TEST-DRIVEN DEVELOPMENT

PRINCIPLES & PRACTICES

 Doug Klugh

 doug@CodeFlair.io

 @DougKlugh

 /Klugh

 /Klugh

 /Klugh

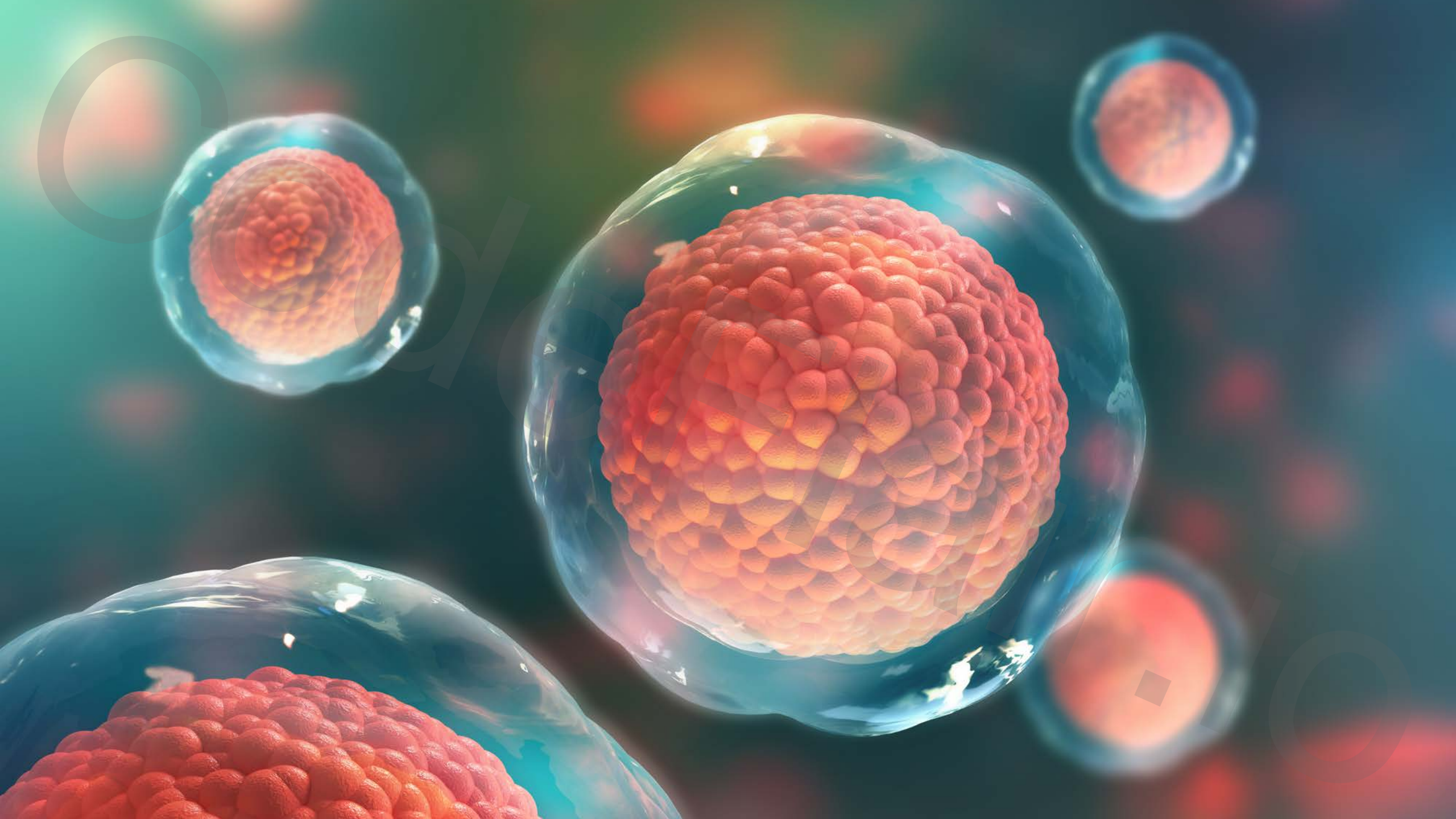
AGENDA

- ✓ Understanding Unit Tests
- ✓ What is Test-Driven Development?
- ✓ TDD by example
- ✓ Conclusion

