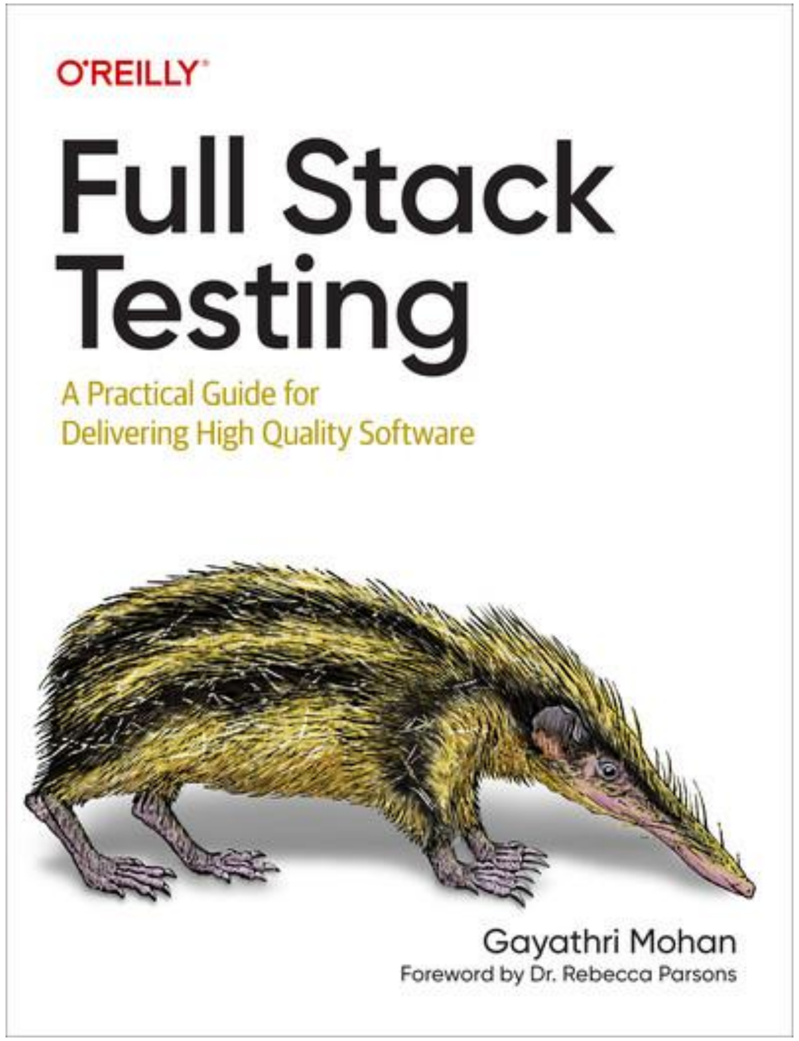
Full Stack Testing in Web Dev

IF3110 Web-Based Application Development
School of Electrical Engineering and Informatics
Institut Teknologi Bandung

