7,2,8],
76                [2,8,7,4,1,9,6,3],
77                [3,4,5,2,8,6,1,7]]
78
79 # check_sudoku should return True
80 # Valid full sudoku
81 valid = [[5,3,4,6,7,8,9,1,2],
82          [6,7,2,1,9,5,3,4,8],
83          [1,9,8,3,4,2,5,6,7],
84          [8,5,9,7,6,1,4,2,3],
85          [4,2,6,8,5,3,7,9,1],
86          [7,1,3,9,2,4,8,5,6],
87          [9,6,1,5,3,7,2,8,4],
88          [2,8,7,4,1,9,6,3,5],
89          [3,4,5,2,8,6,1,7,9]]
90
91 # check_sudoku should return False
92 # Invalid but well-formed
93 invalid = [[5,3,4,6,7,8,9,1,2],
94            [6,7,2,1,9,5,3,4,8],
95            [1,9,8,3,8,2,5,6,7],
96            [8,5,9,7,6,1,4,2,3],
97            [4,2,6,8,5,3,7,9,1],
98            [7,1,3,9,2,4,8,5,6],
99            [9,6,1,5,3,7,2,8,4],
100           [2,8,7,4,1,9,6,3,5],
101           [3,4,5,2,8,6,1,7,9]]
102
103 # Duplicate on collumn but not row
104 invalid_column = [[5,3,4,6,7,8,9,1,2],
105                  [6,7,2,1,9,5,3,4,8],
106                  [1,9,0,3,8,2,5,6,7],
107                  [8,5,9,7,6,1,4,2,3],
108                  [4,2,6,8,5,3,7,9,1],
109                  [7,1,3,9,2,4,8,5,6],
110                  [9,6,1,5,3,7,2,8,4],
111                  [2,8,7,4,1,9,6,3,5],
112                  [3,4,5,2,8,6,1,7,9]]
113
114 # Duplicate on row but not column
115 invalid_row = [[5,3,4,6,7,8,9,1,2],
116               [6,7,2,1,9,5,3,4,8],

```

```

117         [1,9,0,3,8,2,5,6,7],
118         [8,5,9,7,6,1,4,2,3],
119         [4,2,6,8,5,3,7,9,1],
120         [7,1,3,9,2,4,8,5,6],
121         [9,6,1,5,3,7,2,8,4],
122         [2,8,7,4,1,9,6,3,5],
123         [3,4,5,2,0,6,1,7,9]]
124
125 # Duplicate in a block
126 invalid_block = [[0,0,0,0,0,0,0,0,0],
127                  [0,0,0,0,0,0,0,0,0],
128                  [0,0,0,0,0,0,0,0,0],
129                  [0,0,0,0,0,0,0,0,0],
130                  [0,0,0,0,0,0,0,0,0],
131                  [0,0,0,0,0,0,0,0,0],
132                  [0,0,0,1,0,0,0,0,0],
133                  [0,0,0,0,1,0,0,0,0],
134                  [0,0,0,0,0,0,0,0,0]]
135
136 # check_sudoku should return True
137 # valid semi-empty sudoku
138 easy = [[2,9,0,0,0,0,0,7,0],
139          [3,0,6,0,0,8,4,0,0],
140          [8,0,0,0,4,0,0,0,2],
141          [0,2,0,0,3,1,0,0,7],
142          [0,0,0,0,8,0,0,0,0],
143          [1,0,0,9,5,0,0,6,0],
144          [7,0,0,0,9,0,0,0,1],
145          [0,0,1,2,0,0,3,0,6],
146          [0,3,0,0,0,0,0,5,9]]
147
148 # valid empty sudoku
149 empty = [[0,0,0,0,0,0,0,0,0],
150          [0,0,0,0,0,0,0,0,0],
151          [0,0,0,0,0,0,0,0,0],
152          [0,0,0,0,0,0,0,0,0],
153          [0,0,0,0,0,0,0,0,0],
154          [0,0,0,0,0,0,0,0,0],
155          [0,0,0,0,0,0,0,0,0],
156          [0,0,0,0,0,0,0,0,0],
157          [0,0,0,0,0,0,0,0,0]]
158
159 # check_sudoku should return True
160 hard = [[1,0,0,0,0,7,0,9,0],
161          [0,3,0,0,2,0,0,0,8],
162          [0,0,9,6,0,0,5,0,0],
163          [0,0,5,3,0,0,9,0,0],
164          [0,1,0,0,8,0,0,0,2],
165          [6,0,0,0,0,4,0,0,0],
166          [3,0,0,0,0,0,0,1,0],
167          [0,4,0,0,0,0,0,0,7],
168          [0,0,7,0,0,0,3,0,0]]
169
170 # Unsolvble Valid Matrix
171 unsolvable = [[5,3,4,6,7,8,9,1,2],
172               [6,7,2,1,9,5,3,4,8],
173               [1,9,0,3,8,2,5,6,7],
174               [8,5,9,7,6,1,4,2,3],
175               [4,2,6,8,5,3,7,9,1],

```

