How to Introduce TDD to Your Team
Part I: Understanding TDD

PRESENTED BY

James Shore

SCREENCAST: letscodejavascript.com
TWITTER: @jamesshore
EMAIL: jshore@jamesshore.com

Agile 2019
Washington, D.C.
6 August 2019

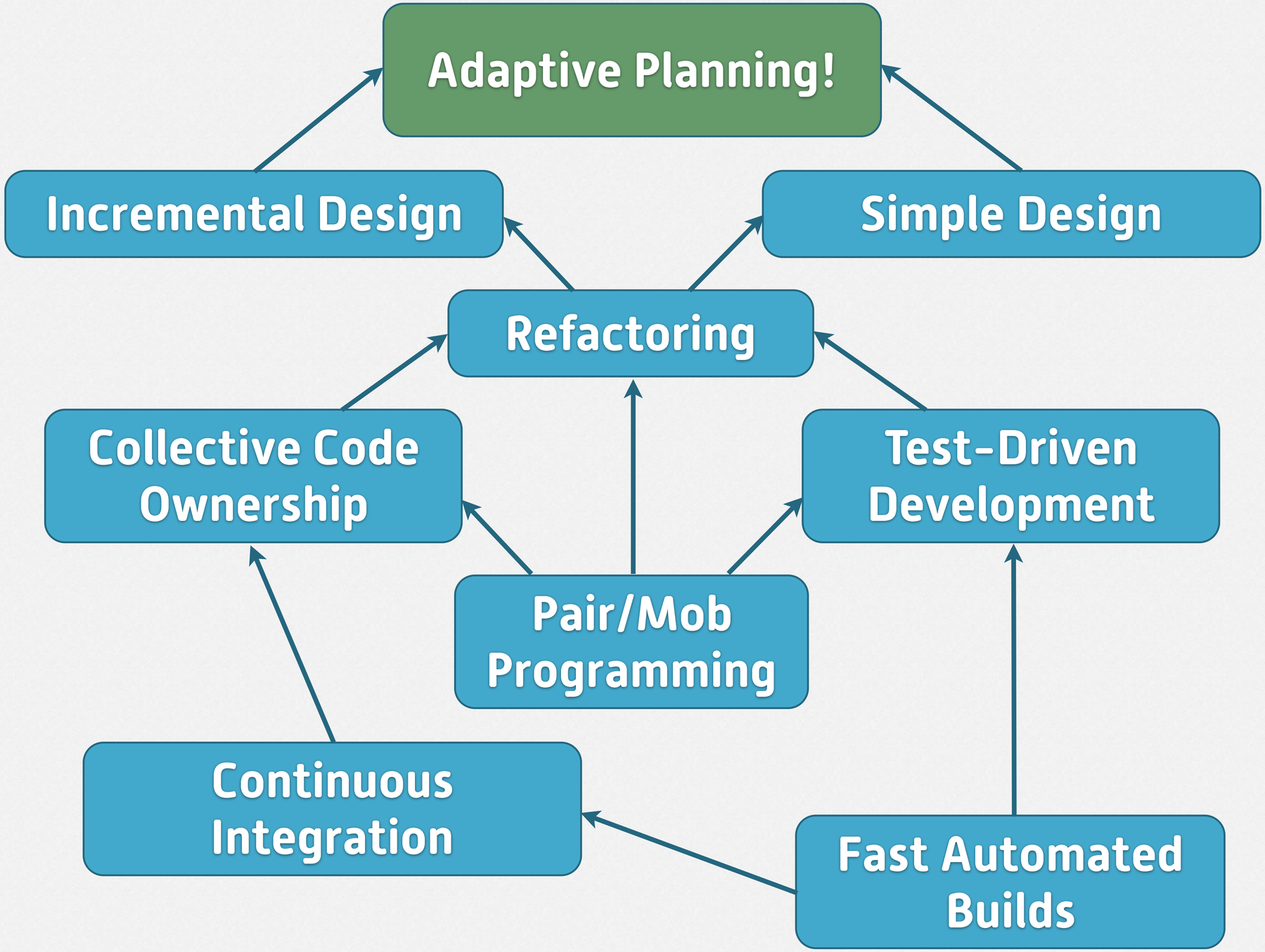
Today's Agenda

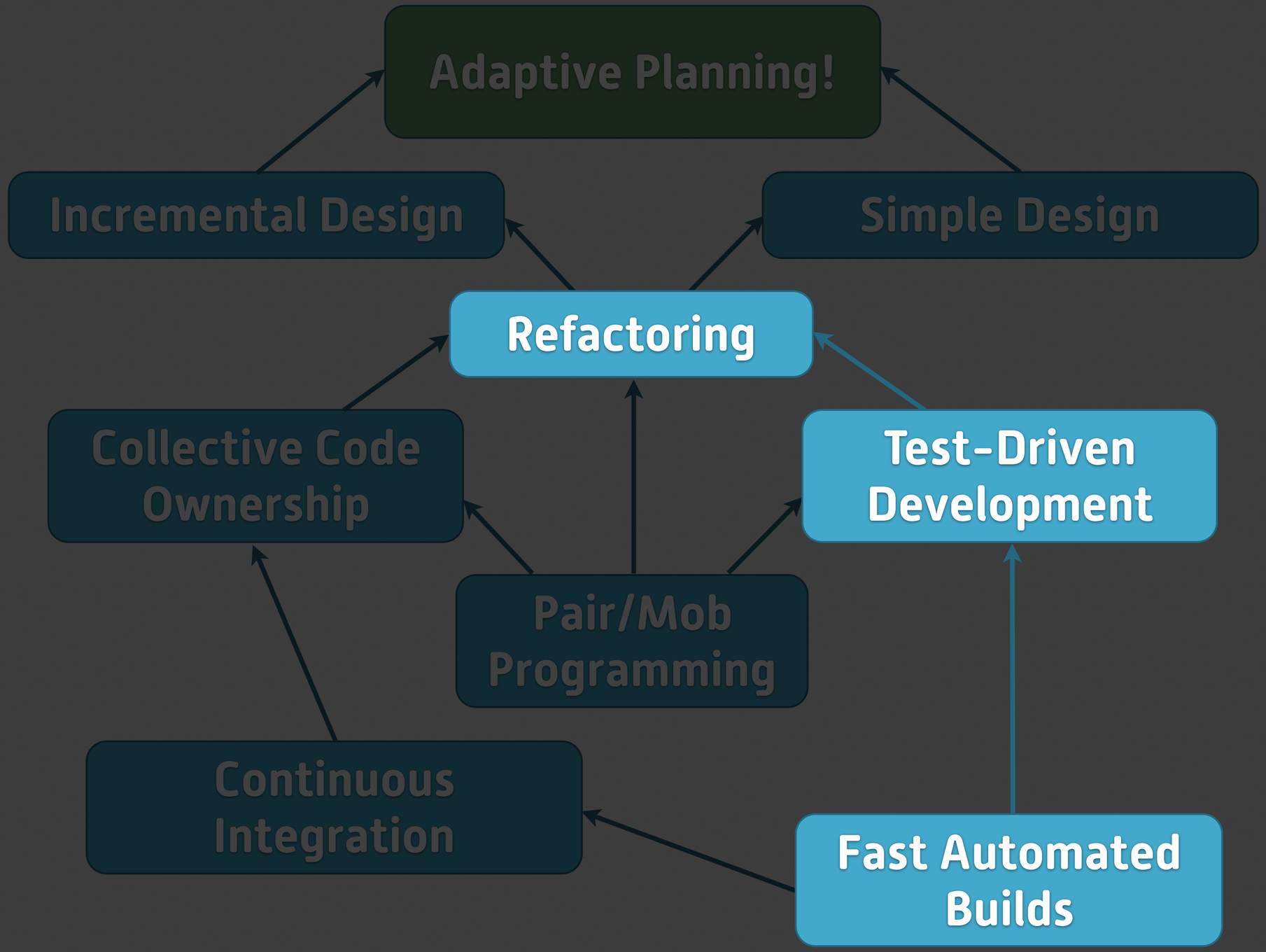
- **Part I: 9:00–10:15**
  Understanding TDD
    - Why TDD
    - The TDD Loop
    - Types of Tests
    - Making Improvements

- **Part II: 10:45–12:00**
  Facilitating a TDD Workshop

I. Understanding TDD

Why TDD?


Adaptive Planning!
Incremental Design
Simple Design
Refactoring
Collective Code Ownership
Test-Driven Development
Pair/Mob Programming
Continuous Integration
Fast Automated Builds


Adaptive Planning!
Incremental Design
Simple Design
Refactoring
Collective Code Ownership
Test-Driven Development
Pair/Mob Programming
Continuous Integration
Fast Automated Builds