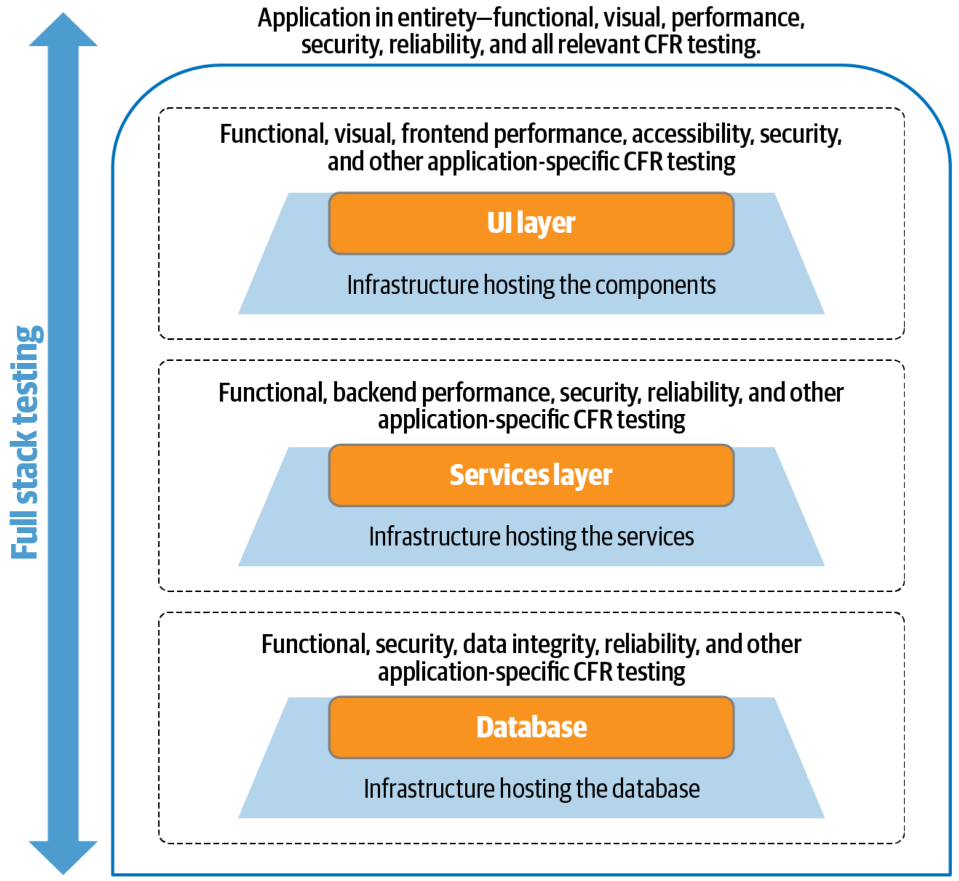
Ref


O'REILLY
Full Stack Testing
A Practical Guide for Delivering High Quality Software
Gayathri Mohan
Foreword by Dr. Rebecca Parsons