```

176         [7,1,3,9,2,4,8,5,6],
177         [9,6,1,5,3,7,2,8,4],
178         [2,8,7,4,1,9,6,3,5],
179         [3,4,5,2,8,6,1,7,9]]
180
181
182 def matrix_trans(grid):
183     """
184     @requires : grid must be an n*n matrix
185     @ensures : grid transposed
186     """
187
188     # Initialisation of the transposed matrix
189     grid_range = len(grid[0])
190     gridTrans = []
191
192     for i in range(grid_range):
193         gridTrans.append([])
194
195     # Reverse line and columns
196     for row in grid:
197         for element in row:
198             gridTrans[row.index(element)].append(element)
199
200     return gridTrans
201
202 def row_validity(row):
203     """
204     @requires : row instance of list
205     @ensures : True if it's a valid sudoku row,
206                False either.
207     """
208
209     # Creation of temporary variable for modification
210     row_tmp = [x for x in row if x != 0]
211     if len(row_tmp) != len(set(row_tmp)): # Checks if no duplicates
212         return False
213
214     return True
215
216 def matrix_validity(grid):
217     """
218     @requires : grid instance of matrix (list of list)
219     @ensures : True if grid is a valid sudoku,
220                False either.
221     """
222
223     # Creation of the transposed grid
224     gridTrans = matrix_trans(grid)
225
226     for row in grid:
227         if row_validity(row) == False: # Checks rows validity
228             return False
229
230     for row in gridTrans:
231         if row_validity(row) == False : # Checks columns validity
232             return False
233
234     return True

```



```

235
236
237 def check_sudoku(grid):
238     """
239     @ensures : True if grid is a valid matrix,
240                False if grid is an invalid matrix,
241                None if grid is an invalid input.
242     """
243
244     # Checking the sanity check
245     if not isinstance(grid, (list)): # Checks that grid is a list
246         return None
247
248     if len(grid) != 9: # Checks the number of columns
249         return None
250
251     for row in grid:
252         if not isinstance(row, (list)) or len(row) != 9: # Checks that row is a
253                                                         # list and the number
254                                                         # of lines
255
256             for element in row:
257                 if element not in [0,1,2,3,4,5,6,7,8,9]: # Checks that elements
258                                                         # are in the domain
259
260             if matrix_validity(grid) == False:
261                 return False
262
263     # Boundary delimitation of the subsets
264
265     for i in [(0,3),(3,6),(6,9)]:
266         for j in [(0,3),(3,6),(6,9)]:
267             # Creation of subsets bases on boundaries
268             sub_grid = [row[i[0]:i[1]] for row in grid[j[0]:j[1]]]
269             sub_row = [row for sub_row in sub_grid for row in sub_row]
270             if row_validity(sub_row) == False:
271                 return False
272
273     return True
274
275
276 def backTracking(grid):
277     """
278     @requires : grid is a valid matrix
279     @modifies : grid
280     @ensures : solved grid if grid is solvable,
281                False either.
282     """
283
284     # Take the first blankBox
285     blankBox = pickBox(grid)
286     if not blankBox:
287         return grid # The sudoku is solved
288     else:
289         # Take coordinates
290         row, column = blankBox
291
292     # Check elements in the box from 1 to 9
293     for element in range(1,10):

```

```

294     # Check if it's a valid element
295     if element_validity(grid, row, column, element):
296         # Add the element to the grid
297         grid[row][column] = element
298
299     # Recursivity
300     if backTracking(grid):
301         return grid # return the sudoku solved
302
303     # If we go to a dead end, come back and try again
304     grid[row][column] = 0
305
306     return False
307
308
309 def element_validity(grid, row, column, element):
310     """
311     @requires : grid is a valid matrix &&
312                 0 <= row < 9 && 0 <= column < 9 &&
313                 0 < element <= 9
314     @ensures : True if the element in this position is valid,
315                 False either.
316     """
317
318     # Check if element not in row
319     for columns in grid[row]:
320         if columns == element and column != grid[row].index(columns):
321             return False
322
323     # Check if element not in column
324     for rows in grid:
325         if element == grid[grid.index(rows)][column] and row != grid.index(rows)
326         :
327             return False
328
329     # Check if element not in box
330     box_x = column // 3
331     box_y = row // 3
332
333     for i in range(box_y*3, box_y*3 + 3):
334         for j in range(box_x * 3, box_x*3 + 3):
335             if grid[i][j] == element and i != row and j != column:
336                 return False
337
338     return True
339
340 def display_sudoku(grid):
341     """
342     @requires grid is a matrix (list of list) or None or False
343     """
344
345     # Create a GUI
346     if (grid is None) or (grid is False):
347         print(grid)
348     else:
349         for i in range(len(grid)):
350             if (i % 3 == 0) and (i != 0):
351                 print("- - - - -")

```

