INTRO TO TDD

What, Why, How

TEST DRIVEN

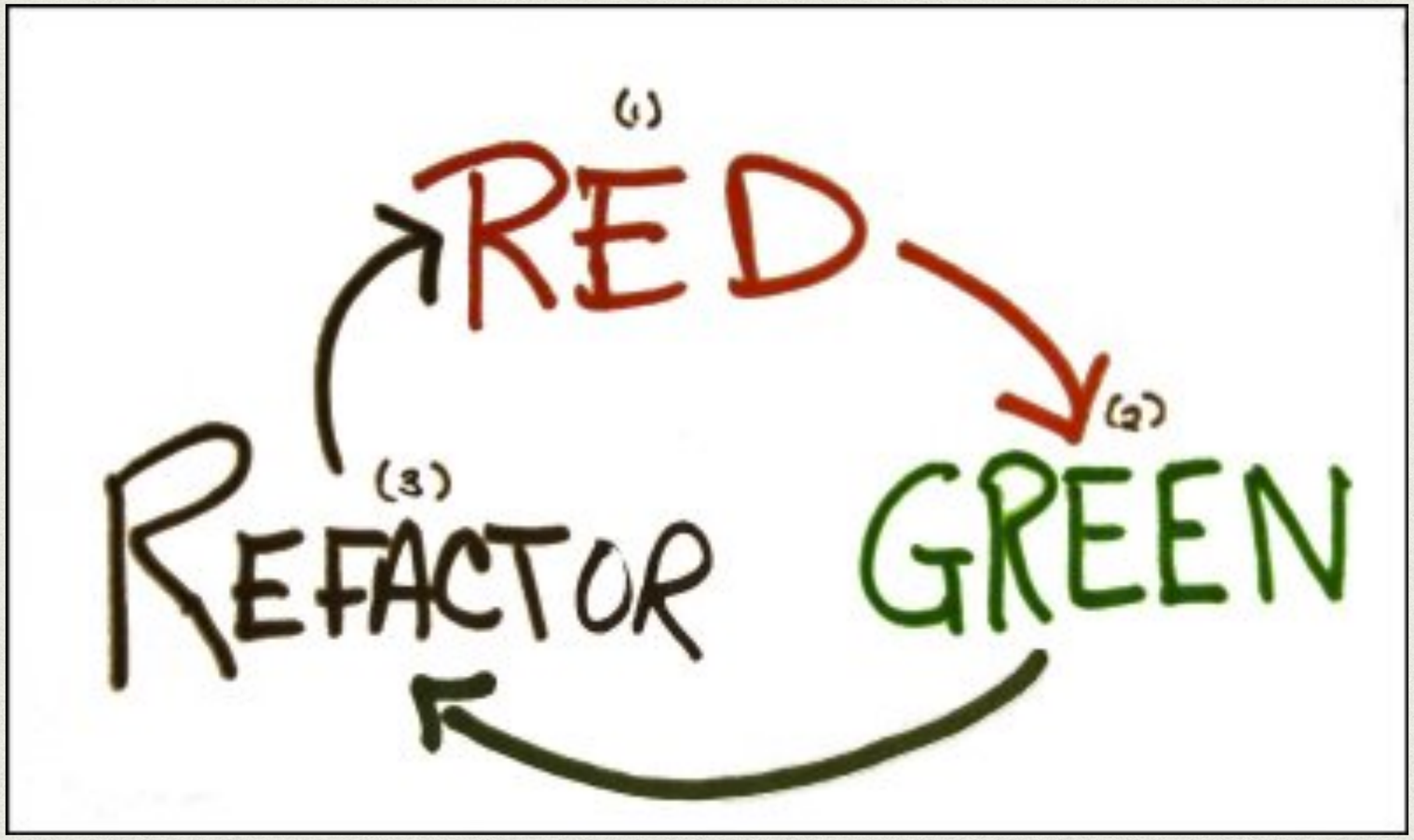
DEVELOPMENT

1. You must write a failing test before you write any production code.

2. You must not write more of a test than is sufficient to fail, or fail to compile.

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

* https://blog.cleancoder.com/uncle-bob/2014/12/17/TheCyclesOfTDD.html



1. Create a unit tests that fails
2. Write production code that makes that test pass.
3. Clean up the mess you just made.

* https://blog.cleancoder.com/uncle-bob/2014/12/17/TheCyclesOfTDD.html

WRITE WHAT KIND OF TEST?

FOR WHAT UNIT OF CODE?