- Credit
  - Most pictures (if not ALL) are taken from this book

Motivation

- Scope and Layers of Testing

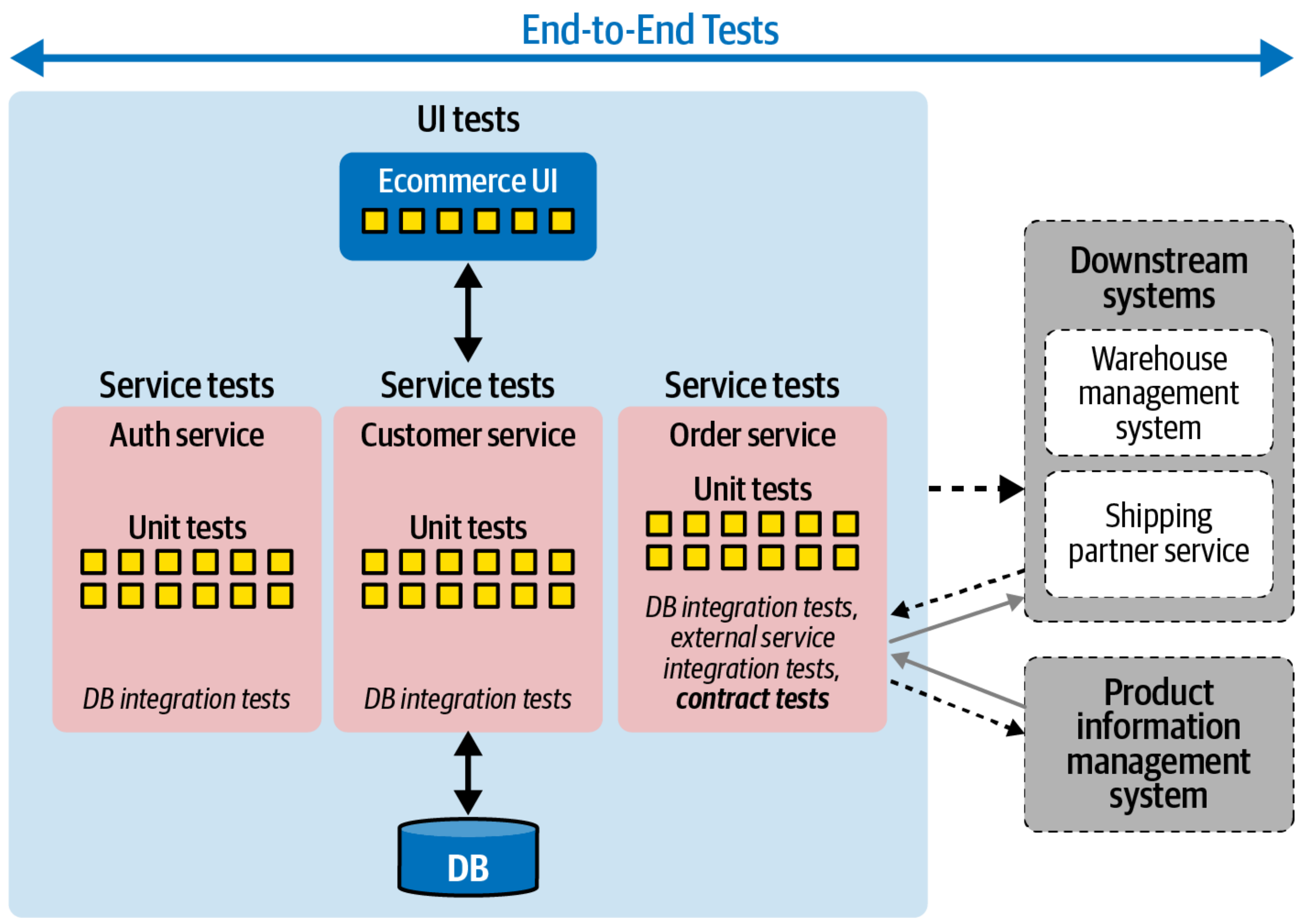


Application in entirety—functional, visual, performance, security, reliability, and all relevant CFR testing.
Functional, visual, frontend performance, accessibility, security, and other application-specific CFR testing
UI layer
Infrastructure hosting the components