```

352
353         for j in range(len(grid[0])):
354             if (j % 3 == 0) and (j != 0):
355                 print(" | ", end="")
356
357             if j == 8:
358                 print(grid[i][j])
359             else:
360                 print(str(grid[i][j]) + " ", end="")
361
362
363 def pickBox(grid):
364     """
365     @requires : grid is a valid matrix
366     @returns : a tuple with the row and the column of an empty box if it exists,
367               None either.
368     """
369
370     # Look for the first empty box
371     for row in grid:
372         for column in grid[grid.index(row)]:
373             if column == 0:
374                 return (grid.index(row), row.index(column))
375
376     # Return None if there is no box left
377     return None
378
379
380 def solve_sudoku(grid):
381     """
382     @return : A solved sudoku if grid is solvable,
383              False if the sudoku is unsolvable,
384              None if the grid is an invalid input.
385     """
386
387     # Check if it's a valid sudoku
388     state = check_sudoku(grid)
389     if (state is None) or (state is False):
390         return state
391
392     # BackTracking Algorithm
393     grid = backTracking(grid)
394
395     return grid
396
397
398 matrix = [not_matrix, not_number_matrix, ill_formed, row_lack, column_lack,
399           valid, invalid, invalid_column, invalid_block, invalid_row, easy,
400           empty, hard, unsolvable]
401
402 for grid in matrix:
403     print(matrix.index(grid))
404     start = time.time()
405     display_sudoku(solve_sudoku(grid))
406     end = time.time()
407     print("%s seconds" %(end - start))

```

Coverage report: 100%

<i>Module ↓</i>	<i>statements</i>	<i>missing</i>	<i>excluded</i>	<i>coverage</i>
SudokuSolver.py	115	0	0	100%
Total	115	0	0	100%

coverage.py v4.5.4, created at 2019-08-15 10:09

FIGURE E.6 – Coverage

E.3.5 Sudoku Solver Bis

```

1  import copy
2  import time
3
4  def check_row(row):
5      t_row = [x for x in row if x != 0]
6      if len(t_row) != len(set(t_row)):
7          return False
8      return True
9
10 def check_sudoku(grid):
11
12     if len(grid) != 9 or type(grid) is not list: #checks if there are 9 columns
13         and the grid is a list
14         print("grid is not a list or number of columns != 9")
15         return None
16
17     for row in grid: #checks if there are 9 rows
18         if len(row) != 9:
19             print("number of rows != 9")
20             return None
21
22     for row in grid:
23         for element in row:
24             if not isinstance(element, (int)): #checks if numbers are intergers
25                 print("numbers are not intergers")
26                 return None
27             elif not (element in [0,1,2,3,4,5,6,7,8,9]): #checks if numbers are
28                 between 0 and 9
29                 print("numbers not between 0 and 9")
30                 return None
31
32     for row in grid:
33         if check_row(row) == False: #checks if all the rows are valid
34             print("some rows are invalid")
35             return False
36
37     trans_grid = zip(*grid)
38     for row in trans_grid:
39         if check_row(row) == False:
40             return False
41
42     for j in [(0,3),(3,6),(6,9)]: #checks each 3x3 subgrids

```

```

41         for i in [(0,3),(3,6),(6,9)]:
42             cuad = [row[i[0]:i[1]] for row in grid[j[0]:j[1]]]
43             l = [v for sub in cuad for v in sub]
44             if check_row(l) == False:
45                 return False
46
47         return True
48
49 def solve_sudoku(grid):
50     state = check_sudoku(grid)
51     if state is None or state is False:
52         return state
53
54     new_grid = copy.deepcopy(grid)
55     # new_grid = grid
56
57     for i in range(9):
58         single_row = new_grid[i]
59         for j in range(9):
60             single_column = [x for x in [new_grid[y][j] for y in range(9)]]
61             if new_grid[i][j] == 0:
62                 # possible values
63                 ns = [x for x in range(1, 10) if x not in single_row and x not
in single_column]
64                 for n in ns:
65                     new_grid[i][j] = n
66                     new = solve_sudoku(new_grid)
67                     if new is not False:
68                         return new
69                 return False
70     return new_grid
71
72
73
74
75 # solve_sudoku should return None
76 ill_formed = [[5,3,4,6,7,8,9,1,2],
77               [6,7,2,1,9,5,3,4,8],
78               [1,9,8,3,4,2,5,6,7],
79               [8,5,9,7,6,1,4,2,3],
80               [4,2,6,8,5,3,7,9], # <—
81               [7,1,3,9,2,4,8,5,6],
82               [9,6,1,5,3,7,2,8,4],
83               [2,8,7,4,1,9,6,3,5],
84               [3,4,5,2,8,6,1,7,9]]
85 # —> None
86
87 # solve_sudoku should return valid unchanged
88 valid = [[5,3,4,6,7,8,9,1,2],
89           [6,7,2,1,9,5,3,4,8],
90           [1,9,8,3,4,2,5,6,7],
91           [8,5,9,7,6,1,4,2,3],
92           [4,2,6,8,5,3,7,9,1],
93           [7,1,3,9,2,4,8,5,6],
94           [9,6,1,5,3,7,2,8,4],
95           [2,8,7,4,1,9,6,3,5],
96           [3,4,5,2,8,6,1,7,9]]
97 # —> True
98

```