Benefits of TDD vs. Traditional

- **Better Tests**
- **Improved Self-Discipline**
- **Fast Feedback**

I. Understanding TDD

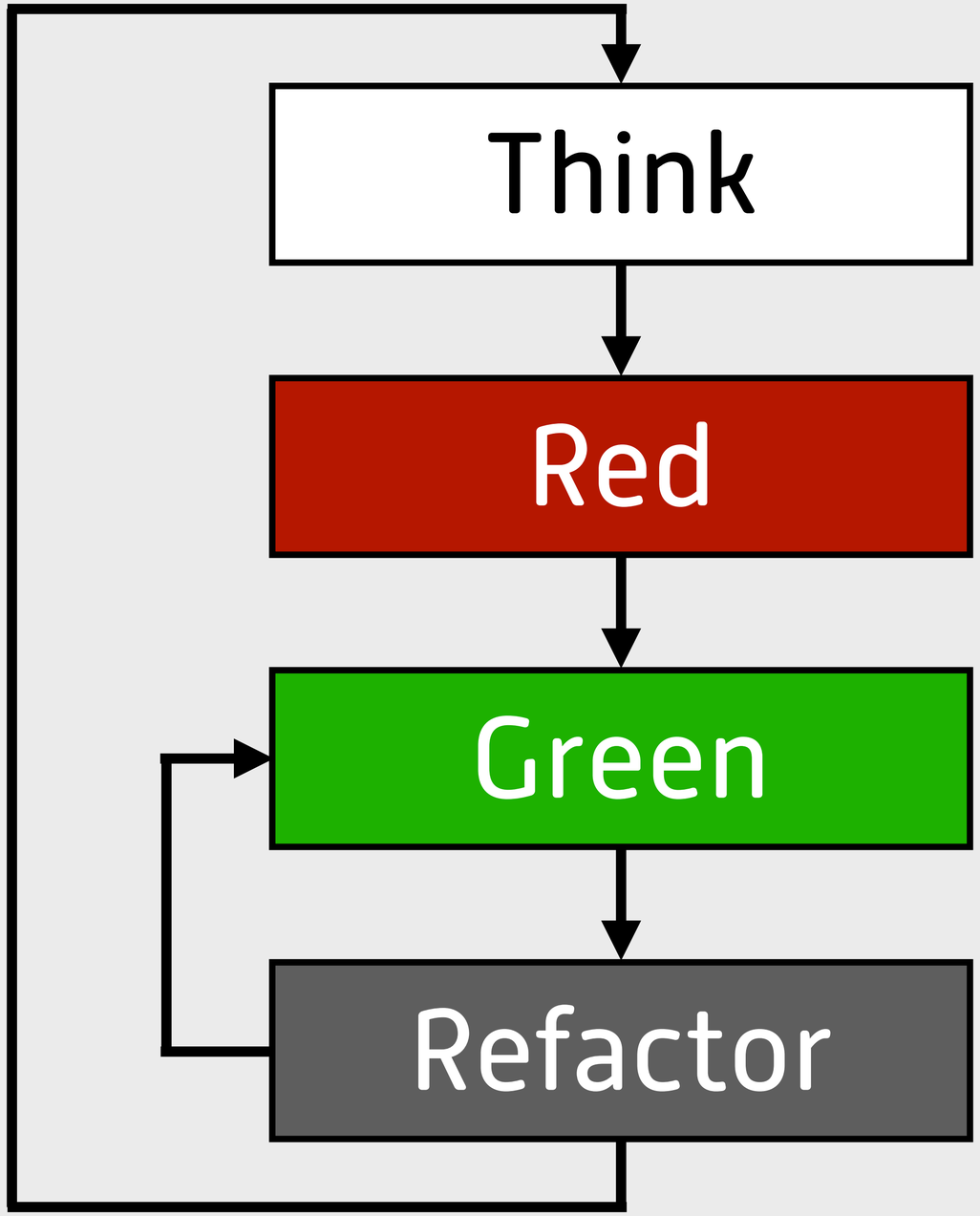
The TDD Loop

Guessing Game v1

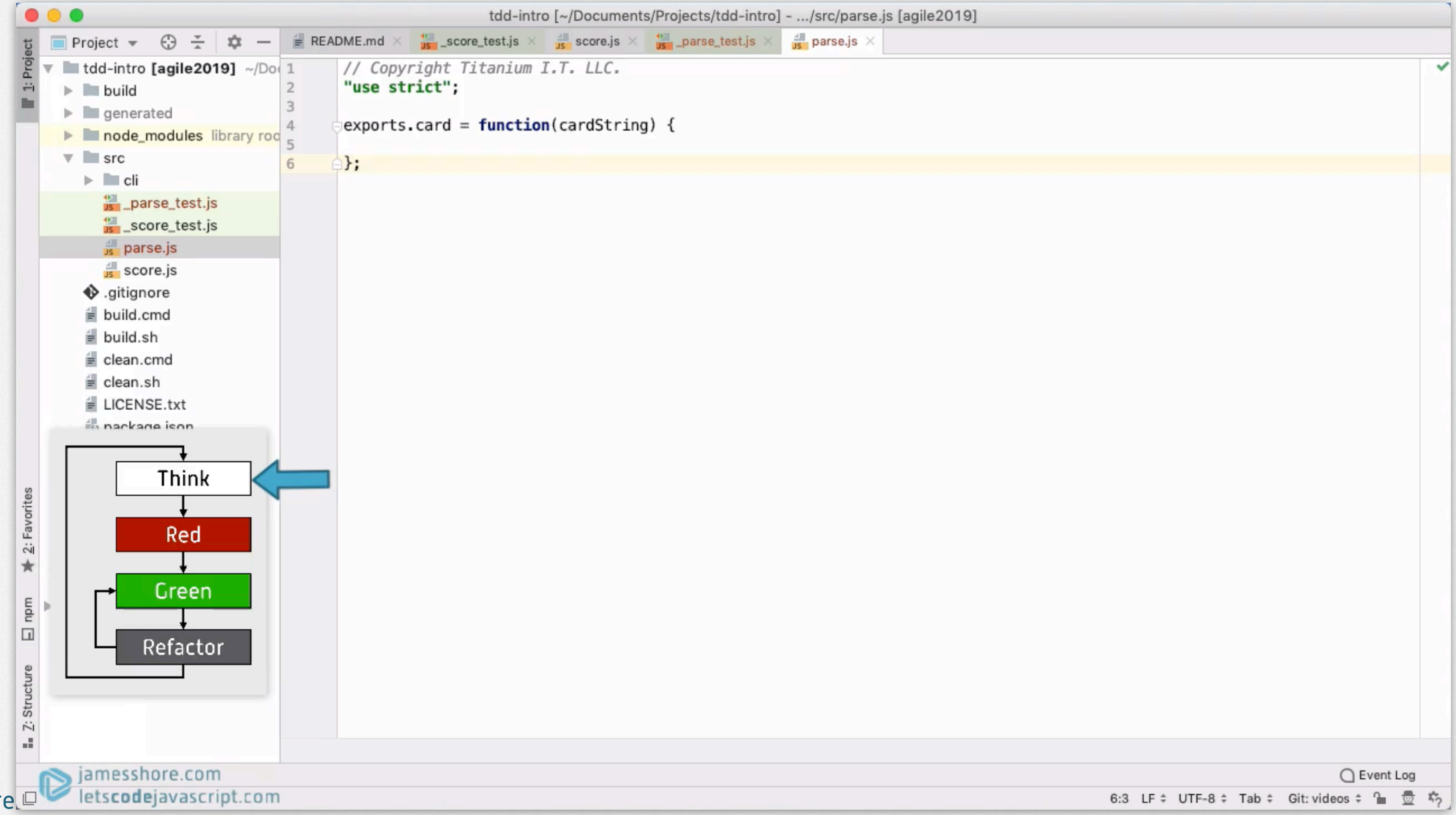
- Person 1: Think of a whole number between 1 and 100.
- Person 2: Make **four different guesses** of the number, each at least 5 digits apart.
  🚫 **51, 52, 53, 54** ✅ **51, 56, 61, 66**
- Person 1: Say how many guesses were high, low, or right on, **but don't say which guess is which.**
- **Repeat,** four guesses at a time, until you've guessed the number, then switch.

Guessing Game v2

- Person 1: Think of a whole number between 1 and 100.
- Person 2: Make **one guess** of the number.
- Person 1: Say if the guess was high, low, or right on.
- **Repeat,** one guess at a time, until you've guessed the number, then switch.


Think
Red
Green
Refactor

TDD is a Series of Validated Hypotheses



Why TDD Works Better

- **Better Tests:** Work is fine-grained, covering more edge cases.
- **Improved Self-Discipline:** It's easier to write tests as you go, and there's less temptation to move on to the next thing.
- **Fast Feedback:** TDD is a series of small, validated hypotheses.

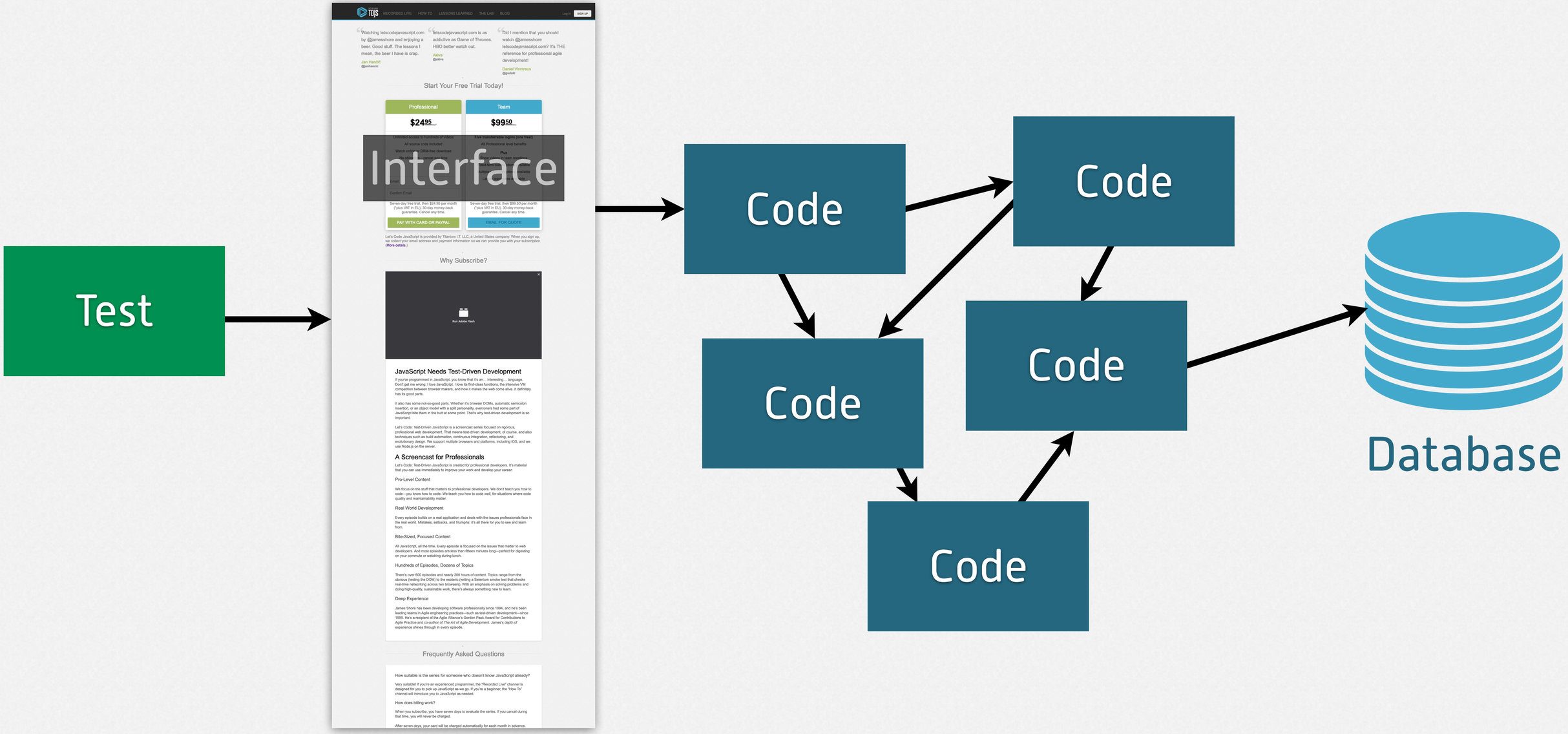
I. Understanding TDD

Types of Tests

Types of Tests

- End-to-end tests
- Focused Integration tests
- Unit tests

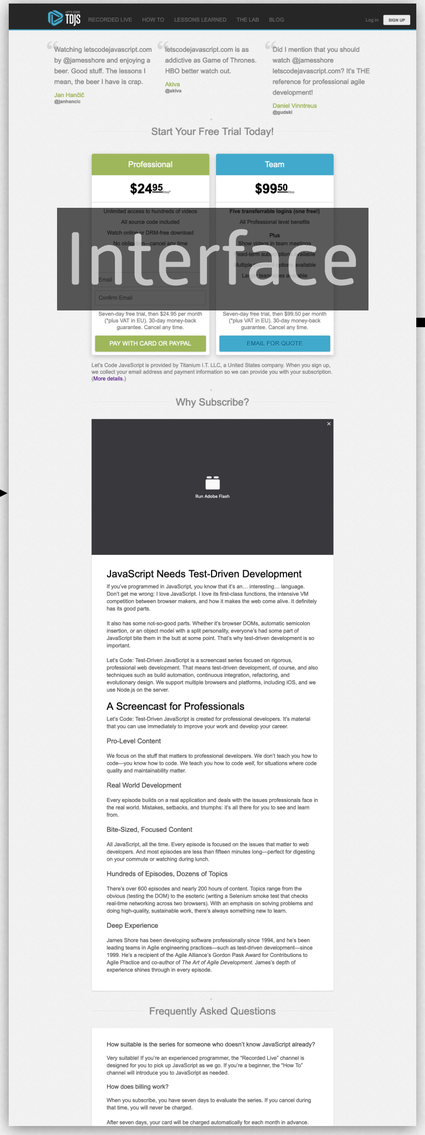
End-to-End Tests



End-to-End Tests

- ✅ Easy to conceive
- ✅ Checks real-world behavior
- ⛔ Expensive to write
- ⛔ Expensive to maintain
- ⛔ Prone to false failure
- ⛔ Very slow: tens of seconds per test
- ⛔ Untargeted feedback

