

Software Testing: Project

Simran Basu
MT2020145

Jyotsna Tata
MT2020020

Md Aqueeb Jawed
MT2020110

Abstract—This report contains the implementation of test cases on a project using open source tools and the test case design strategies.

I. PROBLEM STATEMENT

Mutation-source-code: Projects that use mutation testing, based on mutation operators applied at the level of a statement within a method or a function. The mutated program needs to be strongly killed by the designed test cases. At least three different mutation operators should be used.

II. METHODOLOGY

For this project, A python program for scientific calculator was used which uses many functions such as Sine, Cosine, Logarithm, Factorial, Permutations, etc. The test cases were first designed and built for unit testing of each of these functions. Unit testing is a software testing method by which individual units of source code—sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures—are tested to determine whether they are fit for use. Python's unittest library was used for that. It is a unit testing framework was originally inspired by JUnit and has a similar flavor as major unit testing frameworks in other languages. It supports test automation, sharing of setup and shutdown code for tests, aggregation of tests into collections, and independence of the tests from the reporting framework.

testing tool in Python that generated mutants and computes a mutation score. It supports standard unittest module, generates YAML/HTML reports and has colorful output. After running mut.py, it was found that for 6 different mutation operators, mutants were strongly killed. The mutation operators used were:

- 1) ROR (Relational Operator Replacement): Each occurrence of one of the relational operators (`<`, `<=`, `>`, `>=`, `==`, `!=`) is replaced by each of the other operators and by `falseOp` and `trueOp`.

```

121: # 202: ROM Calculation 2:
122: w = []
123: b = []
124: C = input('Enter the numbers (max of 7 numbers) separated by a space: ')
125: c = c.split()
126: if len(a) == 0:
127:     if len(a) != 0:
128:         return 0
129:     for i in a:
130:         b.append(int(i))
131: arr = np.array(b)
132: self.cube_root = tk.Button(bottom_frame, text = '\n1\ ', **btn_parameters, command = lambda: self.btn_click('\factorial(\ '))
133: self.mod.grid(row=3, column = 0)
134:
135: #cube root button
136: self.cube_root = tk.Button(bottom_frame, text = '\x3\ ', **btn_parameters, command = lambda: self.btn_click('\**(1/3)\ '))
137: self.cube_root = tk.Button(bottom_frame, text = '\x3/2\ ', **btn_parameters, command = lambda: self.btn_click('\**(2/3)\ '))
138: self.cube_root.grid(row=3, column = 1)
139:
140: #cube button
141: self.cube = tk
142: self.cube = tk.Button(bottom_frame, text = '\x3\ ', **btn_parameters, command = lambda: self.btn_click('\**3\ '))
143: self.frac = tk.Button(bottom_frame, **btn_parameters, text = '\x/y\ ', command = '\**\ ')
144: self.frac.grid(row = 4, column = 0)
145:
146: #square root button
147: self.square_root = tk.Button(bottom_frame, **btn_parameters, text = '\x1/2\ ', command = lambda: self.btn_click('\sqrt\ '))
148: self.square_root = tk.Button(bottom_frame, **btn_parameters, text = '\x3/4\ ', command = lambda: self.btn_click('\sqrt[4]\ '))
149: self.square_root.grid(row = 4, column = 1)
150:
151: #square button
152: self.square = tk.Button(bottom_frame, **btn_parameters, text = '\x2\ ', command = lambda: self.btn_click('\**2\ '))
153:
154:

```

Fig. 2. Relational Operator Replacement.

