Intro to Pair Programming and Test-Driven Development

Pair Programming Roles

Driver

Takes control of the keyboard

Navigator

Constantly reviews what is being typed

Constant switching

Expected Benefits

- **Continuous Code Reviews**. Many mistakes get caught as they are being typed in rather than in QA test or in the field, leading to lower end defect count

- **Pair Relaying.** The designs are better and code length shorter

- **Line-of-sight learning.** The people learn significantly more about the system and about software development

Expected Benefits

- **Shared Understanding**. The project ends up with multiple people understanding each piece of the system

- **Improved Collaboration.** The people learn to work together and talk more often together, giving better information flow and team dynamics

- **More enjoyment**

Core Pairing Guidelines

1. **Collaborate; don't critique**
2. Actively contribute
3. Switch roles frequently
4. Find comfortable work stations
5. Rest if you must.

Pairing Smells

1. **Unequal access**, e.g., one person dominates the keyboard

2. **Unhealthy relationship**

3. **Worker-rester relationship**

Pairing Smells

4. **Endless debate**

5. **"Go make me a cup of tea"** syndrome

When is pair programming least effective?

1. **When** working on non-complex mechanical-like tasks

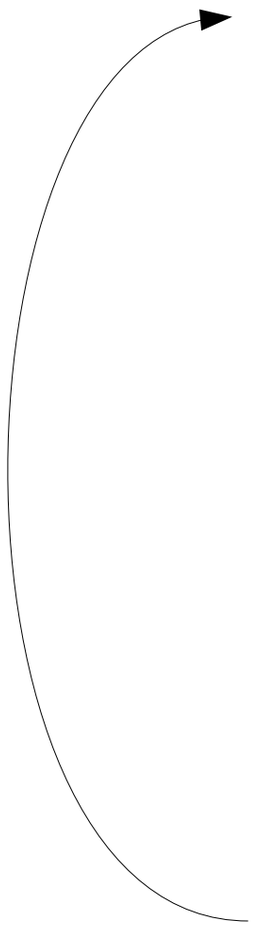
2. **When** both parties do not have the same level of expertise (is this really bad?) - great for training but pairs are more engaged when they have the same level of expertise

3. **When** pairs don't rotate

Pair Programming and Test-Driven Development (TDD)

Can these 2 go together?

The TDD Mantra



RED

Write a test first. See it **fail**

GREEN

Write the simplest code to make the test **pass**

REFACTOR

TDD + Pair Programming (Ping Pong Method)

1 Write a failing test →

2 Make the test pass.
Write a new test ←

3 Make the new test pass.
Write a new test or refactor →

Refactoring

Key Principle:

Don't Repeat Yourself

Refactoring

UpcomingDeparturesViewControllerSpec.m

```
describe( @"UpcomingDeparturesViewController ", ^{

  it( @"has the station's name as the title", ^{
    // Given
    Station *theStation =
      [[Station alloc] initWithName:@"station name"
                               abbr:@"station-abbr"];
    UpcomingDeparturesViewController *theDeparturesVC =
      [[UpcomingDeparturesViewController alloc] initForStation:theStation];

    // Then
    [[theDeparturesVC.title should] equal:@"station name"];
  });

  // ...
```

Refactoring

UpcomingDeparturesViewControllerSpec.m

```
describe( @"UpcomingDeparturesViewController ", ^{

    __block Station *station;
    __block UpcomingDeparturesViewController *departuresVC;

    beforeEach(^{
        station =
          [[Station alloc] initWithName:@"station name" abbr:@"station-abbr"];
        departuresVC =
          [[UpcomingDeparturesViewController alloc] initForStation:station];
    })

    it( @"has the station's name as the title", ^{
        // Then
        [[departuresVC.title should] equal:station.name];
    });


    // ...
```

Choose a template

Android bit.ly/CodingDojoAndroidKata

IOS bit.ly/CodingDojoIOSKata

C# .NET bit.ly/CodingDojoDotNetKata