Focused Integration Tests



Test

Code

Code

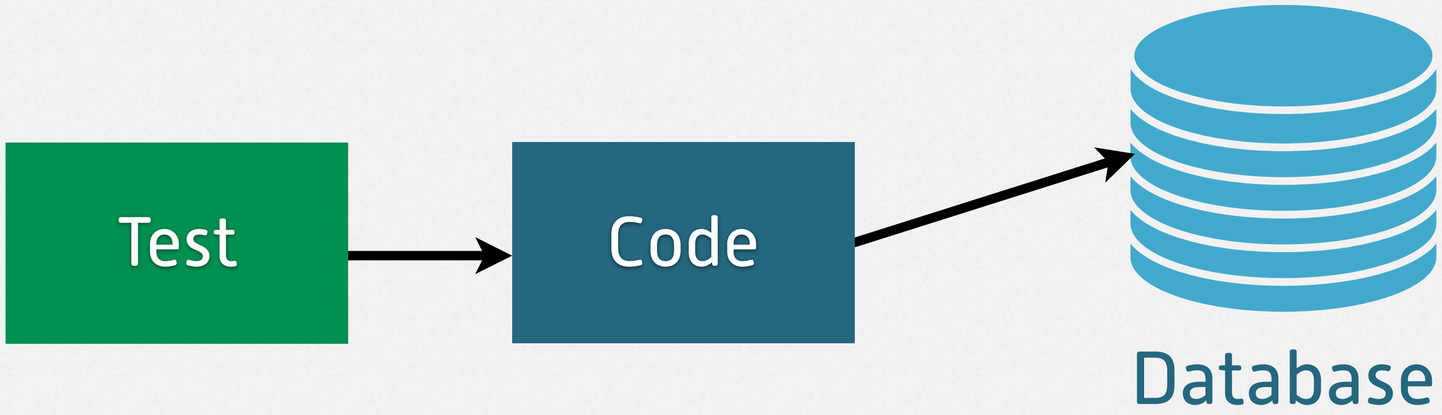
Code

Code

Code

Database

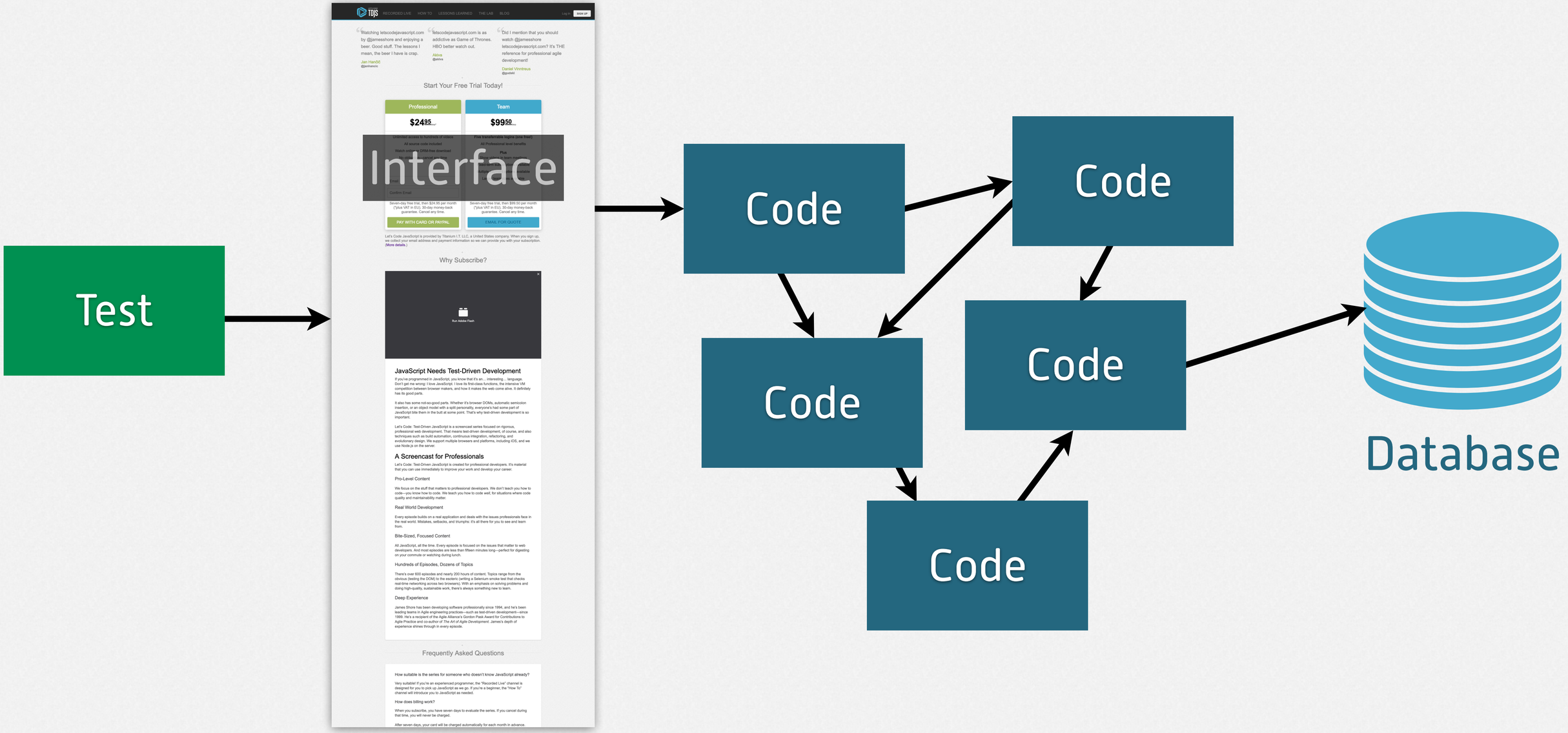
Focused Integration Tests



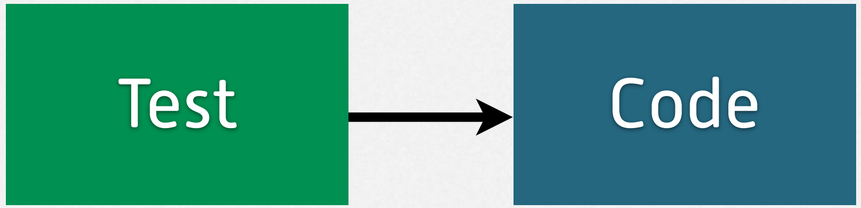
Focused Integration Tests

- ✅ Checks external systems
- ✅ Targeted feedback
- ✅ Faster than end-to-end tests…
- ⛔ …but still pretty slow: seconds per test
- ⛔ Expensive to write