CodeFlair.io


CodeFlair.io


Code Review



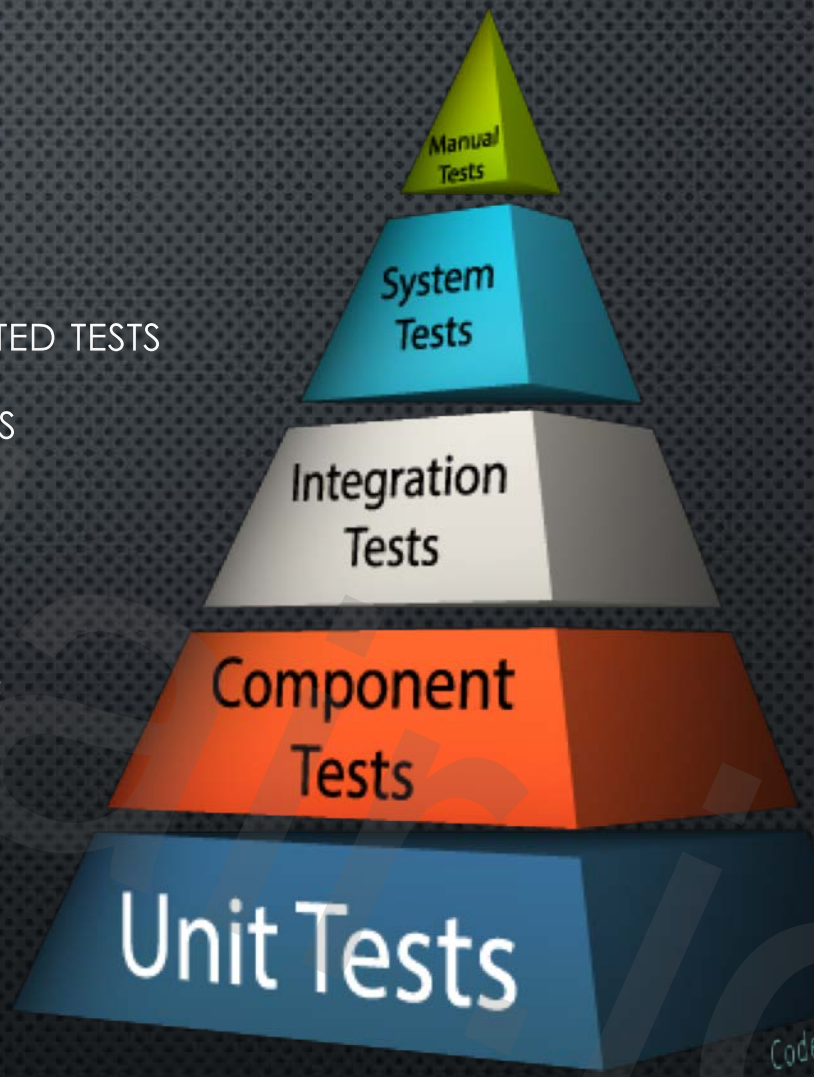
UNDERSTANDING UNIT TESTS

VALIDATING SMALL UNITS OF FUNCTIONALITY

CodeFlair.io

BUILDING CODE COVERAGE

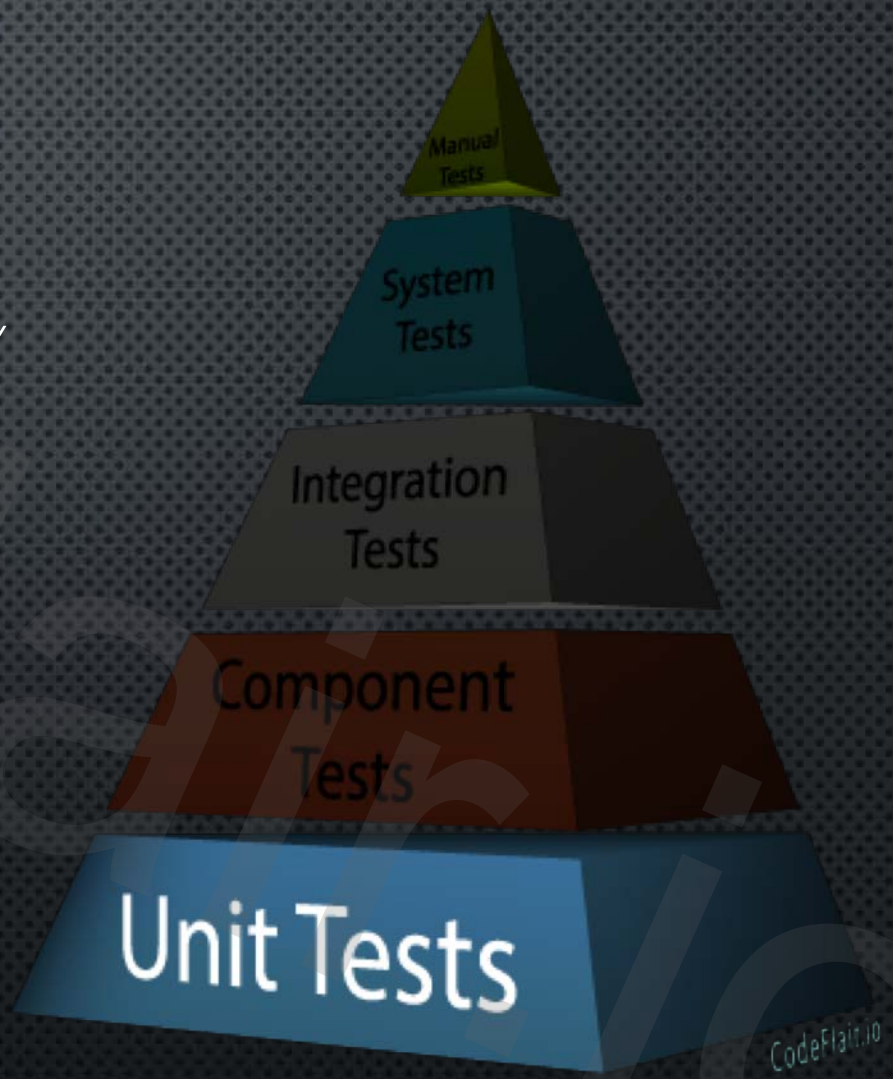
- ✓ To optimize productivity, maximize the number of automated tests
- ✓ Unit tests should compose the majority of your test suites
- ✓ Not all levels have the same probability of a bug