Functional, backend performance, security, reliability, and other application-specific CFR testing
Services layer
Infrastructure hosting the services
Functional, security, data integrity, reliability, and other application-specific CFR testing
Database
Infrastructure hosting the database
Full stack testing

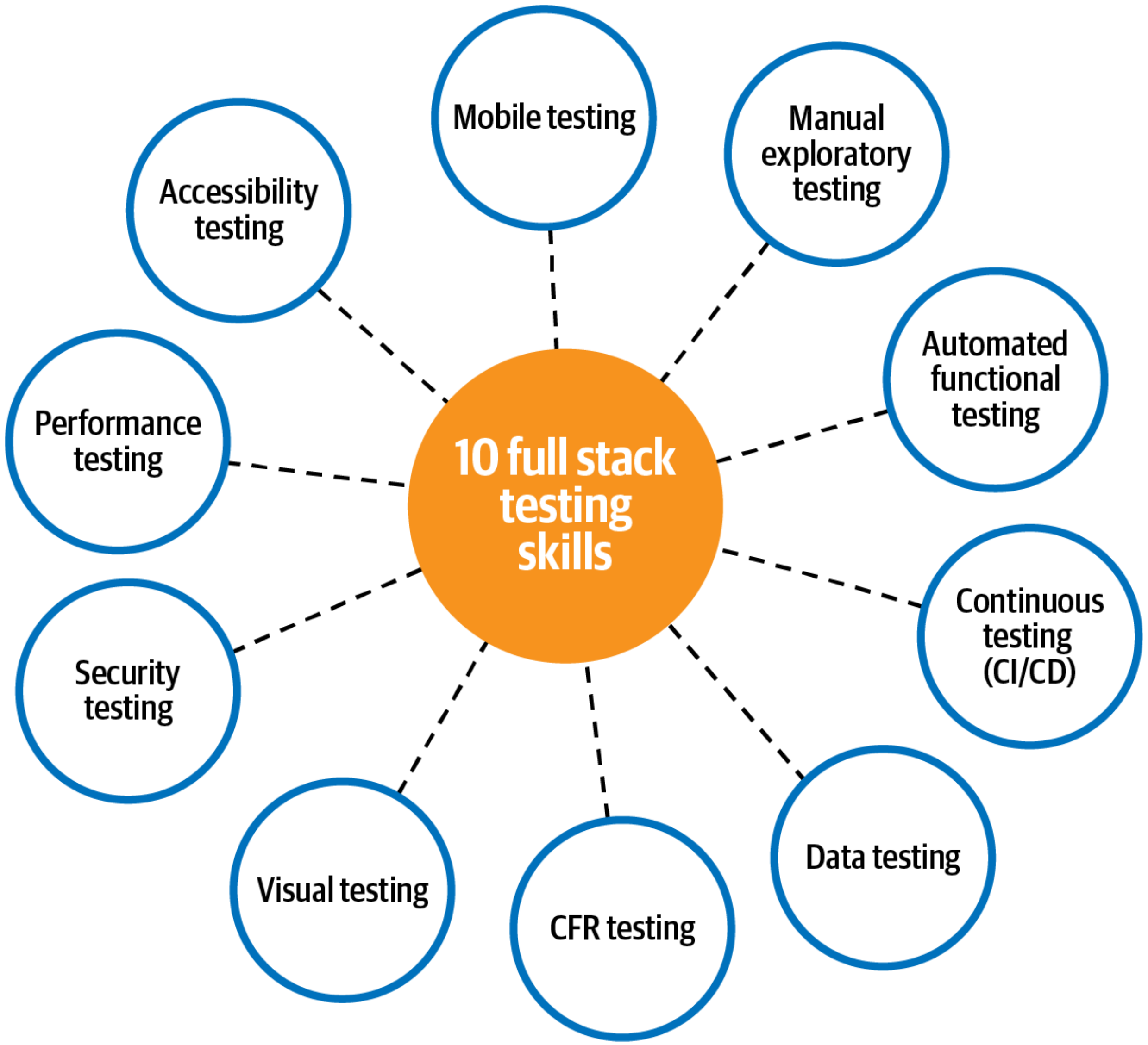
- Full-stack testing testing different aspects of the application’s quality in each layer (database, services, and UI), and the application as a whole.