Unit Tests



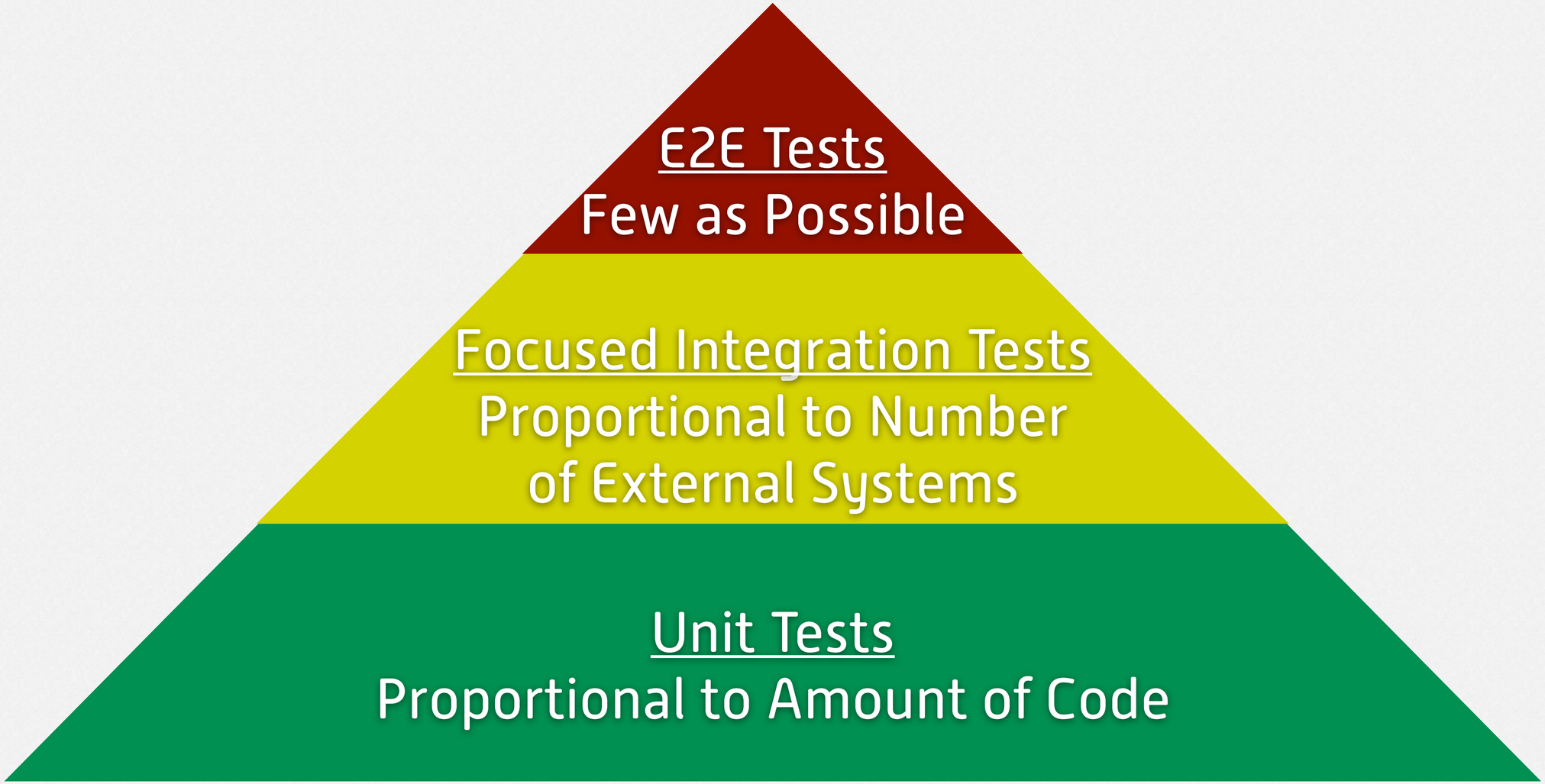
Unit Tests



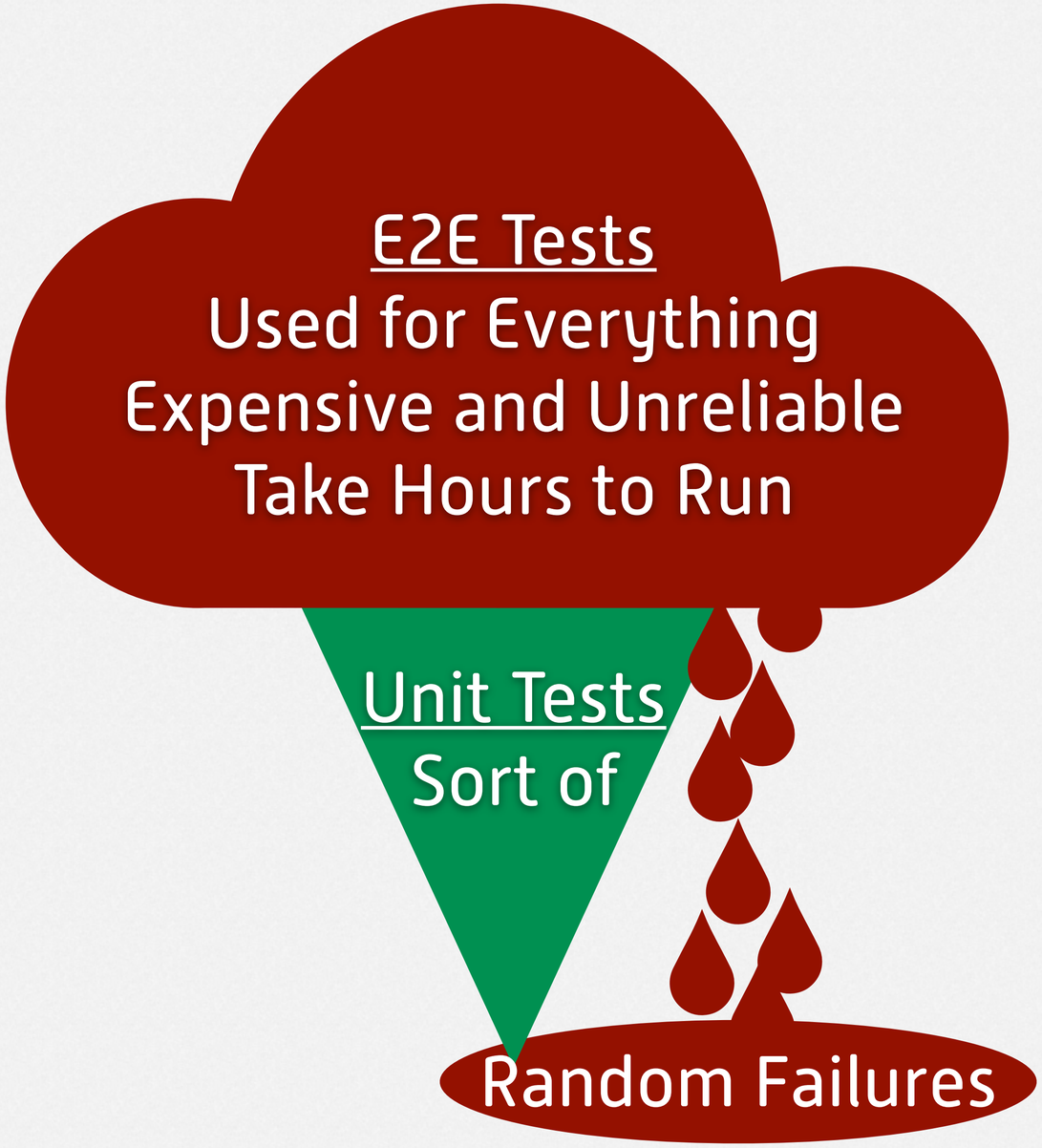
Unit Tests

- ✅ Very fast: hundreds of tests per second
- ✅ Very targeted feedback
- ✅ Cheap to write
- ⛔ Expensive to introduce
- ⛔ Challenging to master

Follow the Test Pyramid



Beware the Test Ice Cream Cone



Types of Tests

Table Discussion:

- What kinds of tests do your teams have?
- Are their tests more like the test pyramid, or the test ice cream cone?
- What results do you see?

I. Understanding TDD

Making Improvements

Enforcing Test Coverage is Easy…

…But Not Effective

- Easiest way to increase code coverage:
  End-to-end tests
- Even easier: **Bad** end-to-end tests

Enforcing code coverage will change behavior, but usually not the way you want.

Instead of Measuring Code Coverage

What do you want to accomplish?

- Better test and code practices
- Better code quality
- Better test discipline
- Add tests to legacy code
- Better meet non-functional requirements
- Create a culture of quality

To Improve Test and Code Practices

Analyze your defects to identify improvements

1. Perform root-cause analysis of escaped defects
2. Improve the design of the code that failed
3. Improve the processes that enabled the failure

To Improve Code Quality

Make it easier to write good tests

- Teach testing skills
- Speed up test feedback

Use simpler, easier-to-understand code design

- Refactor more
- Use evolutionary design

Improve test discipline

- See next slide

To Improve Test Discipline

Habits can’t be enforced, only nurtured

- Use pair programming or mob programming
- Provide technical coaching

To Add Tests to Legacy Code

Focus on the 20% that matters

- Before fixing a bug, add tests first
- When modifying existing code, retrofit it with tests first
- The 20% of code that is touched most often will improve most quickly

To Better Meet Non-Functional Req's

Even perfectly-tested code isn't perfect

- Use real-world telemetry and monitoring
- Write code to fail fast
- Create specialized testbeds

To Transform Your Technical Culture

Agile 2019 session:

Leading a 1,000 Person Technical Culture Transformation Without Resistance

Arlo Belshee & James Shore

Wednesday, 10:45am

Maryland Ballroom C

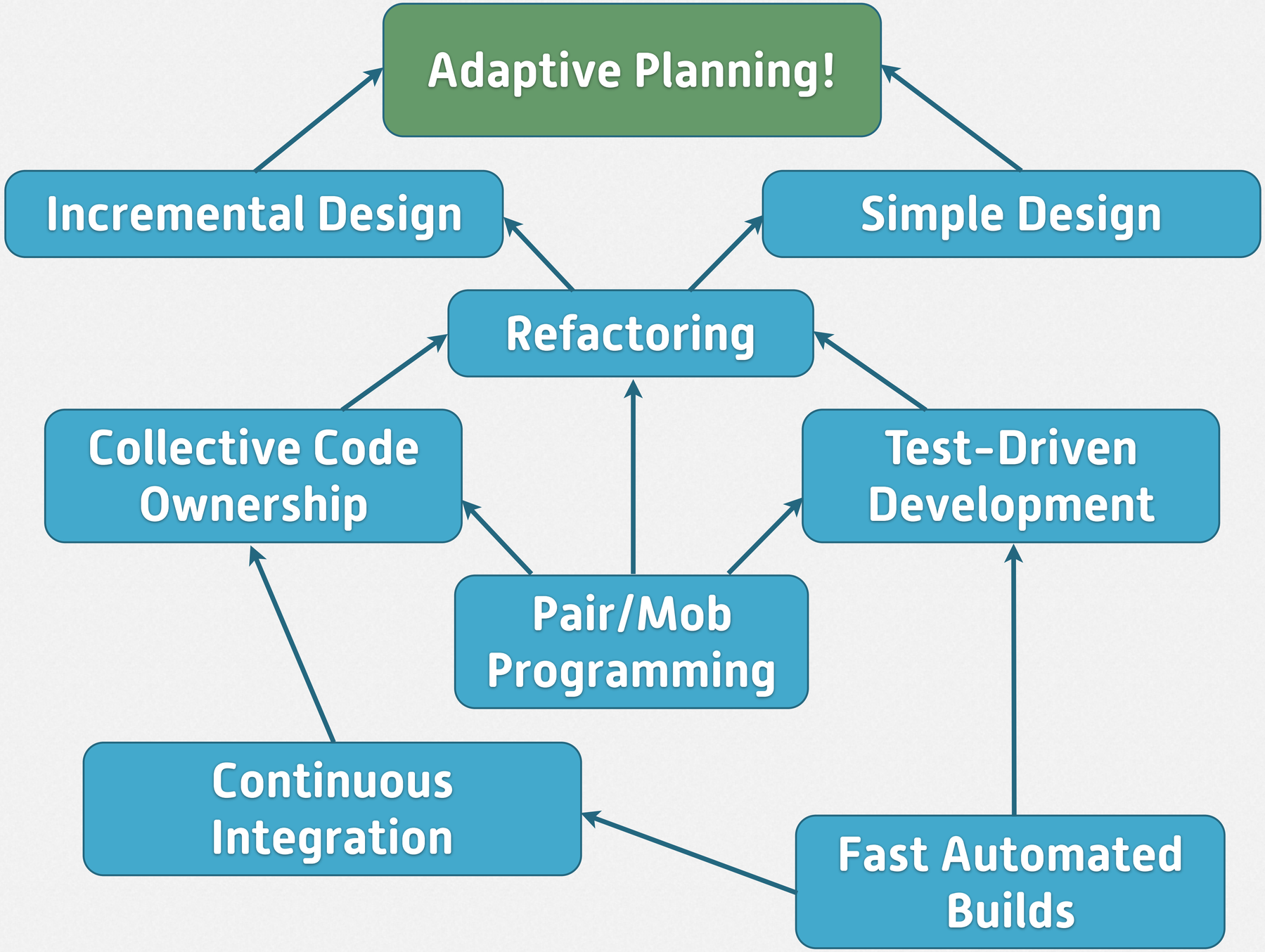
What Do You Want to Improve?