WHAT VALUE ARE WE GETTING OUT OF THIS TEST?

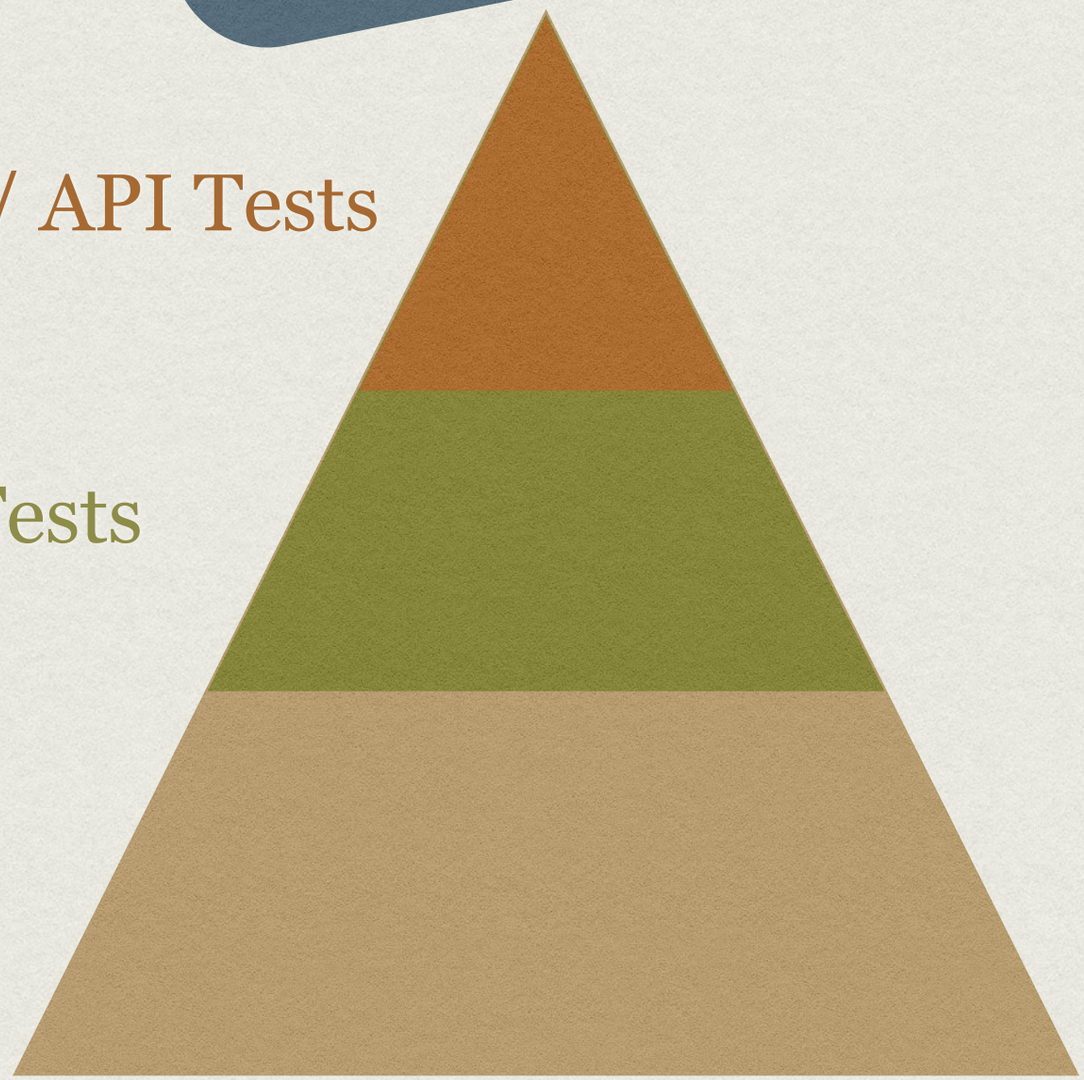
COSTS & BENEFITS

Manual Tests

Browser / API Tests

Integration Tests

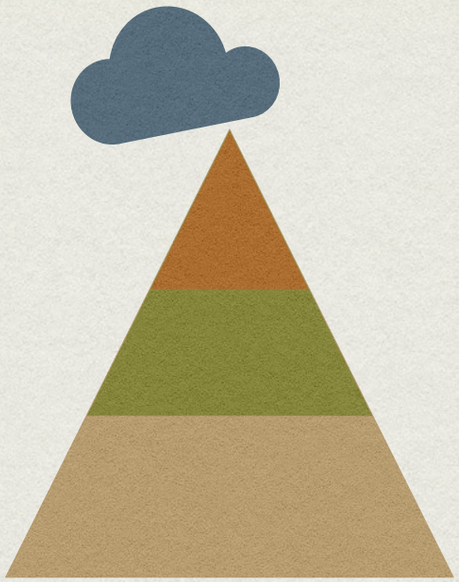
Unit Tests



$$$

¢

* https://github.com/testdouble/contributing-tests/wiki/Testing-Pyramid



Manual Tests

Coordinating with a QA team (hopefully)