```

99 # solve_sudoku should return False
100 invalid = [[5,3,4,6,7,8,9,1,2],
101             [6,7,2,1,9,5,3,4,8],
102             [1,9,8,3,8,2,5,6,7],
103             [8,5,9,7,6,1,4,2,3],
104             [4,2,6,8,5,3,7,9,1],
105             [7,1,3,9,2,4,8,5,6],
106             [9,6,1,5,3,7,2,8,4],
107             [2,8,7,4,1,9,6,3,5],
108             [3,4,5,2,8,6,1,7,9]]
109 # --> False
110
111 # solve_sudoku should return a
112 # sudoku grid which passes a
113 # sudoku checker. There may be
114 # multiple correct grids which
115 # can be made from this starting
116 # grid.
117 easy = [[2,9,0,0,0,0,0,7,0],
118          [3,0,6,0,0,8,4,0,0],
119          [8,0,0,0,4,0,0,0,2],
120          [0,2,0,0,3,1,0,0,7],
121          [0,0,0,0,8,0,0,0,0],
122          [1,0,0,9,5,0,0,6,0],
123          [7,0,0,0,9,0,0,0,1],
124          [0,0,1,2,0,0,3,0,6],
125          [0,3,0,0,0,0,0,5,9]]
126 # --> True
127
128 # Note: this may timeout
129 # in the Udacity IDE! Try running
130 # it locally if you'd like to test
131 # your solution with it.
132
133 hard = [[1,0,0,0,0,7,0,9,0],
134          [0,3,0,0,2,0,0,0,8],
135          [0,0,9,6,0,0,5,0,0],
136          [0,0,5,3,0,0,9,0,0],
137          [0,1,0,0,8,0,0,0,2],
138          [6,0,0,0,0,4,0,0,0],
139          [3,0,0,0,0,0,0,1,0],
140          [0,4,0,0,0,0,0,0,7],
141          [0,0,7,0,0,0,3,0,0]]
142 # --> True
143
144 hard2 = [[0,5,0,0,3,0,0,2,0],
145           [1,0,0,0,0,9,0,0,6],
146           [0,0,0,6,0,7,0,0,0],
147           [0,9,2,0,0,0,1,0,0],
148           [4,0,0,0,0,0,0,0,7],
149           [0,0,5,0,0,0,3,9,0],
150           [0,0,0,7,0,8,0,0,0],
151           [6,0,0,2,0,0,0,0,4],
152           [0,7,0,0,1,0,0,3,0]]
153 # --> True
154
155 hard3 = [[8,0,0,0,0,0,0,0,0],
156           [0,0,3,6,0,0,0,0,0],
157           [0,7,0,0,9,0,2,0,0],

```

```

158     [0,5,0,0,0,7,0,0,0],
159     [0,0,0,0,4,5,7,0,0],
160     [0,0,0,1,0,0,0,3,0],
161     [0,0,1,0,0,0,0,6,8],
162     [0,0,8,5,0,0,0,1,0],
163     [0,9,0,0,0,0,4,0,0]]
164
165 ##### Below I've provided some extra Sudoku grids to check. #####
166
167 valid_but_unsolved = [[5, 3, 0, 0, 7, 0, 0, 0, 0],    # A typical unsolved
168                        Sudoku puzzle (from Wikipedia)
169                        [6, 0, 0, 1, 9, 5, 0, 0, 0],
170                        [0, 9, 8, 0, 0, 0, 0, 6, 0],
171                        [8, 0, 0, 0, 6, 0, 0, 0, 3],
172                        [4, 0, 0, 8, 0, 3, 0, 0, 1],
173                        [7, 0, 0, 0, 2, 0, 0, 0, 6],
174                        [0, 6, 0, 0, 0, 0, 2, 8, 0],
175                        [0, 0, 0, 4, 1, 9, 0, 0, 5],
176                        [0, 0, 0, 0, 8, 0, 0, 7, 9]]
177
178 # --> True
179
180 grid_solved = [[5, 3, 4, 6, 7, 8, 9, 1, 2],    # The same Sudoku puzzle, but
181               solved (from Wikipedia)
182               [6, 7, 2, 1, 9, 5, 3, 4, 8],
183               [1, 9, 8, 3, 4, 2, 5, 6, 7],
184               [8, 5, 9, 7, 6, 1, 4, 2, 3],
185               [4, 2, 6, 8, 5, 3, 7, 9, 1],
186               [7, 1, 3, 9, 2, 4, 8, 5, 6],
187               [9, 6, 1, 5, 3, 7, 2, 8, 4],
188               [2, 8, 7, 4, 1, 9, 6, 3, 5],
189               [3, 4, 5, 2, 8, 6, 1, 7, 9]]
190
191 # --> True
192
193 ##### Most of the Sudoku grids below are my derivations of the above 2 Sudoku
194 puzzles
195
196 grid_not_a_list = ([5, 3, 4, 6, 7, 8, 9, 1, 2],    # Grid is a tuple of lists,
197                  rather than a list of lists
198                  [6, 7, 2, 1, 9, 5, 3, 4, 8],
199                  [1, 9, 8, 3, 4, 2, 5, 6, 7],
200                  [8, 5, 9, 7, 6, 1, 4, 2, 3],
201                  [4, 2, 6, 8, 5, 3, 7, 9, 1],
202                  [7, 1, 3, 9, 2, 4, 8, 5, 6],
203                  [9, 6, 1, 5, 3, 7, 2, 8, 4],
204                  [2, 8, 7, 4, 1, 9, 6, 3, 5],
205                  [3, 4, 5, 2, 8, 6, 1, 7, 9])
206
207 # --> None
208
209 too_few_rows = [[5, 3, 4, 6, 7, 8, 9, 1, 2],    # Only 8 rows
210                 [6, 7, 2, 1, 9, 5, 3, 4, 8],
211                 [1, 9, 8, 3, 4, 2, 5, 6, 7],
212                 [8, 5, 9, 7, 6, 1, 4, 2, 3],
213                 [4, 2, 6, 8, 5, 3, 7, 9, 1],
214                 [7, 1, 3, 9, 2, 4, 8, 5, 6],
215                 [9, 6, 1, 5, 3, 7, 2, 8, 4],
216                 [3, 4, 5, 2, 8, 6, 1, 7, 9]]
217
218 # --> None

```

