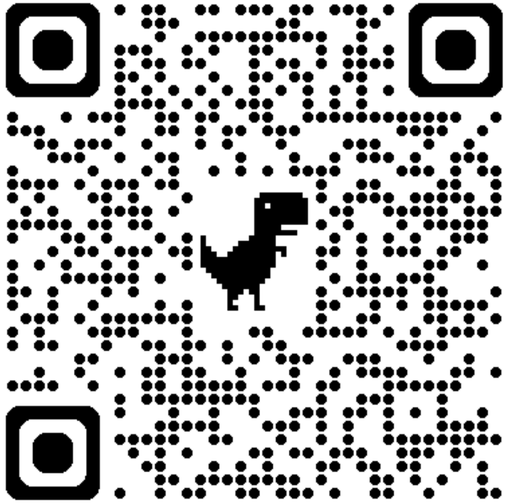

Java & TDD
Daniel Hinojosa
Turning up the intensity of your test process!

About Me

- Developer, Trainer, Coach
- evolutionnext.com
- Java, Scala, Ops, ML, MLOps, Kafka, Event Driven Architecture, Data, IntelliJ
- Available for Consulting, Training and Coaching

In this Workshop

- Test Driven Development
- Process of TDD
- Role of Ai in TDD
- BDD and Ai
- Property Based Testing
- Test Containers
- Test Containers with PBT
- Test Doubles
- Large Applications



Slides and Material: https://github.com/dhinojosa/nfjs-java-tdd

Test Driven Development


The Addison
Test-Driven
Development
By Example

The Process

“

1. Quickly add a test.
2. Run all tests and see the new one fail.
3. Make a little change.
4. Run all tests and see them all succeed.
5. Refactor to remove duplication.

Kent Beck – Test Driven Development By Example 2003

”

“

1. Write a failing test.
2. Write code to make it pass.
3. Repeat steps 1 and 2.
4. Along the way, refactor aggressively.
5. When you can't think of any more tests, you must be done.

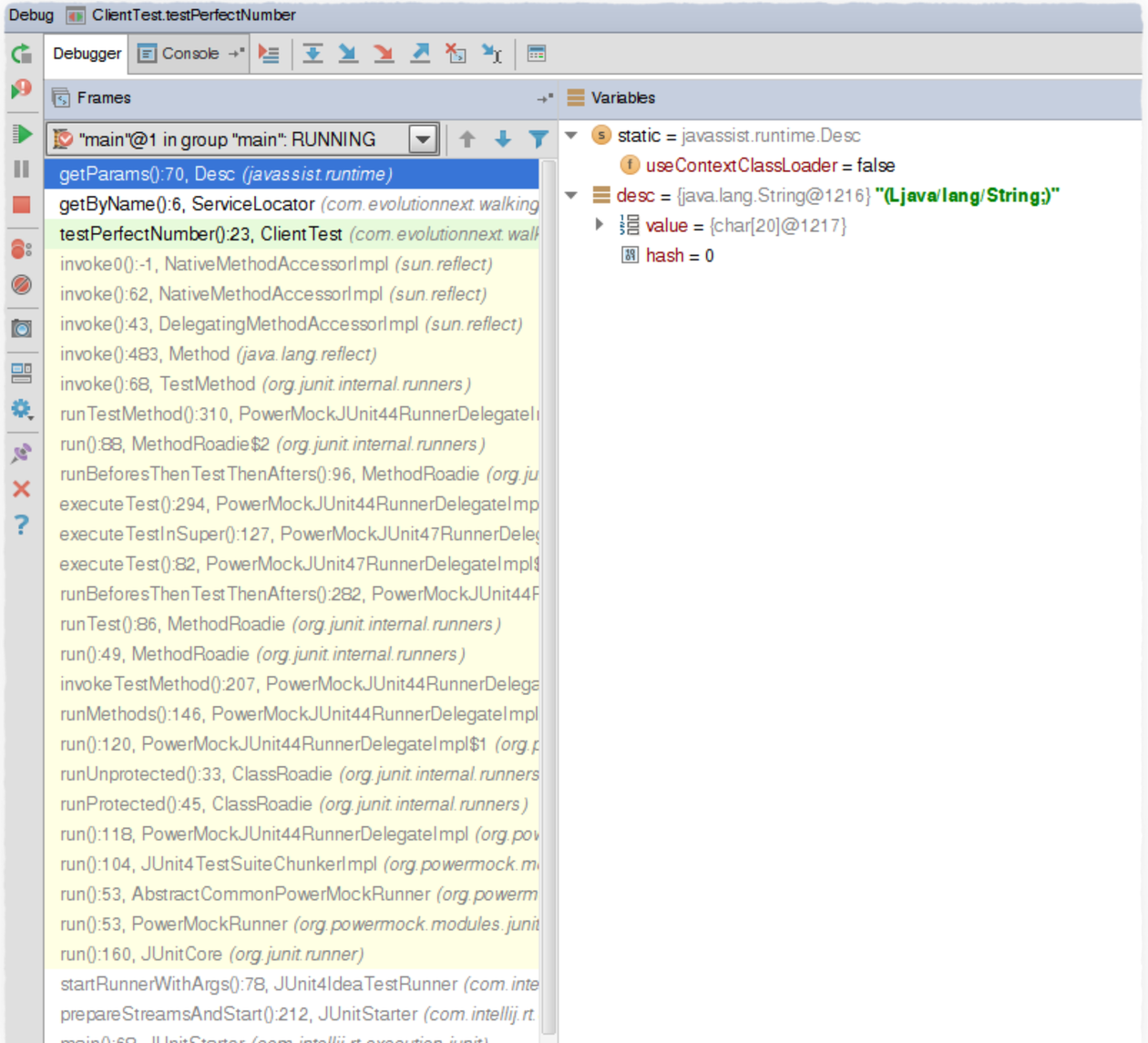
Neal Ford – Evolutionary Architecture and Emergent Design 2009

”

Benefits

- Promotes design decisions up front
- Allows you and your team to understand your code
- Model the API the way you want it to look
- Means of communicating an API before implementation
- Avoids Technical Debt
- Can be used with any programming language

Less or Quality Debugging



Disadvantages

- People find it difficult and unintuitive at first
- Requires team investment
- Many do not see the advantages until it is too late

Three Laws

You may not write production code until you have written a failing unit test.

Bob Martin – Clean Code 2008

You may not write more of a unit test than is sufficient to fail, and not compiling is failing.

Bob Martin – Clean Code 2008

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

Bob Martin – Clean Code 2008

Adoption

Adoption as an Individual

- Practice Makes Perfect
- TDD every new method, class, or function
- Contribute to an open source project for fun
- Use testing when learning a new language!

Adoption as a Team

- For Green Field Projects
  - Start Immediately!
  - Prevent Technical Debt
- For Brown Field Projects
  - Adopt "Testing Thursdays or Fridays" if affordable
  - Powermock in Java if necessary*

* Powermock in Java, or any class manipulation utility is usually a sign of bad design

Measuring

Adoption as an Individual

- Use Code Coverage to check that code is being tested
  - Cobertura
  - Emma
  - **Jacoco**
- Employ Pair Programming
- Employee Code Review
- Non-TDD Code is easy to spot

What Not To Test?

What Not To Test?

- The main method, this is where wiring takes place
- There are varying testing frameworks to test GUIs
  - Selenium Web Driver (Web, JavaScript)
  - Fest-Swing (Java Swing Applications)
  - TestFX (Java FX Applications)
- What about getters and setters?
  - We should be moving to records
  - At the very least, using rich methods

Artificial Intelligence (Ai)

Artificial Intelligence

- AI helps in test case generation and code completion
- Can analyze code patterns to suggest test scenarios
- Helps identify edge cases that human developers might miss
- Provides intelligent code refactoring suggestions
- Assists in maintaining test documentation
- Can predict potential bugs based on historical data
- Helps optimize test suite performance

Technologies

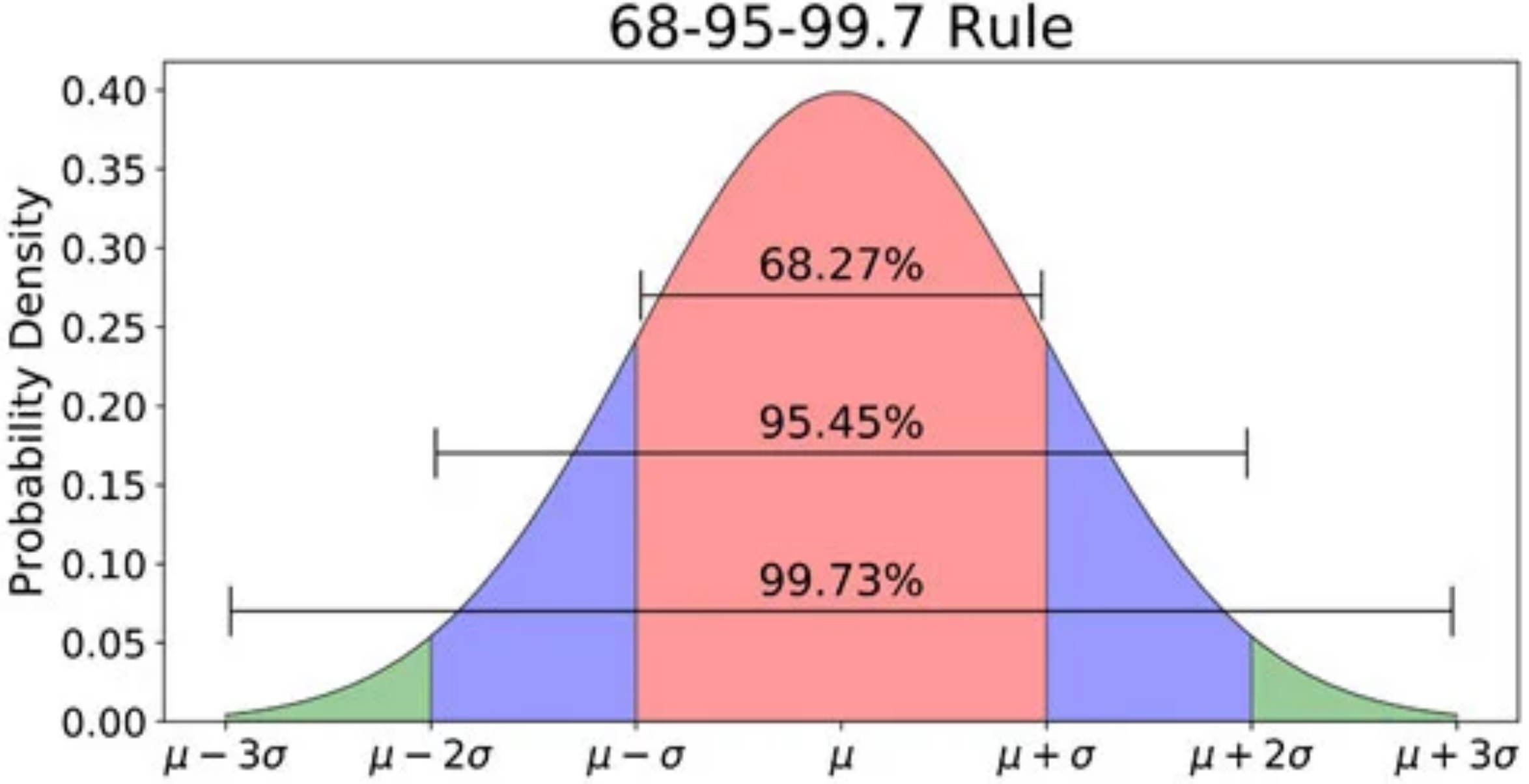
- ChatGPT - OpenAI's language model
- Claude - Anthropic's AI assistant
- Deep Seek - Code generation model
- GitHub Copilot - AI pair programmer
- Amazon CodeWhisperer - AWS AI coding companion
- Tabnine - AI code completion tool

