

# Test-Driven

## Development

YEGOR BUGAYENKO

Lecture #13 out of 16

90 minutes

All videos are in [this YouTube playlist](#).

All visual and text materials presented in this slidedeck are either originally made by the author or taken from public Internet sources, such as website. Copyright belongs to their respected authors.

The Psychology of Testing

Test Driven Development (TDD)

Unit vs. Integration Tests

Test Coverage

Automated Performance Testing

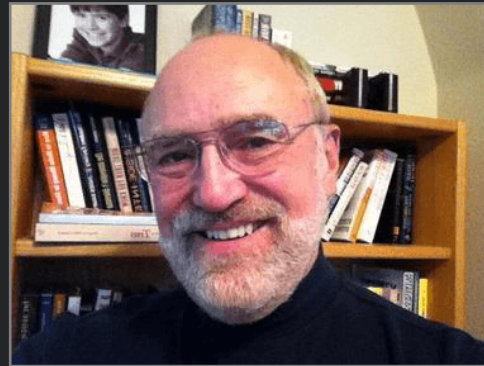
Behavior Driven Development (BDD)

Testing vs. QA

Books, Venues, Call-to-Action

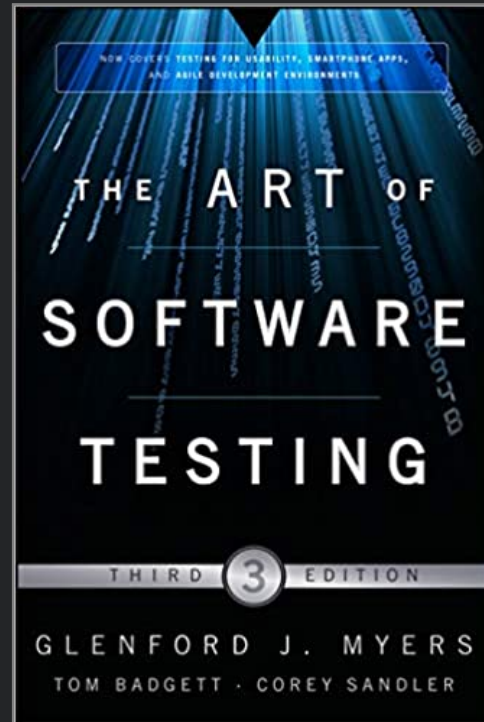
Chapter #1:

# The Psychology of Testing



“One of the primary causes of poor application testing is the fact that most programmers begin with a false definition of the term. They might say: ‘Testing is the process of demonstrating that errors are not present.’”

— Glenford J. Myers, *The Art of Software Testing*



“Don’t test a program to show that it works; rather, start with the assumption that the program contains errors. Testing is the process of executing a program with the intent of finding errors.”

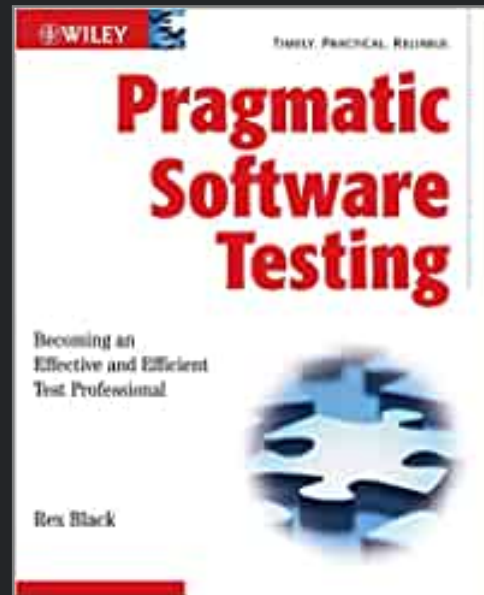
— *The Art of Software Testing*, Glenford J. Myers et al.

