

TEST-DRIVEN DEVELOPMENT

PRINCIPLES & PRACTICES



Doug Klugh



doug@CodeFlair.io



@DougKlugh



/Klugh



/Klugh



/Klugh



CodeFlair.io



DOUG KLUGH

SOFTWARE CRAFTSMAN, LEADER, TEACHER & STUDENT

- ✓ B.S. IN COMPUTER SCIENCE (SOFTWARE ENGINEERING)
- ✓ 34 YEARS DELIVERING FIRMWARE/SOFTWARE FOR SOME OF THE WORLD'S TOP BRANDS
- ✓ 20+ YEARS OF PRACTICE LEADERSHIP SPANNING ALL SDLC DISCIPLINES
- ✓ 15+ YEARS OF AGILE LEADERSHIP
- ✓ DELIVERED CONSUMER & ENTERPRISE SOFTWARE SERVICING > ONE BILLION USERS



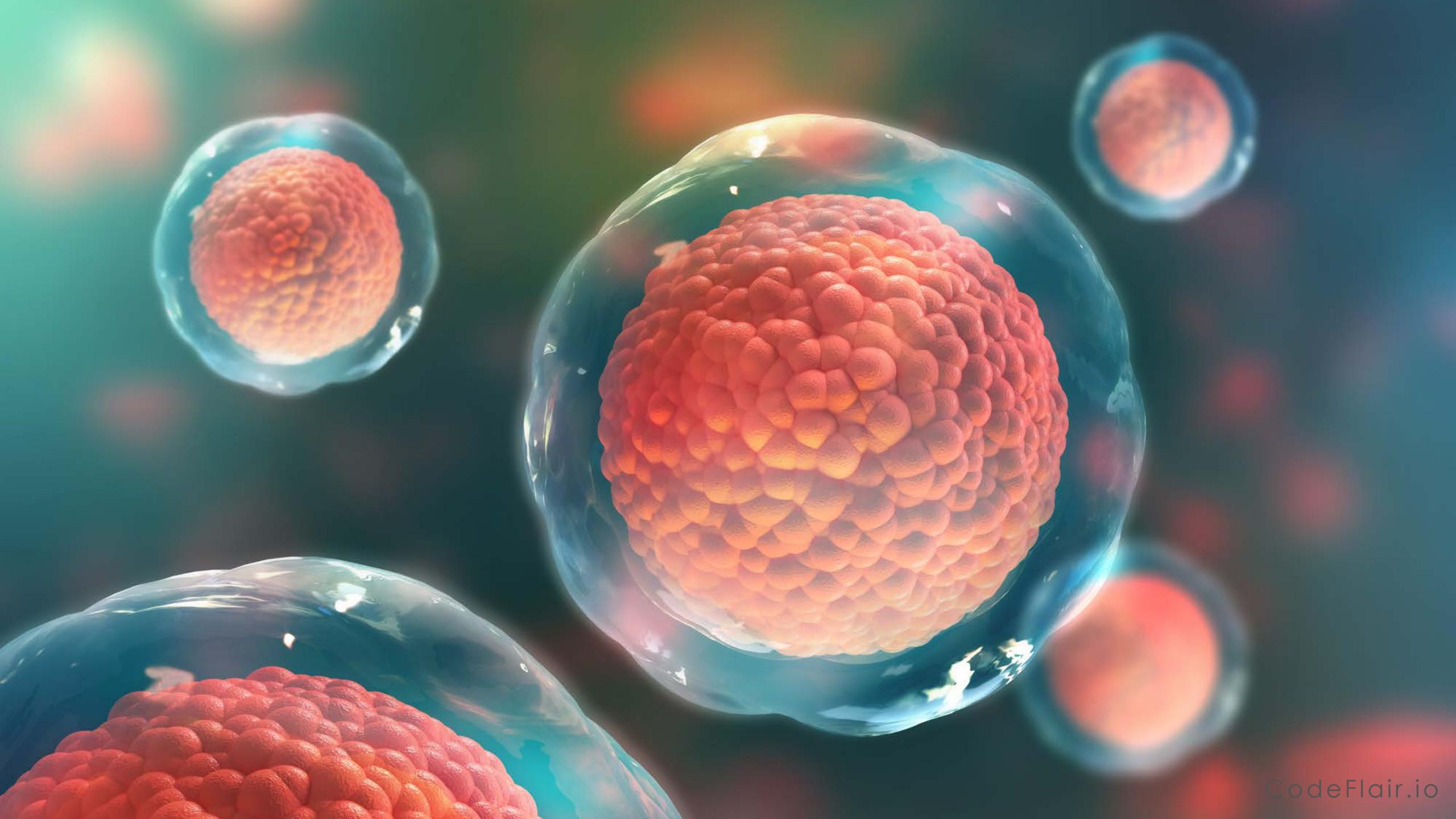
AGENDA

- ✓ UNIT TESTS
- ✓ TEST-DRIVEN DEVELOPMENT
- ✓ TDD vs ATDD
- ✓ LONDON vs CHICAGO
- ✓ TRANSFORMING TO GREEN
- ✓ TDD BY EXAMPLE
- ✓ CONCLUSION