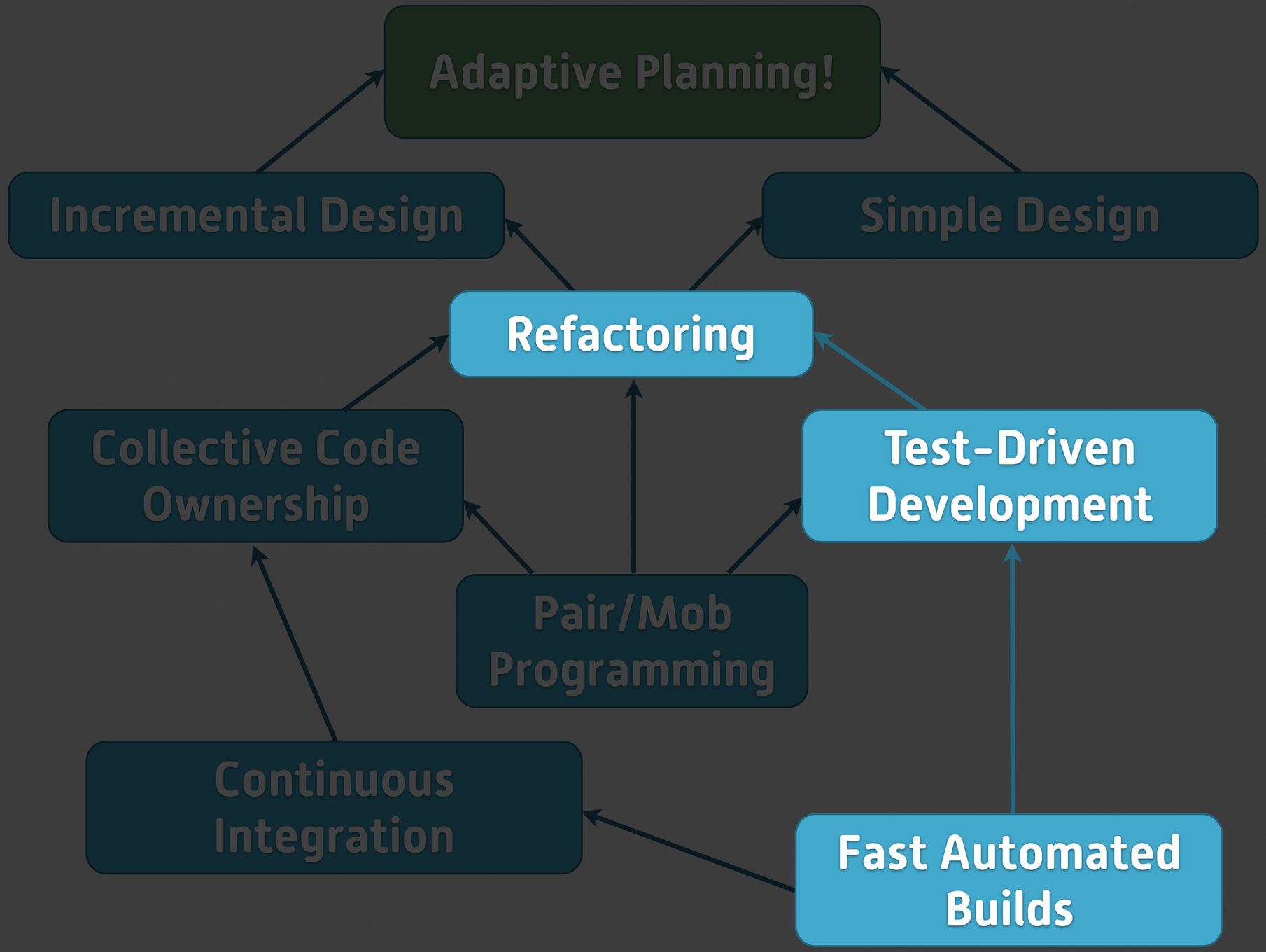
Table Discussion:

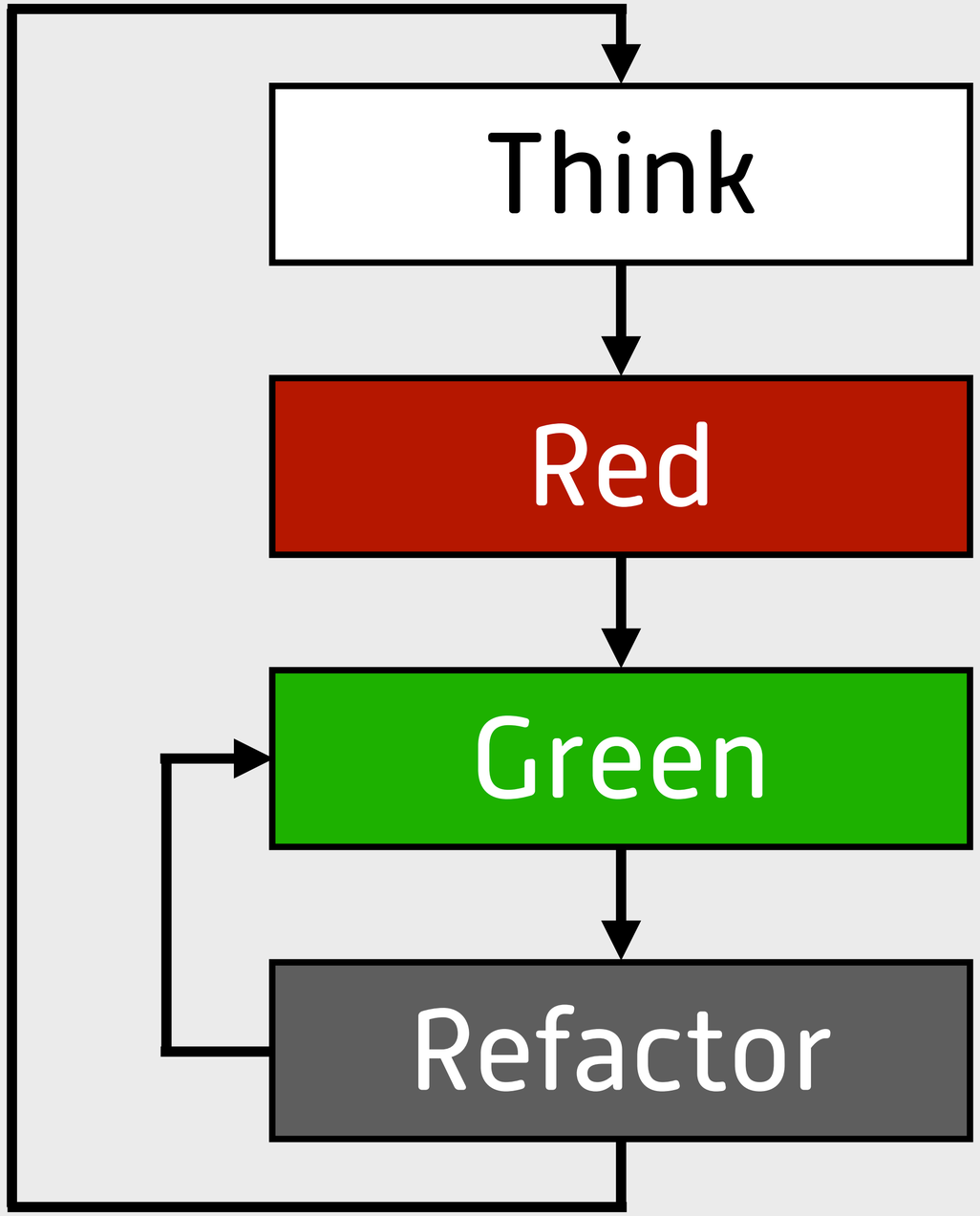
- What do you want your teams to accomplish?
- What can you do to help them?

I. Understanding TDD

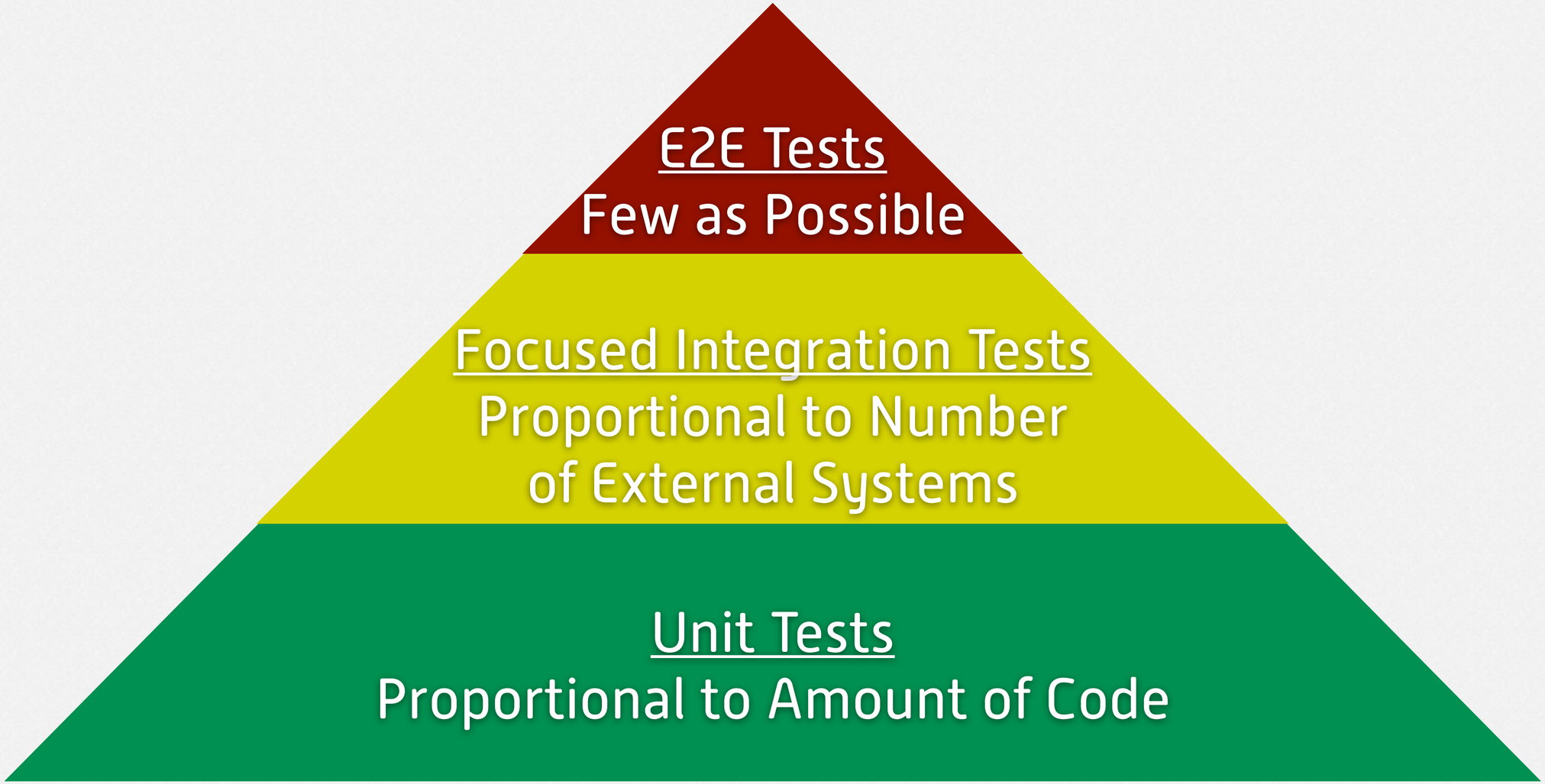
Conclusion


Adaptive Planning!
Incremental Design
Simple Design
Refactoring
Collective Code Ownership
Test-Driven Development
Pair/Mob Programming
Continuous Integration
Fast Automated Builds