“If something is to be delivered, then it is the testers who make the final decision as to whether or not that something is delivered into the live environment.” —**Nick Sewell**, *How to Test a System That Is Never Finished*, 2009

“Testing is an essential activity in software engineering. In the simplest terms, it amounts to observing the execution of a software system to validate whether it behaves as intended ...” —**Antonia Bertolino**, *Software testing research: Achievements, challenges, dreams*, 2007

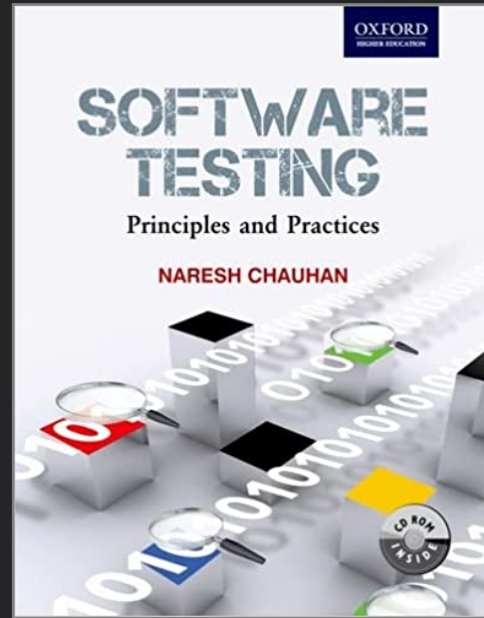
“Software testing is the process of executing a software system to determine whether it matches its specification and executes in its intended environment.” —**James A. Whittaker**, *What Is Software Testing? And Why Is It So Hard?*, 2000

“We distinguish the four major testing models... One model says we test to demonstrate that some version of the software satisfies its specification, two models say we test to detect faults, and the fourth says we test to prevent faults. These three goals need not conflict and, in fact, are all present in the prevention model.” —**David Gelperin**, *The growth of software testing*, 1988



“Software testing is not about proving conclusively that the software is free from any defects, or even about discovering all the defects. Such a mission for a test team is truly impossible to achieve.”

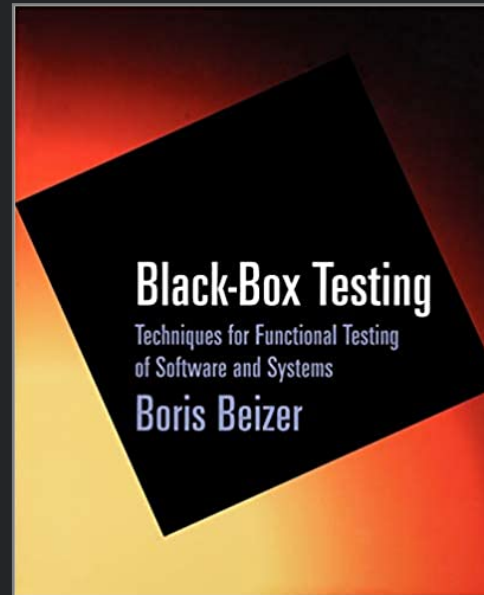
— Rex Black, *Pragmatic Software Testing: Becoming Effective and Efficient Test Professional*



“The immediate goal of testing is to find errors at any stage of software development. More the bugs discovered at an early stage, better will be the success rate of software testing.”

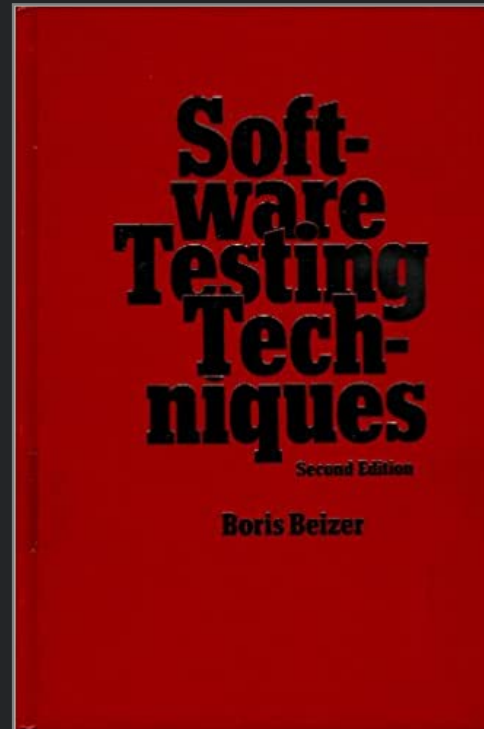
— Naresh Chauhan, *Software Testing: Principles and Practices*





“Anything written by people has bugs. Not testing something is equivalent to asserting that it’s bug-free. Programmers can’t think of everything especially of all the possible interactions between features and between different pieces of software. We try to break software because that’s the only practical way we know of to be confident about the product’s fitness for use.”