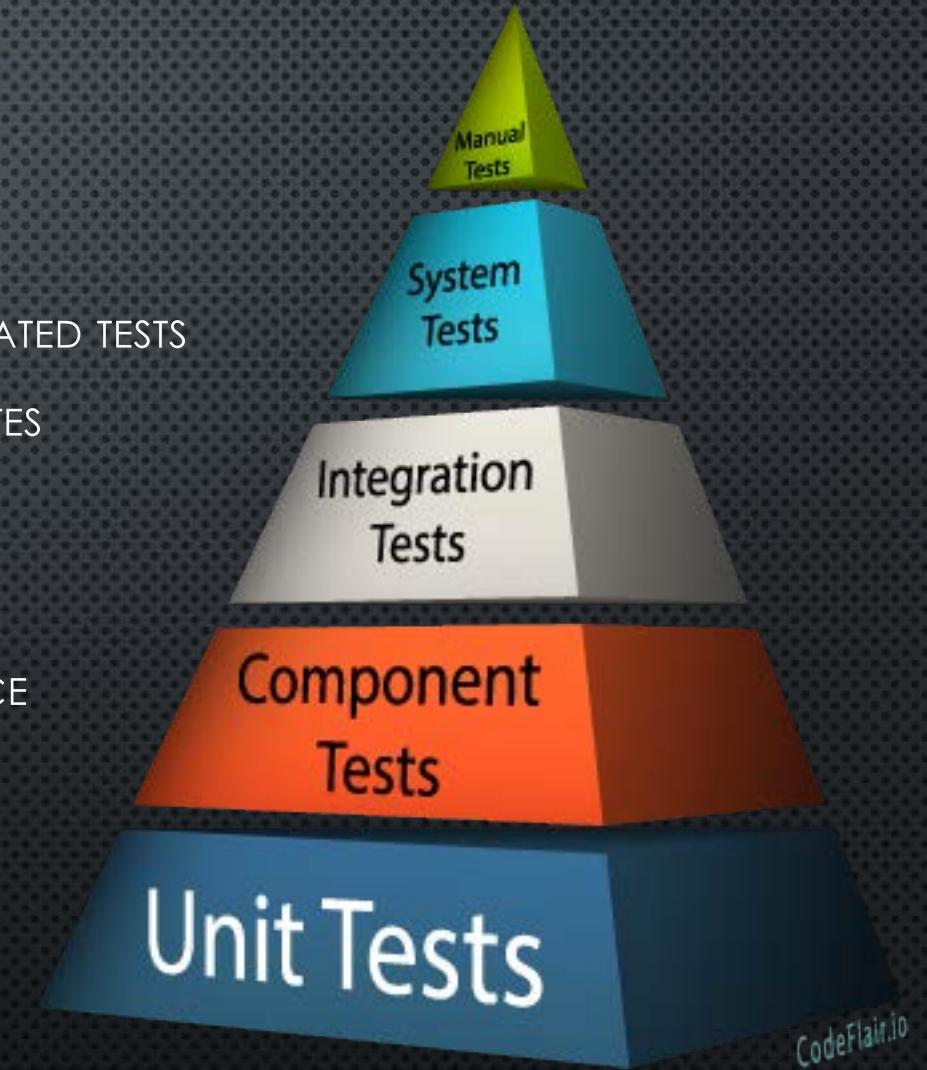


UNDERSTANDING UNIT TESTS

VALIDATING SMALL UNITS OF FUNCTIONALITY

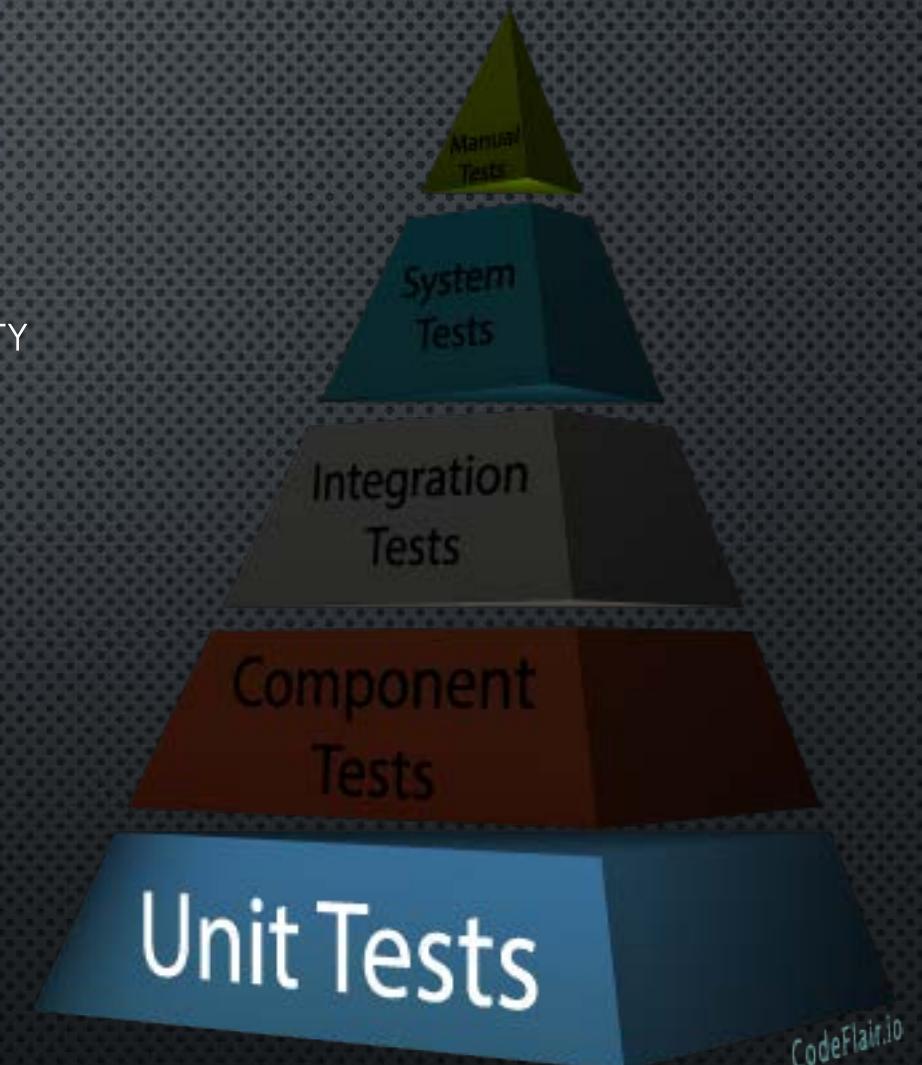
BUILDING CODE COVERAGE

- ✓ TO OPTIMIZE PRODUCTIVITY, MAXIMIZE THE NUMBER OF AUTOMATED TESTS
- ✓ UNIT TESTS SHOULD COMPOSE THE MAJORITY OF YOUR TEST SUITES
- ✓ NOT ALL LEVELS HAVE THE SAME PROBABILITY OF A BUG
- ✓ TESTABILITY IS MORE IMPORTANT THE FARTHER DOWN YOU GO
- ✓ LOWER TESTS SHOULD BE DECOUPLED FROM THE USER INTERFACE



UNIT TESTS

- ✓ VALIDATE SMALL, LOW-LEVEL, ISOLATED UNITS OF FUNCTIONALITY
- ✓ SIMPLE: CYCLOMATIC COMPLEXITY = 1
- ✓ EASY TO WRITE, EASY TO RUN, EASY TO AUTOMATE
- ✓ CAN SIMULATE ALL ERROR CONDITIONS
- ✓ EASY TO REPRODUCE FAILURES – NO DEBUGGER NEEDED
- ✓ EXECUTE IN MILLISECONDS
- ✓ DEVELOPERS CAN RUN THESE AFTER EACH FILE MODIFICATION
- ✓ DOCUMENT THE SYSTEM
- ✓ TARGET 100% CODE COVERAGE





