

Introduction to Test Driven Development



Dario Campagna

Head of Research

Let’s start with

Development



Now add **Test Driven**

Software development practice

Clean code that works

Test first

Small steps, fast feedback



Clean code

Easy to understand

Easy to evolve

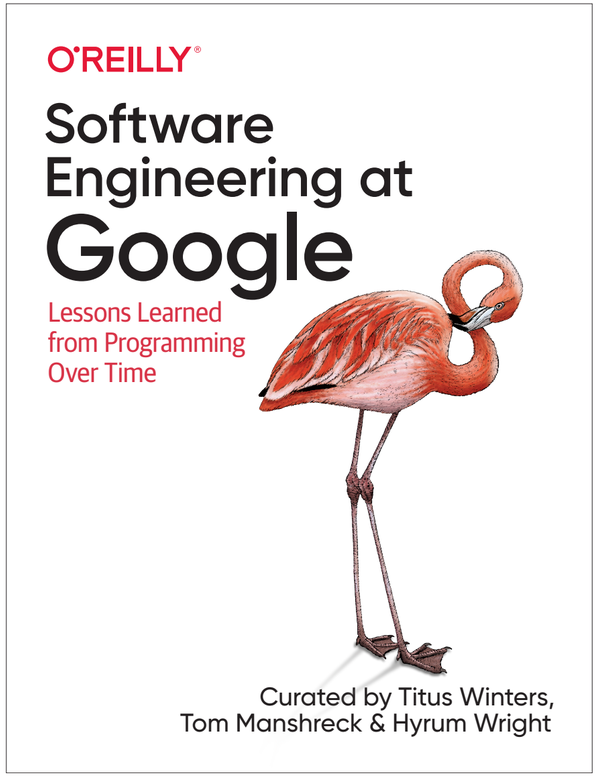
Easy to maintain

Sustains delivery pace



Your project is **sustainable** if, for the expected life span of your software, you are capable of reacting to whatever **valuable change** comes along, for either technical or business reasons.

Software Engineering at Google



Free digital version

Example of Ugly Code

TerrariaClone class from the GitHub repository TerrariaClone.

- > 6500 lines of code
- > 1300 lines of code for init() method
- Deeply nested if and for statements
- Many other “issues”

```
3056                if (ic != null) {
3057                    if (ic.type.equals("workbench")) {
3058                        for (ux=0; ux<3; ux++) {
3059                            for (uy=0; uy<3; uy++) {
3060                                if (mousePos[0] >= ux*40+6 && mousePos[0] < ux*40+46 &&
3061                                    mousePos[1] >= uy*40+inventory.image.getHeight()+46 &&
3062                                    mousePos[1] < uy*40+inventory.image.getHeight()+86) {
3063                                    checkBlocks = false;
3064                                    if (mouseClicked) {
3065                                        mouseNoLongerClicked = true;
3066                                        moveItemTemp = ic.ids[uy*3+ux];
3067                                        moveNumTemp = ic.nums[uy*3+ux];
3068                                        if (moveItem == ic.ids[uy*3+ux]) {
3069                                            moveNum = (short)inventory.addLocationIC(ic, uy*3+ux, moveItem, moveNum, moveDur);
3070                                            if (moveNum == 0) {
3071                                                moveItem = 0;
3072                                            }
3073                                        }
3074                                        else {
3075                                            inventory.removeLocationIC(ic, uy*3+ux, ic.nums[uy*3+ux]);
3076                                            if (moveItem != 0) {
3077                                                inventory.addLocationIC(ic, uy*3+ux, moveItem, moveNum, moveDur);
3078                                            }
3079                                            moveItem = moveItemTemp;
3080                                            moveNum = moveNumTemp;
3081                                        }
3082                                    }
3083                                }
3084                            }
3085                        }
3086                        if (mousePos[0] >= 4*40+6 && mousePos[0] < 4*40+46 &&
3087                            mousePos[1] >= 1*40+inventory.image.getHeight()+46 &&
3088                            mousePos[1] < 1*40+inventory.image.getHeight()+86) {
3089                            checkBlocks = false;
3090                            if (mouseClicked) {
3091                                if (moveItem == ic.ids[9] && moveNum + ic.nums[9] <= MAXSTACKS.get(ic.ids[9])) {
3092                                    moveNum += ic.nums[9];
3093                                    inventory.useRecipeWorkbench(ic);
3094                                }
3095                                if (moveItem == 0) {
3096                                    moveItem = ic.ids[9];
3097                                    moveNum = ic.nums[9];
3098                                    if (TOOLDURS.get(moveItem) != null) {
3099                                        moveDur = TOOLDURS.get(moveItem);
3100                                    }
3101                                    inventory.useRecipeWorkbench(ic);
3102                                }
```

Ugly code is

Rigid

Fragile

Inseparable

Opaque



The nature of code is to grow ugly.

