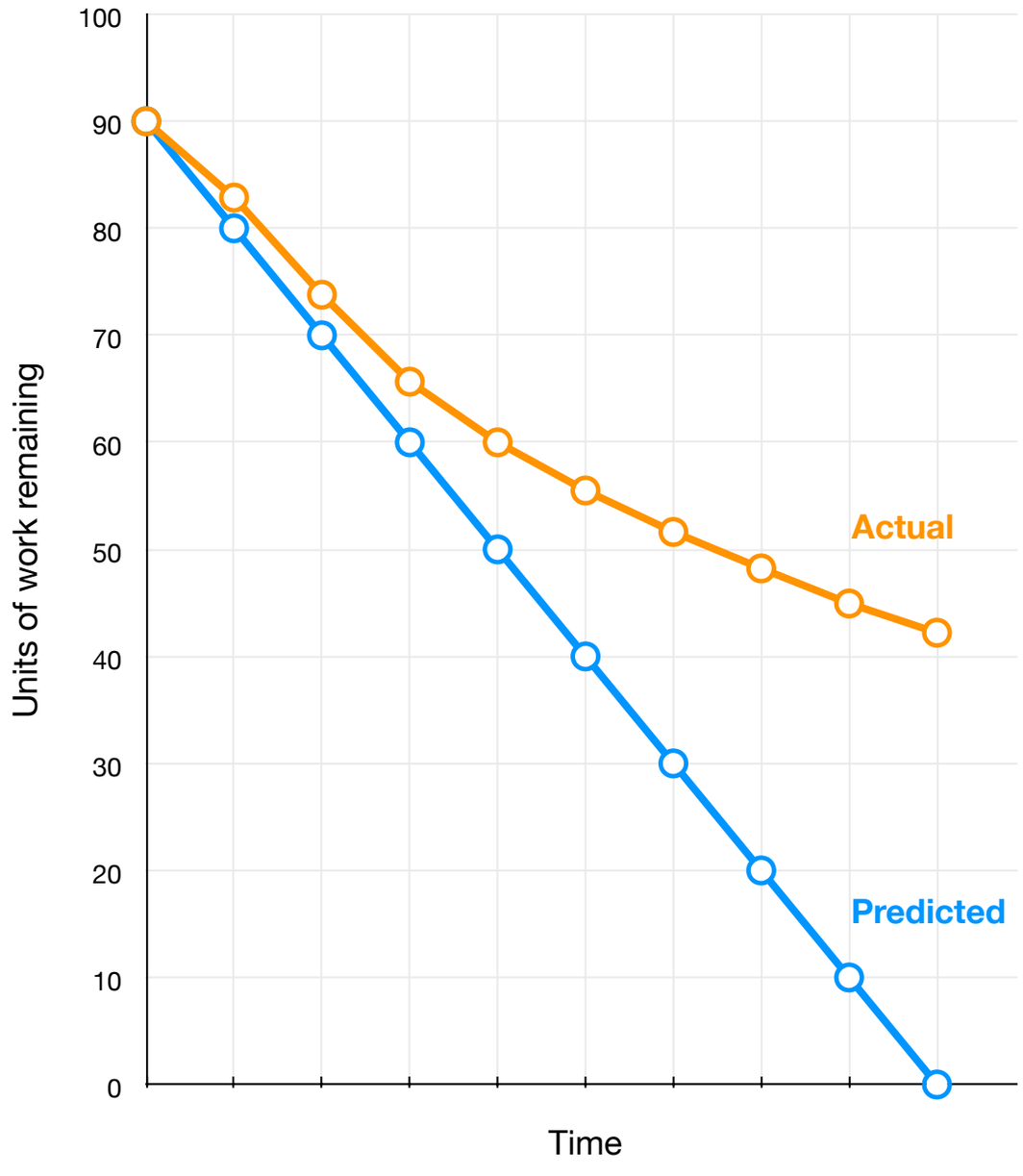
Introduction to TDD

Productivity trap

- We have to go “faster”
- We rush and make a mess
- The code grows and time necessary to add new features grows too
- We go even slower