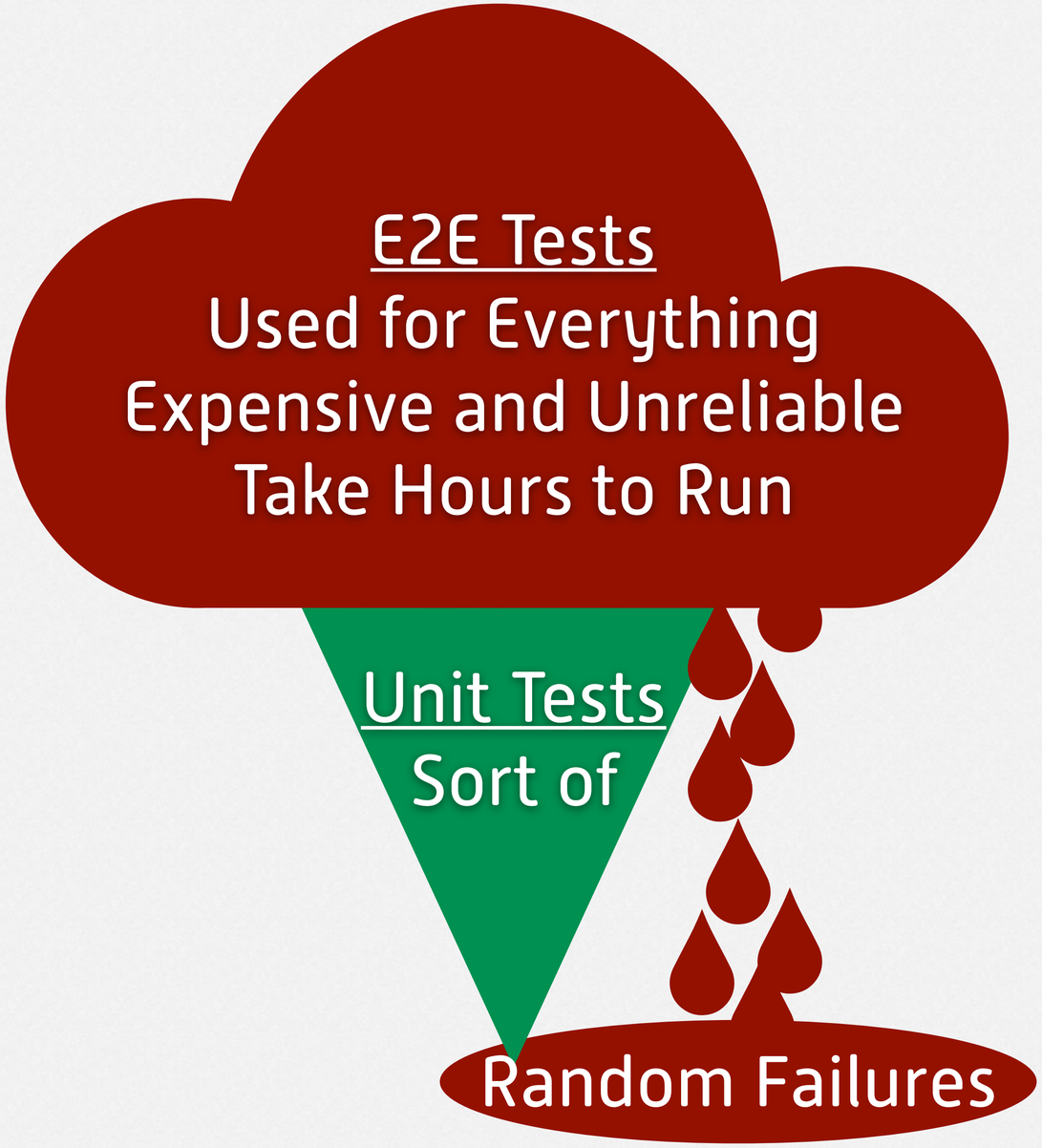

Adaptive Planning!
Incremental Design
Simple Design
Refactoring
Collective Code Ownership
Test-Driven Development
Pair/Mob Programming
Continuous Integration
Fast Automated Builds


Think
Red
Green
Refactor

Follow the Test Pyramid



Beware the Test Ice Cream Cone



Instead of Measuring Code Coverage

What do you want to accomplish?

- Better test and code practices
- Better code quality
- Better test discipline
- Add tests to legacy code
- Better meet non-functional requirements
- Create a culture of quality

Today's Agenda

- **Part I: 9:00–10:15**
  Understanding TDD

- **Part II: 10:45–12:00**
  Facilitating a TDD Workshop

How to Introduce TDD to Your Team
Part I: Understanding TDD

PRESENTED BY

James Shore

WEB: jamesshore.com
TWITTER: @jamesshore
EMAIL: jshore@jamesshore.com

Agile 2019
Washington, D.C.
6 August 2019

How to Introduce TDD to Your Team
Part II: Facilitating a TDD Workshop

PRESENTED BY

James Shore

WEB: jamesshore.com
TWITTER: @jamesshore
EMAIL: jshore@jamesshore.com

Agile 2019
Washington, D.C.
6 August 2019

Today's Agenda

- **Part I: 9:00–10:15**
  Understanding TDD

- **Part II: 10:45–12:00**
  Facilitating a TDD Workshop
    - About the Workshop
    - Preparing for the Workshop
    - Facilitating the Workshop

II. Facilitating a TDD Workshop

About the Workshop

The TDD Workshop

Four pre-recorded videos to assist you:

1. **Start Here** (4m 08s)
2. **Test Infrastructure** (10m 54s)
3. **The Core TDD Cycle** (8m 46s)
4. **TDD in Practice** (43m 57s)

Total time: 3+ hours

INTRODUCTION TO TDD EP2

Start Here (Facilitated Workshop Version)

Learning Goals

1. **Understand the core TDD loop**
2. **Use tests to drive design**
3. **Take small, predictable steps**

Not included:

Types of tests (the test pyramid / test ice cream cone)

Ways to improve

Video Contents

1. **Start Here** (4m 08s)
   Overview
   Setting up for the workshop
2. **Test Infrastructure** (10m 54s)
   Tooling needed
   Test structure (Arrange/Act/Assert)
3. **The Core TDD Cycle** (8m 46s)
   Think-Red-Green-Refactor-Repeat
4. **TDD in Practice** (43m 57s)
   Code-along example

Workshop Structure

1. **Understanding TDD**
   First three videos (25 min)
   Two exercises (10-15 min)
   Section Time: 45 min
2. **Practicing TDD**
   Final video (45 min)
   Several coding exercises (90+ min)
   Section Time: 2 hr 15 min+

Total time: 3+ hours

Your Role

1. **Prepare the Workshop**
   Select a venue
   Ensure computers are ready
   (Optional) Choose a tech lead to assist
2. **Facilitate the Workshop**
   Show the videos
   Guide the exercises
   Watch for common mistakes

Other Options

- **Self-Study Variant**
  letscodejavascript.com/v3/episodes/tdd_intro

- **Bring Me On-Site**
  Web: jamesshore.com
  Email: jshore@jamesshore.com
  Twitter: @jamesshore

Resources

- Videos:
  letscodejavascript.com/v3/episodes/tdd_intro
- GitHub Repository w/ code & materials:
  github.com/jamesshore/tdd-intro/

II. Facilitating a TDD Workshop

Preparing for the Workshop

Select the Venue

- Decide: pair programming or mob programming
- Provide a projector and sound system
- Ensure people can work comfortably and see screen
- Convey a professional mood (imagine fine dining)
- Find a way to prevent interruptions
- For maximum impact, especially for full-day version:
  - Provide food and drink
  - Go off-site

Ensure Computers Are Ready

1. Install Git: **git-scm.com**
2. Install Node.js: **nodejs.org**
3. Clone repository:
   git clone https://github.com/jamesshore/tdd-intro.git

- Email template:
  https://github.com/jamesshore/tdd-intro/blob/facilitator/facilitator/prep-email.md
- Double-check before workshop begins

Preparing for the Workshop

Make a personal checklist:

- What do you need to do to prepare for this workshop at your company?
- Do you have a tech lead that you can partner with? How would you get them involved?

Then discuss at your table.

II. Facilitating a TDD Workshop

Facilitating the Workshop

Workshop Structure

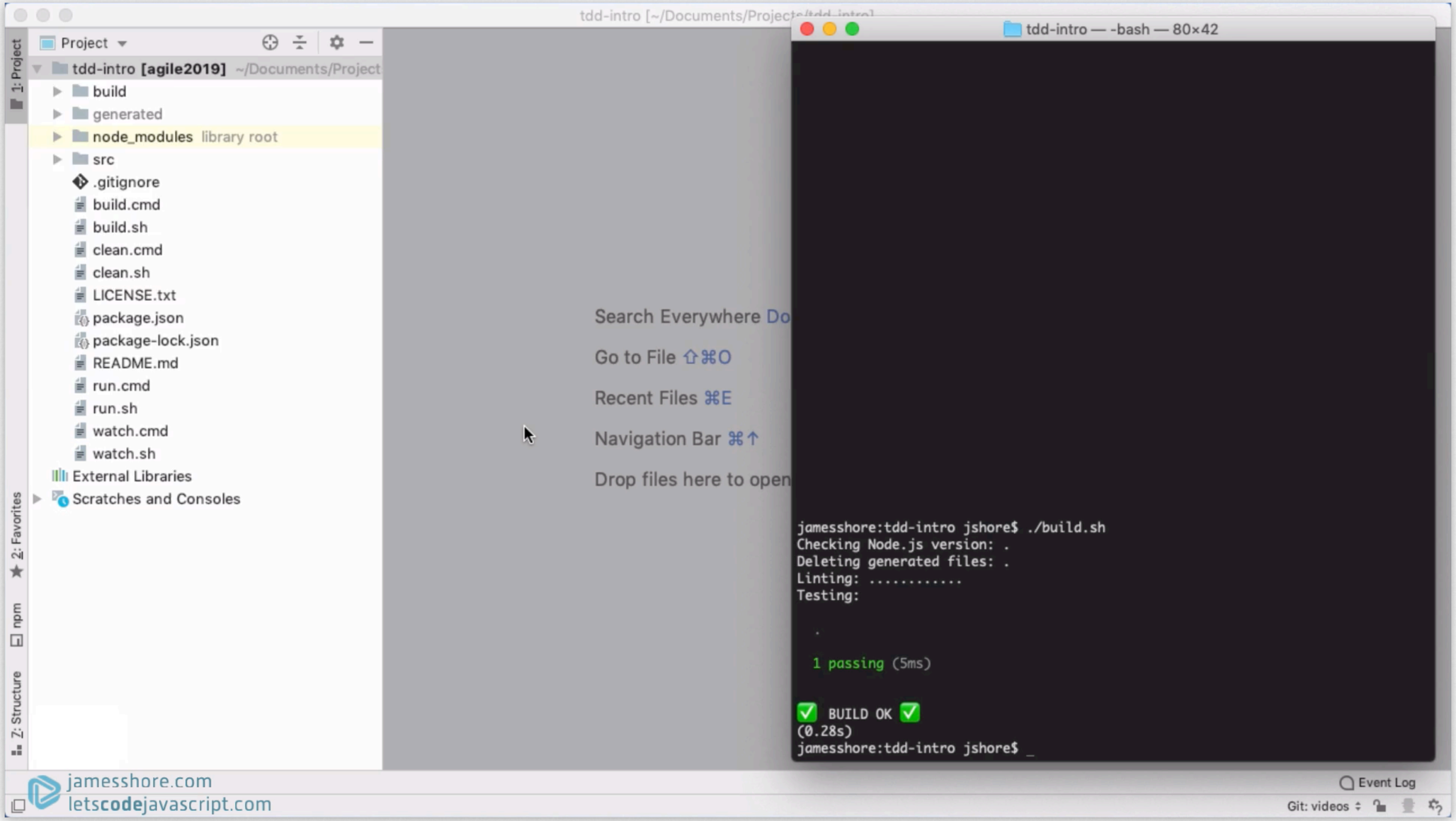
1. **Understanding TDD**
   First three videos (25 min)
   Two exercises (10-15 min)
   Section Time: 45 min
2. **Practicing TDD**
   Final video (45 min)
   Several coding exercises (90+ min)
   Section Time: 2 hr 15 min+