Bas Vodde



Why does code **grow ugly**?

We have no time to clean it

We need to go “faster”

We are afraid of breaking it

Fear prevents us to clean it



Code Kata

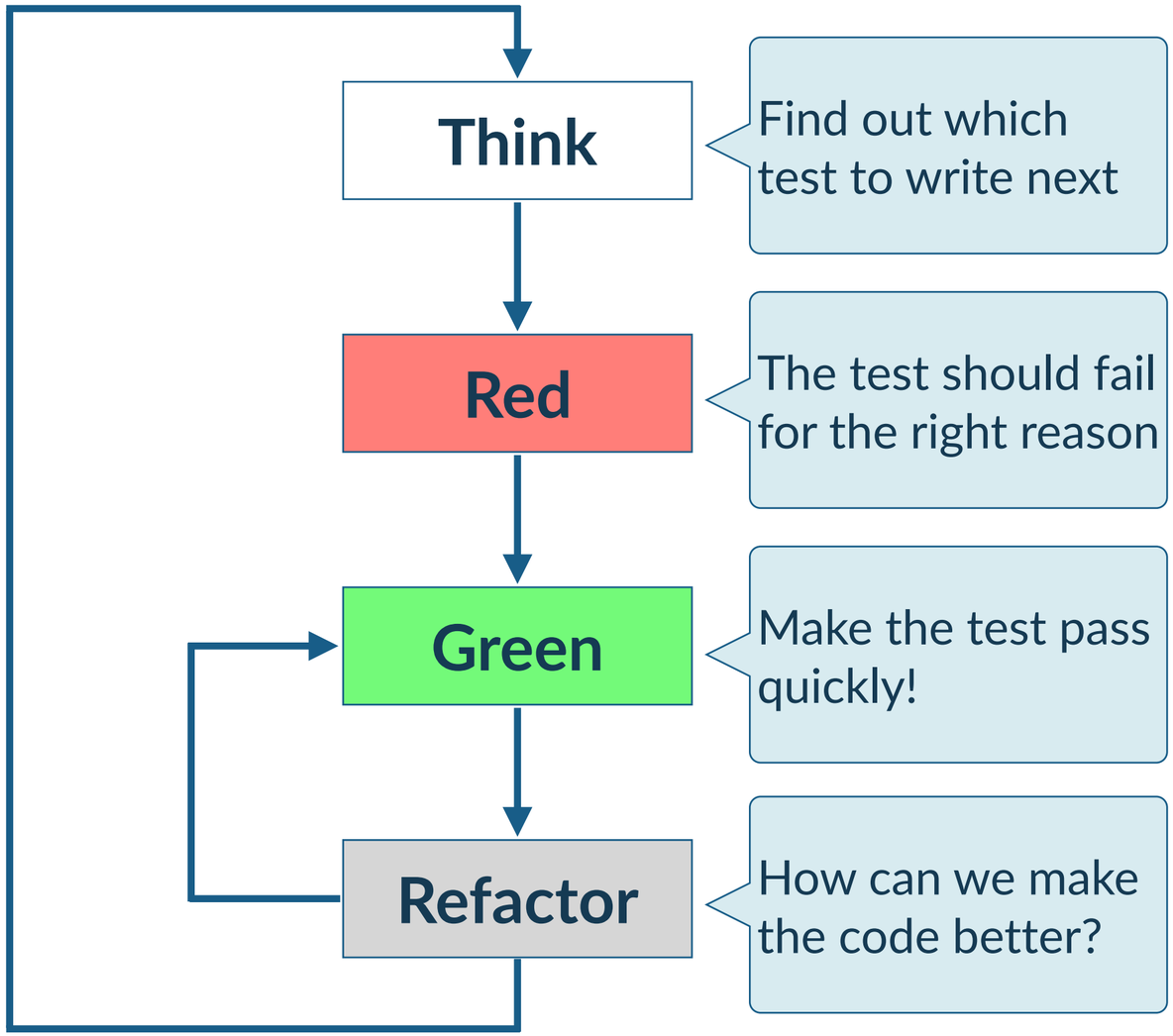
A system of coding practice incorporating techniques and notions that have been cultivated and polished for decades.