TEST DRIVEN DEVELOPMENT

ACCELERATING SOFTWARE DELIVERY

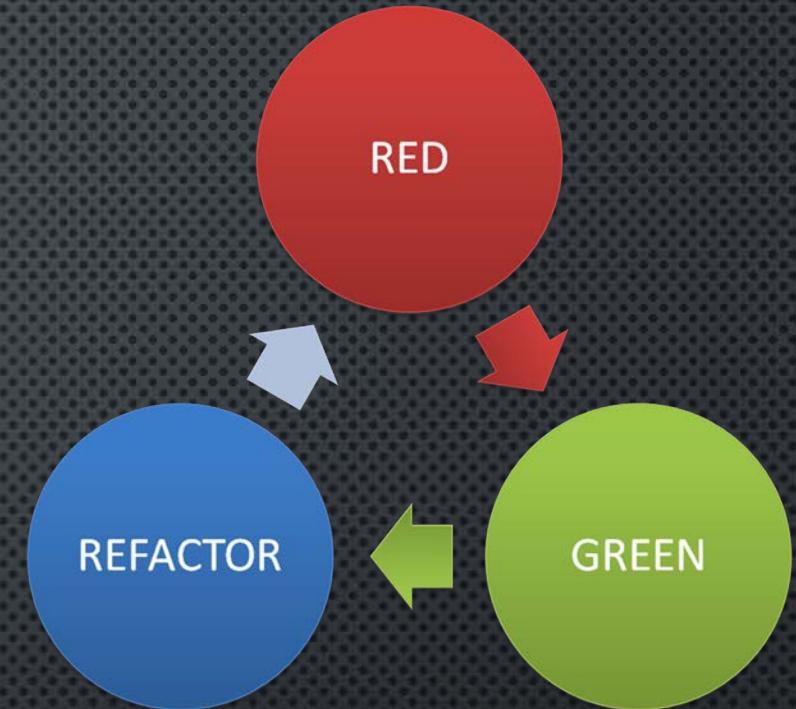
TEST-DRIVEN DEVELOPMENT

- ✓ EXTREME PROGRAMMING
- ✓ IT'S NOT DEVELOPMENT-DRIVEN TESTING
- ✓ ...IT'S MORE ABOUT DESIGN
- ✓ ARCHITECTURE AND IMPLEMENTATION EVOLVE FROM TESTS THAT ARE WRITTEN PRIOR TO WRITING THE CODE
- ✓ SHORT DEVELOPMENT CYCLE
- ✓ VALIDATE THE TESTS THRU FAILURE

Going well beyond quality control
TDD results in a high quality design that is easily maintainable, extensible, and yes, **testable**.



RED → GREEN → REFACTOR



1. MAKE IT FAIL – WRITE A FAILING TEST
2. MAKE IT WORK – WRITE THE SIMPLEST CODE TO PASS THE TEST
3. MAKE IT BETTER – CLEAN, IMPROVE, AND OPTIMIZE THE CODE JUST WRITTEN

THE THREE LAWS OF TDD

1. WRITE NO PRODUCTION CODE EXCEPT TO PASS A FAILING TEST
2. WRITE ONLY ENOUGH OF A TEST TO DEMONSTRATE A FAILURE
3. WRITE ONLY ENOUGH PRODUCTION CODE TO PASS THE TEST

Discipline of Test-Driven Development

The three laws of Test-Driven Development define the discipline of TDD.