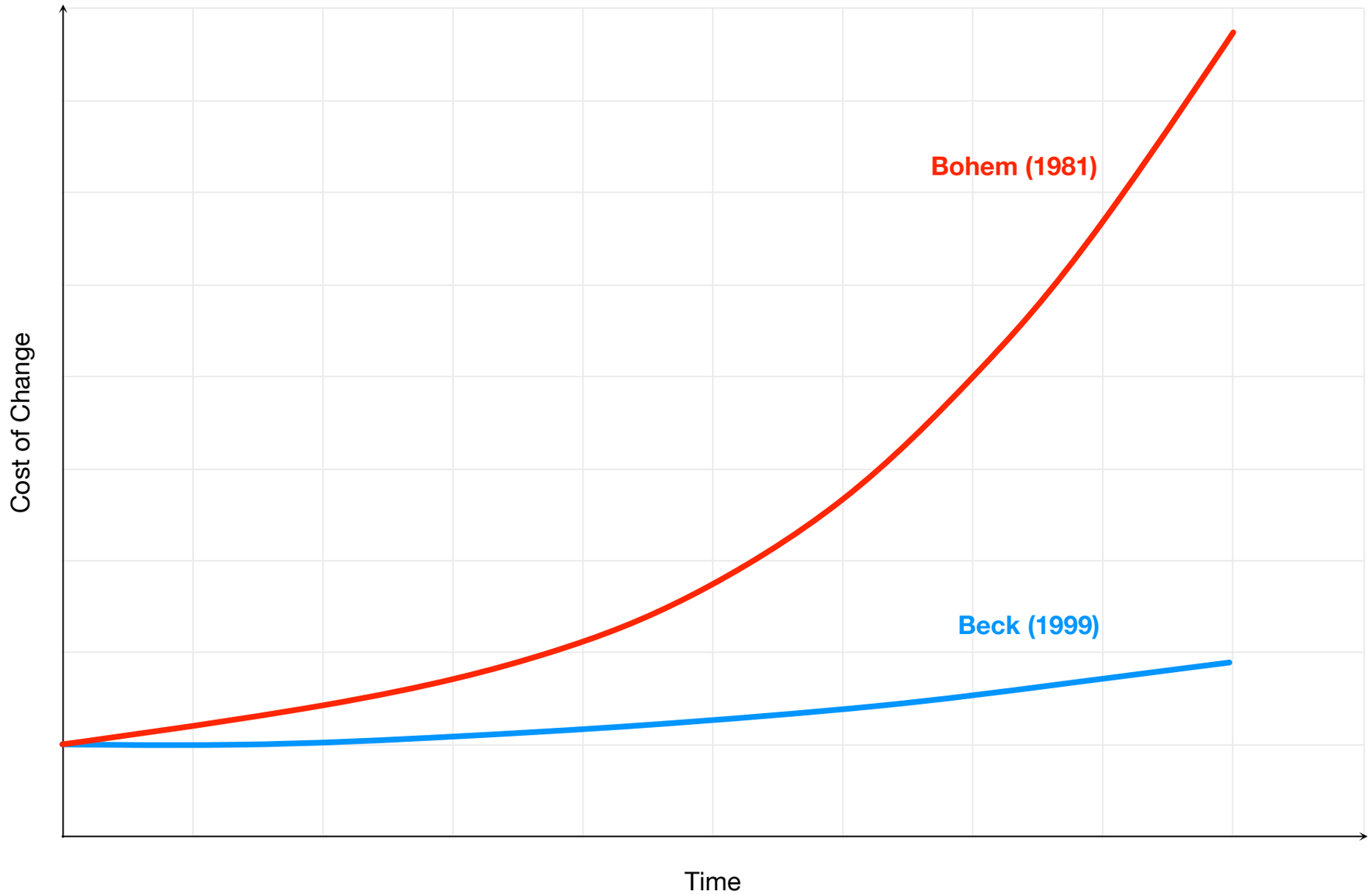
Code Rots

Over time it becomes

- Rigid
- Fragile
- Inseparable
- Opaque

Why?

- We ruin it
- We do not fix it because we are afraid of breaking it
- Fear prevents us to clean it and we leave it rotting



Cost of Change

How to reduce it?

TDD

- Drives development to get to clean code that works
- It is a way of managing fear during programming

This is a hard problem and I can't see the end from the beginning.

- It is an awareness of the gap between decision and feedback, and techniques to control the gap

Write a Test

```
public class AddIntegersTest {
 @Test
 public void twoPlusThree() {
  Adder a = new Adder();
  assertEquals(5, a.add(2,3));
 }
}
```

Now it Compiles

```
public class AddIntegersTest {
  @Test
  public void twoPlusThree() {
    Adder a = new Adder();
    assertEquals(5, a.add(2,3));
  }
}

public class Adder {
  public int add(int a, int b) { return 0; }
}
```

Red Bar!

```
public class AddIntegersTest {
 @Test
 public void twoPlusThree() {
  Adder a = new Adder();
  assertEquals(5, a.add(2,3));
 }
}

public class Adder {
 public int add(int a, int b) { return 0; }
}
```