Dave Nicolette

- The purpose is to practice and internalize programming *techniques*
- (Some are) Designed to reflect programming problems that have particular *shapes*



Test Driven Development (TDD) Cycle

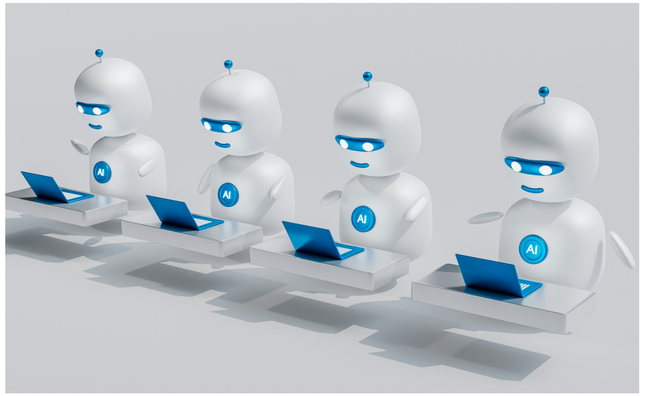


What about coding agents?

It’s still unclear whether TDD in the agent loop adds value or not (see here and here).

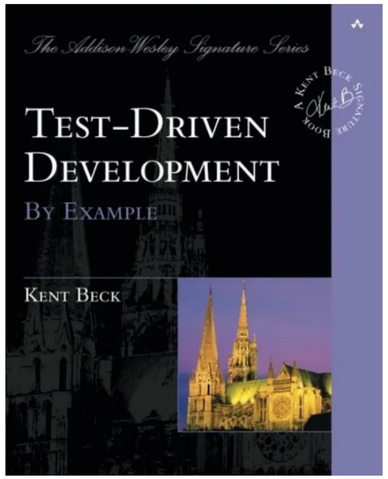
At the same time…

- Tests can increase the confidence in the code produced
- Clean code can reduce the cost of changes
- Clean code can be easier to review
- TDD helps us work in small batches

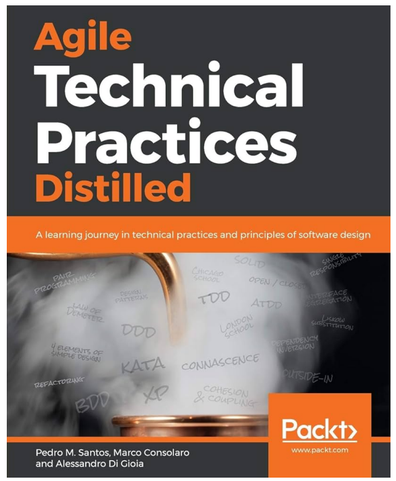


Some books about TDD (and more)

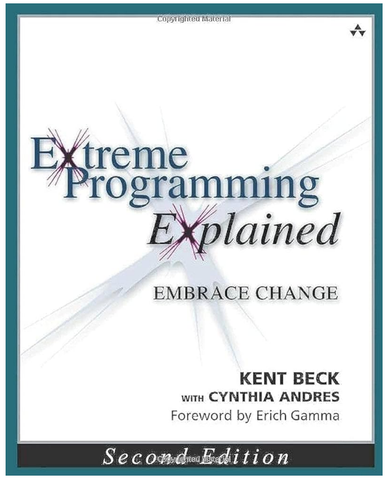
Look for them on http://www.biblio.units.it/



Test Driven Development by Example by Kent Beck



Agile Technical Practices Distilled by Pedro Moreira Santos, Marco Consolaro, Alessandro Di Gioia



Extreme Programming Explained by Kent Beck

ESTECO

Test Infrastructure

Things we need to practice TDD	Why?
Automated build	To run the tests, as fast as possible
Test framework	To build the test suite
Assertion library	To check test status
Arrange/Act/Assert	A way to write tests



How to choose the first test?

1. Look at the problem you will work on and list the behaviors you will need to solve it.
2. List the tests which, when passed, will demonstrate the presence of code you are confident will implement the desired behaviors.
3. Choose the first test to write.

- Usually a nano-test.
- Simple, short, essential.
- Degenerate but useful example.
- Don’t underestimate the implicit complexity.



Good tests

Describe

Tests should have names that describe a business feature or behavior.

Avoid

Technical names and leaking implementation details.

Communicate

Tests should clearly express required functionalities to the reader.



Ways to move forward

Fake it

Just return the exact value you need.

Something that works is better than something that doesn’t work!