JERRY BRONWENHEIMER PRODUCTION

GONE IN 60 SECONDS

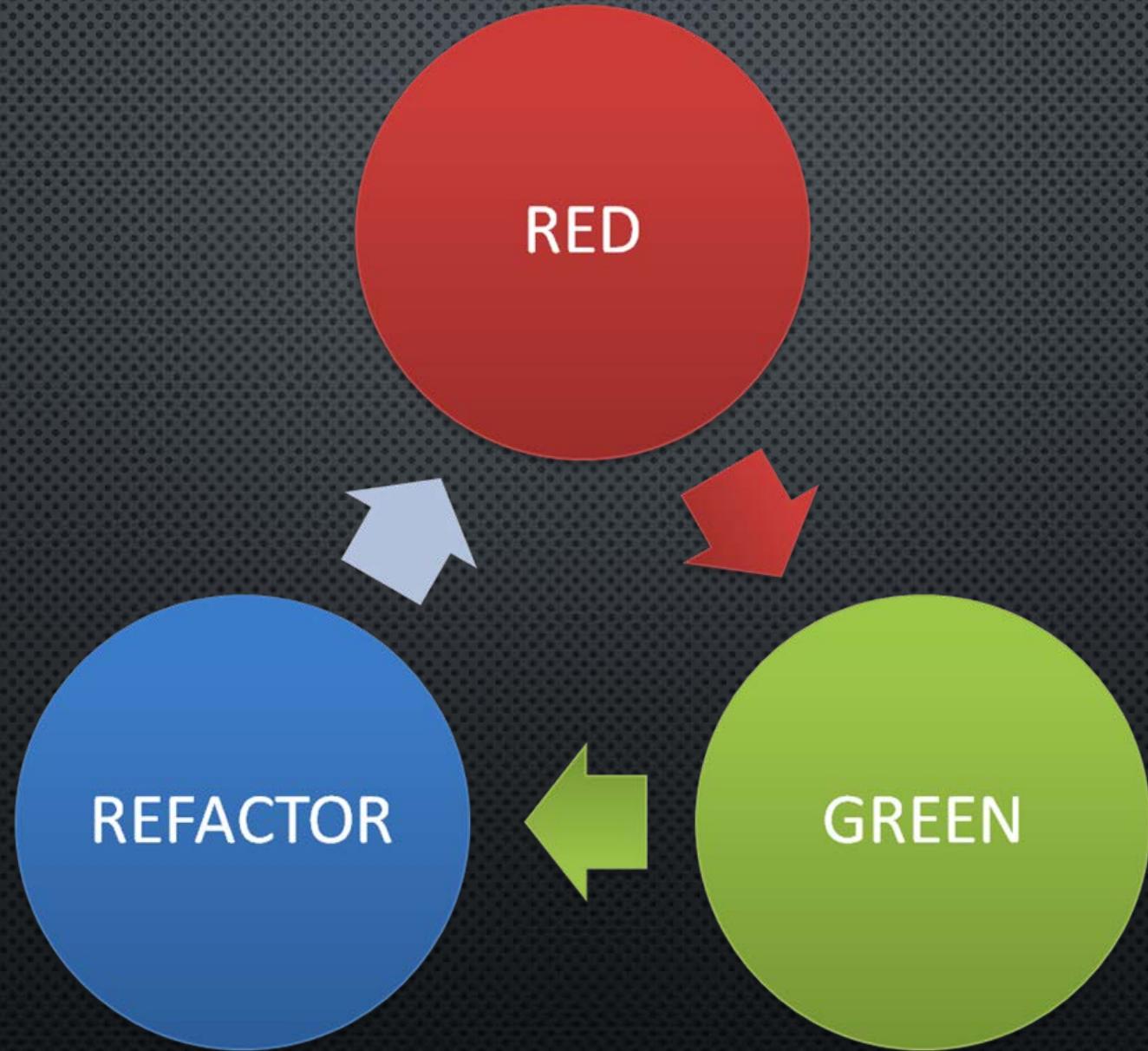
NICOLAS CAGE



ANGELINA JOLIE

GIOVANNI RIBISI

ROBERT DUVALL



THE THREE LAWS OF TDD

1. WRITE NO PRODUCTION CODE EXCEPT TO PASS A FAILING TEST
2. WRITE ONLY ENOUGH OF A TEST TO DEMONSTRATE A FAILURE
3. WRITE ONLY ENOUGH PRODUCTION CODE TO PASS THE TEST