- ✓ Testability is more important the farther down you go
- ✓ Lower tests should be decoupled from the user interface



CodeFlair.io

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



CodeFlair.io

TEST DRIVEN DEVELOPMENT

ACCELERATING SOFTWARE DELIVERY

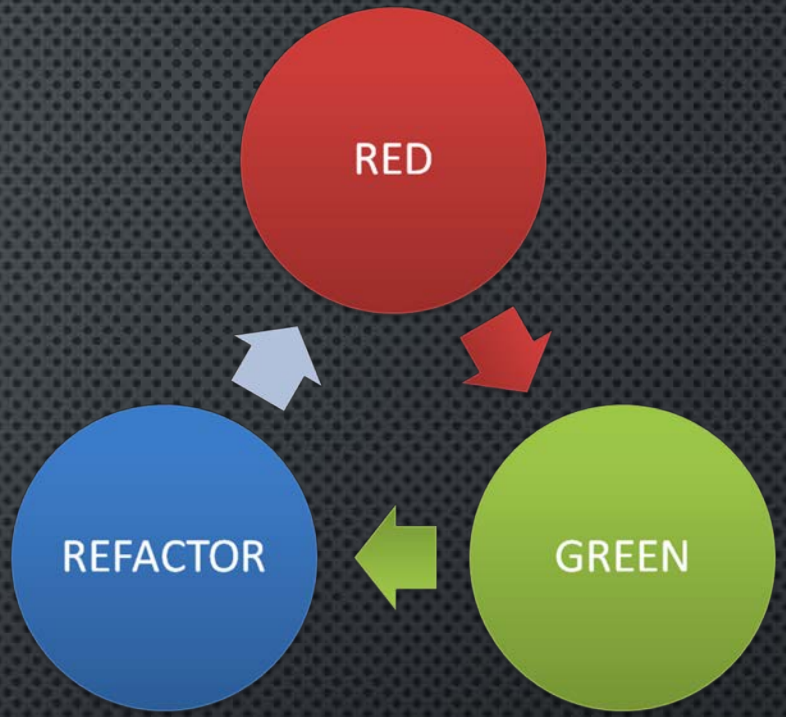
CodeFlair.io

TEST-DRIVEN DEVELOPMENT

- ✓ eXtreme Programming
- ✓ It's not just about 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