— Boris Beizer, *Black-Box Testing: Techniques for Functional Testing of Software and Systems*



“The probability of showing that the software works decreases as testing increases; that is, the more you test, the likelier you are to find a bug. Therefore, if your objective is to demonstrate a high probability of working, that objective is best achieved by not testing at all!”

— Beizer, Boris, *Software Testing Techniques*



“Program testing can be used to show the presence of bugs, but never to show their absence!”

— Edsger W. Dijkstra, *Notes on Structured Programming*, 1970

*Software Testing Philosophy*

Bee Mobile Meetup

Moscow, Russia, 7 November 2018



*Software Testing Pitfalls*

JPoint

Moscow, Russia, 5 April 2019



*Testing and Testers*

TestCon

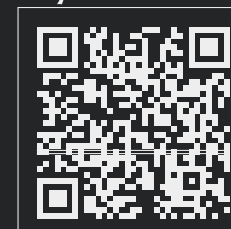
Moscow, Russia, 16 September 2020



*Quality Assurance vs. Testing*

QA Fest

Kyiv, Ukraine, 20 September 2019



Chapter #2:

## Test Driven Development (TDD)

```
1 (test one
2   (is (= 1 (f 1))))
3 (test two
4   (is (= 1 (f 2))))
5 (test fifteen
6   (is (= 610 (f 15))))
```

Can you put some code here?

```
7 (test one
8   (is (= 1 (f 1))))
9 (test two
10  (is (= 1 (f 2))))
11 (test fifteen
12  (is (= 610 (f 15))))
```

```
13 (defun f (n)
14   (cond
15     ((= n 1) 0)
16     ((= n 2) 1)
17     (+
18      (f (- n 1))
19      (f (- n 2)))))
```



“Test-first fundamentalism is like abstinence-only sex ed: An unrealistic, ineffective morality campaign for self-loathing and shaming.”

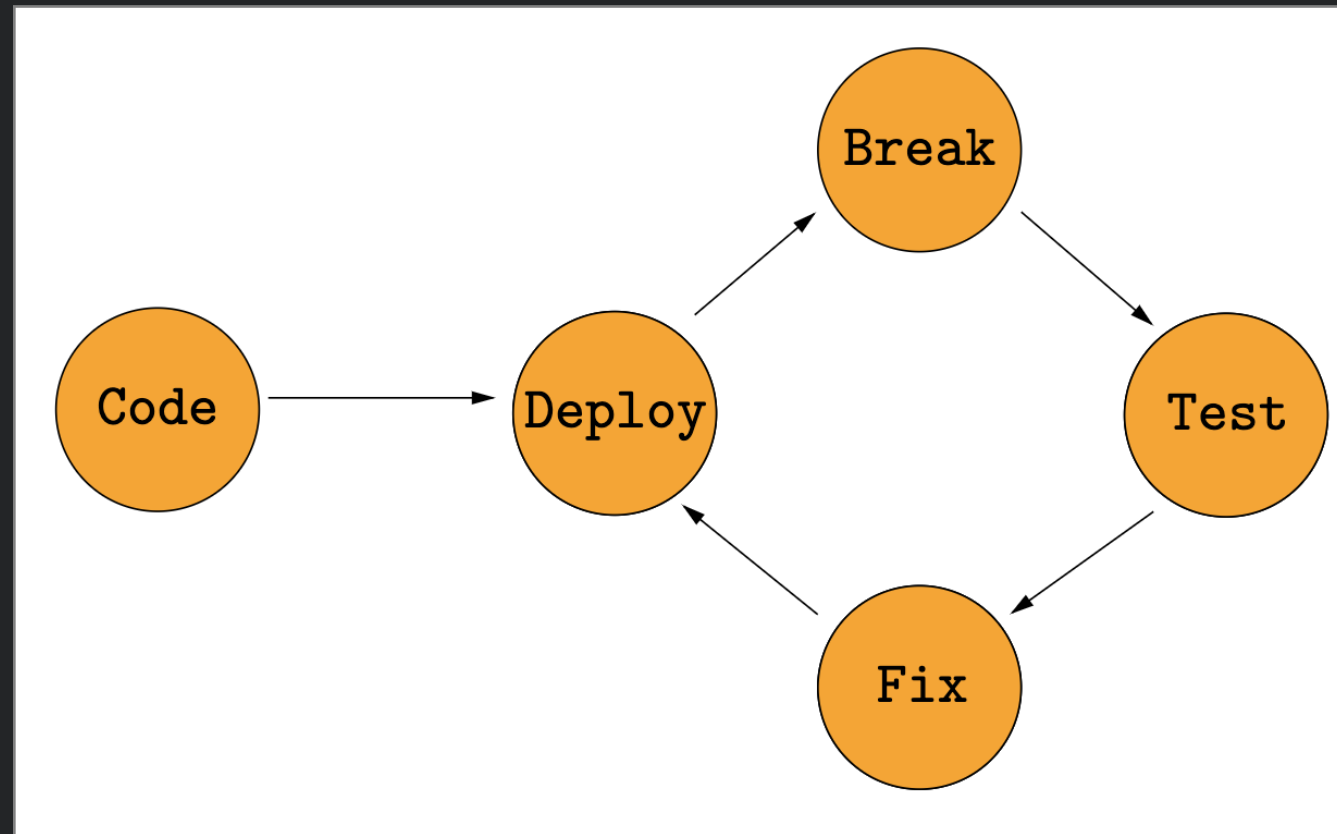
— David Heinemeier Hansson, *TDD is dead. Long live testing*