IDEs

- All JetBrains Products
- Cursor
- VS Code
- Eclipse with AI plugins
- Visual Studio with IntelliCode
- Neovim with AI extensions

TDD Disadvantages with Ai

- Too overzealous - It wants to solve everything and doesn't give you time to understand
- Inconsistent and wrong like Ai can be
- Kent Beck - An “unpredictable genie” that grants your “wishes” — but oftentimes in unexpected (and illogical) ways!


68-95-99.7 Rule
Probability Density
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
68.27%
95.45%
99.73%
$\mu - 3\sigma$
$\mu - 2\sigma$
$\mu - \sigma$
μ
$\mu + \sigma$
$\mu + 2\sigma$
$\mu + 3\sigma$

Guidance and Instructions

Some Ai tools will allow you to post instructions and/or guardrails so that all responses adhere to your rules

https://docs.github.com/en/copilot/customizing-copilot/adding-repository-custom-instructions-for-github-copilot

Markdown

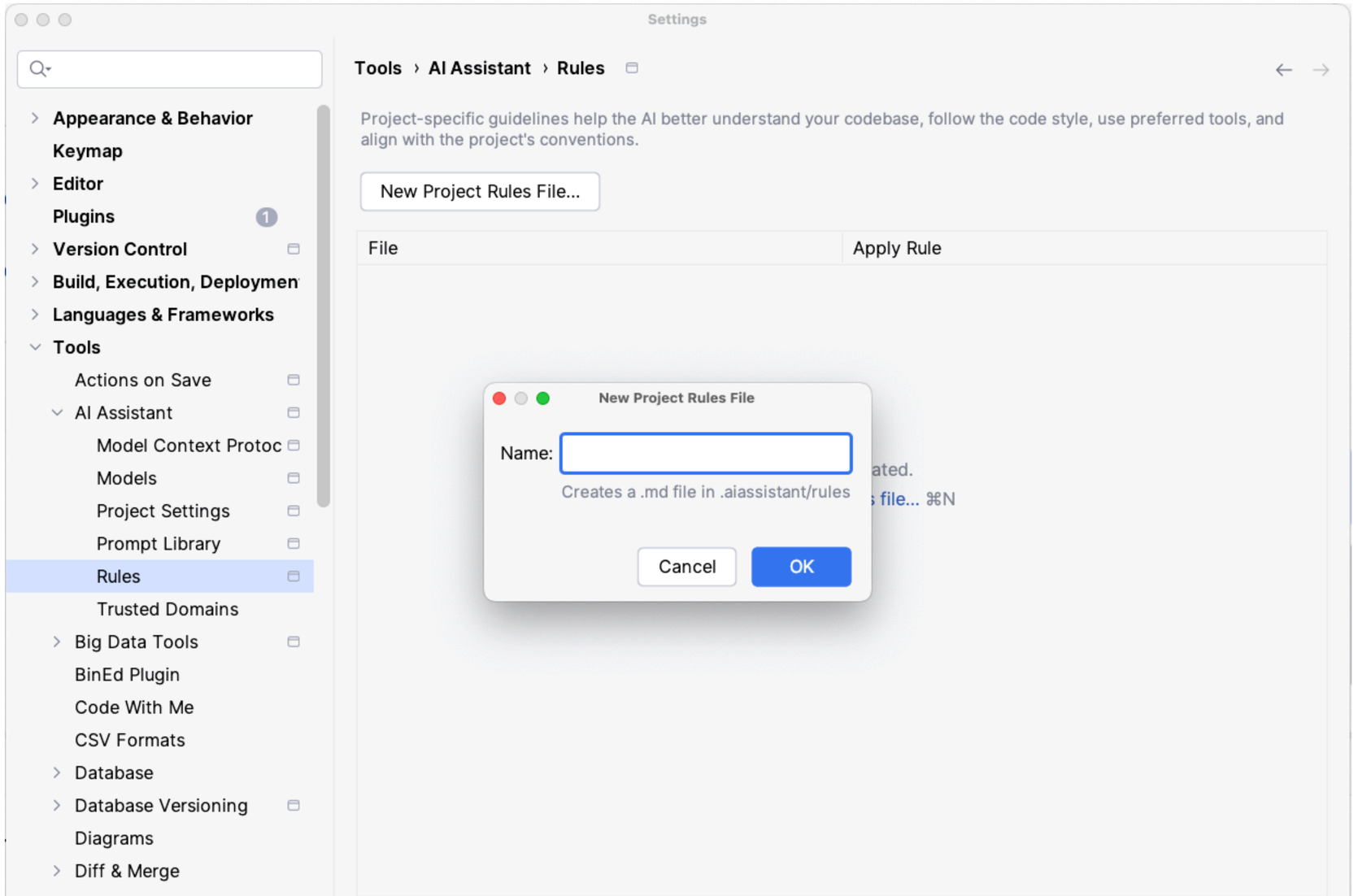
```
Your task is to "onboard" this repository to Copilot coding agent by adding a
.github/copilot-instructions.md file in the repository that contains
information describing how a coding agent seeing it for the first time can work
most efficiently.

You will do this task only one time per repository and doing a good job can
SIGNIFICANTLY improve the quality of the agent's work, so take your time, think
carefully, and search thoroughly before writing the instructions.

<Goals>
- Reduce the likelihood of a coding agent pull request getting rejected by the
user due to
generating code that fails the continuous integration build, fails a validation
pipeline, or
having misbehavior.
- Minimize bash command and build failures.
- Allow the agent to complete its task more quickly by minimizing the need for
exploration using grep, find, str_replace_editor, and code search tools.
```

JetBrains Products

JetBrains has something similar



Cursor

Cursor has cursor rules to adhere to your preferences

https://cursor.com/docs/context/rules

Core

Rules

Rules provide system-level instructions to Agent. They bundle prompts, scripts, and more together, making it easy to manage and share workflows across your team.

Cursor supports four types of rules:

Project Rules

Stored in `.cursor/rules`, version-controlled and scoped to your codebase.

User Rules

Global to your Cursor environment. Used by Agent (Chat).

Team Rules

Team-wide rules managed from the dashboard. Available on Team and

AGENTS.md

Agent instructions in markdown format. Simple alternative to `.cursor/rules`.

Behavior Driven Development


¡THREE AMIGOS!
Developer
Tester
Product Owner

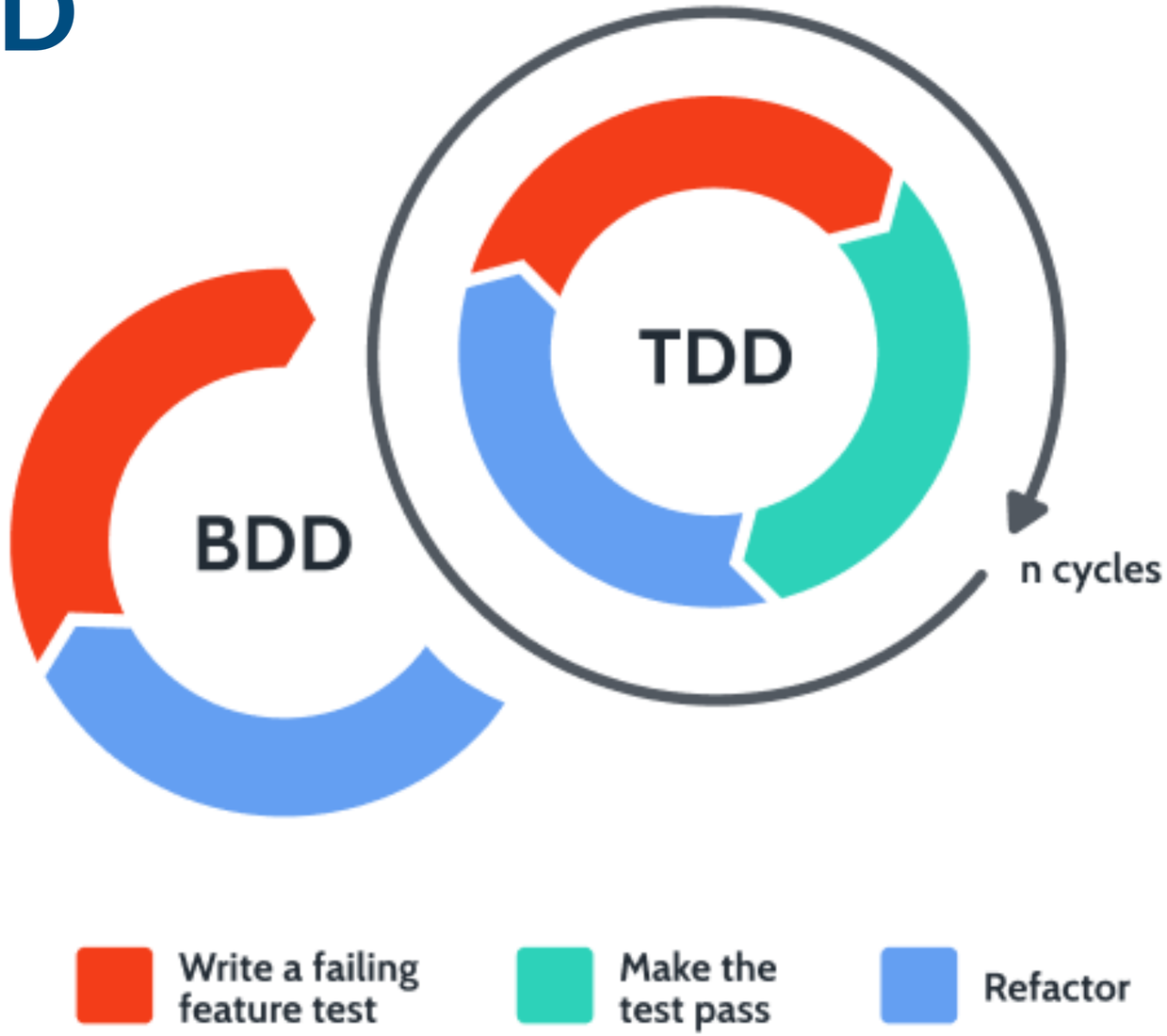
Gherkin and BDD

```
Given the account has a balance of $23.95
When a message of "balance" comes in from a 404-333-2222
Then a text is sent back: "Your balance is $23.95"

Given the account has a balance of $23.95
When any message comes in from an unknown number 233-123-1121
Then log message and phone number, and do nothing

Given the account has a balance of $23.95
When any message other than "balance" comes in from a 404-333-2222
Then send text back: "Didn't understand message, send BALANCE for current balance."
```

