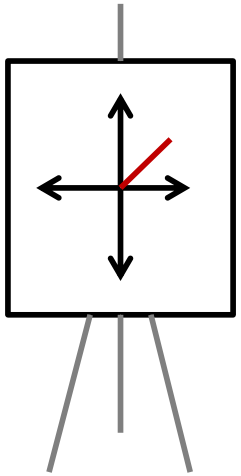
The Testing Circle



Whiteboard

English

// Create side (0,0) – (3,4)
// Verify length

Testing Circle

Result

Code

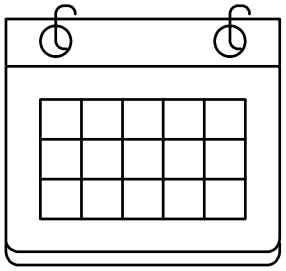
Side (0,0) – (3,4) length = 5

```
Side s = new Side(0,0,3,4);
Approvals.Verify($"{s}.length = {s.Length}");
```

4 Benefits of Tests

1. Specification
2. Feedback
3. Regression
4. Granularity

Rules for Test Scenarios
(1/6)



In the Past

The past already happened. No ifs or conditionals, no branches.

~~*Sam might go to a store sometime tomorrow.*~~

Sam went to the video store at 11:15 yesterday

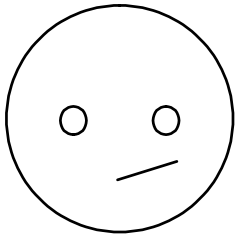
Specific Details

The devil is in the details.
Make sure they are clear

~~*Create a game board.*~~

Create a crossword board that is 18 x 23

Rules for Test Scenarios
(3/6)



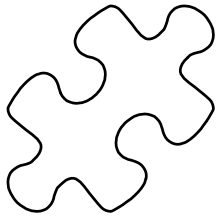
Avoid Symmetry

Symmetry is a smell in test as you can easily get a false positive

~~*Place a piece at (5,5).*~~

Place a piece at (4,5)

Rules for Test Scenarios
(4/6)



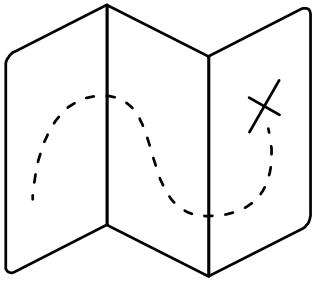
Start Simple

You are building complexity, don't start with it.

$3*5/4^2 = 0.9375$

$1+1 = 2$

Rules for Test Scenarios
(5/6)



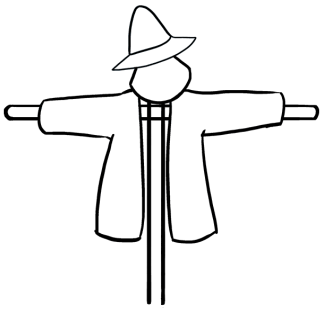
Happy Path

Sketch out the main things that can go right before focusing on what can go wrong

~~*It's illegal to place a piece on an ...*~~

Player 1 puts an X in the middle to win

Rules for Test Scenarios
(6/6)



Reality is Optional

You only need to use realistic situations when they are also convenient. Otherwise, don't

~~*Password = 2FX_V?Az8Wm/9CuZ%*~~

Password = Password

Parts of a Test

Do

Verify

Triangles

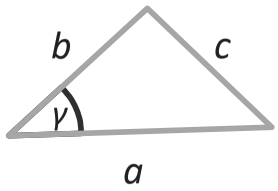
Side

1. A side has a distance
2. A side has endpoints

Triangle

3. 3 points
4. 3 sides
5. Perimeter
6. Get sides touching a point
7. Get sides opposite a point
8. The angle for the 2 sides touching a point

$\gamma = \arccos((a^2+b^2-c^2) / 2ab)$

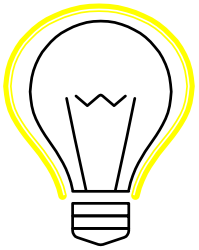


9. 3 angles
10. Right Triangle

Bowling

1. _____
2. _____
3. _____
4. _____
5. _____
6. _____
7. _____
8. _____
9. _____
10. _____
11. _____
12. _____
13. _____
14. _____

Rules for Creating Code with Consume First
(1/6)



Use your imagination

Write the code you ***want*** to exist, regardless of what currently does

~~*array.isEmpty() ? null : array.get(0)*~~

array.first()

Rules for Creating Code with Consume First
(1/6)

Use it then create it

~~*public Piece[][] board;*~~

board[x][y] = piece;

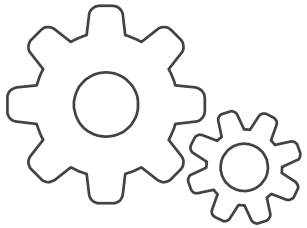
Rules for Creating Code with Consume First
(3/6)

Evaluate the consequences

Ask yourself "what are the resulting ***classes*** and ***methods*** from this implementation?"

Do you like the implications?

Rules for Creating Code with Consume First
(4/6)



Use Tools

Most of your code can be completed or generated by your tools.

Use **Autocomplete** and **QuickFix**

(ctrl+space) (alt+enter)

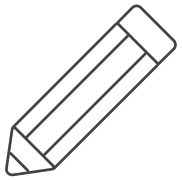
Rules for Creating Code with Consume First
(5/6)

Programming By Red

Read your errors, let them ***guide*** you

“Failure helps us see what success should look like”

Rules for Creating Code with Consume First
(6/6)

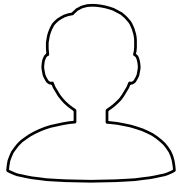


Y.A.G.N.I

"You ain't going to need it"

Do the simplest thing that can possibly work. You can improve it later.

Rules for Translating Test Scenarios
(1/5)



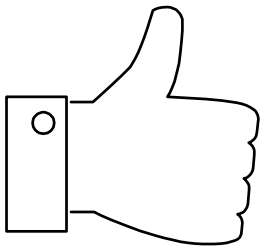
From the user perspective

You are writing what the user does, not what the program does

~~*1.Get Shovel 2. Dig Hole 3. Place Pole…*~~

setupFlagPole()

Rules for Translating Test Scenarios
(2/5)



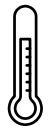
Edit

Improve on your 1st draft.
The better your English is, the better the code will be.
There will never be an easier time to refactor

~~*Ask to make a game for the category of TicTacToe*~~

Create a TicTacToe game

Rules for Translating Test Scenarios
(3/5)



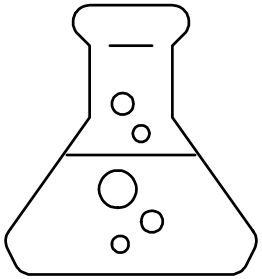
Verify Effect

Make sure you have the correct outcome

Place a 'X' at 1,2

✚ *Check X is at 1,2*

Rules for Translating Test Scenarios
(4/5)



Verify Cause

Make sure it happened for the right reasons

✚*Check 1,2 is blank*

Place a 'X' at 1,2

Check X is at 1,2

Rules for Translating Test Scenarios
(5/5)

Complete

The world begins and ends with your test.

Make sure it has everything it needs

✚*Create a board*

Check 1,2 is blank

Place a ‘X’ at 1,2

Check X is at 1,2