```

212 row_not_a_list = [(5, 3, 4, 6, 7, 8, 9, 1, 2),      # Grid is a list of tuples,
                    rather than a lists of lists
213                 (6, 7, 2, 1, 9, 5, 3, 4, 8),
214                 (1, 9, 8, 3, 4, 2, 5, 6, 7),
215                 (8, 5, 9, 7, 6, 1, 4, 2, 3),
216                 (4, 2, 6, 8, 5, 3, 7, 9, 1),
217                 (7, 1, 3, 9, 2, 4, 8, 5, 6),
218                 (9, 6, 1, 5, 3, 7, 2, 8, 4),
219                 (2, 8, 7, 4, 1, 9, 6, 3, 5),
220                 (3, 4, 5, 2, 8, 6, 1, 7, 9)]
221 # --> None
222
223 row_incomplete = [[5, 3, 4, 6, 7, 8, 9, 1],      # 1st row is incomplete (only 8
                    elements, rather than 9)
224                  [6, 7, 2, 1, 9, 5, 3, 4, 8],
225                  [1, 9, 8, 3, 4, 2, 5, 6, 7],
226                  [8, 5, 9, 7, 6, 1, 4, 2, 3],
227                  [4, 2, 6, 8, 5, 3, 7, 9, 1],
228                  [7, 1, 3, 9, 2, 4, 8, 5, 6],
229                  [9, 6, 1, 5, 3, 7, 2, 8, 4],
230                  [2, 8, 7, 4, 1, 9, 6, 3, 5],
231                  [3, 4, 5, 2, 8, 6, 1, 7, 9]]
232 # --> None
233
234 non_integer = [[5.0, 3, 0, 0, 7, 0, 0, 0, 0],    # 1st element is a float (5.0)
                rather than an int (5)
235               [6, 0, 0, 1, 9, 5, 0, 0, 0],
236               [0, 9, 8, 0, 0, 0, 0, 6, 0],
237               [8, 0, 0, 0, 6, 0, 0, 0, 3],
238               [4, 0, 0, 8, 0, 3, 0, 0, 1],
239               [7, 0, 0, 0, 2, 0, 0, 0, 6],
240               [0, 6, 0, 0, 0, 0, 2, 8, 0],
241               [0, 0, 0, 4, 1, 9, 0, 0, 5],
242               [0, 0, 0, 0, 8, 0, 0, 7, 9]]
243 # --> None
244
245 out_of_range = [[50, 3, 0, 0, 7, 0, 0, 0, 0],    # 1st element is '50', which is
                out of the range of 0..9
246               [6, 0, 0, 1, 9, 5, 0, 0, 0],
247               [0, 9, 8, 0, 0, 0, 0, 6, 0],
248               [8, 0, 0, 0, 6, 0, 0, 0, 3],
249               [4, 0, 0, 8, 0, 3, 0, 0, 1],
250               [7, 0, 0, 0, 2, 0, 0, 0, 6],
251               [0, 6, 0, 0, 0, 0, 2, 8, 0],
252               [0, 0, 0, 4, 1, 9, 0, 0, 5],
253               [0, 0, 0, 0, 8, 0, 0, 7, 9]]
254 # --> None
255
256 trivial_all_zeros = [[0, 0, 0, 0, 0, 0, 0, 0, 0], # Trivial case of a grid
                      consisting of lists of nothing but zeros.
257                     [0, 0, 0, 0, 0, 0, 0, 0, 0],
258                     [0, 0, 0, 0, 0, 0, 0, 0, 0],
259                     [0, 0, 0, 0, 0, 0, 0, 0, 0],
260                     [0, 0, 0, 0, 0, 0, 0, 0, 0],
261                     [0, 0, 0, 0, 0, 0, 0, 0, 0],
262                     [0, 0, 0, 0, 0, 0, 0, 0, 0],
263                     [0, 0, 0, 0, 0, 0, 0, 0, 0],
264                     [0, 0, 0, 0, 0, 0, 0, 0, 0]]
265 # --> True

```