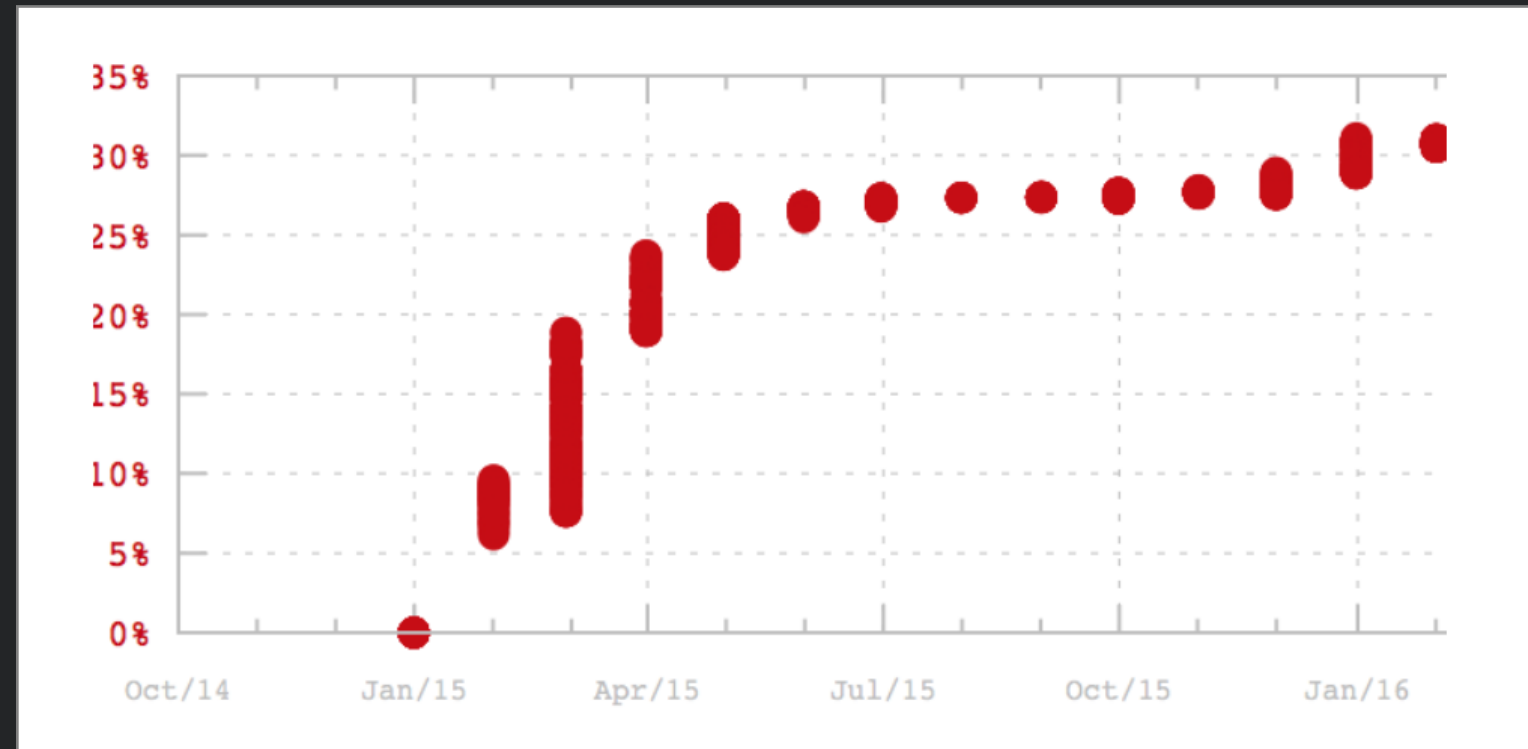
“It would not surprise me if, one day, TDD had the force of law behind it.”