Total time: 3+ hours

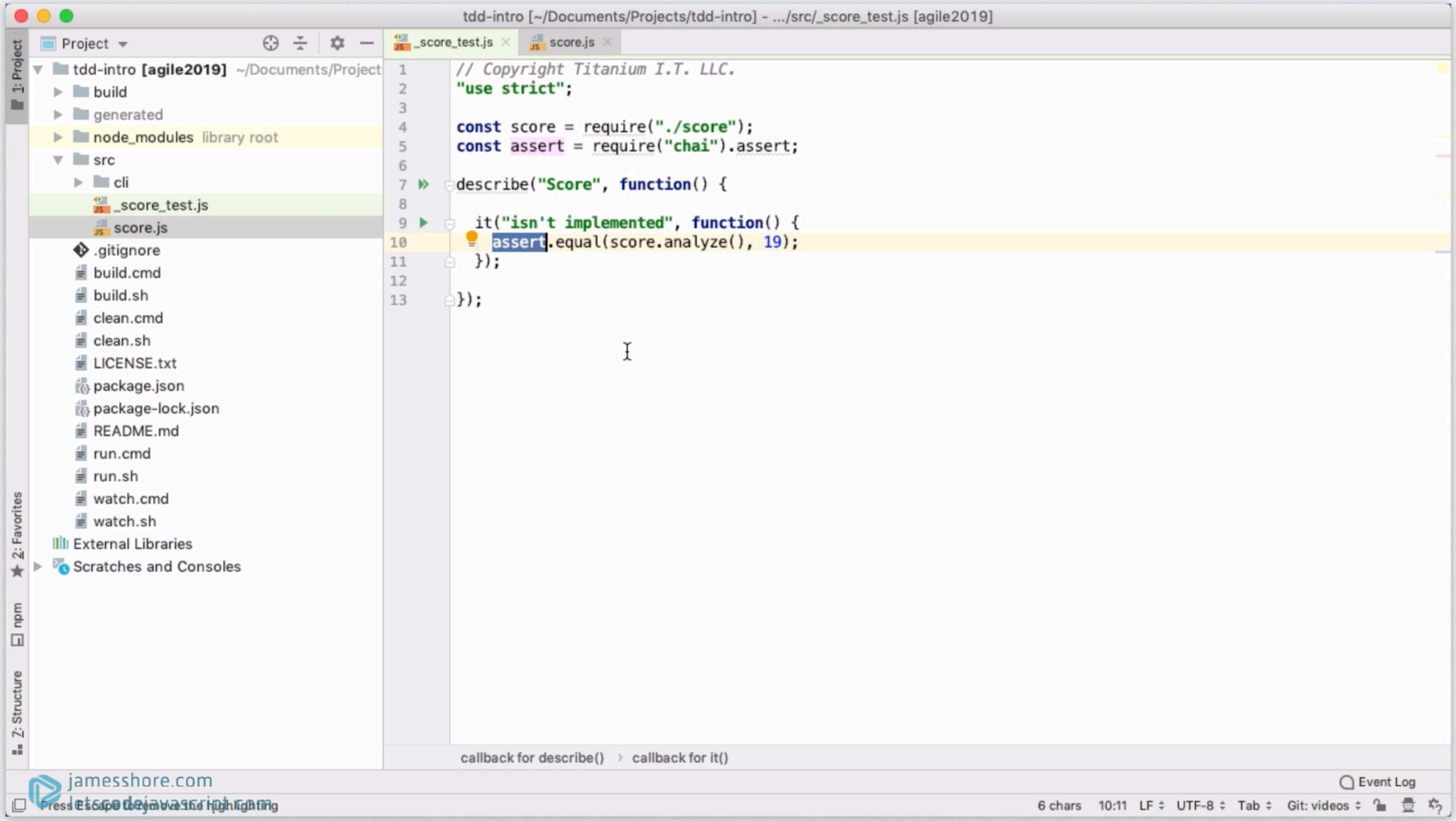
Workshop Part 1: Understanding TDD

1. **Understanding TDD**
   Video: Start Here (4 min) (Optional)
   Video: Test Infrastructure (11 min)
   Exercise: Think about your current infrastructure (5-10 min)
   Video: The Core TDD Cycle (9 min)
   Exercise: Reflect on the TDD cycle (5-10 min)
   Section Time: 45 min

The Test Infrastructure Video



The Test Infrastructure Exercise



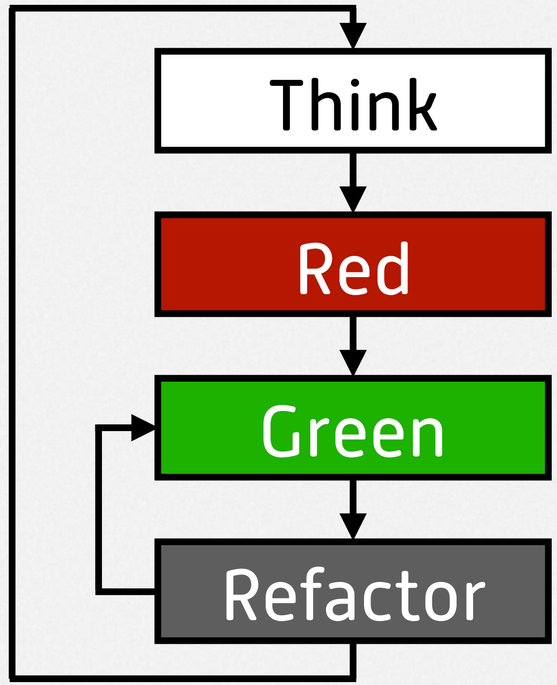
The Test Infrastructure Exercise

Make a few notes to yourself:

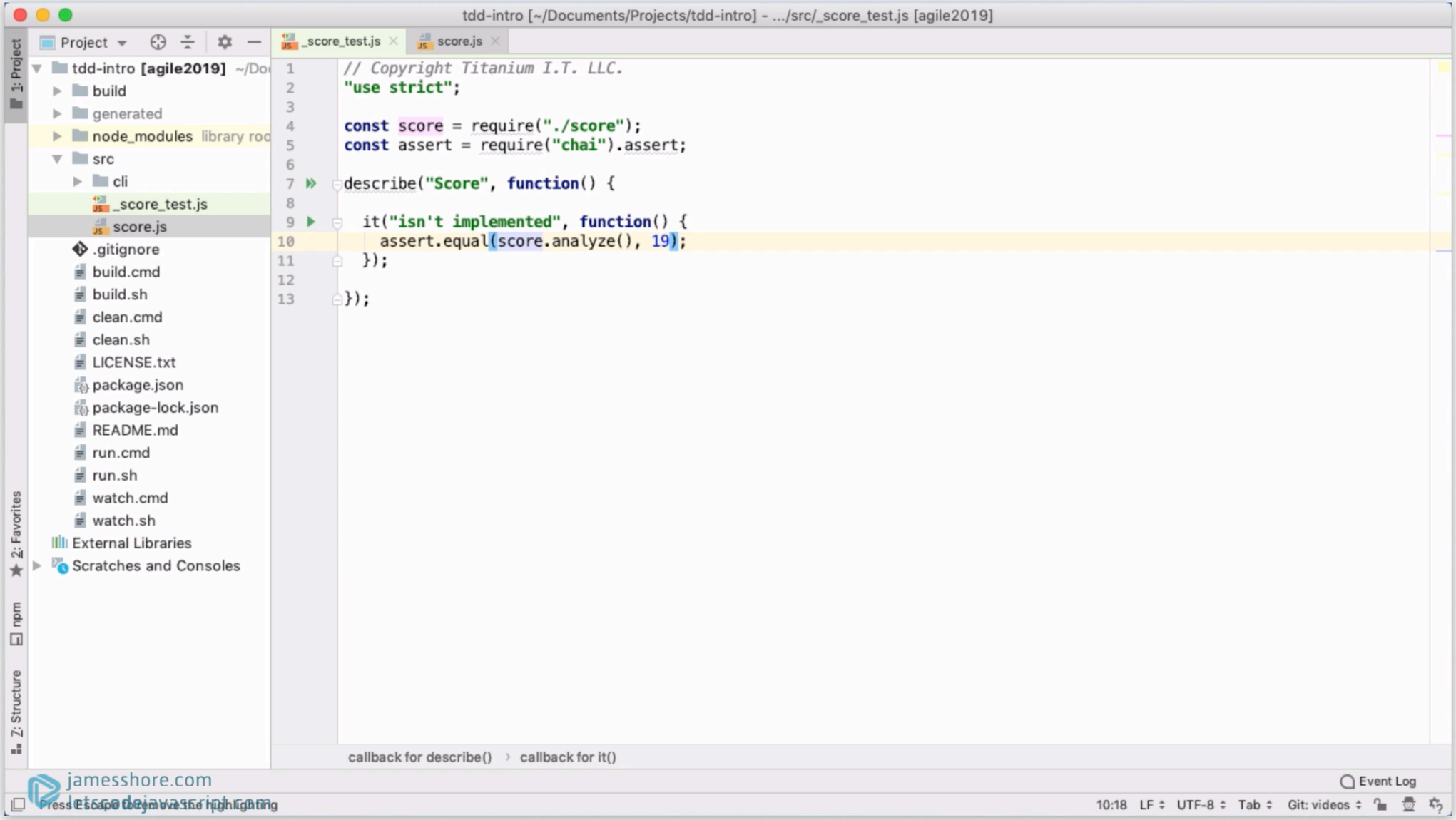
- How would you facilitate the test infrastructure exercise?

The TDD Cycle Video

- First half: Red-Green-Refactor explanation
- Second half: A very simple example (starts at 6m 15s)



The TDD Cycle Exercise



Facilitating Workshop Part 1

Make a few notes to yourself:

- How would you facilitate the TDD loop exercise?

Discuss at your table:

- Ideas for facilitating the whole “Understanding TDD” part of the workshop.

Workshop Part 2: Practicing TDD

2. Practicing TDD

Video: TDD in Practice (44 min, but interrupted with exercises)
Introduce the problem
- Exercise @2m18s: How can we break down the problem? **(Think)**
Break whole program down into “parse” and “score”
Break down “parse” into “parse many cards” and “parse one card”
Break down “parse one card” into “parse rank” and “parse suit”
Break down “parse rank” into “Create ‘parse’ file”
Create tests, ‘parse’ file, and ‘card’ file.
Solve “parse rank” problem.

Workshop Part 2: Practicing TDD

2. Practicing TDD (cont.)

…Solve “parse rank” problem.

- Exercise @18m40s: How can we refactor the code? **(Refactor)**
- (Optional) Exercise @22m20s: How can we refactor? **(Refactor)**
- Exercise @24m34s: What should we do next? **(Think)**

Answer: solve “parse suit” problem.