- 2) CRP (Constant Replacement): Each occurrence of a constant (1,0,2,-1) is replaced by other constants similar to it.

```

.....
95:         return 'Wrong Input!'
96:     else
97:
98: def Sqrt(x):
99:     return x ** 0.5
100:     return x ** 1.5
101:
102:
103: def Log(x):
104:     return m.log(x)
105:
106:     self.mod = tk.Button(bottom_frame, text = '\n1\'.', **btn_parameters, command = lambda: self.btn_click('\Factorial(V')
107:     self.mod.grid(row=3, column = 0)
108:
109:
110:     wcube_root = tk.Button(bottom_frame, text = '\sqrt\'.', **btn_parameters, command = lambda: self.btn_click('\sqrt(3)/\')
111:     self.cube_root = tk.Button(bottom_frame, text = '\sqrt[3]{x}\'.', **btn_parameters, command = lambda: self.btn_click('\sqrt[3](x)/\')
112:     self.cube_root.grid(row=3, column = 1)
113:
114:
115:     wcube_button
116:     self.cube_root = tk.Button(bottom_frame, text = '\sqrt[3]{x}\'.', **btn_parameters, command = lambda: self.btn_click('\sqrt[3](x)/\')
117:     self.frac = tk.Button(bottom_frame, **btn_parameters, text = '\sqrt{x}/\'.', command = '\sqrt{x}/\'.')
118:     self.frac.grid(row = 4, column = 0)
119:
120:
121:     tk.Button(bottom_frame, **btn_parameters, text = '\sqrt{x}\'.', command = lambda: self.btn_click('\sqrt{x}/\')
122:     self.root_x = tk.Button(bottom_frame, **btn_parameters, text = '\sqrt[3]{x}\'.', command = lambda: self.btn_click('\sqrt[3](x)/\')
123:     self.root_x.grid(row = 4, column = 1)
124:
125:
126:     square_button
127:     self.square = tk.Button(bottom_frame, **btn_parameters, text = '\sqrt{x}^2\'.', command = lambda: self.btn_click('\sqrt{x}^2/\')
128:     self.squared = tk.Button(bottom_frame, **btn_parameters, text = '\sqrt{x}^2\'.', command = lambda: self.btn_click('\sqrt{x}^2/\')
129:
130:
131:
132:
133:
134:
135:
136:
137:
138:
139:
140:
141:
142:
143:
144:
145:
146:
147:
148:
149:
150:
151:
152:
153:
154:
155:
156:
157:
158:
159:
160:
161:
162:
163:
164:
165:
166:
167:
168:
169:
170:
171:
172:
173:
174:
175:
176:
177:
178:
179:
180:
181:
182:
183:
184:
185:
186:
187:
188:
189:
190:
191:
192:
193:
194:
195:
196:
197:
198:
199:
200:
201:
202:
203:
204:
205:
206:
207:
208:
209:
210:
211:
212:
213:
214:
215:
216:
217:
218:
219:
220:
221:
222:
223:
224:
225:
226:
227:
228:
229:
230:
231:
232:
233:
234:
235:
236:
237:
238:
239:
240:
241:
242:
243:
244:
245:
246:
247:
248:
249:
250:
251:
252:
253:
254:
255:
256:
257:
258:
259:
260:
261:
262:
263:
264:
265:
266:
267:
268:
269:
270:
271:
272:
273:
274:
275:
276:
277:
278:
279:
280:
281:
282:
283:
284:
285:
286:
287:
288:
289:
290:
291:
292:
293:
294:
295:
296:
297:
298:
299:
300:
301:
302:
303:
304:
305:
306:
307:
308:
309:
310:
311:
312:
313:
314:
315:
316:
317:
318:
319:
320:
321:
322:
323:
324:
325:
326:
327:
328:
329:
330:
331:
332:
333:
334:
335:
336:
337:
338:
339:
340:
341:
342:
343:
344:
345:
346:
347:
348:
349:
350:
351:
352:
353:
354:
355:
356:
357:
358:
359:
360:
361:
362:
363:
364:
365:
366:
367:
368:
369:
370:
371:
372:
373:
374:
375:
376:
377:
378:
379:
380:
381:
382:
383:
384:
385:
386:
387:
388:
389:
390:
391:
392:
393:
394:
395:
396:
397:
398:
399:
400:
401:
402:
403:
404:
405:
406:
407:
408:
409:
410:
411:
412:
413:
414:
415:
416:
417:
418:
419:
420:
421:
422:
423:
424:
425:
426:
427:
428:
429:
430:
431:
432:
433:
434:
435:
436:
437:
438:
439:
440:
441:
442:
443:
444:
445:
446:
447:
448:
449:
450:
451:
452:
453:
454:
455:
456:
457:
458:
459:
460:
461:
462:
463:
464:
465:
466:
467:
468:
469:
470:
471:
472:
473:
474:
475:
476:
477:
478:
479:
480:
481:
482:
483:
484:
485:
486:
487:
488:
489:
490:
491:
492:
493:
494:
495:
496:
497:
498:
499:
500:
501:
502:
503:
504:
505:
506:
507:
508:
509:
510:
511:
512:
513:
514:
515:
516:
517:
518:
519:
520:
521:
522:
523:
524:
525:
526:
527:
528:
529:
530:
531:
532:
533:
534:
535:
536:
537:
538:
539:
540:
541:
542:
543:
544:
545:
546:
547:
548:
549:
550:
551:
552:
553:
554:
555:
556:
557:
558:
559:
560:
561:
562:
563:
564:
565:
566:
567:
568:
569:
570:
571:
572:
573:
574:
575:
576:
577:
578:
579:
580:
581:
582:
583:
584:
585:
586:
587:
588:
589:
590:
591:
592:
593:
594:
595:
596:
597:
598:
599:
600:
601:
602:
603:
604:
605:
606:
607:
608:
609:
610:
611:
612:
613:
614:
615:
616:
617:
618:
619:
620:
621:
622:
623:
624:
625:
626:
627:
628:
629:
630:
631:
632:
633:
634:
635:
636:
637:
638:
639:
640:
641:
642:
643:
644:
645:
646:
647:
648:
649:
650:
651:
652:
653:
654:
655:
656:
657:
658:
659:
660:
661:
662:
663:
664:
665:
666:
667:
668:
669:
670:
671:
672:
673:
674:
675:
676:
677:
678:
679:
680:
681:
682:
683:
684:
685:
686:
687:
688:
689:
690:
691:
692:
693:
694:
695:
696:
697:
698:
699:
700:
701:
702:
703:
704:
705:
706:
707:
708:
709:
710:
711:
712:
713:
714:
715:
716:
717:
718:
719:
720:
721:
722:
723:
724:
725:
726:
727:
728:
729:
730:
731:
732:
733:
734:
735:
736:
737:
738:
739:
740:
741:
742:
743:
744:
745:
746:
747:
748:
749:
750:
751:
752:
753:
754:
755:
756:
757:
758:
759:
760:
761:
762:
763:
764:
765:
766:
767:
768:
769:
770:
771:
772:
773:
774:
775:
776:
777:
778:
779:
780:
781:
782:
783:
784:
785:
786:
787:
788:
789:
790:
791:
792:
793:
794:
795:
796:
797:
798:
799:
800:
801:
802:
803:
804:
805:
806:
807:
808:
809:
810:
811:
812:
813:
814:
815:
816:
817:
818:
819:
820:
821:
822:
823:
824:
825:
82
```