Obvious implementation

When you are sure of the code you need to write, write it, and see the test go green!

Triangulation

Write a new and more specific test that forces the code to be more generic.



How to choose the next test?

Every test is a question we can ask to the system, a chance to learn something about: the domain, the code, the design.

- Use your test list.
- Look for the simple thing that could possibly break.
- The choice is heavily influenced by previous tests.



Refactor to remove duplication

Types of duplication

Code, data, knowledge.

Wait

Avoid removing duplication too soon, as this may lead you to extract the wrong abstractions.

Rule of Three

Extract duplication only when you see it for the third time.



Duplication of knowledge

```
public class Cylinder {

    private final double radius;
    private final double height;

    public Cylinder(double radius, double height) {
        this.radius = radius;
        this.height = height;
    }

    public double volume() {
        return Math.PI * Math.pow(radius, 2) * height;
    }

    public double surface() {
        return 2 * Math.PI * Math.pow(radius, 2) + 2 * Math.PI * radius * height;
    }
}
```



Duplication of knowledge

Extract method.

```
public class Cylinder {

    private final double radius;
    private final double height;

    public Cylinder(double radius, double height) {
        this.radius = radius;
        this.height = height;
    }

    public double volume() {
        return baseSurface() * height;
    }

    public double surface() {
        return 2 * baseSurface() + 2 * Math.PI * radius * height;
    }

    private double baseSurface() {
        return Math.PI * Math.pow(radius, 2);
    }
}
```



Duplication of hard coded data

```
@Test
public void productNotFound() throws Exception {
    Display display = new Display();
    Sale sale = new Sale(display);

    sale.onBarcode("99999");

    assertEquals("Product not found for 99999", display.getText());
}
```

```
public class Sale {

    private Display display;

    public Sale(Display display) {
        this.display = display;
    }

    public void onBarcode(String barcode) {
        display.setText("Product not found for 99999");
    }
}
```



Duplication of hard coded data

Replace literal value with variable.

```
@Test
public void productNotFound() throws Exception {
    Display display = new Display();
    Sale sale = new Sale(display);

    sale.onBarcode("99999");

    assertEquals("Product not found for 99999", display.getText());
}
```

```
public class Sale {

    private Display display;

    public Sale(Display display) {
        this.display = display;
    }

    public void onBarcode(String barcode) {
        display.setText("Product not found for " +
                barcode);
    }
}
```

Tests in TDD

Should be…

Isolated and composable

Fast and automated

Behavioral and structure-insensitive

Specific and deterministic

Inspiring and predictive

Writable and readable

Beware of

Databases

Network communications

File system

Other shared fixtures

Configurations



Examples of “not-so-good” tests

```
//Test metodo canBePlaced
@Test
void canBePlacedTest(){
    Piece p= PieceSet.getPossibleSet()[1];
    Board tabella= new Board();
    assertTrue(tabella.canBePlaced(p, new Double[]{2.0,2.0})); //can be placed case
    tabella.getGameBoard()[2][3].setColor(Color.yellow);
    assertFalse(tabella.canBePlaced(p, new Double[]{2.0,2.0})); //occupied tile case
    assertFalse(tabella.canBePlaced(p, new Double[]{3.0,9.0})); //out of gameboard case
}

//Test metodi place e canBePlaced
@Test
void placeTest(){
    Piece p=PieceSet.getPossibleSet()[1];
    Board tabella=new Board();
    assertTrue(tabella.canBePlaced(p, new Double[]{2.0,2.0})); //can be placed case
    tabella.place(p, new Double[]{2.0, 1.0});
    assertFalse(tabella.canBePlaced(p, new Double[]{2.0,0.0})); //verifico che lo stesso pezzo non possa più essere posizionato
    assertFalse(tabella.canBePlaced(p, new Double[]{2.0,1.0}));  // negli spazi già occupati
    assertFalse(tabella.canBePlaced(p, new Double[]{2.0,2.0}));
    assertTrue(tabella.canBePlaced(p, new Double[]{2.0,3.0}));
}
```

Examples of “not-so-good” tests

```
//Test metodo addPoints
@Test
void testAddingPoint()
{
    if (!GraphicsEnvironment.isHeadless()){
        Game game = Game.getInstance();
        int initialScore = game.getScore().points;
        game.addPoints(10);
        assertEquals(10, game.getScore().points - initialScore);
    }

    assertTrue(true);

}


//Test metodo addPoints
@Test
void testPointsAddedByPiece()
{
    if (!GraphicsEnvironment.isHeadless()) {
        Game game = Game.getInstance();
        int initialScore = game.getScore().points;
        Piece p = PieceSet.getPossibleSet()[2];
        game.addPoints(p.getSize());
        assertEquals(3, game.getScore().points - initialScore); // Punti primo test + punti secondo
    }

    assertTrue(true);
}
```



Tests and Java exceptions

How to check that an exception is thrown?

assertThrows

- Available in JUnit 5
- Returns the exception
- Enables fine-grained control

expected attribute

- Available in JUnit 4
- Just checks that an exception is thrown

ExpectedException

- Available in JUnit 4
- To also check the exception message



Should you **always** practice
Test Driven Development?