— Robert Martin, *Professionalism and TDD (Reprise)*



“I only create tests later when my users express the need for them by reporting bugs.”

<https://www.yegor256.com/2017/03/24/tdd-that-works.html>



“ I don’t need tests at the beginning of the project”

<https://www.yegor256.com/2017/03/24/tdd-that-works.html>

## Safety Net

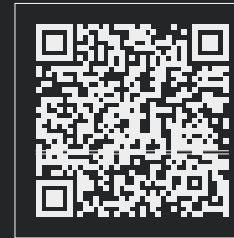


Chapter #3:

## Unit vs. Integration Tests

## Good Tests Are:

- 1) **Short:** less than  $x$  lines each
- 2) **Fast:** less than  $y$  milliseconds each
- 3) **Independent:** runs alone and in a suite
- 4) **Portable:** runs on your laptop and on mine
- 5) **Careful:** side effect free, doesn't leave temp files
- 6) **Isolated:** doesn't touch my files



<https://www.kenneth-truymers.net/2012/12/15/key-qualities-of-a-good-unit-test/> →

```
20 import java.nio.file.Files;
21 class Book {
22     String title() {
23         return Files.readAllLines(
24             Paths.get("/my-data/book.txt")
25         )[0];
26     }
27 }
```

```
28 import org.junit.jupiter.api.Assertions;
29 import org.junit.jupiter.api.Test;
30 class BookTest {
31     @Test
32     void canRetrieveTitle() {
33         String t = new Book().title();
34         Assertions.assertEquals(
35             "Object Thinking", t
36         );
37     }
38 }
```

Is it Short, Fast, Independent, Portable, Careful, and Isolated?

```
39 import java.nio.file.Files;
40 class Book {
41     private Path file;
42     Book(Path f) {
43         this.file = f;
44     }
45     String title() {
46         return Files.readAllLines(
47             this.file
48         )[0];
49     }
50 }
```

```
51 import org.junit.jupiter.api.Assertions;
52 import org.junit.jupiter.api.Test;
53 class BookTest {
54     @Test
55     void canRetrieveTitle() {
56         Path f = Paths.get("/tmp/temp.txt");
57         String title = "Object Thinking";
58         Files.write(f, title.getBytes());
59         String t = new Book().title();
60         Assertions.assertEquals(title, t);
61     }
62 }
```

Is it Short, Fast, Independent, Portable, Careful, and Isolated?



```
63 import java.nio.file.Files;
64 class Book {
65     private Path file;
66     Book(Path f) {
67         this.file = f;
68     }
69     String title() {
70         return Files.readAllLines(
71             this.file
72         )[0];
73     }
74 }
```

```
75 import org.junit.jupiter.api.Assertions;
76 import org.junit.jupiter.api.Test;
77 import org.junit.jupiter.api.io.TempDir;
78 class BookTest {
79     @Test
80     void canRead(@TempDir Path dir) {
81         Path f = dir.resolve("temp.txt");
82         String title = "Object Thinking";
83         Files.write(f, title.getBytes());
84         String t = new Book().title();
85         Assertions.assertEquals(title, t);
86     }
87 }
```