Fig. 3. Constant Replacement.

3) COI (Conditional Operator Insertion): Each occurrence of each logical operator (and-&&, or——, and with no conditional evaluation-&, or with no conditional evaluation- —, not equivalent- $\hat{=}$) is inserted.

```
Anaconda Powershell Prompt (anaconda)
(base) PS F:\Python-Scientific-Calculator-master> python -m unittest test.py
.....[-1.  1.5]
...
-----
Ran 24 tests in 0.013s

OK
(base) PS F:\Python-Scientific-Calculator-master>
```

Fig. 1. Unit Testing.

After ensuring that none of the test cases for unit testing were failing, mutation testing was used. Mutation testing is used to design new software tests and evaluate the quality of existing software tests. Mutation testing involves modifying a program in small ways. Each mutated version is called a mutant and tests detect and reject mutants by causing the behavior of the original version to differ from the mutant. Python's mutpy was used for this. Mutpy is a Mutation

Fig. 4. Conditional Operator Insertion.

Fig. 7. Arithmetic Operator Deletion.

Fig. 5. Assignment Operator Replacement.

Fig. 6. Arithmetic Operator Replacement.

6) AOD (Arithmetic Operator Deletion): Each occurrence of one of the arithmetic operators +, -, *, /, ** and % is deleted.

A total of 203 mutants were generated out of which 42 mutants were strongly killed with a Mutation Score of 22.2%. We were able to target 4 different classes of mutation operators, namely Relational, Conditional, Arithmetic, Assignment Operators, along with Constant Replacement. The detailed report for the same can be viewed in the HTML file namely "index.html" in the Mutation_Documentation folder.

- [1] Source Code: Python Scientific Calculator - <https://github.com/samuelabidemi/Python-Scientific-Calculator.git>.
- [2] unittest: Unit testing framework - <https://docs.python.org/3/library/unittest.html>.
- [3] Mutation Testing using Mutpy Module in Python - <https://www.geeksforgeeks.org/mutation-testing-using-mutpy-module-in-python/>.
- [4] MutPy - <https://pypi.org/project/MutPy/>.