TDD is a **tool**

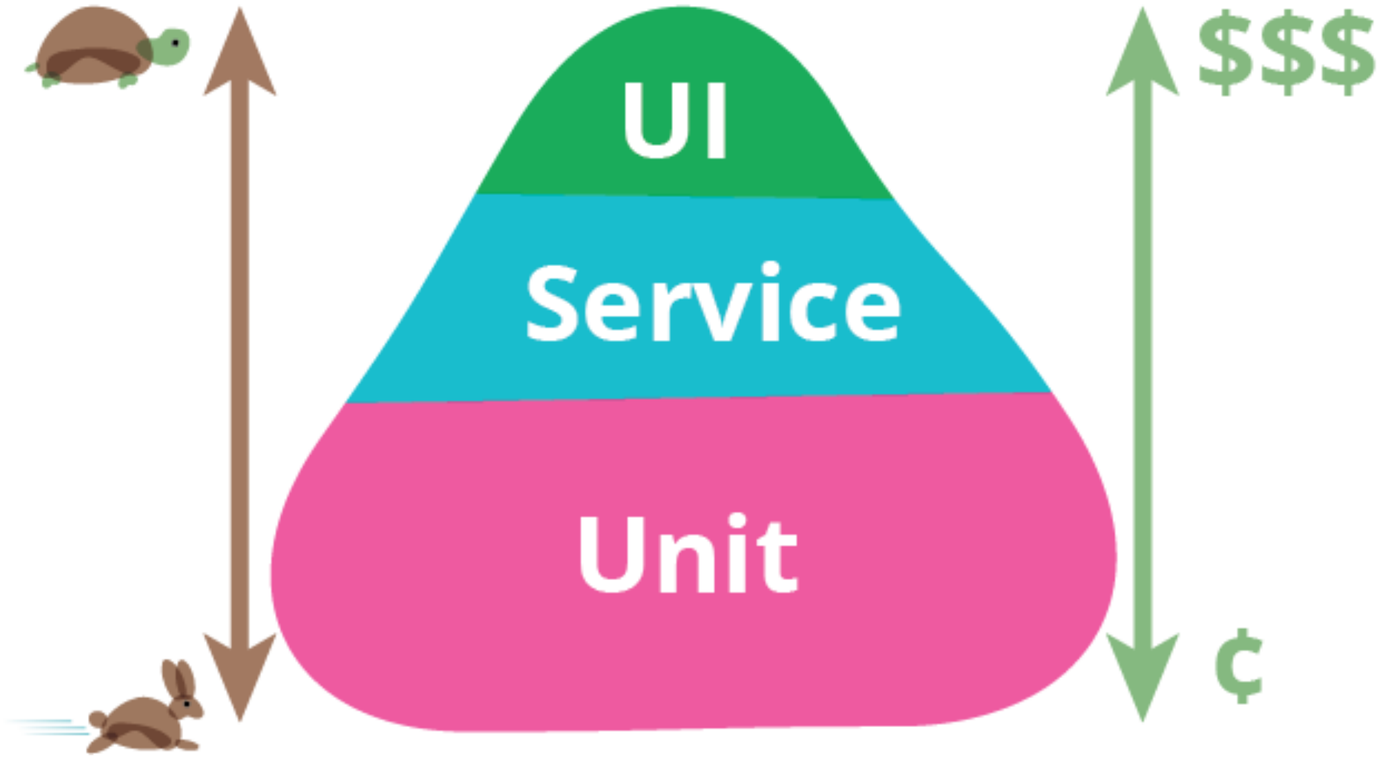
To get value from a tool, it's necessary to:

1. Choose the right tool for the job.
2. Use the tool properly.

Dave Nicolette, "Against TDD", https://neopragma.com/2019/09/against-tdd/



Test Pyramid



https://martinfowler.com/bliki/TestPyramid.html
Copyright © 2012 Martin Fowler