Chapter #4:

## Test Coverage

```
195     loop do
196       json = @api.block(block)
197       if json[:orphan]
198         steps = 4
199         @log.info("Orphan block found at #{block}, moving #{steps} steps back...")
200         wrong << block
201         steps.times do
202           block = json[:previous]
203           wrong << block
204           @log.info("Moved back to #{block}")
205           json = @api.block(block)
206         end
207       next
208     end
209     checked = 0
210     checked_outputs = 0
211     json[:txns].each do |t|
212       t[:outputs].each_with_index do |o, i|
213         address = o[:address]
214         checked_outputs += 1
```

<https://codecov.io/gh/yegor256/sibit/tree/master/lib>

## Coverage Criteria

### **Function** Coverage

Has each function (or subroutine) in the program been called?

### **Statement** Coverage

Has each statement in the program been executed?

### **Edge** Coverage

Has every edge in the control-flow graph been executed?

### **Branch** Coverage

Has each branch of each control structure been executed?

### **Condition** Coverage

Has each Boolean sub-expression evaluated both to true and false?

## Mutation Testing + Coverage

```
88 | def f(n)  
89 |     n * n + 1  
90 | end
```

```
91 | f(0) == 1  
92 | f(1) == 2  
93 | f(2) == 5
```

## Mutation Testing + Coverage

```
94 def f(n)
95   n * n + 1
96 end
97
98 # Mutant no. 1
99 def f(n)
100   n + n + 1
101 end
102
103 # Mutant no. 2
104 def f(n)
105   n * n - 1
106 end
```

```
107 f(0) == 1
108 f(1) == 2
109 f(2) == 5
```

Test coverage: 100%

Mutation coverage = 50%

Chapter #5:

## Automated Performance Testing

## Some Tools

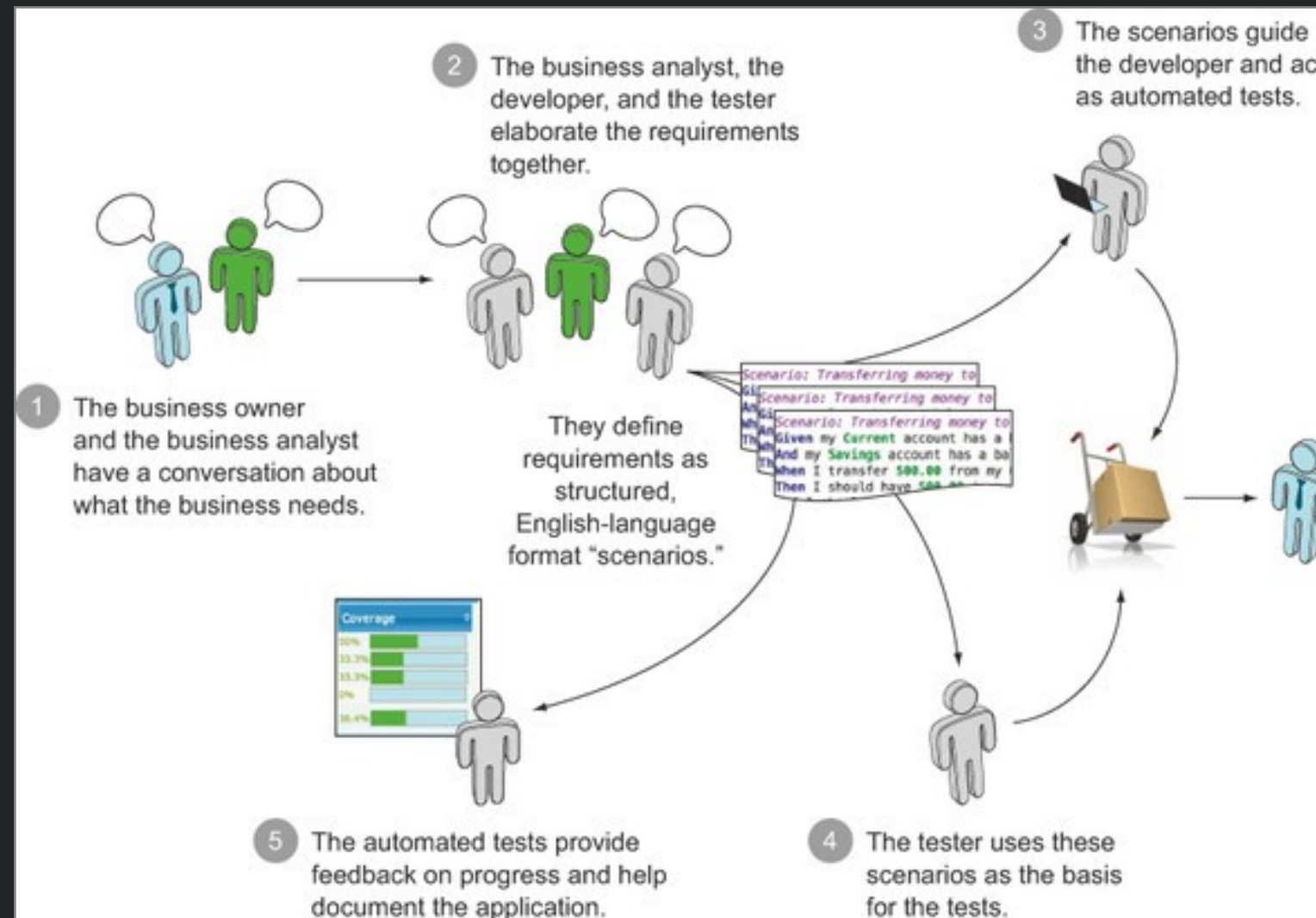
WebLOAD  
LoadNinja  
ReadyAPI Performance  
LoadView  
StormForge  
Keysight's Eggplant  
Apache JMeter  
LoadRunner  
Appvance