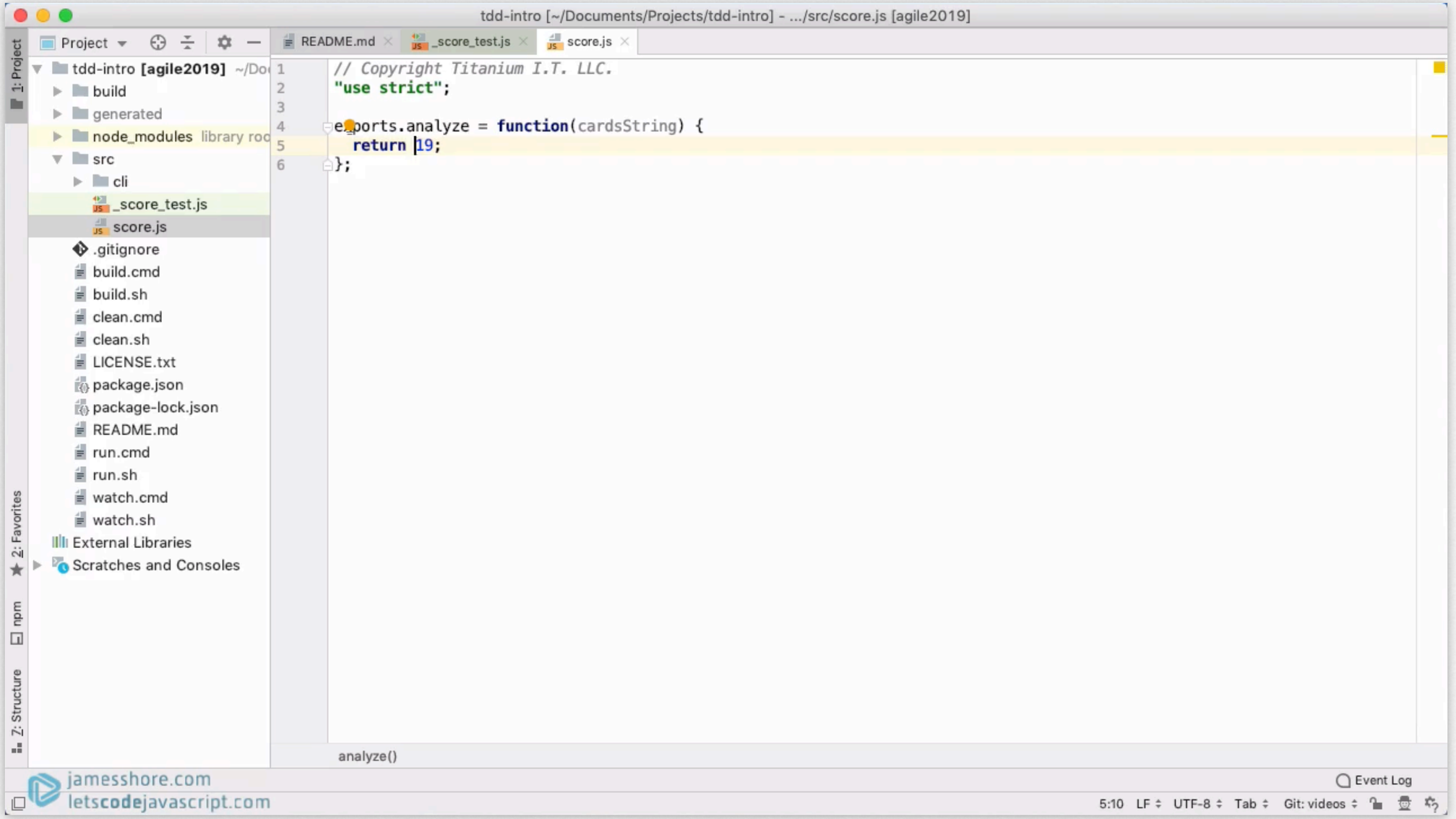
- Exercise @24m44s: Code next step yourself.

Discussion: best way to write tests

- Exercise @40m42s: What do you think is best way to write tests?
- Final exercise: Continue coding on your own.

Section Time: 2 hr 15 min+

Example “TDD in Practice” Prompt



Facilitating the Video Prompts

Make a few notes to yourself:

- How would you facilitate the video prompts?

The video prompts:

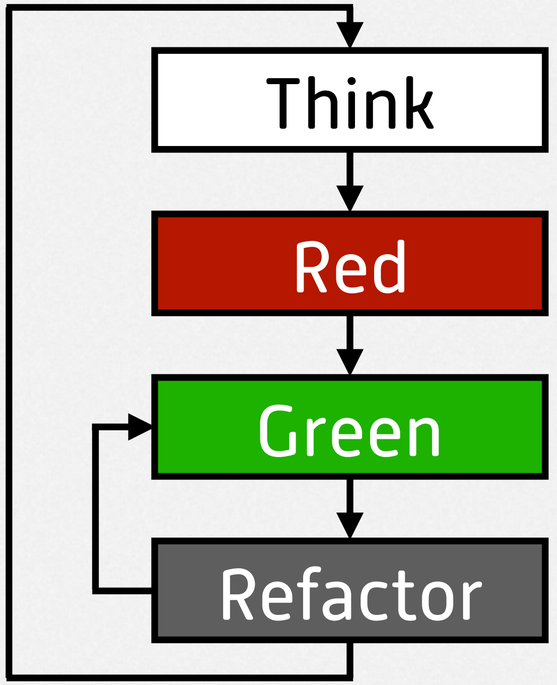
Exercise @2m18s: How can we break down the problem? **(Think)**
Exercise @18m40s: How can we refactor the code? **(Refactor)**
(Optional) Exercise @22m20s: How can we refactor? **(Refactor)**
Exercise @24m34s: What should we do next? **(Think)**

Common TDD Pitfalls

Observe:
Spending too long on “Red”

Common Mistake:
Taking big steps instead of little steps

How to facilitate:
Ask, “How could you break this problem down smaller?”

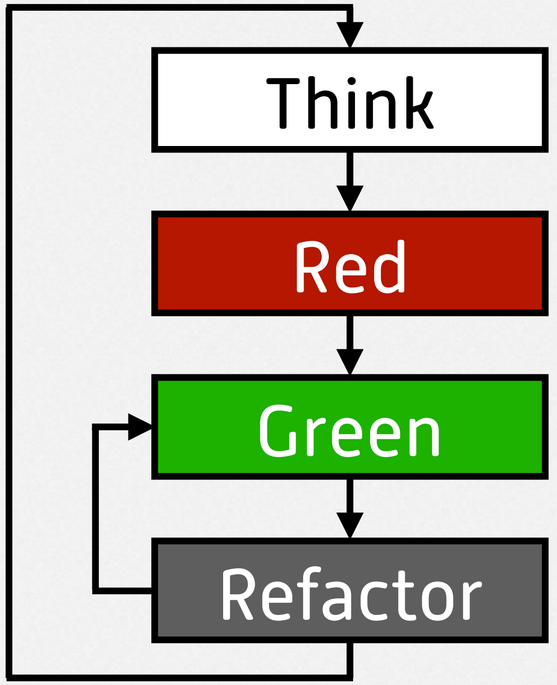


Common TDD Pitfalls

Observe:
Spending too long on “Green”

Common Mistake:
Seeking perfection rather than waiting to refactor

How to facilitate:
Ask, “Could you solve this problem in a simple, dumb way, then refactor?”

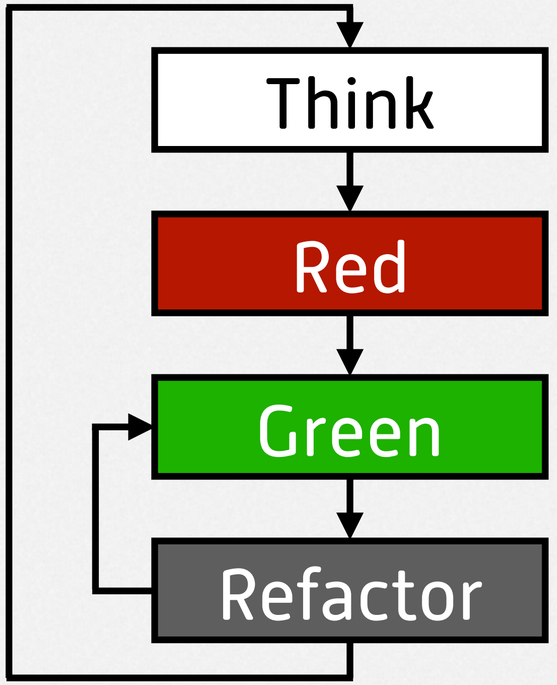


Common TDD Pitfalls

Observe:
Not running the tests very often

Common Mistake:
Tests that are too big or writing too many tests

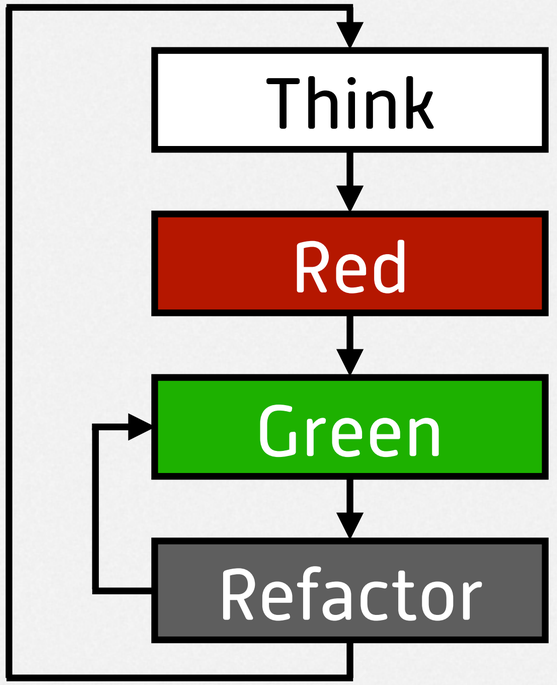
How to facilitate:
Ask, “How could you build up this test incrementally?”



Facilitating the Coding Exercises

What to do after the video ends:

- Solve “parse all cards” problem
- Break down “score” into multiple pieces
- Solve each piece of “score” (in any order):
  - Score pairs of cards
  - Score straights
  - Score flushes
  - Score Jacks (“nobs”)
  - Score 15s
  - Score multiple combinations
- Connect it all together



Workshop Structure

1. **Understanding TDD**
   First three videos (25 min)
   Two exercises (10-15 min)
   Section Time: 45 min
2. **Practicing TDD**
   Final video (45 min)
   Several coding exercises (90+ min)
   Section Time: 2 hr 15 min+

Total time: 3+ hours

II. Facilitating a TDD Workshop

Conclusion

Workshop Structure

1. **Understanding TDD**
   First three videos (25 min)
   Two exercises (10-15 min)
   Section Time: 45 min
2. **Practicing TDD**
   Final video (45 min)
   Several coding exercises (90+ min)
   Section Time: 2 hr 15 min+

Total time: 3+ hours

Your Role

1. **Prepare the Workshop**
   Select a venue
   Ensure computers are ready
   (Optional) Choose a tech lead to assist
2. **Facilitate the Workshop**
   Show the videos
   Guide the exercises
   Watch for common mistakes

Other Options

- **Self-Study Variant**
  letscodejavascript.com/v3/episodes/tdd_intro

- **Bring Me On-Site**
  Web: jamesshore.com
  Email: jshore@jamesshore.com
  Twitter: @jamesshore

How to Introduce TDD to Your Team
Part II: Facilitating a TDD Workshop

PRESENTED BY

James Shore

WEB: jamesshore.com
TWITTER: @jamesshore
EMAIL: jshore@jamesshore.com

Agile 2019
Washington, D.C.
6 August 2019