BDD & TDD


TDD
BDD
n cycles
Write a failing feature test
Make the test pass
Refactor

Understanding Spec-Driven-Development: Kiro, spec-kit, and Tessl

Birgitta Böckeler

Birgitta is a Distinguished Engineer and AI-assisted delivery expert at Thoughtworks. She has over 20 years of experience as a software developer, architect and technical leader.

I've been trying to understand one of the latest AI coding buzzword: Spec-driven development (SDD). I looked at three of the tools that label themselves as SDD tools and tried to untangle what it means, as of now.

https://martinfowler.com/articles/exploring-gen-ai/sdd-3-tools.html

Notable Resources

AssertJ - fluent assertions java library

Version 1.0

1. AssertJ Overview

AssertJ is composed of several modules:

- A core module to provide assertions for JDK types (String, Iterable, Stream, Path, File, Map, ...)
- A Guava module to provide assertions for Guava types (Multimap, Optional, ...)
- A Joda Time module to provide assertions for Joda Time types (DateTime, LocalDateTime)
- A Neo4J module to provide assertions for Neo4J types (Path, Node, Relationship, ...)
- A DB module to provide assertions for relational database types (Table, Row, Column, ...)
- A Swing module provides a simple and intuitive API for functional testing of Swing user interfaces

https://assertj.github.io/doc/

cyber-dojo.org

the place to practice programming

setup a new practice session

enter a practice session

review a practice session

100% of your donation buys Raspberry Pi computers to help children learn to program

please donate

Hamcrest

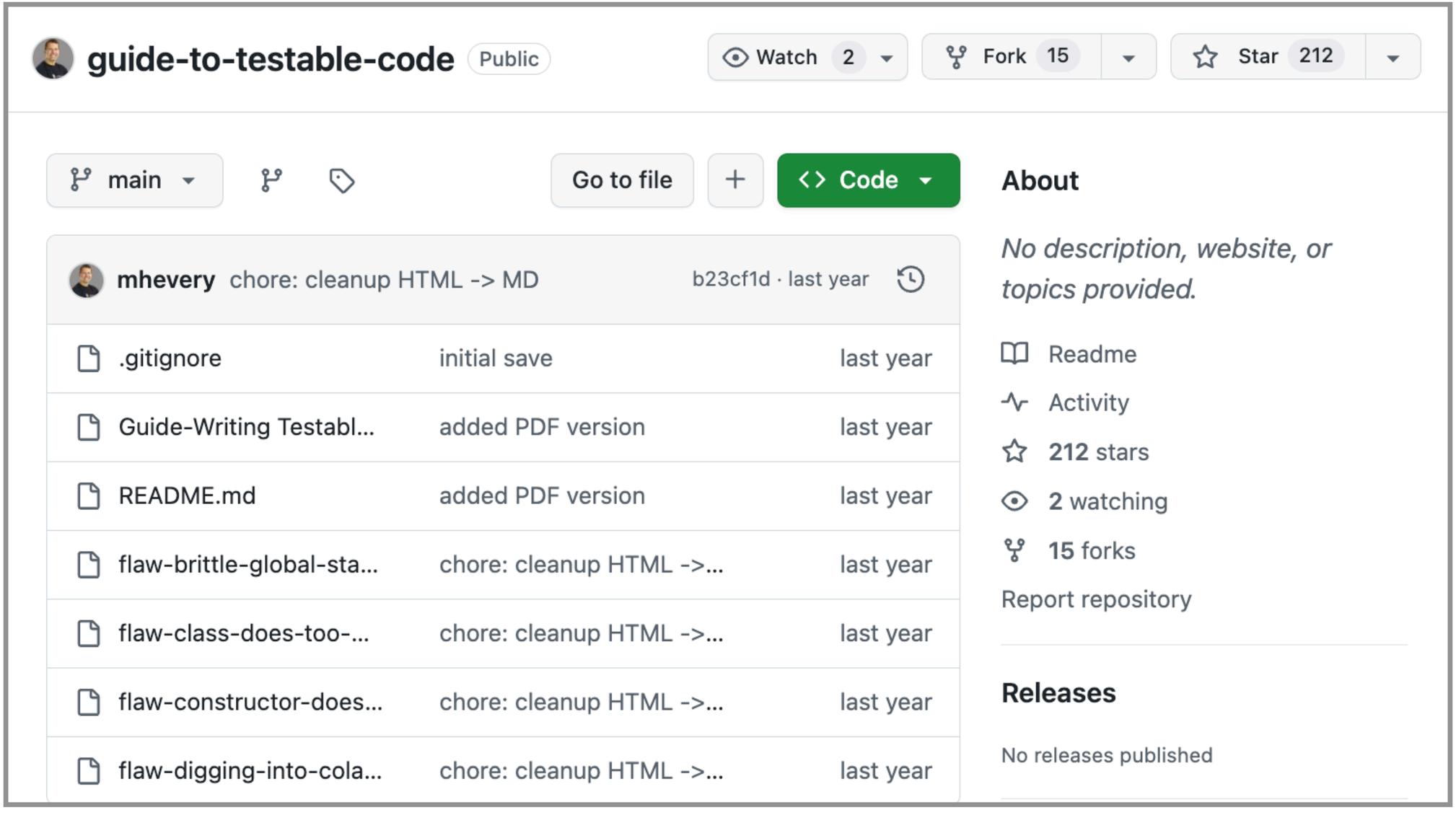
Fork me on GitHub

Matchers that can be combined to create flexible expressions of intent

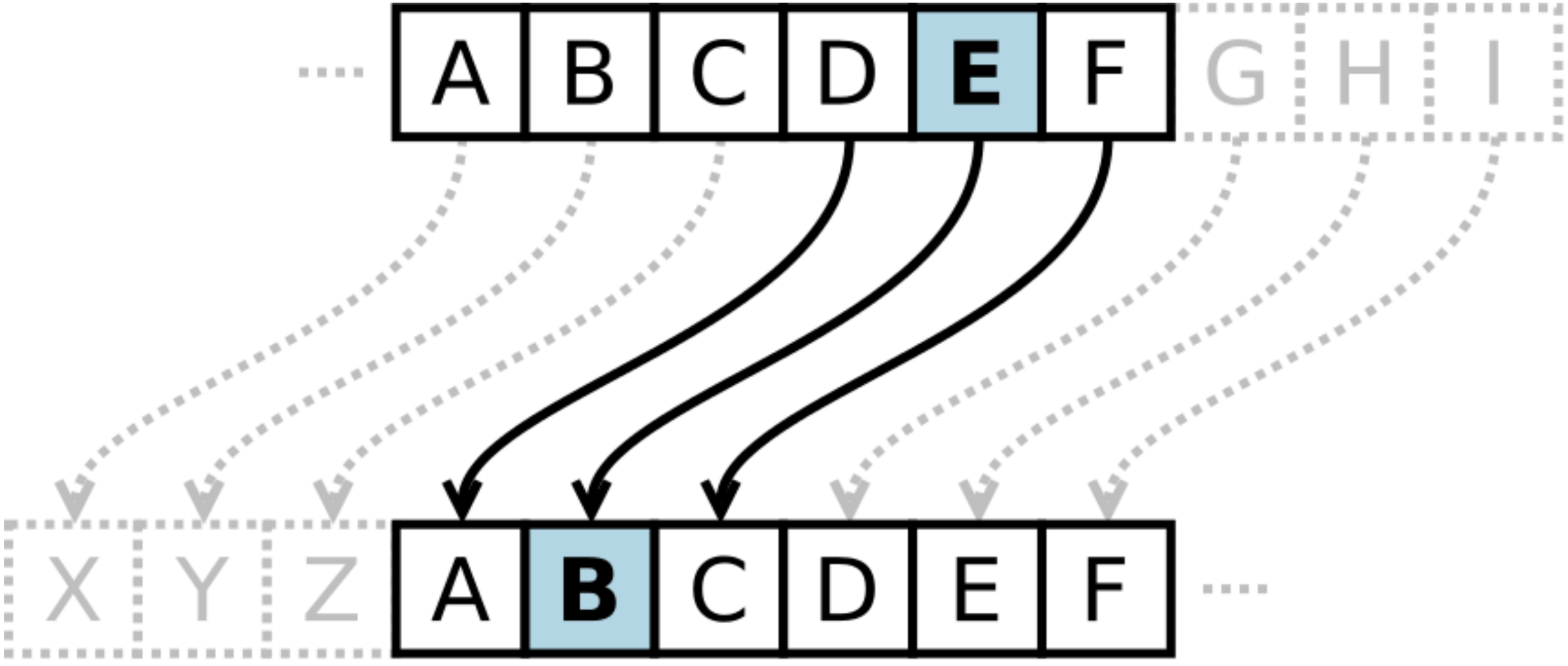
Born in Java, Hamcrest now has implementations in a number of languages.