CodeFlair.io

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

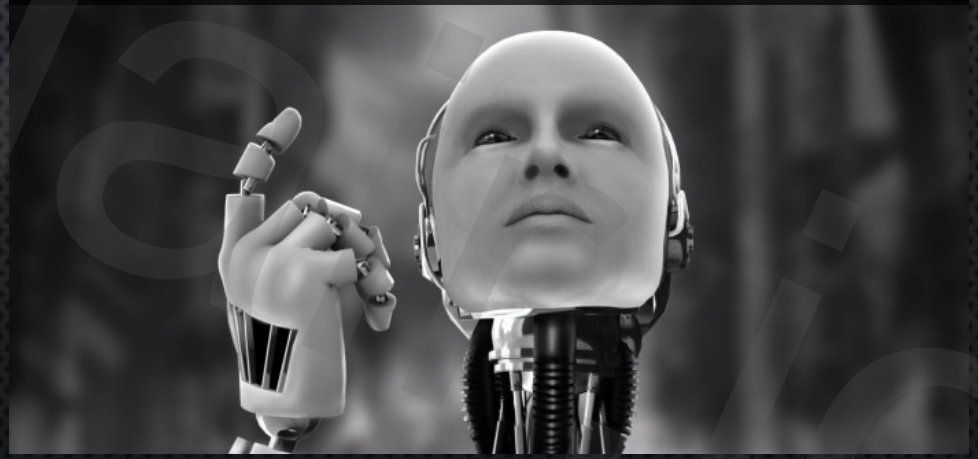
CodeFlair.io

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

Disciplines of Test-Driven Development

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



CodeFlair.io

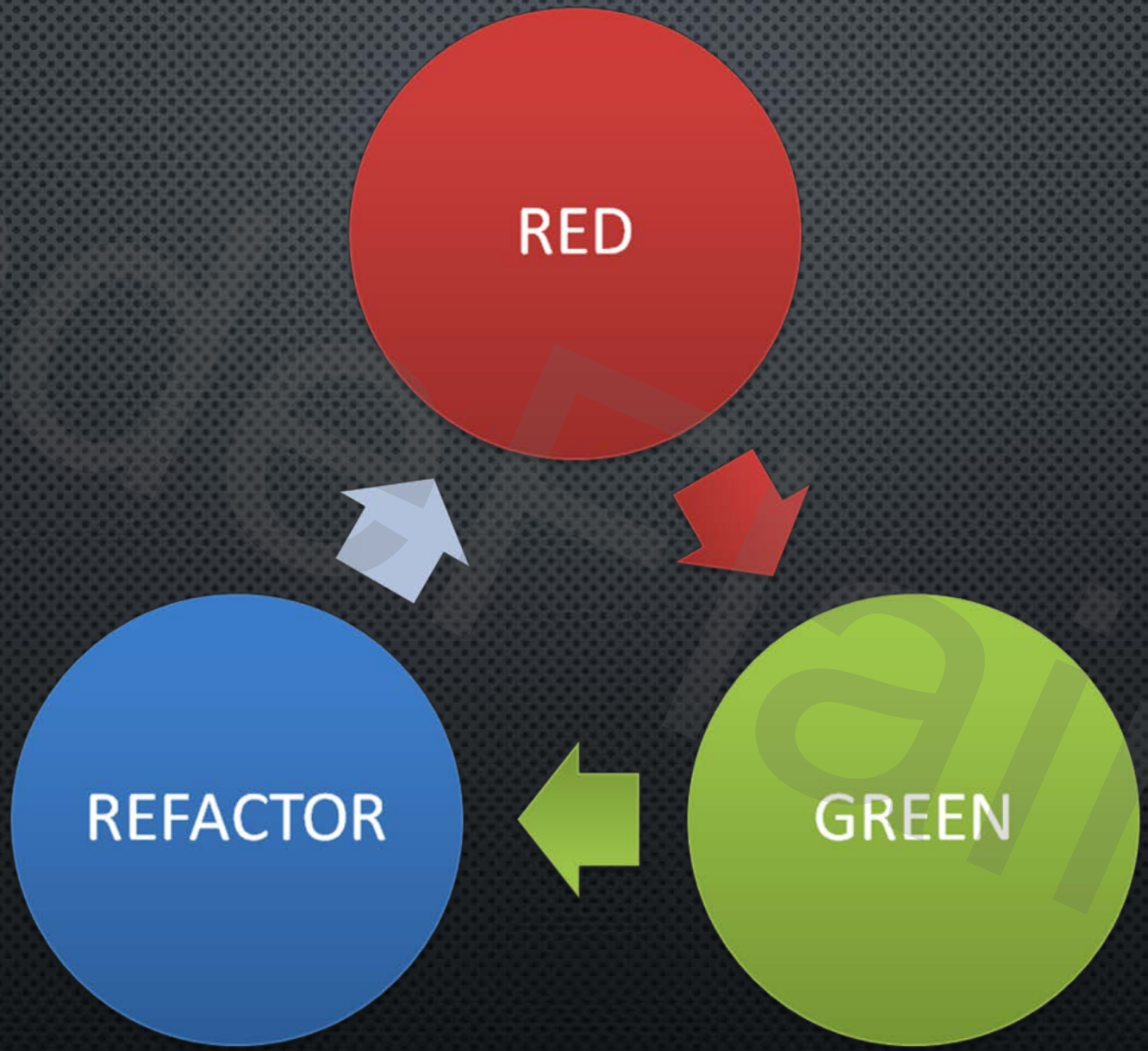
CodeFlair.io

A JERRY BRUCKHEIMER PRODUCTION

GONE IN 60 SECONDS

NICOLAS CAGE

ANGELINA JOLIE GIOVANNI RIBISI ROBERT DUVALL



CodeFlair.io

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)
- ✓ A suite of tests you can trust