NeoLoad  
LoadComplete  
WAPT  
Loadster  
k6  
Rational Performance Tester  
Testing Anywhere  
Apache Bench



Chapter #6:

## Behavior Driven Development (BDD)



<https://www.agilealliance.org/glossary/bdd>

Chapter #7:

## Testing vs. QA



<https://www.youtube.com/watch?v=jZitXMQaXvE> →

## Testing $\neq$ Quality Assurance (QA)



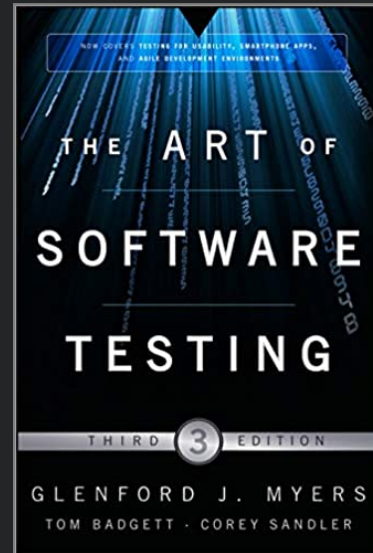
Quality Assurance is the process of auditing the quality requirements and the results from quality control measurements to ensure that appropriate quality standards and operational definitions are used.



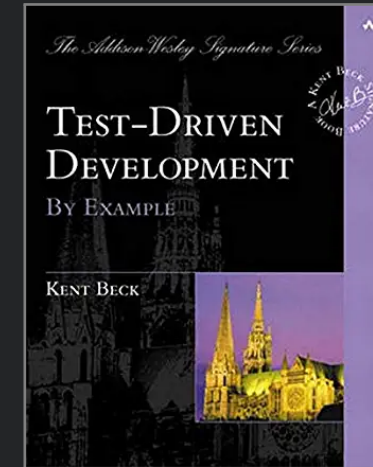
Quality Control is the process of monitoring and recording results of executing the quality activities to assess performance and recommend necessary changes.

Chapter #8:

## Books, Venues, Call-to-Action



“The Art of Software Testing, 3rd Edition” by GLENFORD J. MYERS ET AL.



“Test Driven Development: By Example” by KENT BECK

## Where to publish:

International Symposium on Software Testing and Analysis (ISSTA)

## Call to Action:

Integrate mutation coverage control into your build.



## Still unresolved issues:

- How to test performance right?
- How to create tests automatically?
- How to motivate programmers write tests?
- How to control the quality of testing?