Expected 5, was 0

Do the simplest thing

```
public class AddIntegersTest {
 @Test
 public void twoPlusThree() {
  Adder a = new Adder();
  assertEquals(5, a.add(2,3));
 }
}

public class Adder {
 public int add(int a, int b) { return 5; }
}
```

Remove duplication

```
public class AddIntegersTest {
 @Test
 public void twoPlusThree() {
  Adder a = new Adder();
  assertEquals(5, a.add(2,3));
 }
}

public class Adder {
 public int add(int a, int b) { return a+b; }
}
```

TDD Cycle

1. Write a test
2. See it fail

3. Make it pass

4. Refactoring (remove duplication)

TDD Cycle is

- Tight
- Fast
- Guided by tests
- Refactoring oriented

The Three Laws of TDD

1. You may not write production code until you have written a failing unit test.

2. You may not write more of a unit test than is sufficient to fail, and not compiling is failing.

3. You may not write more production code than is sufficient to pass the currently failing test.

Unit Tests

- Programs that test the functionalities of a unit
- Should be
  - Fast
  - Independent
  - Repeatable
  - Self-validating
  - Timely

A Test is **NOT** a Unit Test if…

- …it talks to a database
- …it communicates across the network
- …it touches the file system
- …you have to do things to your environment to run it (e.g., change configuration files)

Tests that do this are *integration tests*.

Refactoring

Safely improve the design of existing code

Refactoring

Safely improve the design of existing code

Refactoring

Safely improve the design of existing code

Take baby steps, keep test bar green

Refactoring

improve the design

Safely of existing code

Refactoring

improve the design

Safely of existing code

Does not add functionalities

Refactoring

Safely improve the design of existing code

Refactoring

Safely improve the design of existing code

It is not rewriting from scratch

Refactoring

- We need unit tests to do it safely
- We need to do it on tests too
- Keeps code from rotting

Simple Design

1. Runs all the tests

2. Has no duplicated logic (minimize duplication)

3. States every intention important to the programmers (maximize clarity)

4. Has the fewest possible classes and methods (in this order)

```java
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;
    }
}
```

Minimize duplication

Duplication of knowledge

```java
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;
    }
}
```

Minimize duplication

Duplication of knowledge

```java
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);
    }
}
```

Minimize duplication

Extract method

```java
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);
    }
}
```

Minimize duplication

Extract method

```java
@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());
}
```

```java
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");
    }
}
```

Minimize Duplication

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");
    }
}
```

Minimize Duplication

Duplication of hard coded data.

```java
@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());
}
```

```java
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);
    }
}
```

Minimize Duplication

Replaced literal value with variable.

```java
@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());
}
```

```java
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);
    }
}
```

Minimize Duplication

Replaced literal value with variable.

```
private void displayPrice(String barcode) {
    String priceAsText = pricesByBarcode.get(barcode);
    display.setText(priceAsText);
}
```

Maximize clarity

Method does more than what suggested by its name.

```
private void displayPrice(String barcode) {
    String priceAsText = pricesByBarcode.get(barcode);
    display.setText(priceAsText);
}
```

Find

Maximize clarity

Method does more than what suggested by its name.

```
private void displayPrice(String barcode) {
    String priceAsText = pricesByBarcode.get(barcode);
    display.setText(priceAsText);
}
```

Find

Display

Maximize clarity

Method does more than what suggested by its name.

```
private void findPriceAndDisplayAsText(String barcode) {
    String priceAsText = pricesByBarcode.get(barcode);
    display.setText(priceAsText);
}
```

Maximize clarity

Conjunction tells us that method has more than one responsibility.

```
private String findPrice(String barcode) {
    return pricesByBarcode.get(barcode);
}

private void displayPrice(String priceAsText) {
    display.setText(priceAsText);
}
```

Maximize clarity

Two methods with one responsibility each.

Code Smells

Smell	Refactorings
Long Method	Extract method,…
Dead Code	Delete
Primitive Obsession	Replace Data Value with Object,…
Switch Statements	Polymorphism,…
Shotgun Surgery	Move Method and Move Field,…
Feature Envy	Move Method,…
…	…

Code Smells

- https://refactoring.com/catalog/

- https://sourcemaking.com/refactoring/smells

TDD

- Reduces debug time
- Creates low level design documents
- Leads to decoupled code
- Eliminates the fear of change

Pair Programming

Two people working together,

at the same computer,

solving the same problem.

Pair Programming

Driver

Navigator

Pair Programming

How to get used to swap?

- Use a timer and swap every 10 minutes, take a break every hour or so
- Try ping-pong programming
- Try the Pomodoro technique swapping after each pomodoro