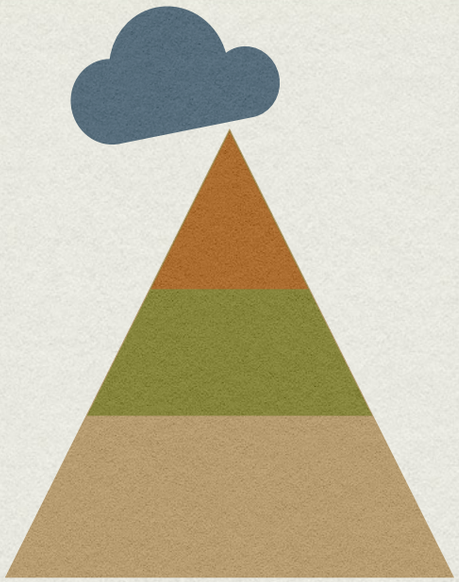
Slowest feedback loop

Excellent for finding experience issues

Great for experimenting

Poor for development cycle

* https://github.com/testdouble/contributing-tests/wiki/Testing-Pyramid



Browser / API Tests

Test entire application

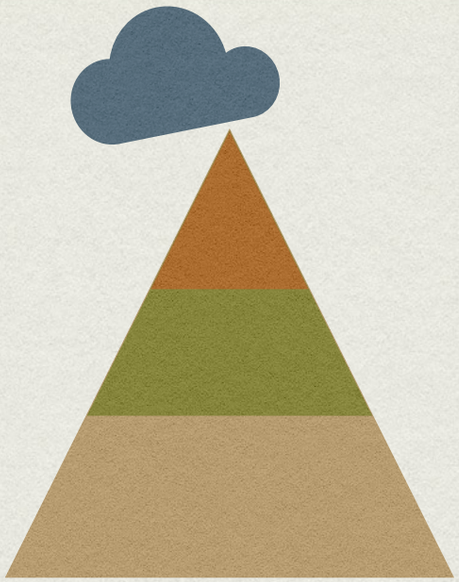
Slow to run

Data setup is super hard

Lots of regression confidence

Focus on business critical paths

* https://github.com/testdouble/contributing-tests/wiki/Testing-Pyramid



Integration Tests

Test a single feature

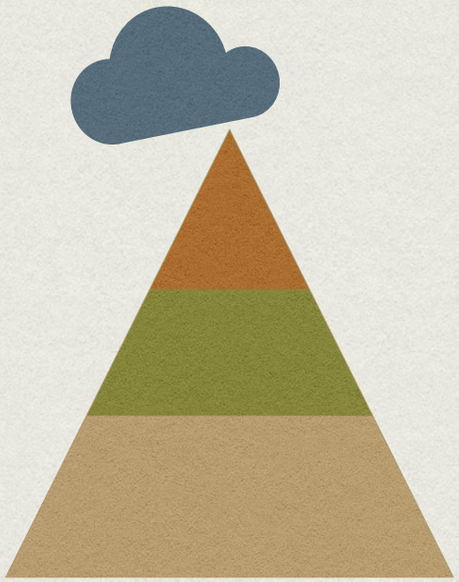
Faster to run

Data setup is difficult

Tests that pieces integrate properly

Focus on interfaces and error handling

* https://github.com/testdouble/contributing-tests/wiki/Testing-Pyramid



Unit Tests

Test a single unit of code

Very fast

Data setup is easy