- Java
- Python
- Ruby
- Objective-C
- PHP
- Common Lisp
- Erlang
- Swift
- Rust
- JavaScript (JsHamcrest)
- JavaScript (Hamjest)
- GO (Gocrest)
- C# (NHamcrest)

https://hamcrest.org/



https://github.com/mhevery/guide-to-testable-code


A B C D E F G H I
X Y Z A B C D E F

Lab: Test Driven Development

- Let’s go through a test Driven Development Example
- We will be using Caesar Shift

Property Based Testing

Challenge: Absolute Value of a Number

Challenge: Reversing a String

Challenge: Concatenating Two Lists

Challenge: Building a Custom Map

Fizz Buzz Test

The "Fizz-Buzz test" is an interview question designed to help filter out the 99.5% of programming job candidates who can't seem to program their way out of a wet paper bag. The text of the programming assignment is as follows:

> *"Write a program that prints the numbers from 1 to 100. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz"."*

Don't believe it,
it's more like 60%

Challenge: How would do the property for FizzBuzz?

Challenge: How would do the property for FizzBuzz?

All divisible by 3 values should start with “Fizz”

All divisible by 5 values should end with “Buzz”

Property Based Testing

- Automatically generates many test cases from a single property.
- Property being: *What should always be true*, not specific inputs/outputs.
- Uncovers edge cases you wouldn't think to write by hand.
- Shrinks failing cases to the minimal example for fast debugging.
- **Encourages modeling behavior instead of scripting examples.**
  - **You shouldn't say "Bob", "Smith", but have the test framework create values!**

Property Testing Smell

- If you find yourself reimplementing the production code in your property you are doing it wrong.
- A property should be a formal definition that holds without reimplementing the code on the outside in the test


JQwik

JQwik

- Property Based Testing for the JVM
- Random, but for the purpose of generating edge cases
- Runs 1000 tests by default with a single property
- Shrinks results if possible to something that you can patch up with a unit test
- Varying range of APIs, called Arbitrary that you can use to create your own data

Shrinking

- Shrinking reduces a failing test case to its simplest form that still fails
- Helps developers quickly understand the root cause of a failure
- Shrinking is automatic in frameworks like jQwik and can be customized
- It turns a mountain of random data into a focused, minimal example

Types Automatically Supported by JQwik

- `int`, `Integer`
- `long`, `Long`
- `double`, `Double`
- `float`, `Float`
- `byte`, `Byte`
- `short`, `Short`
- `boolean`, `Boolean`
- `char`, `Character`
- `enum`
- `String`
- `Character`
- `BigInteger`
- `BigDecimal`
- `LocalDate`
- `LocalTime`
- `LocalDateTime`
- `ZonedDateTime`
- `Duration`
- `Instant`
- `List<T>`
- `Set<T>`
- `Map<K, V>`
- `Optional<T>`
- Arrays (`T[]`, including `int[]`, `String[]`, etc.)

Demo: Basic JQwik

- Let’s take a look at some basic use of JQwik to perform testing
- Let’s take a look at some supporting types out of the box
- We then move our attention to the Lifecycle methods available in JQwik.
- Finally, let’s take a look at Caesar Shift with JQwik and see the benefit of using JQwik

Arbitrary

- `Arbitrary<T>` defines how to generate random values of type `T`
- Used to control both value generation and shrinking behavior
- Supports composing complex types using Combinators
- Can be customized with filters, maps, flatMaps, and constraints
- Built-in support for primitives, collections, enums, and strings
- Allows recursive and edge-case-aware generation with `lazyOf` and `injectNull`

Organizing Arbitraries

- You can put arbitrary definitions anywhere
  - In the test, for local visibility
  - In the test class, still with local visibility
  - As a separate utility of factory classes for reuse across tests

Composition with FlatMap

- Use `flatMap` when generating a value depends on another generated value
- Enables dependent composition, where one `Arbitrary<T>` influences another `Arbitrary<U>`
- For example: generate a `City` first, then create a `Temperature` range based on that city

Parallelizing with Combiners

- `Combinators.combine` is ideal when generated values are *independent*
- Combines multiple `Arbitrary<T>` instances into a single structured type
- Produces tuples or custom objects like `User(firstName, lastName, email)`
- Improves readability and reuse for flat data models
- Use `flatMap` instead when one value depends on another (e.g., `city` → `temperature range`)
- `Combinators.combine` avoids nested lambdas and keeps test data construction declarative

Avoid `.sample()` in Arbitrary

- Eager evaluation: `.sample()` draws a single concrete value during test setup — not during test execution — which means:
  - Every generated `Weather` instance will have the same city.
  - It disables the generator's ability to explore input space over time.
  - Shrinking is broken: jqwik cannot shrink a sampled value, because it's no longer connected to the generation tree.
  - Composition is lost and it breaks the functional pipeline.
  - Test determinism is compromised: sample() will not respect your seed, if applied to the test

Demo: JQwik Arbitraries

- Let’s create some Arbitraries to customize what we want from JQwik.
- We will start with an Student Example, then move onto some interesting problem sets:
  - OneOf, parameterization, recursion, lazyOf, and frequencies

Test Containers

What are TestContainers?

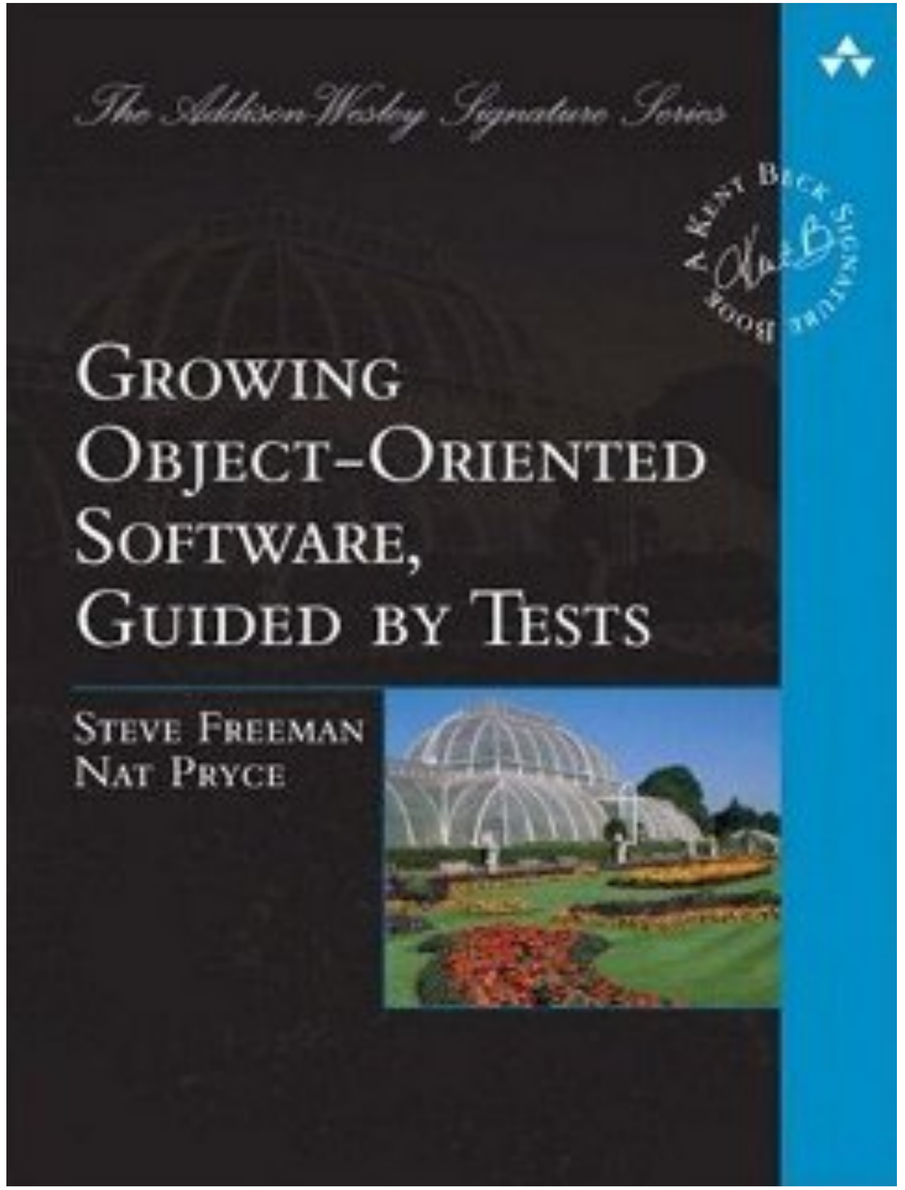
- Library that bootstraps containers (e.g. Docker) for local development and testing
- Benefits!
  - Actual databases, message queues, pub/subs that you use in production
  - No mocking
  - No replacement databases
  - No memory services
  - No XML database frameworks

Advantages

- On-demand isolated infrastructure provisioning: Testcontainers will provide the required services before running your tests.
- Consistent experience on both local and CI environments
  - **Reliable test setup using wait strategies:** Testcontainers modules already implement the relevant wait strategies for a given technology, and you can always implement your own or create a composite strategy if needed.
  - **Advanced networking capabilities:** Testcontainers libraries map the container's ports onto random ports available on the host machine so that your tests connect reliably to those services. You can even create a (Docker) network and connect multiple containers together so that they talk to each other via static Docker network aliases.
  - **Automatic clean up:** The Testcontainers library takes care of removing any created resources (containers, volumes, networks etc.) automatically after the test execution is complete by using the Ryuk sidecar container.

