Test-Driven

Development

Yegor Bugayenko

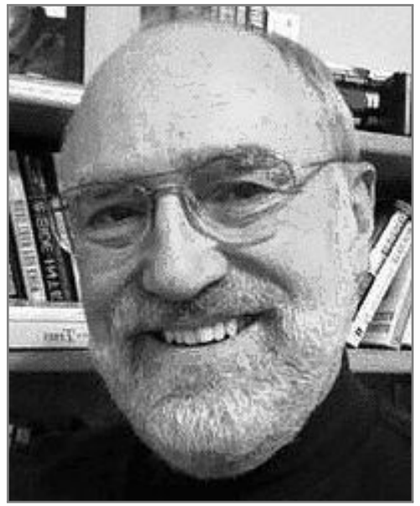
Lecture #13 out of 16
80 minutes

The slidedeck was presented by the author in this YouTube Video

- 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

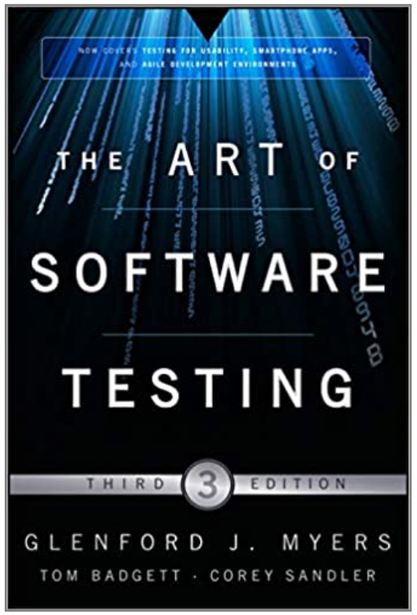
Chapter #1:

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.”

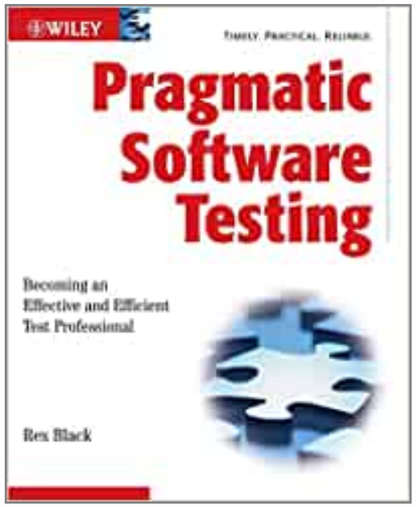
— Glenford J. Myers, Tom Badgett, Todd M. Thomas, and Corey Sandler. *The Art of Software Testing*. Wiley, 2 edition, 2012. doi:10.5555/2161638

"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

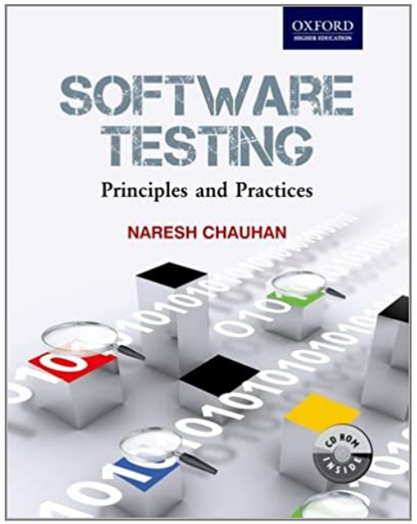
"Software testing is the process of executing a software system to determine whether it matches its specification and executes in its intended environment." —James A. Whittaker, *What Is Software Testing? And Why Is It So Hard?*, 2000

"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



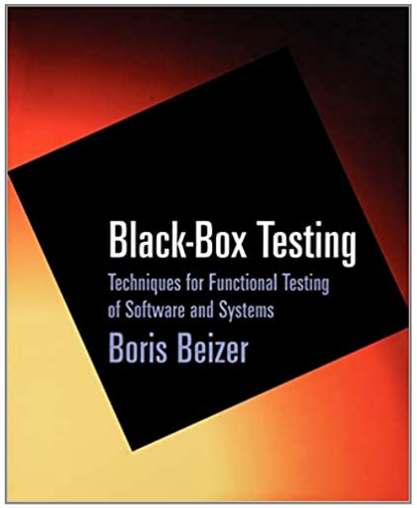
"Software testing is not about proving conclusively that the software is free from any defects, or even about discovering all the defects. Such a mission for a test team is truly impossible to achieve."