Motivation

- End to End Tests

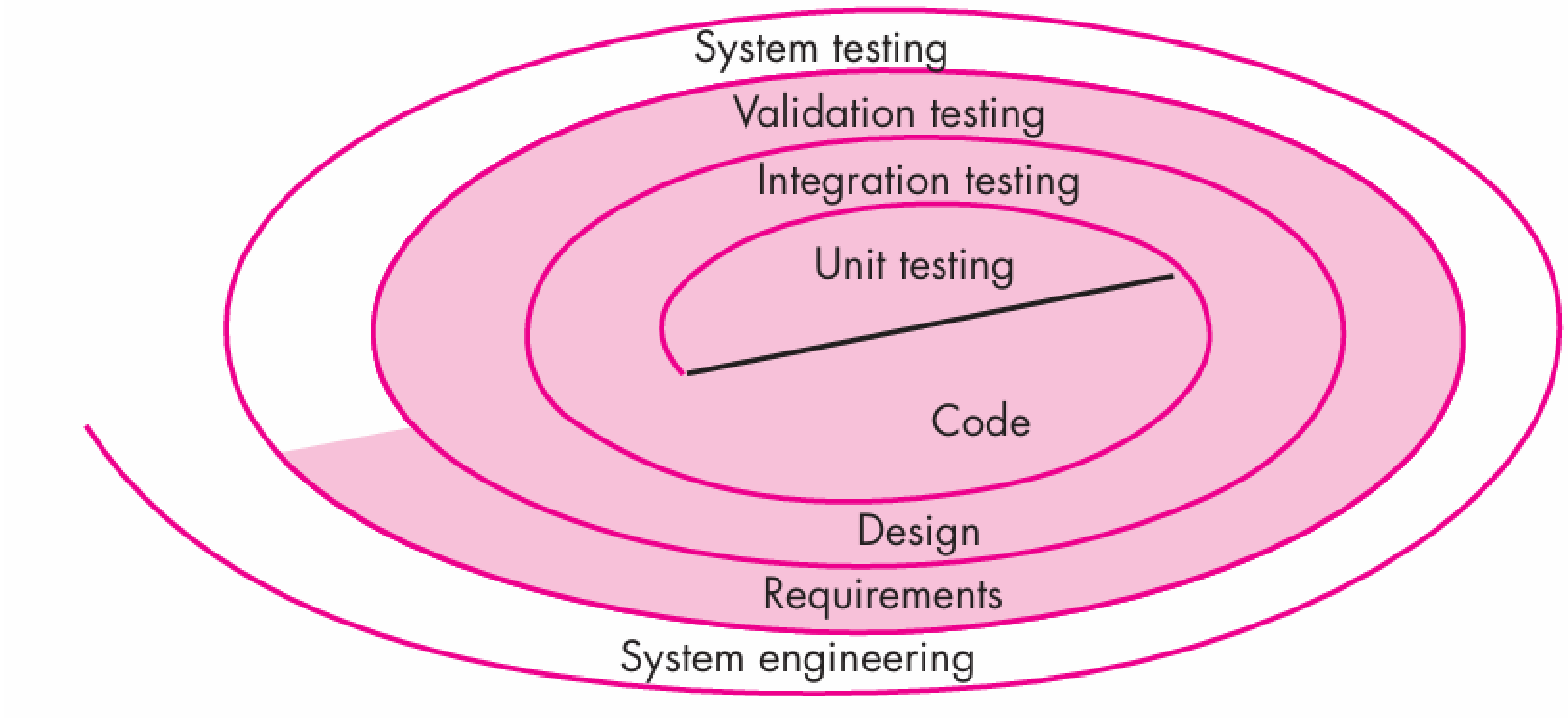


Applicable Tests for Full Stack


10 full stack testing skills
Mobile testing
Manual exploratory testing
Automated functional testing
Continuous testing (CI/CD)
Data testing
CFR testing
Visual testing
Security testing
Performance testing
Accessibility testing

Testing Strategy

© Pressman



Test Pyramid

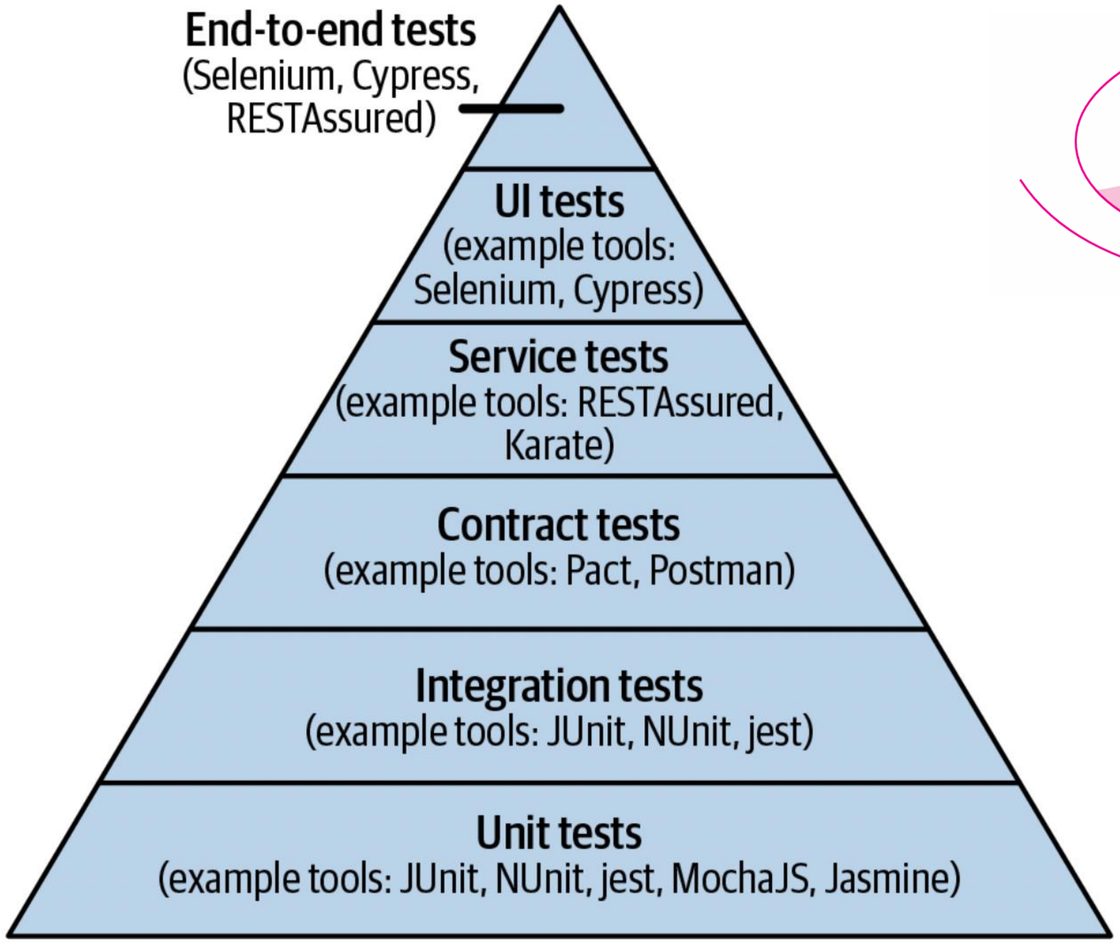
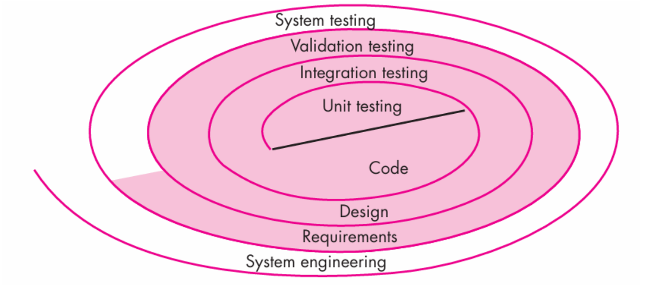