Test Driven Development

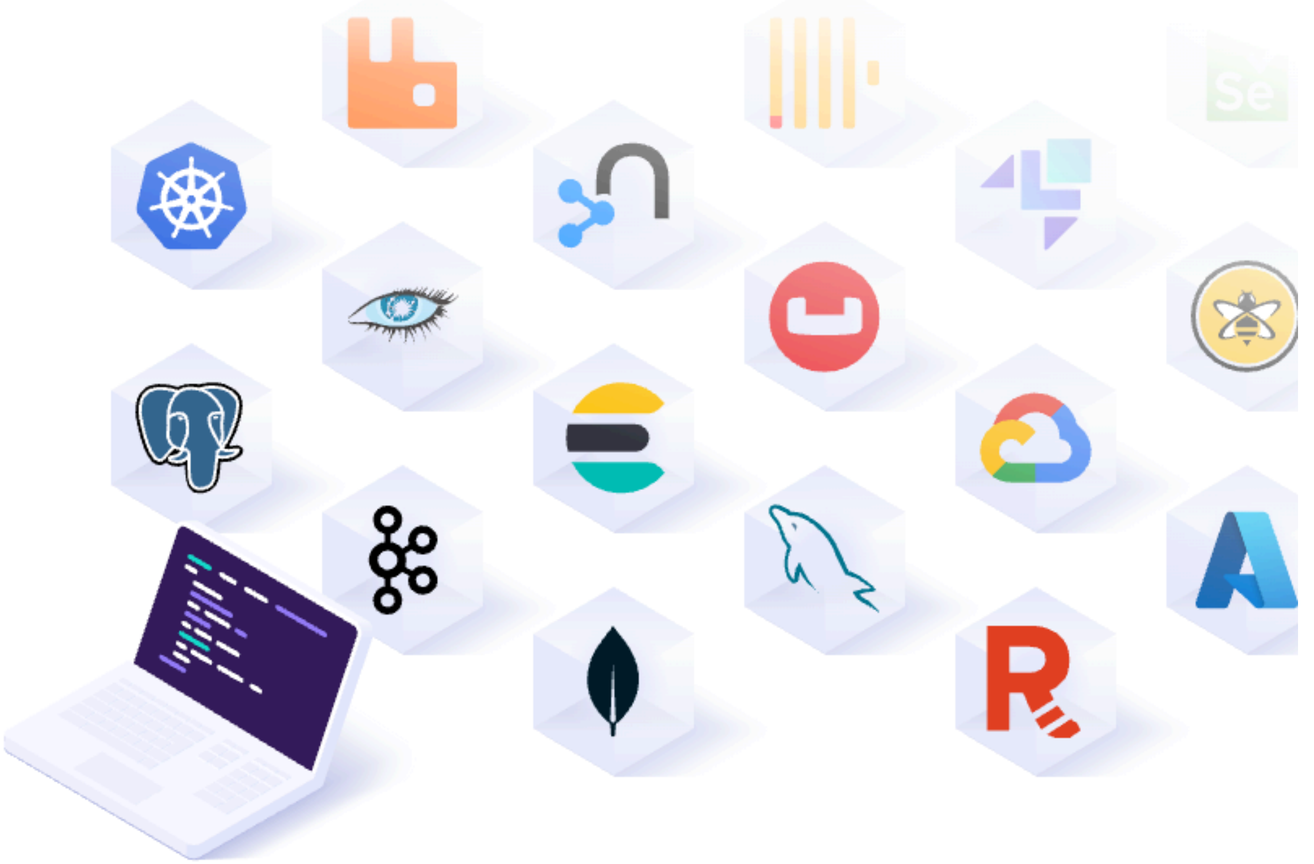
- Don’t write tests for components or behavior that belong to external libraries, frameworks, or systems you don’t control.
- **“We must check that we are using the third-party API correctly, and adjust our abstraction to fit if we find that our assumptions are incorrect.”**
- At a TDD level, a test doesn’t have to be a unit test, it can be an integration test
- In other words, you can perform TDD with actual databases




2 UNIT TESTS, 0 INTEGRATION TESTS
via reddit.com/r/programm

Unit tests with real dependencies

Testcontainers is an open source library for providing throwaway, lightweight instances of databases, message brokers, web browsers, or just about anything that can run in a Docker container.



HOW IT WORKS

Test dependencies as code

Java Go .NET Node.js Python Rust Haskell Ruby Clojure Elixir PHP Native

Demo: Test Containers

- First, let’s look at the modules available for test containers
- Now, let us do a test against a database, and see how that works
- Then let use try out other data storage and messaging systems
  - Kafka
  - Neo4j

Combining JQwik and TestContainers


Reese's
MILK CHOCOLATE
PEANUT BUTTER
CUP

Benefits of Using Both JQwik and TestContainers

- Testing real system with real data stores
- Realistic Infrastructure
- Ensures schema and logic hold under chaos
- Tests real world(ish) payloads, schema, and possibly bugs
- JQwik alone is just aggressively testing data
- TestContainer alone is testing actual data storage systems
- **Together is aggressive testing with data with actual storage systems**

Roundtrip and Identity

- For every inserted record, retrieving it by ID should return the same record.
- For every update, the new value should be visible upon retrieval.
- For every deletion, the record should not be found by ID.
- Insert → Retrieve → Delete → Retrieve = Not Found.

Count and Membership

- If `N` records contain `firstName = "Paul"`, querying by that name should return `N` results.
- For every record inserted into the database, it must exist in the result of a `SELECT *`.
- Count after insert = Count before + `N` of inserts
- Filtered queries should never return more than unfiltered queries.

Symmetry / Reversibility

- Serialization then deserialization yields the same object.
- Insert → Delete → Insert again = Same result.
- Insert → Update → Undo Update → Retrieve = Original state.

Uniqueness & Constraints

- For every inserted email, there should be no duplicates if a `UNIQUE` constraint exists.
- Attempting to insert a duplicate key must raise an exception.
- For every foreign key reference, the referenced entity must exist.

Schema and Behavior

- Fields marked `NOT NULL` must reject null values.
- **String fields should not truncate data (i.e., length must not decrease after roundtrip).**
- If a record is valid by schema, it should be insertable.

State and Accumulative

- If I insert 3 records and delete 1, there should be 2 left.
- If I count all orders for customer A, and then insert another for A, count should increase by 1.

Auto Generative Fields

- If a field is auto-generated (e.g., `createdAt`, `UUID`), it should never be `null` or duplicated.
- If I insert identical records with autogenerated fields, the fields should differ (e.g., `ID` or timestamps).
- Note: These may fail the “Don’t test things you don’t own”

Domain Invariants

- **After running through services test all invariants in the database and in sure data is of the quality that you expect!**
- All inserted records should obey known invariants (e.g., `orderTotal >= 0`)
- For every order line, `lineTotal == quantity * unitPrice`
- For every order, `sum(lineTotal) == orderTotal`

Streaming Data

- For every produced event, a matching event should be consumed.
- For every batch of messages produced by key, order must be preserved per key.

Security

- If `N` accounts are created with the same password, the number of distinct password hashes should also be `N`.
- If `N` accounts are created with unique passwords, then the number of distinct password hashes should also be `N`.
- If a password is hashed and stored during registration, and the same raw password is provided at login, it should match.
- Given a hashed password, a different raw password should never validate.
- If encrypted fields are stored in DB, they should not be readable or reveal length/ pattern of original data

Test Doubles

No evidence of a dependency

```
public void method1() {
      new Employee("Roger", "Moore");
}
```

No evidence of a static dependency

```
public void method2() {
    Resource resource =
        ServerInstance.find("/accounts/resource");
    resource.addDeposit(3000.00);
}
```

Static Dependencies Are Hard to Control

```
public Resource method3() {
    Resource resource =
       ServerInstance.find("/accounts/resource");
    resource.addDeposit(3000.00);
    return resource;
}
```

Evidence of a Dependency

```
public Resource method4(Resource resource) {
    resource.addDeposit(3000.00);
    return resource;
}
```

Evidence of a Dependency

```
public Employee method4() {
  return new Employee("Sean", "Connery");  //OK
}
```

Evidence of a Dependency

```
public List<Employee> method5() {
   return Arrays.asList(
                    new Employee("Sean", "Connery"),
                    new Employee("George", "Lazenby"),
                    new Employee("Pierce", "Brosnan"),
                    new Employee("Roger", "Moore"),
                    new Employee("Timothy", "Dalton"),
                    new Employee("Daniel", "Craig"));
}
```

Fakes

Objects that actually have working implementations, but usually take some shortcut which makes them not suitable for production (an in memory database is a good example).

Martin Fowler - http://martinfowler.com/articles/mocksArentStubs.html

Dummies

“

Objects that are passed around but never actually used. Usually they are just used to fill parameter lists.

”

Martin Fowler - http://martinfowler.com/articles/mocksArentStubs.html

Stubs

> Stubs provide canned answers to calls made during the test, usually not responding at all to anything outside what's programmed in for the test. Stubs may also record information about calls, such as an email gateway stub that remembers the messages it 'sent', or maybe only how many messages it 'sent'.

Martin Fowler - http://martinfowler.com/articles/mocksArentStubs.html

Mocks

Mocks are ... objects pre-programmed with expectations which form a specification of the calls they are expected to receive.

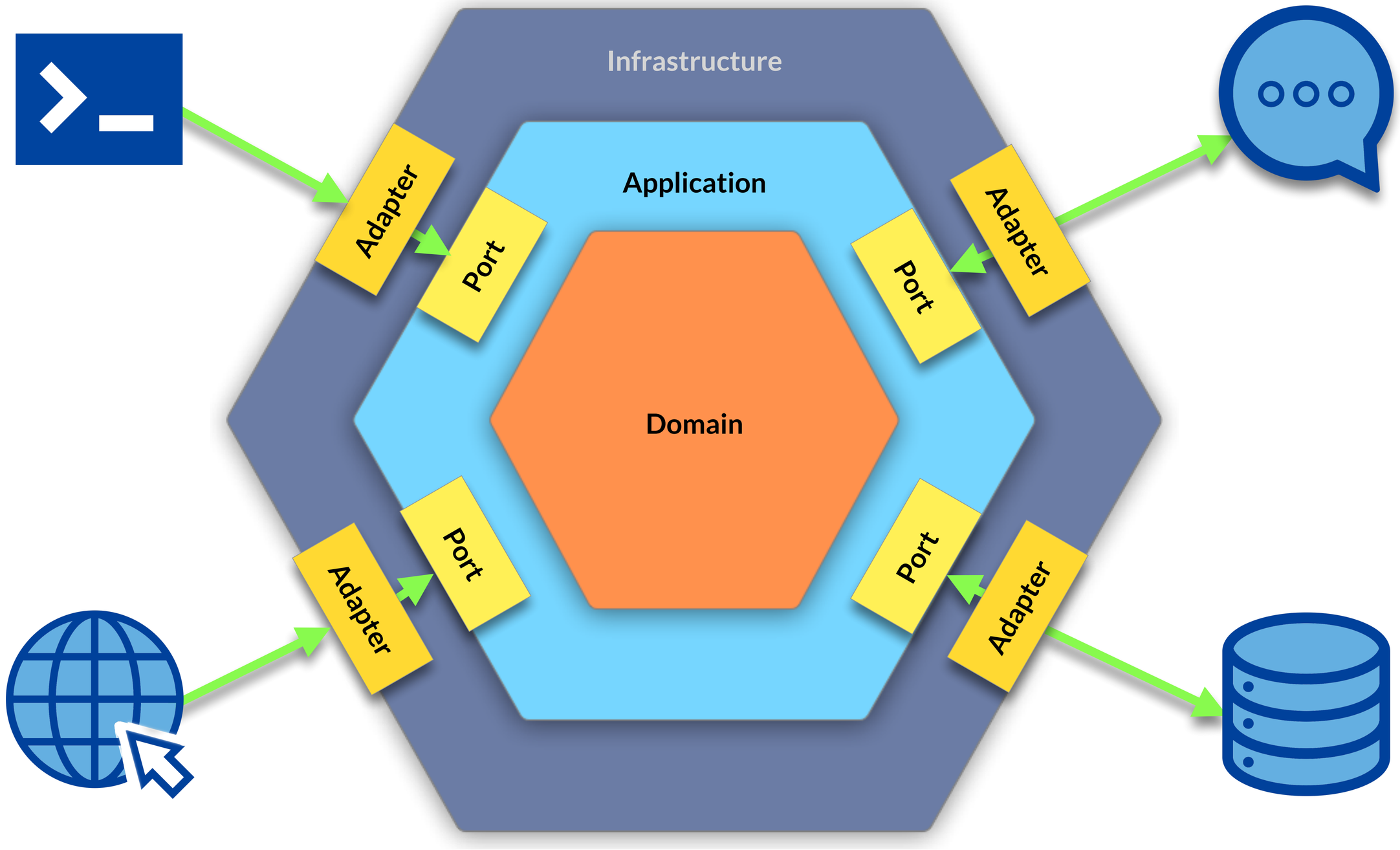
Martin Fowler - http://martinfowler.com/articles/mocksArentStubs.html