— Rex Black. *Pragmatic Software Testing: Becoming an Effective and Efficient Test Professional.* John Wiley & Sons, Inc., 2007. doi:10.5555/1215210



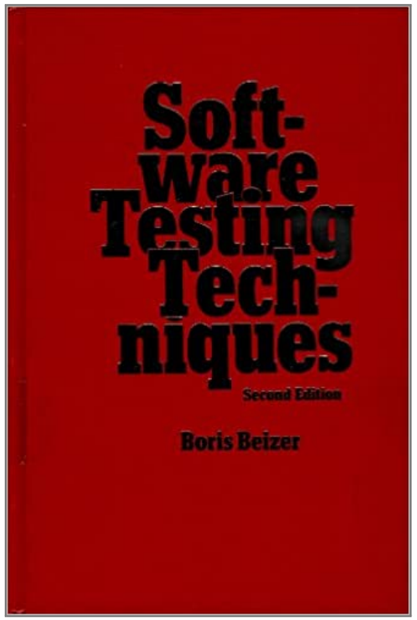
“The immediate goal of testing is to find errors at any stage of software development. More the bugs discovered at an early stage, better will be the success rate of software testing.”

— Naresh Chauhan. *Software Testing: Principles and Practices.* Oxford University Press, 2010



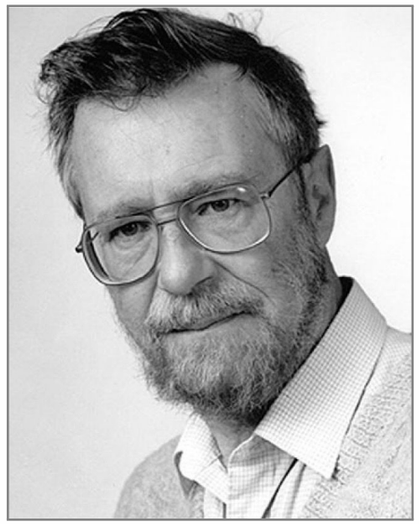
"Anything written by people has bugs. Not testing something is equivalent to asserting that it's bug-free. Programmers can't think of everything especially of all the possible interactions between features and between different pieces of software. We try to break software because that's the only practical 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.* John Wiley & Sons, Inc., 1995. doi:10.5555/202699



“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!”

— Boris Beizer. *Software Testing Techniques.* Van Nostrand Reinhold, 2 edition, 1990. doi:10.5555/79060



“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)

```
(test one
  (is (= 1 (f 1))))
(test two
  (is (= 1 (f 2))))
(test fifteen
  (is (= 610 (f 15))))
```

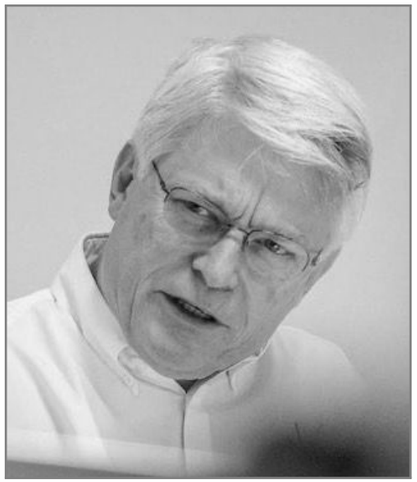
Can you put some code here?

```
(test one
  (is (= 1 (f 1))))
(test two
  (is (= 1 (f 2))))
(test fifteen
  (is (= 610 (f 15))))
```

```
(defun f (n)
  (cond
    ((= n 1) 0)
    ((= n 2) 1)
    (+
      (f (- n 1))
      (f (- n 2)))))
```