```

266
267 duplicate_in_row = [[5, 3, 0, 0, 7, 0, 0, 0, 5],      # '5' duplicated in 1st row
268                    [6, 0, 0, 1, 9, 5, 0, 0, 0],
269                    [0, 9, 8, 0, 0, 0, 0, 6, 0],
270                    [8, 0, 0, 0, 6, 0, 0, 0, 3],
271                    [4, 0, 0, 8, 0, 3, 0, 0, 1],
272                    [7, 0, 0, 0, 2, 0, 0, 0, 6],
273                    [0, 6, 0, 0, 0, 0, 2, 8, 0],
274                    [0, 0, 0, 4, 1, 9, 0, 0, 5],
275                    [0, 0, 0, 0, 8, 0, 0, 7, 9]]
276 # --> False
277
278 duplicate_in_col = [[5, 3, 0, 0, 7, 0, 0, 0, 0],      # '5' duplicated in 1st
279                    column
280                    [6, 0, 0, 1, 9, 5, 0, 0, 0],
281                    [0, 9, 8, 0, 0, 0, 0, 6, 0],
282                    [8, 0, 0, 0, 6, 0, 0, 0, 3],
283                    [4, 0, 0, 8, 0, 3, 0, 0, 1],
284                    [7, 0, 0, 0, 2, 0, 0, 0, 6],
285                    [0, 6, 0, 0, 0, 0, 2, 8, 0],
286                    [0, 0, 0, 4, 1, 9, 0, 0, 5],
287                    [5, 0, 0, 0, 8, 0, 0, 7, 9]]
288 # --> False
289
290 duplicate_in_grid = [[5, 3, 0, 0, 7, 0, 0, 0, 0],    # 5' duplicated in 1st
291                    sub-grid
292                    [6, 0, 0, 1, 9, 5, 0, 0, 0],
293                    [0, 9, 5, 0, 0, 0, 0, 6, 0],
294                    [8, 0, 0, 0, 6, 0, 0, 0, 3],
295                    [4, 0, 0, 8, 0, 3, 0, 0, 1],
296                    [7, 0, 0, 0, 2, 0, 0, 0, 6],
297                    [0, 6, 0, 0, 0, 0, 2, 8, 0],
298                    [0, 0, 0, 4, 1, 9, 0, 0, 5],
299                    [0, 0, 0, 0, 8, 0, 0, 7, 9]]
300 # --> False
301
302 no_soln1 = [[1, 2, 3, 4, 5, 6, 7, 8, 0],
303            [0, 0, 0, 0, 0, 0, 0, 0, 9],
304            [0, 0, 0, 0, 0, 0, 0, 0, 0],
305            [0, 0, 0, 0, 0, 0, 0, 0, 0],
306            [0, 0, 0, 0, 0, 0, 0, 0, 0],
307            [0, 0, 0, 0, 0, 0, 0, 0, 0],
308            [0, 0, 0, 0, 0, 0, 0, 0, 0],
309            [0, 0, 0, 0, 0, 0, 0, 0, 0],
310            [0, 0, 0, 0, 0, 0, 0, 0, 0]]
311 # --> False
312
313 no_soln2 = [[1, 2, 3, 0, 0, 0, 0, 0, 0],
314            [4, 5, 0, 0, 0, 0, 6, 0, 0],
315            [0, 0, 0, 6, 0, 0, 0, 0, 0],
316            [0, 0, 0, 0, 0, 0, 0, 0, 0],
317            [0, 0, 0, 0, 0, 0, 0, 0, 0],
318            [0, 0, 0, 0, 0, 0, 0, 0, 0],
319            [0, 0, 0, 0, 0, 0, 0, 0, 0],
320            [0, 0, 0, 0, 0, 0, 0, 0, 0]]
321 # --> False

```

```

322 sudoku_grids = [ill_formed, valid, invalid, easy, hard, hard2, hard3,
    valid_but_unsolved, grid_solved,
323                 grid_not_a_list, too_few_rows, row_not_a_list, row_incomplete,
    non_integer,
324                 out_of_range, trivial_all_zeros, duplicate_in_row,
    duplicate_in_col,
325                 duplicate_in_grid, no_soln1, no_soln2]
326
327 for grid in sudoku_grids:
328     start = time.time()
329     print(solve_sudoku(grid))
330     end = time.time()
331     print("%s seconds" %(end - start))

```

./Code/SudokuSolverBis.py

E.3.6 Sudoku Randomizer