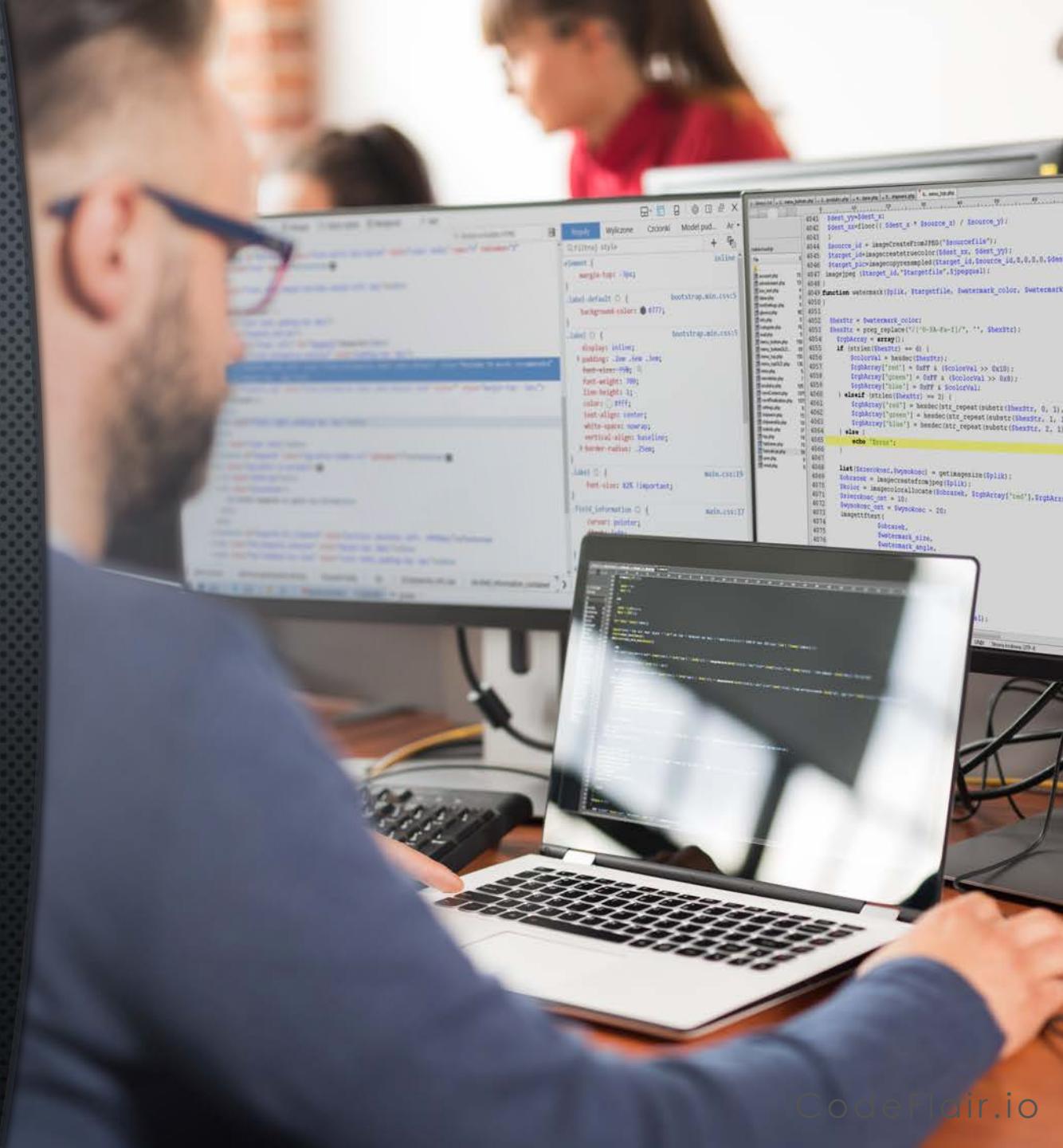


ATDD ≠ TDD

TWO SEPARATE & DISTINCT PRACTICES

TEST DRIVEN DEVELOPMENT

- ✓ VALIDATES PROPER IMPLEMENTATION
- ✓ OFTEN PERFORMED BY A SINGLE DEVELOPER
- ✓ TESTS NEED ONLY BE UNDERSTOOD BY DEVELOPERS
- ✓ SMALL EFFORT
- ✓ SHORT DEVELOPMENT CYCLE



ACCEPTANCE TEST DRIVEN DEVELOPMENT

- ✓ VALIDATES EXPECTED BEHAVIOR
- ✓ REQUIRES TEAM & CUSTOMER COLLABORATION
- ✓ TESTS SHOULD BE UNDERSTOOD BY THE CUSTOMER
- ✓ RELATIVELY LARGE EFFORT
- ✓ LONG DEVELOPMENT CYCLE



TDD & ATDD

SIDE BY SIDE

TEST DRIVEN DEVELOPMENT

- ✓ VALIDATES PROPER IMPLEMENTATION
- ✓ OFTEN PERFORMED BY A SINGLE DEVELOPER
- ✓ TESTS NEED ONLY BE UNDERSTOOD BY DEVELOPERS
- ✓ SMALL EFFORT
- ✓ SHORT DEVELOPMENT CYCLE

ACCEPTANCE TEST DRIVEN DEVELOPMENT

- ✓ VALIDATES EXPECTED BEHAVIOR
- ✓ REQUIRES TEAM & CUSTOMER COLLABORATION
- ✓ TESTS SHOULD BE UNDERSTOOD BY THE CUSTOMER
- ✓ RELATIVELY LARGE EFFORT
- ✓ LONG DEVELOPMENT CYCLE



LONDON VS. CHICAGO

TWO SCHOOLS OF TEST-DRIVEN DEVELOPMENT

LONDON SCHOOL

1. BEHAVIORAL FOCUSED
2. COMMAND-QUERY SEPARATION
3. TEST DOUBLES
4. FRAGILE TESTS
5. DIFFICULT REFACTORING



Behavior Verification

The London School provides a formal, top-down approach to TDD.