Functions

- Lambda Expressions are available since Java 8
- Potential to minimize the use of mocks & stubs, save time

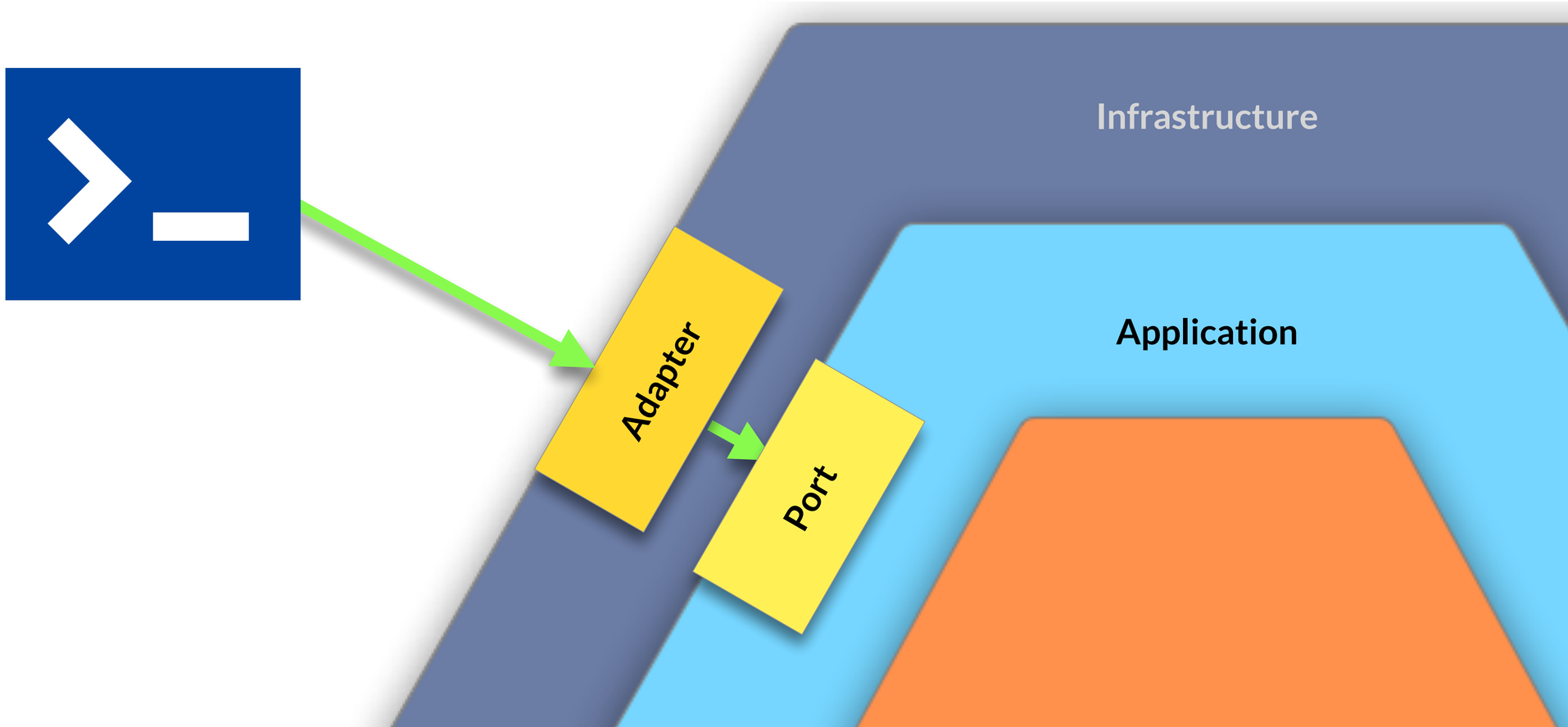
What about Powermock?

Larger Applications


Infrastructure
Application
Domain
Adapter
Port
Adapter
Port
Port
Adapter
Port
Adapter

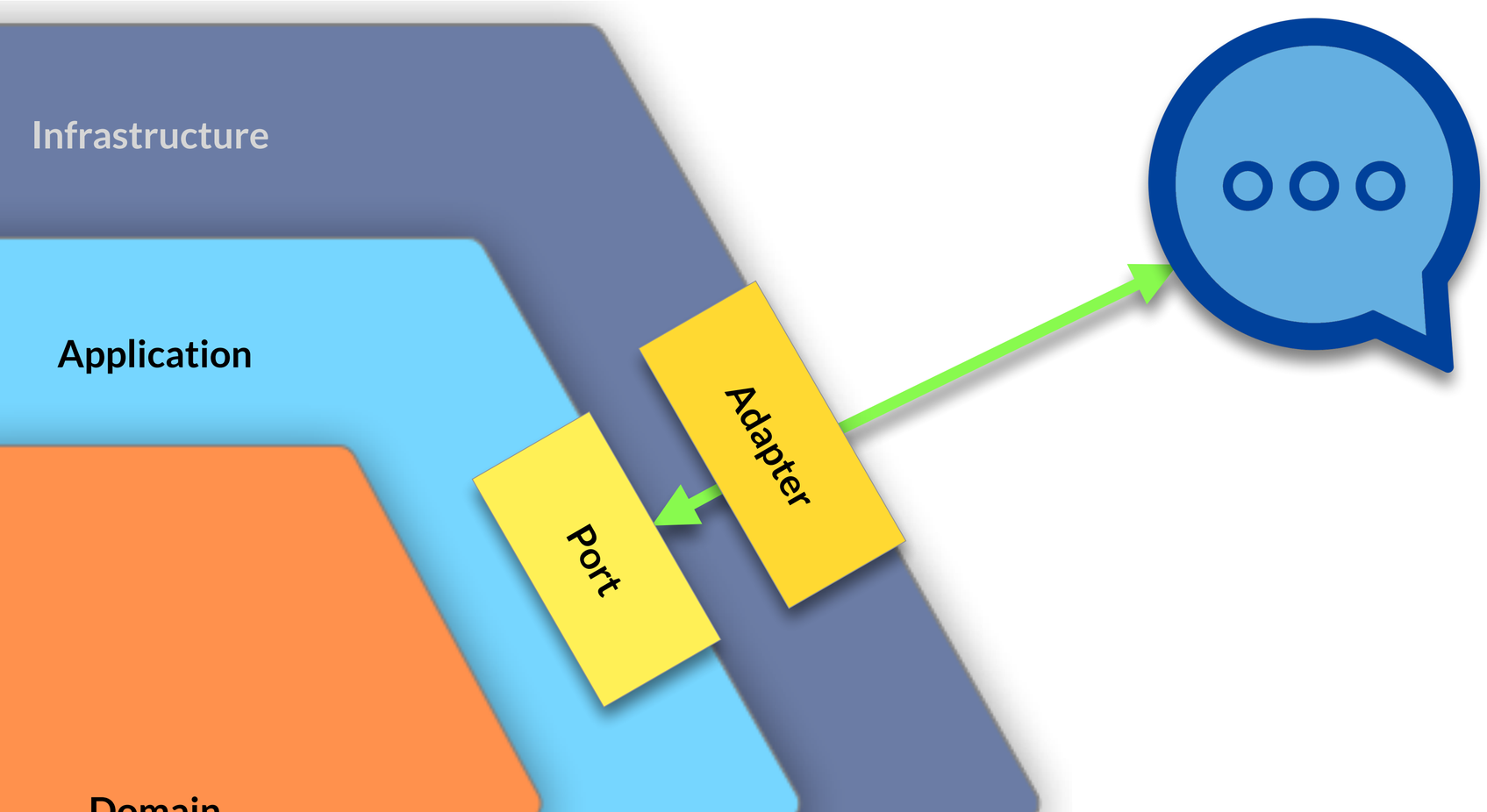
The “In” Adapter (Driving)

- In this case the CLI is the adapter, and it driving the port, which in this case **is in the Application Layer**