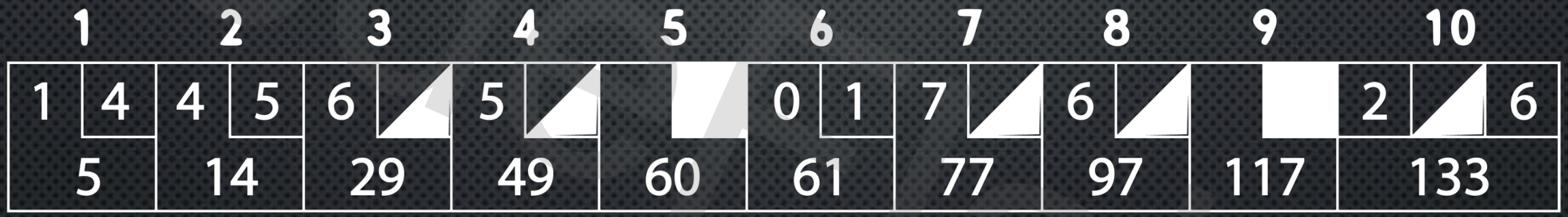
CodeFlair.io

THE BOWLING GAME

BUILDING AN APPLICATION USING TDD

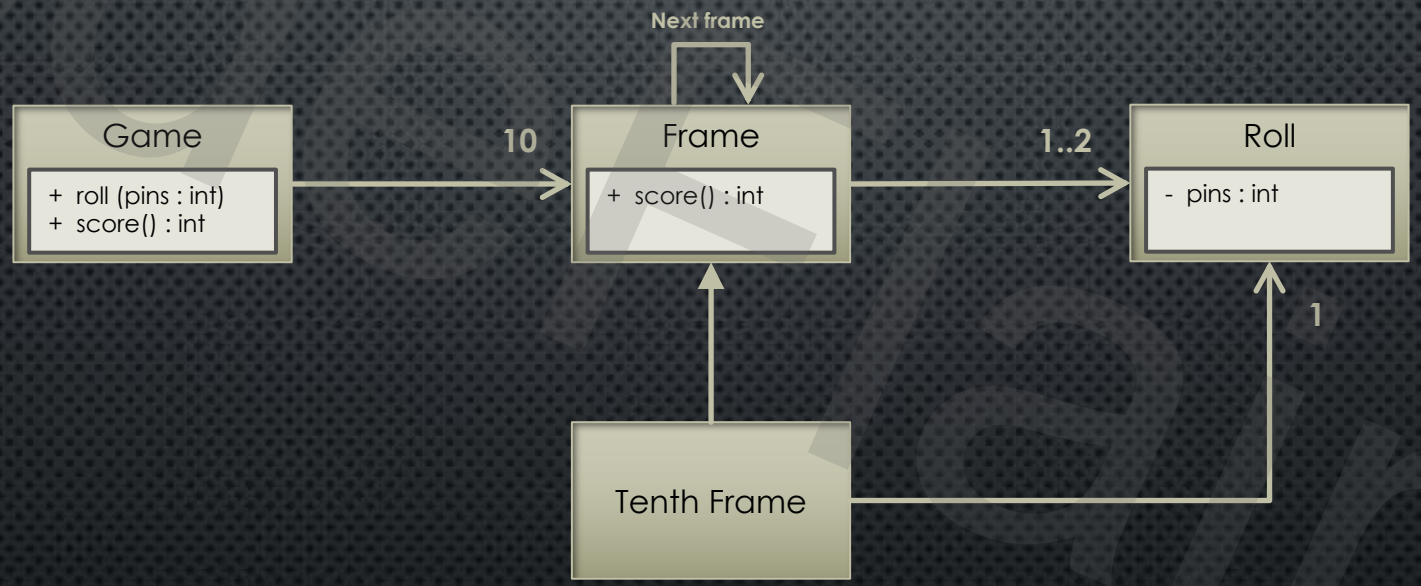
CodeFlair.io

SCORING



CodeFlair.io

DESIGNING THE BOWLING GAME



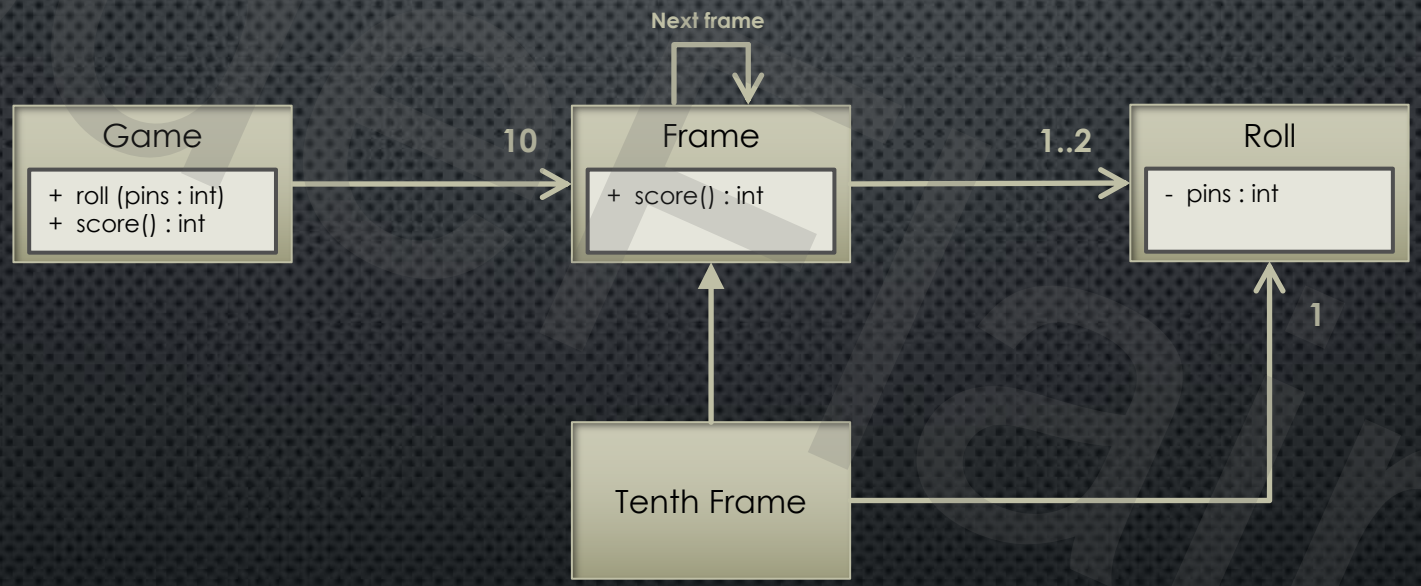
CodeFlair.io

TDD BY EXAMPLE

LET'S START CODING

CodeFlair.io

DESIGNING THE BOWLING GAME



CodeFlair.io

CONCLUSION

REALIZING VALUE FROM TEST-DRIVEN DEVELOPMENT

CodeFlair.io

TEST-DRIVEN DEVELOPMENT

- ✓ Constantly working in a clean code-base
- ✓ Less time debugging
- ✓ Evolution of simple design
- ✓ Evolution of testable software
- ✓ Basing decisions on Swiss Cheese
- ✓ Eliminating the fear of refactoring

CodeFlair.io

THANK YOU

www.CodeFlair.io/why-bother-with-tdd/

www.github.com/Klugh/Course-TestDrivenDevelopment

Doug Klugh

doug@CodeFlair.io

@DougKlugh

/Klugh

/Klugh

/Klugh