CHICAGO SCHOOL

1. STATE FOCUSED
2. STRONG SAFETY NET
3. HIGH COHESION
4. TEST DOUBLES
5. YAGNI



State Verification

The Chicago School provides an informal, inside-out approach to TDD.



TRANSFORMATIONS

TRANSFORMING TO GREEN

TRANSFORMATIONS

- 1. NULL
- 2. NULL TO CONSTANT
- 3. CONSTANT TO VARIABLE
- 4. ADD COMPUTATION
- 5. SPLIT FLOW
- 6. VARIABLE TO ARRAY
- 7. ARRAY TO CONTAINER
- 8. IF TO WHILE
- 9. RECURSE
- 10. ITERATE
- 11. ASSIGN
- 12. ADD CASE

BENEFITS OF TDD

- ✓ GREATER CODE COVERAGE
- ✓ FEWER DEFECTS
- ✓ SHORTER DEBUG TIMES – FASTER TTM
- ✓ BETTER AND MORE RELIABLE LOW-LEVEL DOCUMENTATION
- ✓ CODE THAT IS EASY TO TEST (I.E. TESTABLE)
- ✓ RELIABLE SAFETY NET
- ✓ A SUITE OF TESTS YOU CAN TRUST



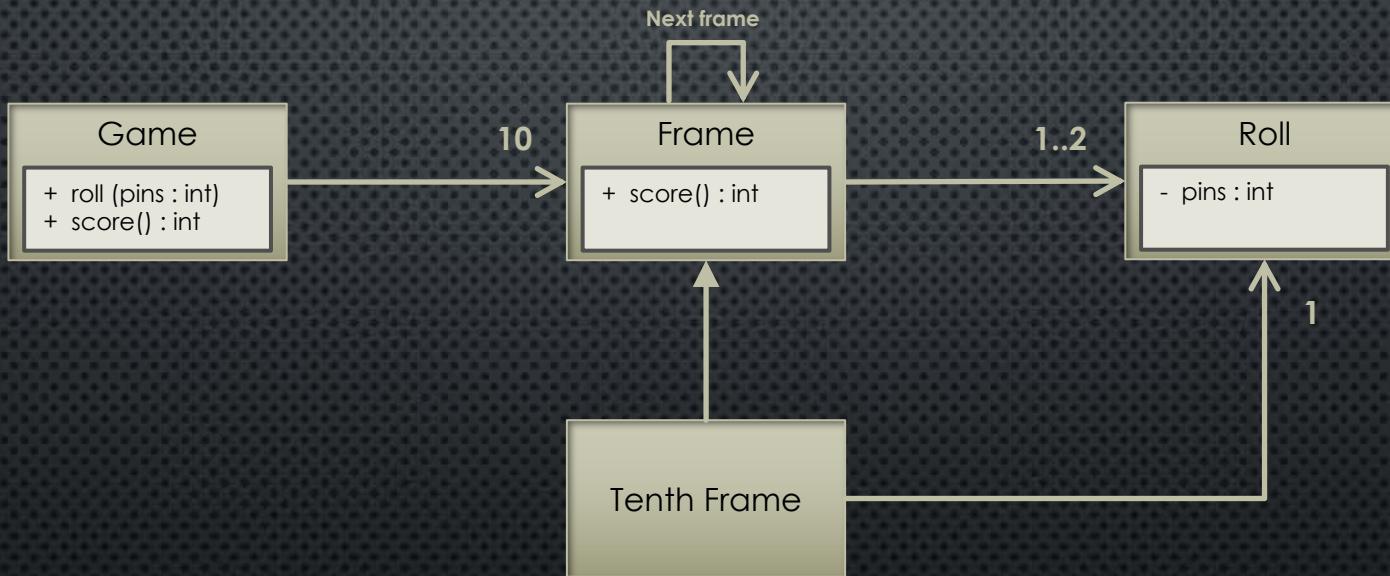
THE BOWLING GAME

BUILDING AN APPLICATION USING TDD

SCORING

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|--|--|-----|-----|-----|