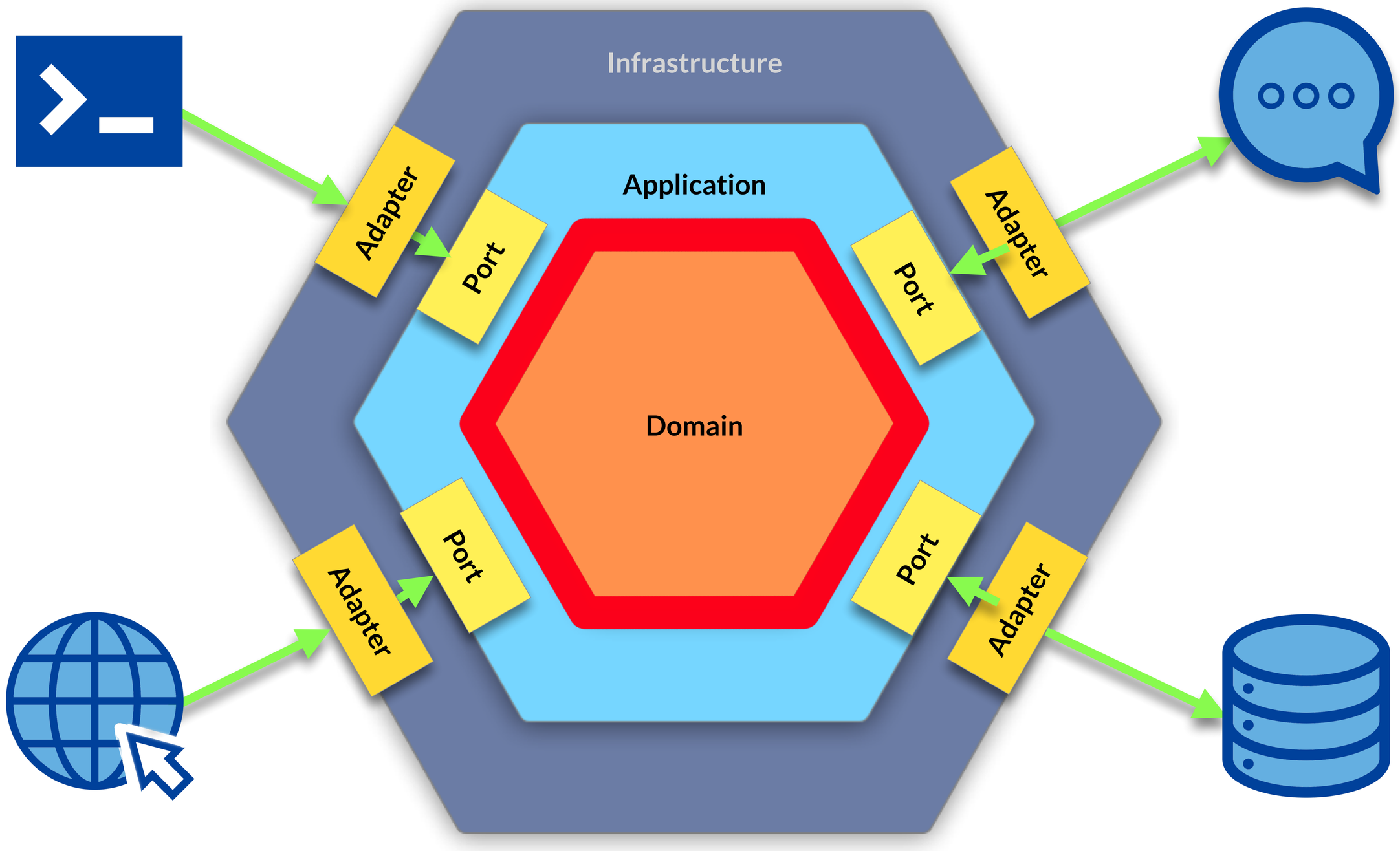
The “Out” Adapter (Driven)

- In this case the CLI is the adapter, and it driving the port, which in this case is in the **Infrastructure Layer.**



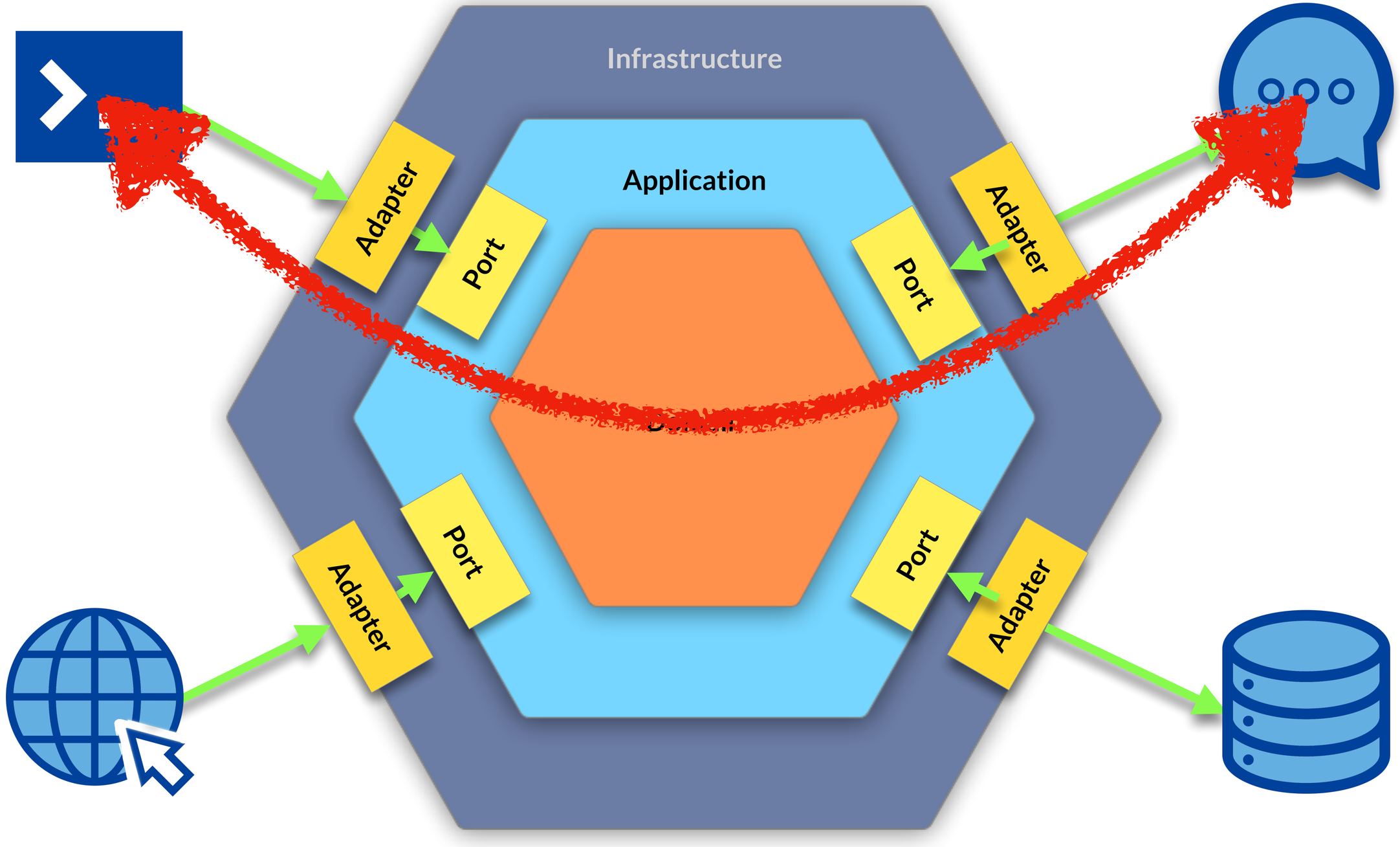
Test Driven Development

- Core business logic lives in the domain layer, free of infrastructure
- TDD is easy because the domain uses pure functions and value objects
- Input ports can be tested with mocks of output ports
- Enables fast, isolated, unit tests without databases or HTTP


Infrastructure
Application
Domain
Adapter
Port
Adapter
Port
Port
Adapter
Port
Adapter

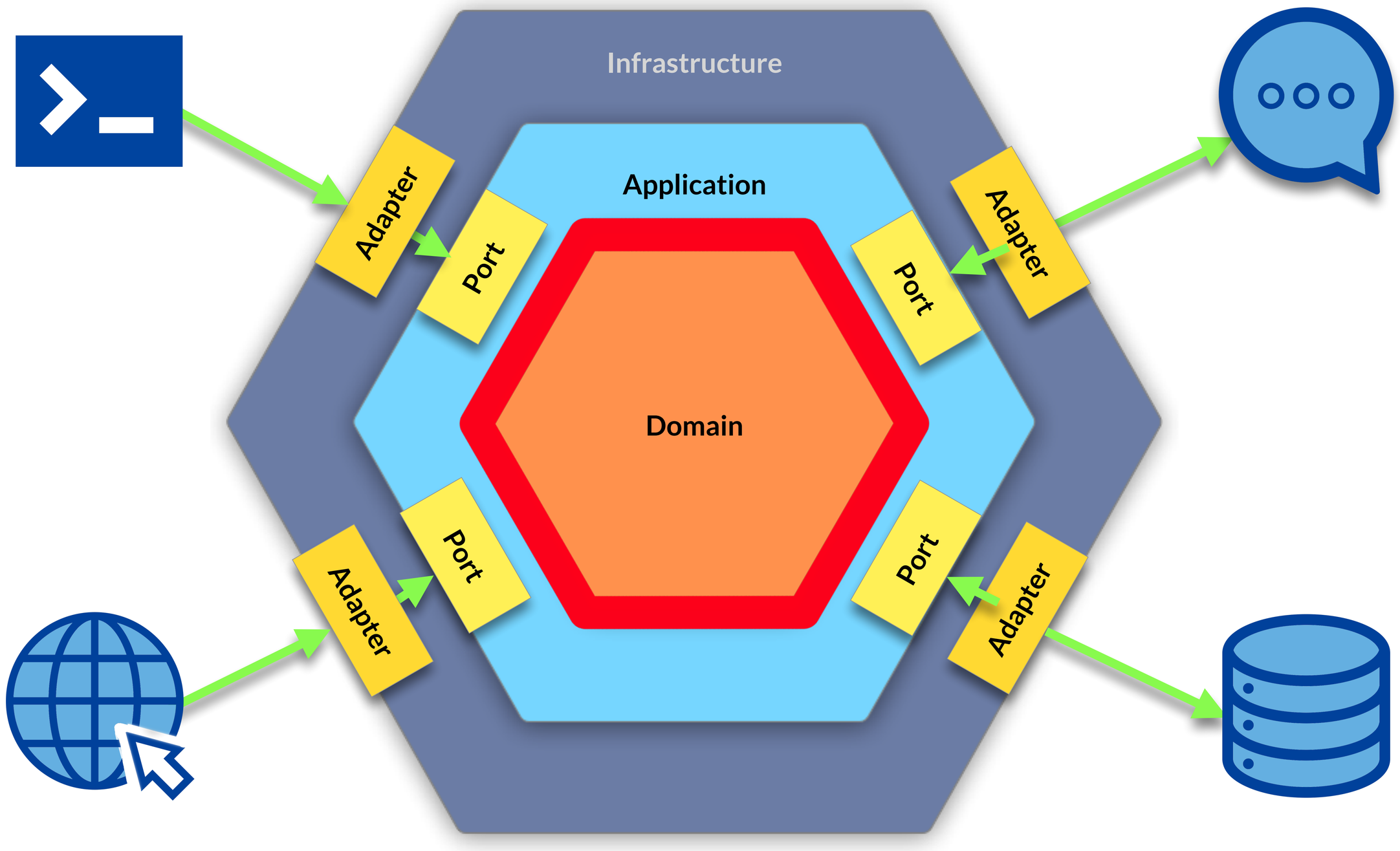
End-to-End Testing

- With adapters plugged in, ports allow for full system flow testing
- Tests go through UI or API → input port → domain → output port → DB
- Architecture keeps dependencies clear — fewer surprises
- Great for testing real-world workflows with full stack
- Another great idea is you can also swap out the DB with an in-memory database