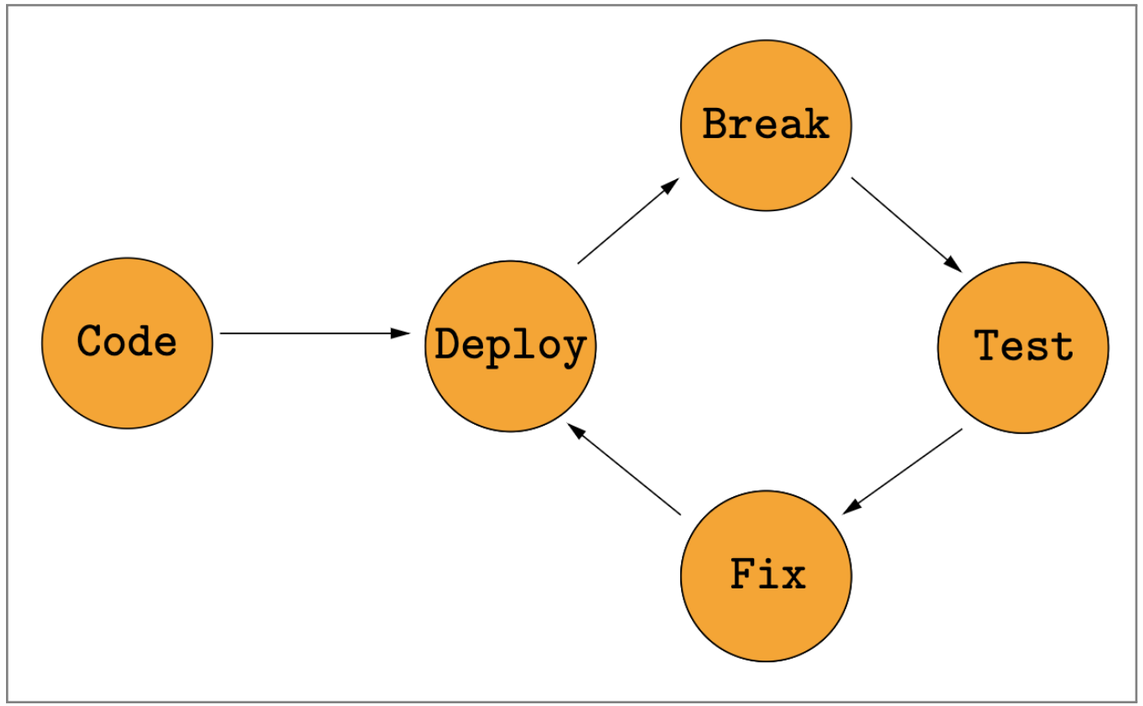
“Test-first fundamentalism is like abstinence-only sex ed: An unrealistic, ineffective morality campaign for self-loathing and shaming.”

— David H. Hansson. TDD Is Dead. Long Live Testing. https://dhh.dk/2014/tdd-is-dead-long-live-testing.html, 2014. [Online; accessed 04-05-2025]