| 1 4 4 5 6 5 5 0 1 7 6 6 6 | 4 5 5 6 6 5 4 1 1 7 7 7 7 | 4 5 5 6 6 5 4 0 1 7 7 7 7 | 5 5 5 6 6 5 4 0 1 7 7 7 7 | 6 6 6 6 6 6 6 6 6 6 6 6 6 | 14 29 49 60 61 77 97 117 133 | 29 49 60 61 77 97 117 133 | 133 | 133 | 133 |
| 5 | 14 | 29 | 49 | 60 | 61 | 77 | 97 | 117 | 133 |

DESIGNING THE BOWLING GAME

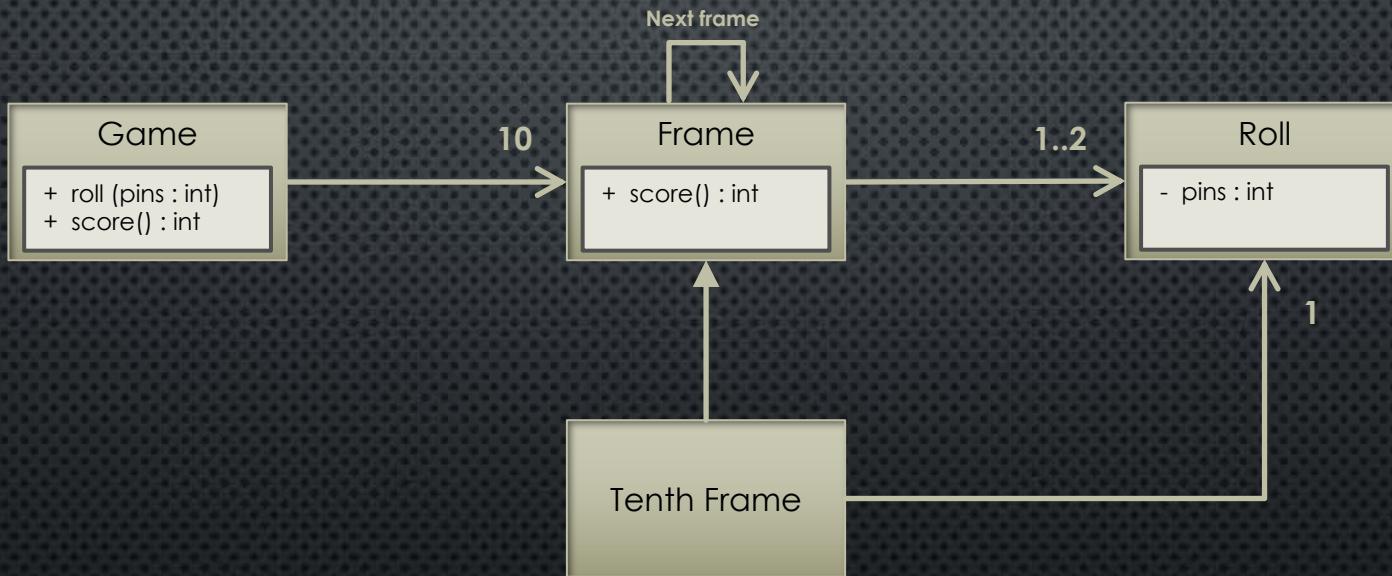




TDD BY EXAMPLE

LET'S START CODING

DESIGNING THE BOWLING GAME





CONCLUSION

REALIZING VALUE FROM TEST-DRIVEN DEVELOPMENT

TEST-DRIVEN DEVELOPMENT

- ✓ CONSTANTLY WORKING IN A CLEAN CODE-BASE
- ✓ LESS TIME DEBUGGING
- ✓ EVOLUTION OF SIMPLE DESIGN
- ✓ EVOLUTION OF TESTABLE SOFTWARE
- ✓ RELIABLE LOW-LEVEL DOCUMENTATION
- ✓ BASING DECISIONS ON SWISS CHEESE
- ✓ ELIMINATING THE FEAR OF REFACTORING

THANK YOU

www.CodeFlair.io/CodeKata/TestDrivenBowling

www.CodeFlair.io/Articles/WhyBotherWithTDD

www.github.com/Klugh/Course-TestDrivenDevelopment



Doug Klugh



doug@CodeFlair.io



@DougKlugh



/Klugh



/Klugh



/Klugh