Infrastructure
Application
Adapter
Port
Port
Adapter
Adapter
Port
Port
Adapter

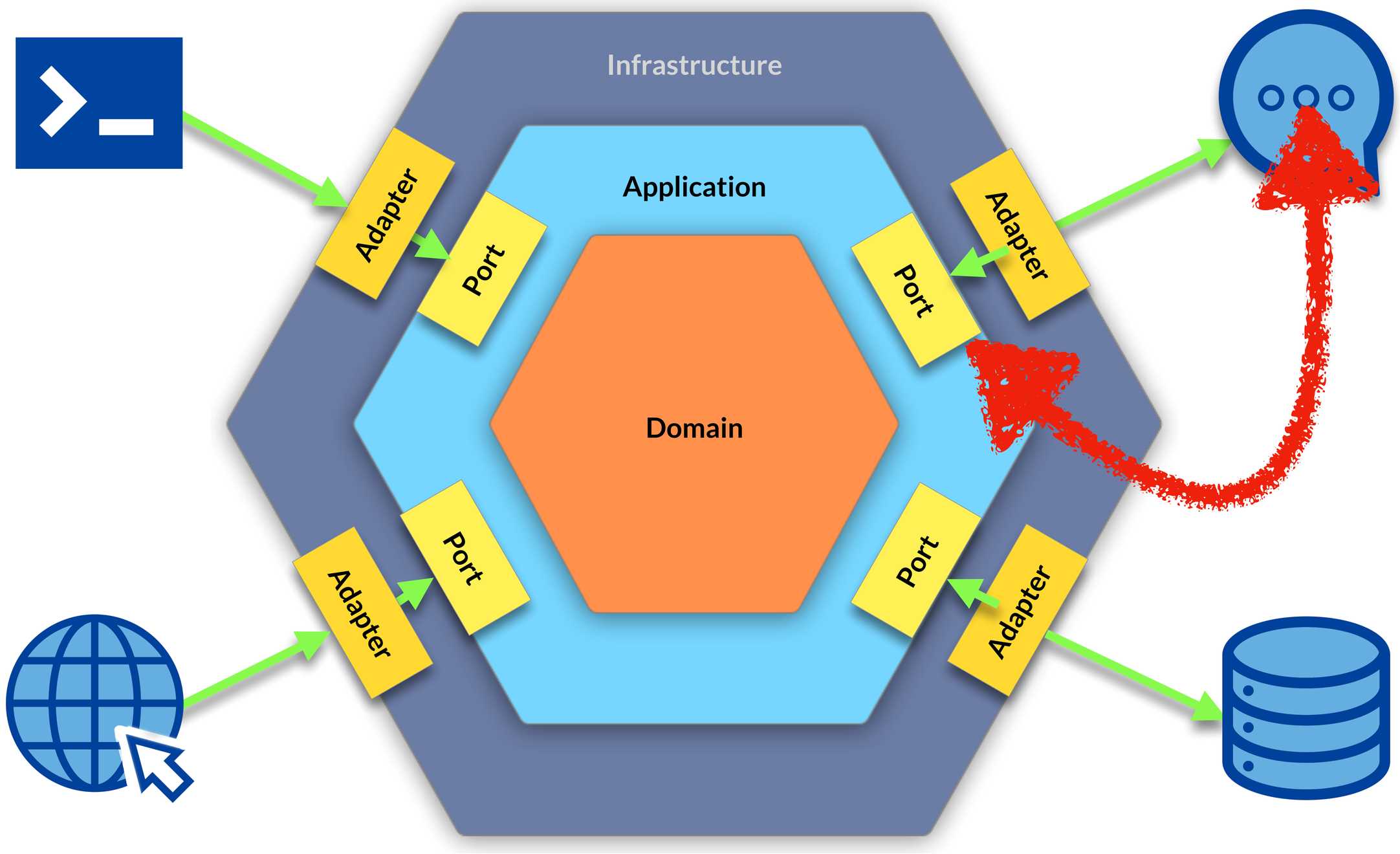
Behavior Driven Development

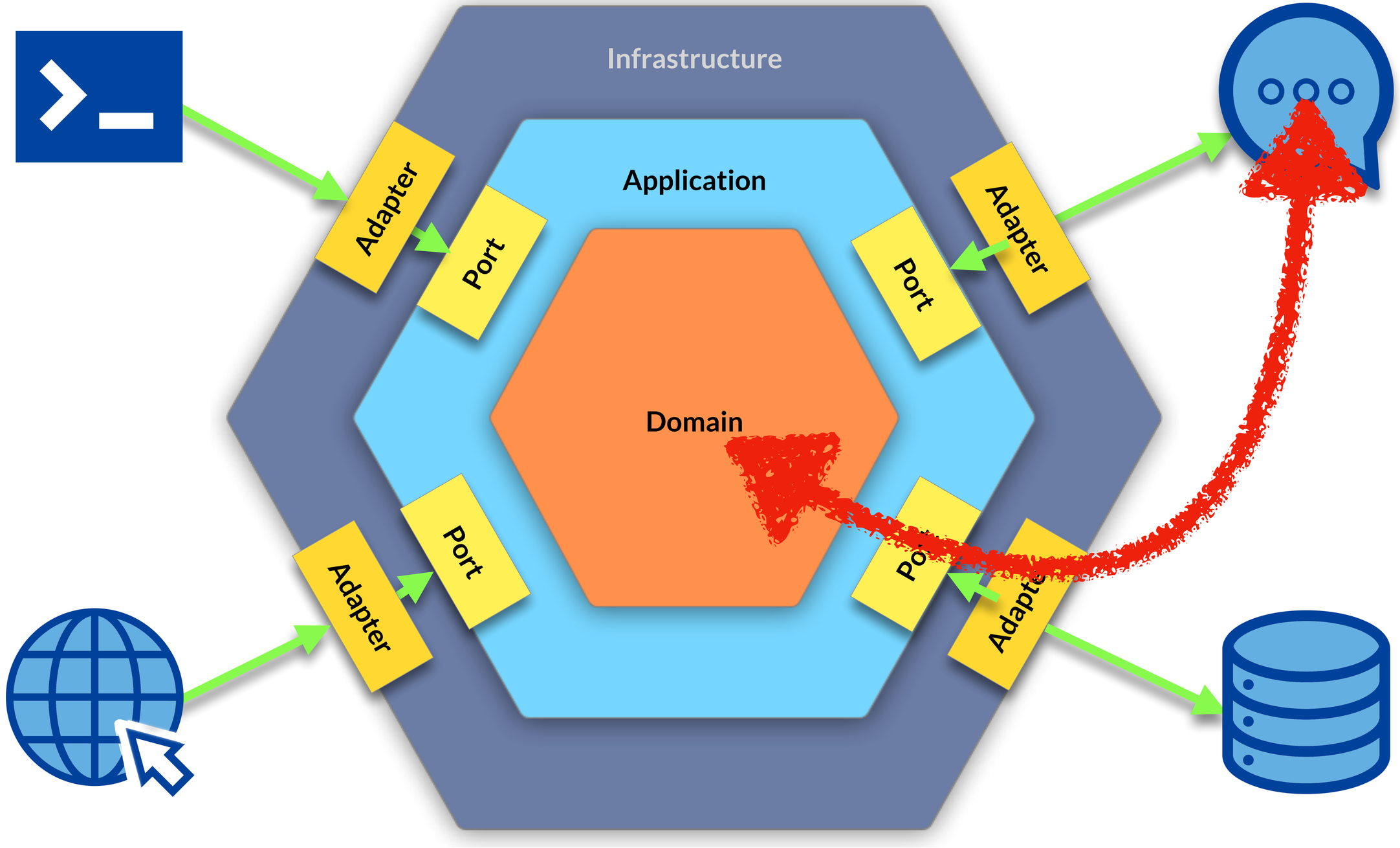
- Input ports represent use cases — perfect for scenario-driven specs
- Use cases can be exercised directly, without UI or DB
- Tests express expected business behavior, not internal steps
- Matches well with `Given-When-Then` frameworks like Cucumber or JUnit 5's `@DisplayName`


Infrastructure
Application
Domain
Adapter
Port
Adapter
Port
Port
Adapter
Port
Adapter

Data Driven Testing

- Output ports let you swap in real infrastructure for adapter testing
- Use **TestContainers** to run real DBs, APIs, brokers in test environments
- Contracts are enforced by testing the adapter separately
- Great for adapter-level integration tests with reproducible environments


Infrastructure
Application
Domain
Adapter
Port
Adapter
Port
Port
Adapter
Port
Adapter


Infrastructure
Application
Domain
Adapter
Port
Adapter
Port
Port
Adapter
Adapter
Port

Testing Strategies

- Isolate the test, load all the data, tear down all the data for each property
- Use lifecycle methods when in doubt
- Not one property should depend on another
- You can create a docker container preloaded with the data you need
  - Great for bootstrapping data with “issues” you wish to duplicate
- You can also load data with a init.sql script, but that is dependent on the type of data store test container you are using
- Use of schema evolution tools (e.g. Flyway) can work well if applied first
- Using your framework’s ephemeral loaders works amazingly well
- Fakers are great, but you lose the edge cases by doing so, and that may be alright.

Note about using both JQwik and TestContainers

- You will need to use the jqwik-testcontainers dependency

```
<dependency>
    <groupId>net.jqwik</groupId>
    <artifactId>jqwik-testcontainers</artifactId>
    <version>0.5.2</version>
    <scope>test</scope>
</dependency>
```

- Ensure that in the test that you use the JQwik version of the container annotations

```
import net.jqwik.testcontainers.Container;
import net.jqwik.testcontainers.Testcontainers;
```

Demo: Test Containers and JQwik Larger Applications

- First, let’s view the setup of using both TestContainers and JQwik in a much larger application.
- Let’s view the different layers of testing with JQwik and Test Containers
- For fun, let’s also look at TestContainers using docker-compose for create e2e tests as well

Conclusion

Conclusion

- TDD is still the same but comes with new friends and techniques
- **jQwik** lets you explore massive input spaces with confidence
- **Testcontainers** gives you real databases with production-like behavior
- **Ai** gives you guidance but can throw you off

Thank You

- Email: dhinojosa@evolutionnext.com
- Github: https://www.github.com/dhinojosa
- Linked In: http://www.linkedin.com/in/dhevolutionnext