```

1  import random
2
3  from SudokuChecker import check_sudoku
4
5  def random_sudoku():
6      sudoku = []
7      bias = -0.2
8
9      for i in range(9):
10         sudoku.append([])
11
12     for row in sudoku:
13         for i in range(9):
14             if (random.random() < 0.5 + bias):
15                 row.append(random.randint(1,9))
16                 bias -= 0.35
17             else:
18                 row.append(0)
19                 bias += 0.05
20
21     bias = -0.2
22
23     return sudoku
24
25 for i in range(10000):
26     print(check_sudoku(random_sudoku()))

```

./ProblemSets/ProblemSet3/PS3.2/SudokuRandomizer.py

E.4 Fuzzer

```

1  #!/usr/bin/python
2
3  # 5-line fuzzer below is from Charlie Miller's
4  # "Babysitting an Army of Monkeys":
5  # Part 1 - http://www.youtube.com/watch?v=Xnwodi2CBws
6  #
7  # Part 2 - http://www.youtube.com/watch?v=lK5fgCvS2N4
8
9
10 # List of files to use as initial seed
11 file_list = ["C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/AutomatedWhiteBoxTesting.jpg",
12             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/BugTriage.jpg",
13             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/Coverage_PS2_1.PNG",
14             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/Coverage_PS2_2.PNG",
15             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/Coverage_PS3_1.PNG",
16             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/Coverage_Sudoku_Randomizer.PNG",
17             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/CoverageTesting.jpg",
18             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/DefensiveCoding.png",
19             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/DriverScript.jpg",
20             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/EquivalenceClasses.png",
21             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/FuzzingTimeline.jpg",
22             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/GUIApplication.png",
23             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/InputOutput.png",
24             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/InputValidity.jpg",
25             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/NotCovered_Sudoku_Randomizer.PNG",
26             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/RandomSystemTester.jpg",
27             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/RandomTesting.jpg",
28             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/Rep_RandomTesting_1.jpg",
29             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/Rep_RandomTesting_2.jpg",
30             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/TestCase_Reduction.jpg",
31             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/TestingMethodes.png",
32             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/TestingPractice.jpg",
33             "C://Users/Utilisateur/Desktop/Reports/SoftwareTesting/ProblemSets/
    ProblemSet4/Seed/TuningProbability.jpg"] # "x.pdf"

```

```

34
35 # List of applications to test
36 apps = "C://Program Files/GIMP 2/bin/gimp-2.10.exe" # "/path/application"
37
38 fuzz_output = "fuzz.jpg"
39
40 FuzzFactor = 100
41 num_tests = 2880
42
43 bugs = {}
44 bugsOcc = {}
45
46 ##### end configuration #####
47
48 import math
49 import random
50 import string
51 import subprocess
52 import time
53 import sys
54
55 log = "-" * 100
56 log += "\n"
57 log += ' ' * ((100 - len("_____ START FUZZING _____") - 1) / 2)
58 log += "_____ START FUZZING _____"
59 log += "\n"
60 log += "-" * 100
61 log += "\n"
62
63 fh = open("./log.txt", 'w')
64
65 fh.close()
66
67 start = time.time()
68
69 for i in range(num_tests):
70     file_choice = random.choice(file_list)
71
72     if file_choice[-4:] == ".jpg":
73         fuzz_output = fuzz_output[:-4] + ".jpg"
74     elif file_choice[-4:] == ".png":
75         fuzz_output = fuzz_output[:-4] + ".png"
76     elif file_choice[-4:] == ".PNG":
77         fuzz_output = fuzz_output[:-4] + ".PNG"
78
79     app = apps
80
81     buf = bytearray(open(file_choice, 'rb').read())
82
83     # Start Charlie Miller code
84     numwrites = random.randrange(math.ceil((float(len(buf)) / FuzzFactor))) + 1
85
86     # Because of Tuning Rules and Probabilities
87     fuzz_choice = random.choice(['start', 'end', 'middle', 'random'])
88     begin = None
89     if fuzz_choice == 'start':
90         begin = 0
91     if fuzz_choice == 'end':
92         begin = len(buf) - numwrites - 1

```