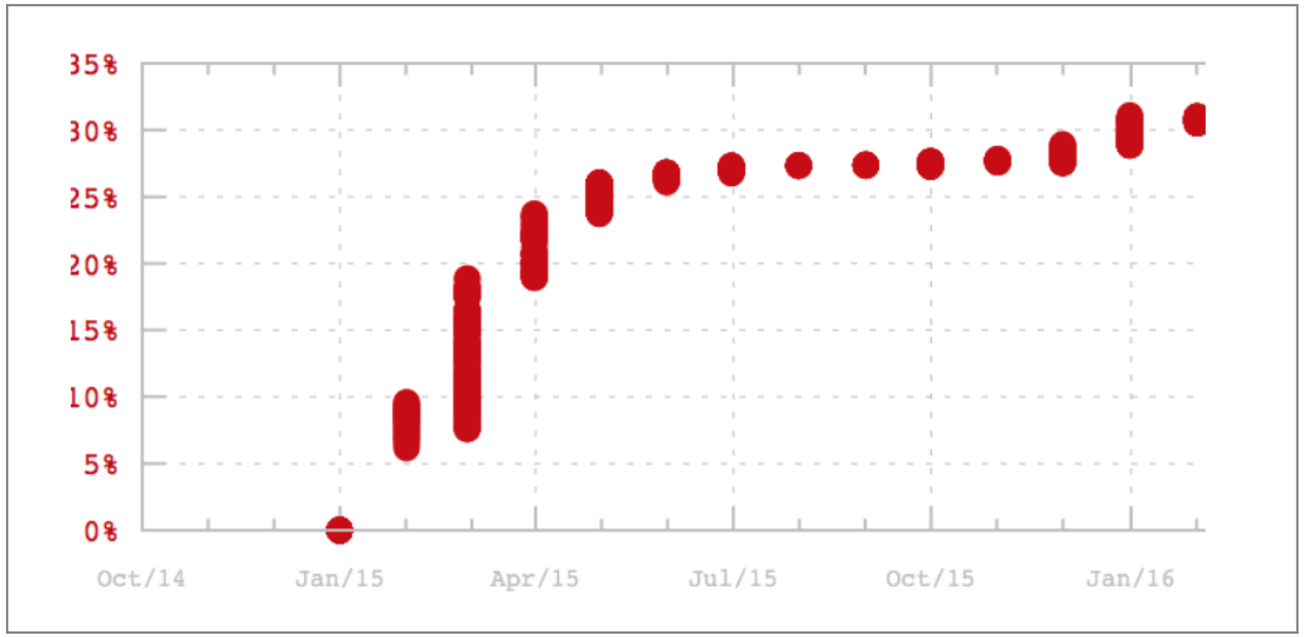


"It would not surprise me if, one day, TDD had the force of law behind it."

— Robert C. Martin. Professionalism and TDD (Reprise). https://shorturl.at/nGS3G, 2014. [Online; accessed 04-05-2025]



"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-truyers.net/2012/12/15/key-qualities-of-a-good-unit-test/ →

```
import java.nio.file.Files;
class Book {
  String title() {
    return Files.readAllLines(
      Paths.get("/my-data/book.txt")
    )[0];
  }
}
```

```
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
class BookTest {
  @Test
  void canRetrieveTitle() {
    String t = new Book().title();
    Assertions.assertEquals(
      "Object Thinking", t
    );
  }
}
```

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

```
import java.nio.file.Files;
class Book {
  private Path file;
  Book(Path f) {
    this.file = f;
  }
  String title() {
    return Files.readAllLines(
      this.file