Tests that a single unit functions properly

Focus on how various inputs affect behavior

* https://github.com/testdouble/contributing-tests/wiki/Testing-Pyramid

TEST DRIVEN

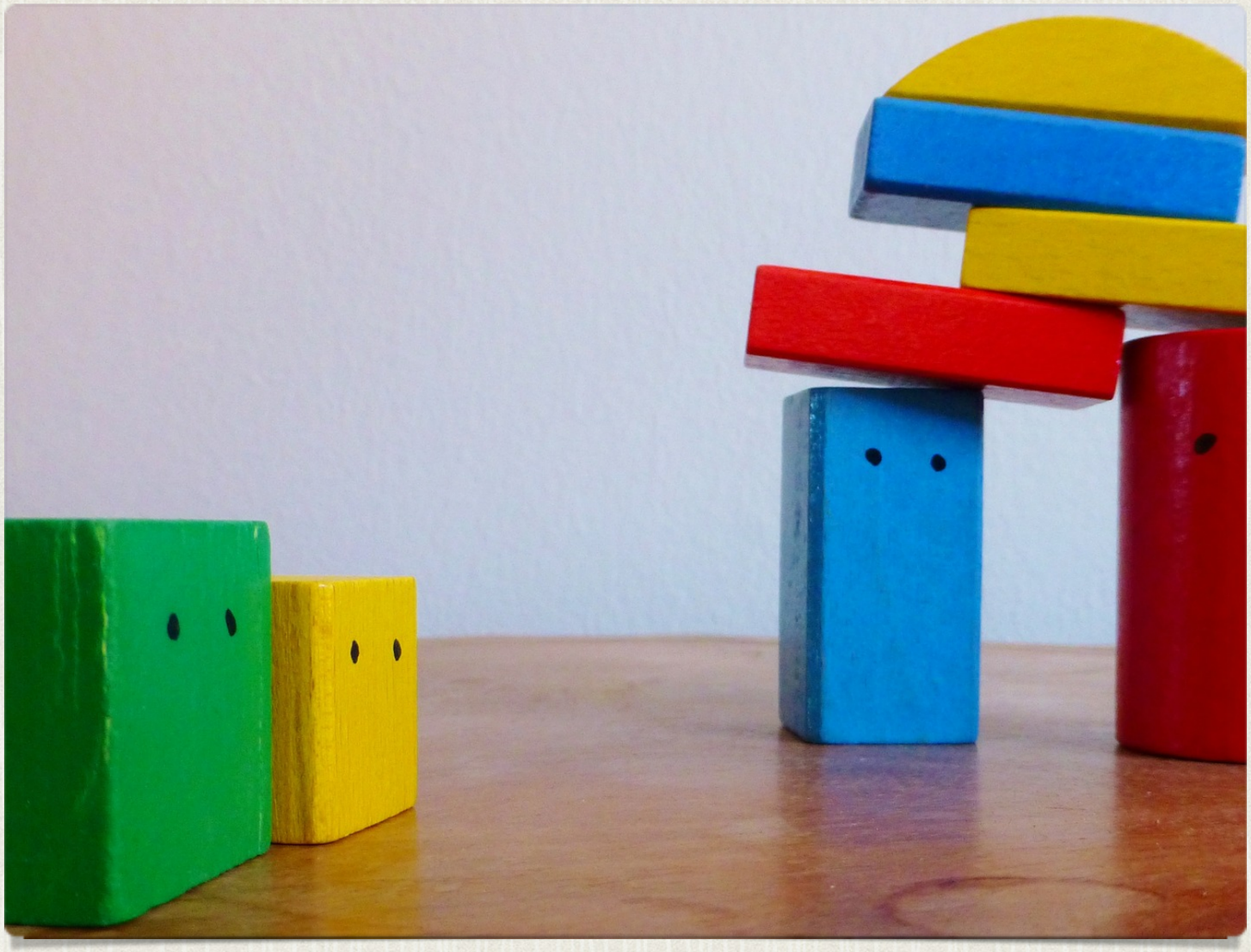
DESIGN

UNITS ARE WELL DESIGNED

Test Driven Design

UNITS INTERACT EASILY

Test Driven Design

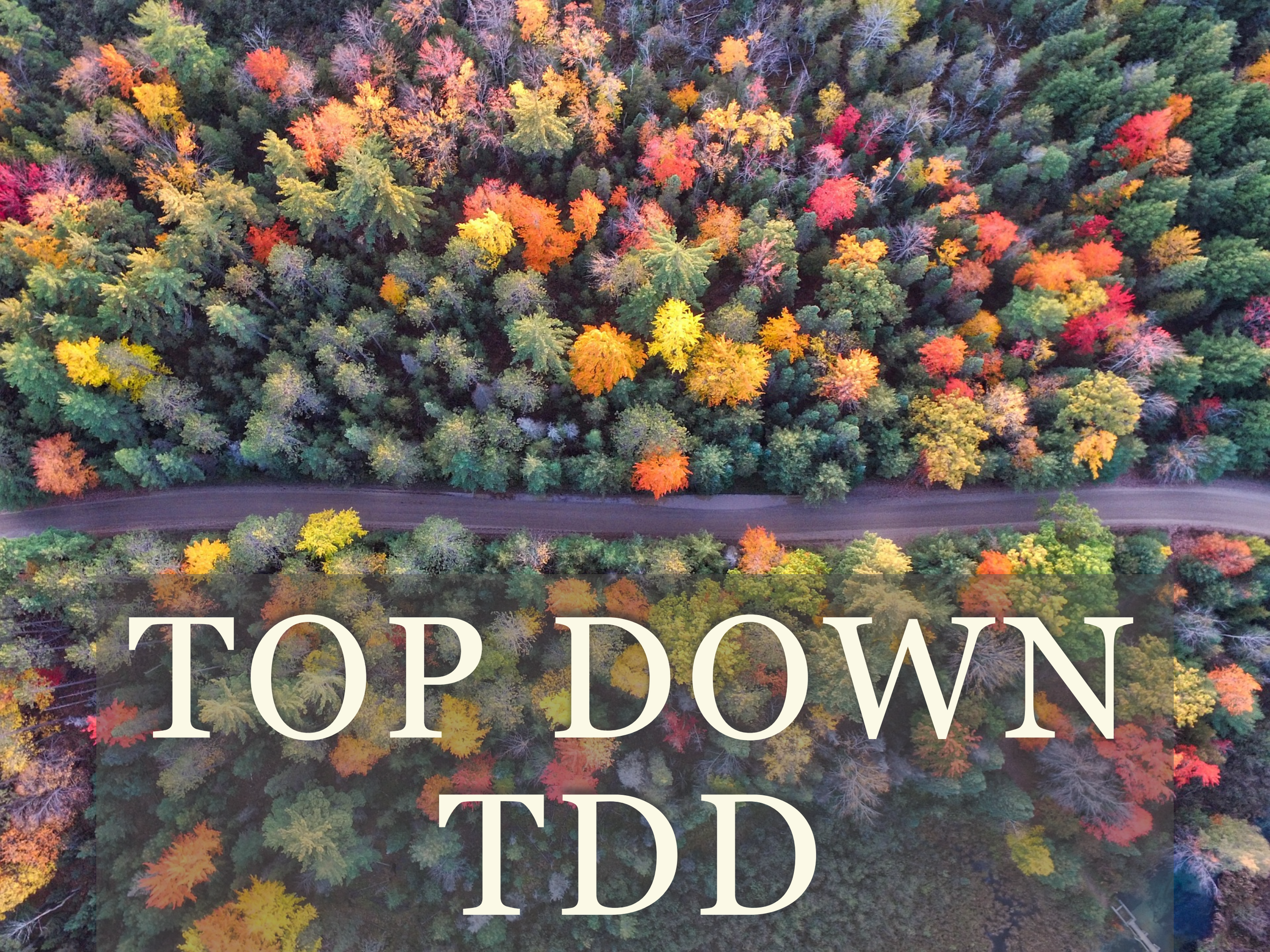


SEPARATED INTO COMPONENTS

Test Driven Design

COMPONENTS ARE FOCUSED

Test Driven Design



TOP DOWN TDD

PROCESS

1. Write a test

2. Make it pass

3. Repeat

WHAT KIND OF TEST?

Start with an acceptance test

Describe what would prove that the functionality of the feature is correct

Define the public API for the feature

WHAT KIND OF TEST?

Now an integration test

Describe what the first unit will accomplish

Define the interactions it will have with collaborating objects

FOR WHAT CODE?

The entry point

Implement the first unit to pass

Interact with the described collaborating objects

FOR WHAT CODE?

Repeat

For each collaborating unit, repeat integration testing process.

When you can't break the problem up anymore write a unit test

WHAT VALUE ARE WE GETTING OUT OF THESE TESTS?

ACCEPTANCE TEST

- Our unit is complete
- Our unit will not break
- Our unit is documented

COLLABORATING UNIT TESTS

- Components integrate easily
- Components are domain concepts

INTEGRATION TESTS

- Components integrate easily
- Components are domain concepts

UNIT TESTS

- Units function properly
- Every possible input has a documented outcome

THOUGHTFUL INTERACTIONS

Tests are a place to design interactions

NO MORE REFACTORING

Code design emerges naturally

CLEAN INTERNAL API

No fighting and refactoring awkward APIs

BIT.LY/SAC_TDD