```

93     if fuzz_choice == 'middle':
94         begin = random.randrange(len(buf)-numwrites)
95
96     for j in range(numwrites):
97         rbyte = random.randrange(256)
98
99         if begin is None:
100             rn = random.randrange(len(buf))
101         else:
102             rn = begin
103             begin += 1
104
105         buf[rn] = "%c"%(rbyte)
106
107     # End Charlie Miller code
108
109     open(fuzz_output, 'wb').write(buf)
110
111     print("Fuzzed App : %s \n File: %s Fuzz Type: %s #writes=%d" % (app,
112 file_choice, fuzz_choice, numwrites))
113
114     process = subprocess.Popen([app, fuzz_output], stdout=subprocess.PIPE, stderr
115 =subprocess.PIPE)
116
117     time.sleep(10)
118     crashed = process.poll()
119
120     if not crashed:
121         process.terminate()
122
123         errout = process.stderr.read()
124
125         if not errout and FuzzFactor:
126             FuzzFactor -= 100
127         elif FuzzFactor > 100:
128             FuzzFactor += 100
129
130         if FuzzFactor >= 100:
131             FuzzFactor = 101
132
133         code = ""
134         flag1 = -1
135         flag2 = -1
136         flag3 = -1
137         flag4 = -1
138         for i in range(len(errout)):
139             code += errout[i]
140
141             if flag1 == -1 and errout[i] in "0123456789" and code[i-15:-1] == "
142 (script-fu.exe:":
143                 flag1 = i - 15
144
145             if flag1 != -1 and flag2 == -1 and errout[i] == ':':
146                 flag2 = i
147
148             if flag2 != -1 and flag3 == -1 and errout[i] in "0123456789":
149                 flag3 = i - 24

```

```

149         if flag3 != -1 and flag4 == -1 and errout[i] == ':' and errout[i+1]
    == ',':
150             flag4 = i
151
152             err_tmp = errout[:flag1 - 1]
153             err_code = errout[flag1:flag2]
154             err_time = errout[flag3:flag4]
155             err_msg = errout[flag4 + 1:]
156
157             checkList = []
158
159             for meta in bugs.values():
160                 checkList.append(meta[0])
161
162             if bugs == {} or (err_tmp not in checkList and file_choice not in bugs.
    keys()):
163                 bugs[file_choice] = [[err_tmp, app, fuzz_choice, numwrites, err_code
    , err_time, err_msg]]
164                 elif err_tmp not in checkList and file_choice in bugs.keys():
165                     bug_tmp = bugs[file_choice]
166                     bug_tmp.append([err_tmp, app, fuzz_choice, numwrites, err_code,
    err_time, err_msg])
167                     bugs[file_choice] = bug_tmp
168                 elif err_tmp in checkList and file_choice not in bugs.keys():
169                     bugs[file_choice] = [[err_tmp, app, fuzz_choice, numwrites, err_code
    , err_time, err_msg]]
170
171                 if err_tmp not in bugsOcc.keys():
172                     bugsOcc[err_tmp] = 1
173                 else:
174                     bugsOcc[err_tmp] += 1
175
176 end = time.time()
177
178 print("Number of Process : %i"%(num_tests))
179
180 log += "-"*100
181 log += "\n"
182
183
184 for key in bugsOcc:
185     for value in bugs:
186         for error in bugs[value]:
187             if key in [error][0]:
188                 middleValue = int(math.ceil(len(value)/2))
189                 value1 = value[:middleValue - 1]
190                 value2 = value[middleValue:]
191                 log += "Fuzzed App : %s \nFile: %s\n%s Fuzz Type: %s #writes=%d\
    n"%(error[1], value1, value2,
192
193                     error[2], error[3])
194
195                 log += "-"*100
196                 log += "\n"
197
198                 log += "ERROR : \n\n %s\n\n"%(key)
199
200     for value in bugs:
201         for error in bugs[value]:

```

```

201         if key in [error][0]:
202             log += "%s:%s:%s"%(error[4], error[5], error[6])
203
204
205         log += "Occurs : %i"%(bugsOcc[key])
206
207         log += "\n"
208         log += "-" * 100
209         log += "\n"
210
211     fh = open("./log.txt", 'a')
212
213     fh.write(log)
214
215     fh.close()
216
217     log = "-" * 100
218     log += "\n"
219     log += ' '*((100 - len("xx.xxxxxx sec - xxxxxx Tests"))/2)
220     log += "%f sec - %i Tests"%((end - start), num_tests)
221     log += "\n"
222     log += ' '*((100 - len("_____ END FUZZING _____"))/2)
223     log += "_____ END FUZZING _____"
224     log += "\n"
225     log += "-" * 100
226     log += "\n"
227
228     fh = open("./log.txt", 'a')
229
230     fh.write(log)
231
232     fh.close()

```

./ProblemSets/ProblemSet4/Fuzzer.py

Coverage for **SudokuRandomizer.py** : 96%

114 statements 110 run 4 missing 0 excluded

FIGURE E.7 – Coverage

```

88     # Checking the sanity check
89     if not isinstance(grid, (list)): # Checks that grid is a list
90         return None
91
92     if len(grid) != 9: # Checks the number of columns
93         return None
94
95     for row in grid:
96         if not isinstance(row, (list)) or len(row) != 9: # Checks that row is a
97             return None                                     # list and the number
98                                                         # of lines
99
100     for element in row:
101         if element not in [0,1,2,3,4,5,6,7,8,9]: # Checks that elements
102             return None                               # are in the domain

```

FIGURE E.8 – Coverage