    )[0];
  }
}
```

```
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
class BookTest {
  @Test
  void canRetrieveTitle() {
    Path f = Paths.get("/tmp/temp.txt");
    String title = "Object Thinking";
    Files.write(f, title.getBytes());
    String t = new Book().title();
    Assertions.assertEquals(title, t);
  }
}
```

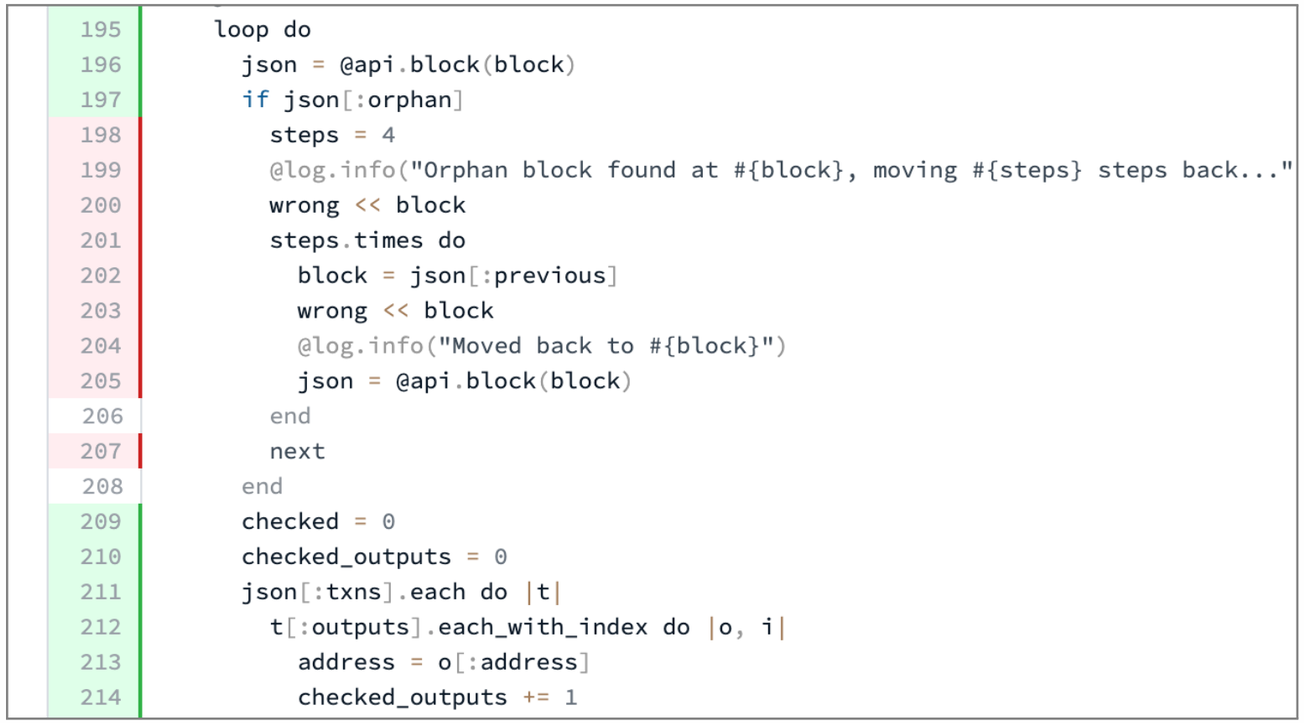
Is it Short, Fast, Independent, Portable, Careful, and Isolated?

```
import java.nio.file.Files;
class Book {
  private Path file;
  Book(Path f) {
    this.file = f;
  }
  String title() {
    return Files.readAllLines(
      this.file
    )[0];
  }
}
```

```
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.io.TempDir;
class BookTest {
  @Test
  void canRead(@TempDir Path dir) {
    Path f = dir.resolve("temp.txt");
    String title = "Object Thinking";
    Files.write(f, title.getBytes());
    String t = new Book().title();
    Assertions.assertEquals(title, t);
  }
}
```

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

```
1 def f(n)
2   n * n + 1
3 end
```

```
1 f(0) == 1
2 f(1) == 2
3 f(2) == 5
```

Mutation Testing + Coverage

```ruby
def f(n)
  n * n + 1
end

# Mutant no. 1
def f(n)
  n + n + 1
end

# Mutant no. 2
def f(n)
  n * n - 1
end
```

```ruby
f(0) == 1
f(1) == 2
f(2) == 5
```

Test coverage: 100%

Mutation coverage = 50%

Chapter #5:

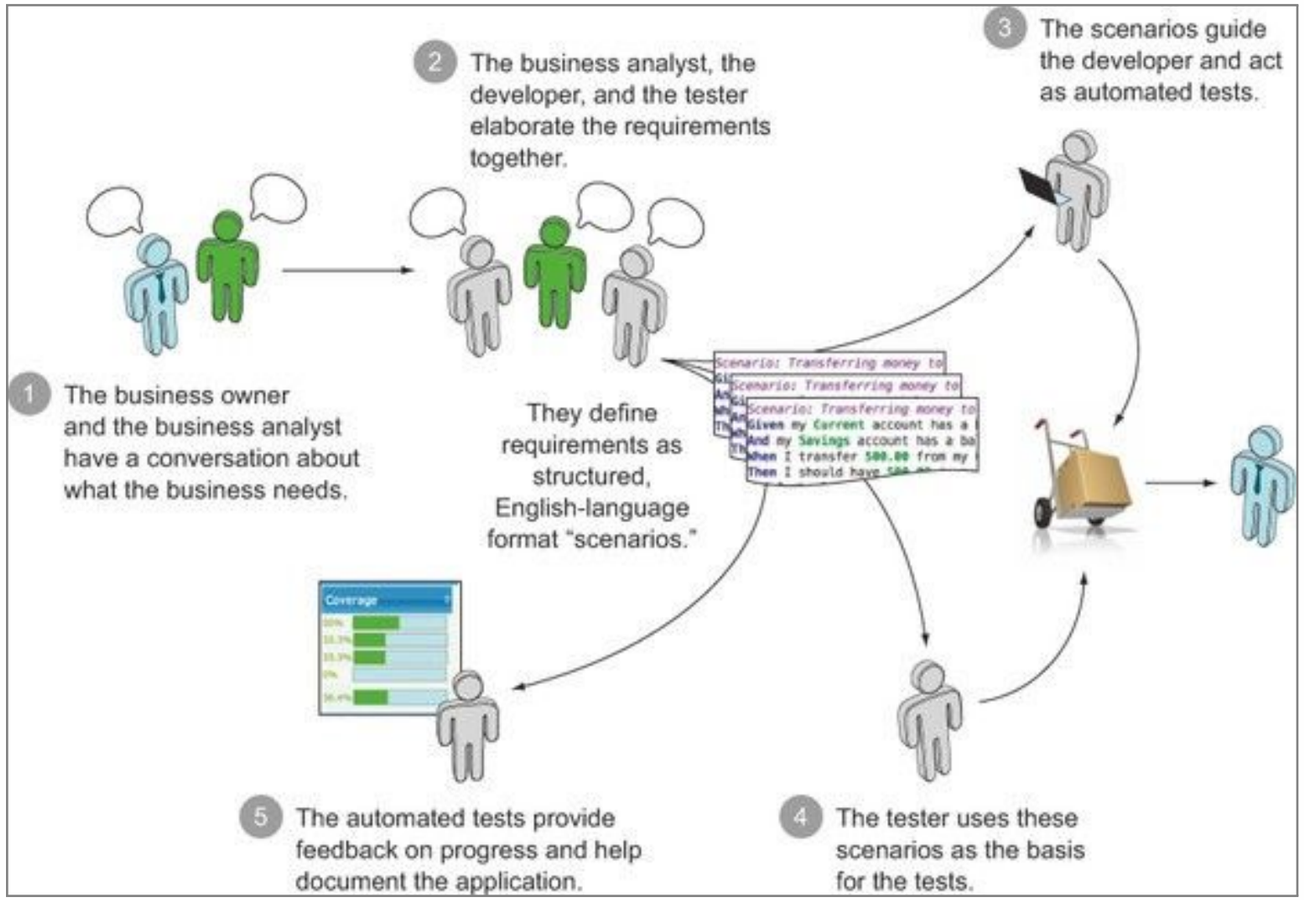
Automated Performance Testing

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:

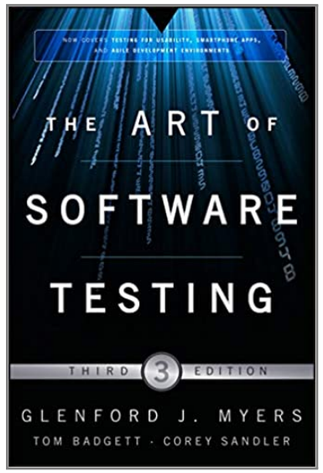
Testing vs. QA

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

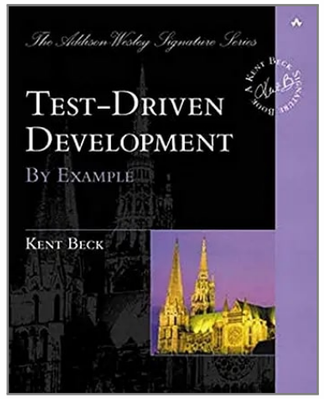
Testing ≠ Quality Assurance (QA)

Chapter #8:

Books, Venues, Call-to-Action



Glenford J. Myers, Tom Badgett, Todd M. Thomas, and Corey Sandler. *The Art of Software Testing*. Wiley, 2 edition, 2012. doi:10.5555/2161638



Kent Beck. *Test Driven Development: By Example*. Addison-Wesley, 2002. doi:10.5555/579193

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?

Bibliography

Kent Beck. *Test Driven Development: By Example*. Addison-Wesley, 2002. doi:10.5555/579193.

Boris Beizer. *Software Testing Techniques*. Van Nostrand Reinhold, 2 edition, 1990. doi:10.5555/79060.

Boris Beizer. *Black-Box Testing: Techniques for Functional Testing of Software and Systems*. John Wiley & Sons, Inc., 1995. doi:10.5555/202699.

Rex Black. *Pragmatic Software Testing: Becoming an Effective and Efficient Test Professional*. John Wiley & Sons, Inc., 2007. doi:10.5555/1215210.

Naresh Chauhan. *Software Testing: Principles and Practices*. Oxford University Press, 2010.

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

David H. Hansson. TDD Is Dead. Long Live Testing. `https://dhh.dk/2014/tdd-is-dead-long-live-testing.html`, 2014. [Online; accessed 04-05-2025].

Robert C. Martin. Professionalism and TDD (Reprise). `https://shorturl.at/nGS3G`, 2014. [Online; accessed 04-05-2025].

Glenford J. Myers, Tom Badgett, Todd M. Thomas, and Corey Sandler. *The Art of Software Testing*. Wiley, 2 edition, 2012. doi:10.5555/2161638.