End-to-end tests
(Selenium, Cypress, RESTAssured)
UI tests
(example tools: Selenium, Cypress)
Service tests
(example tools: RESTAssured, Karate)
Contract tests
(example tools: Pact, Postman)
Integration tests
(example tools: JUnit, NUnit, jest)
Unit tests
(example tools: JUnit, NUnit, jest, MochaJS, Jasmine)

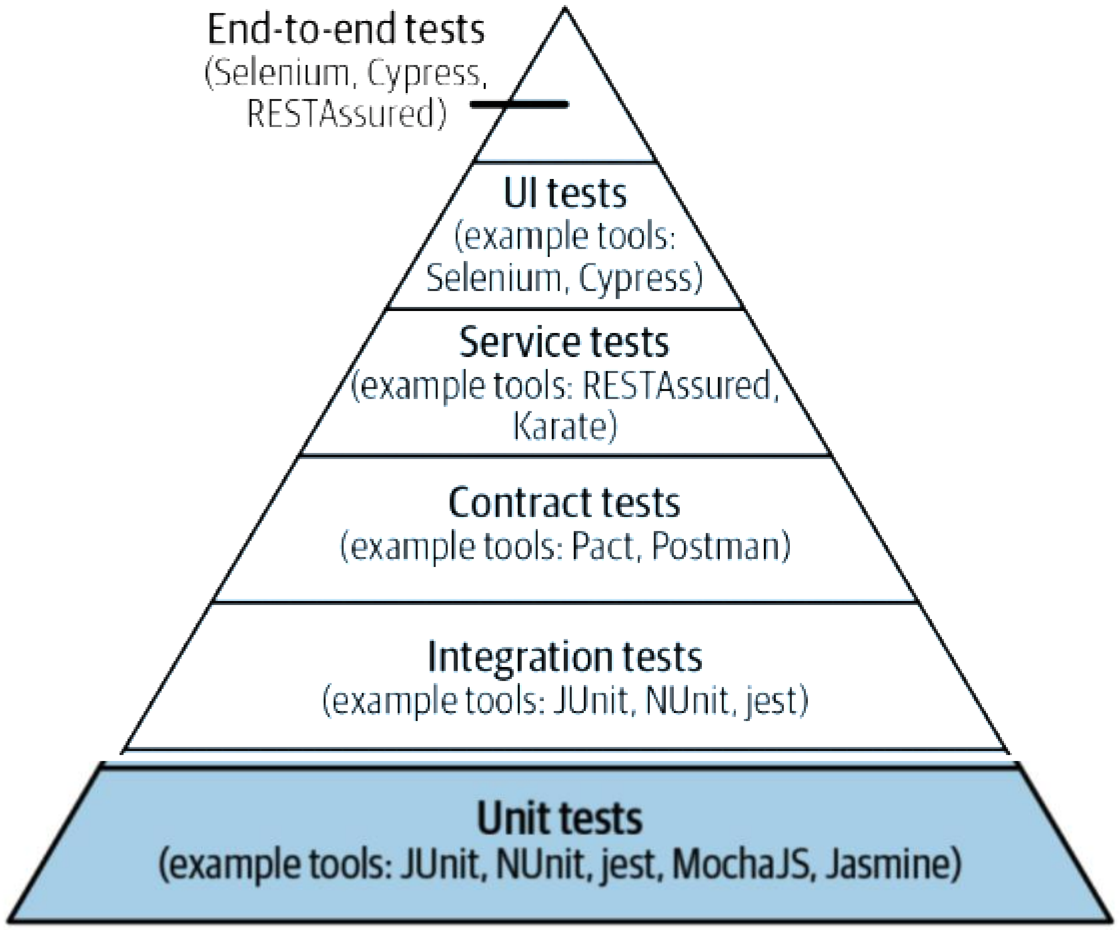
Figure 3-2. Test pyramid for a service-oriented web application

© Pressman



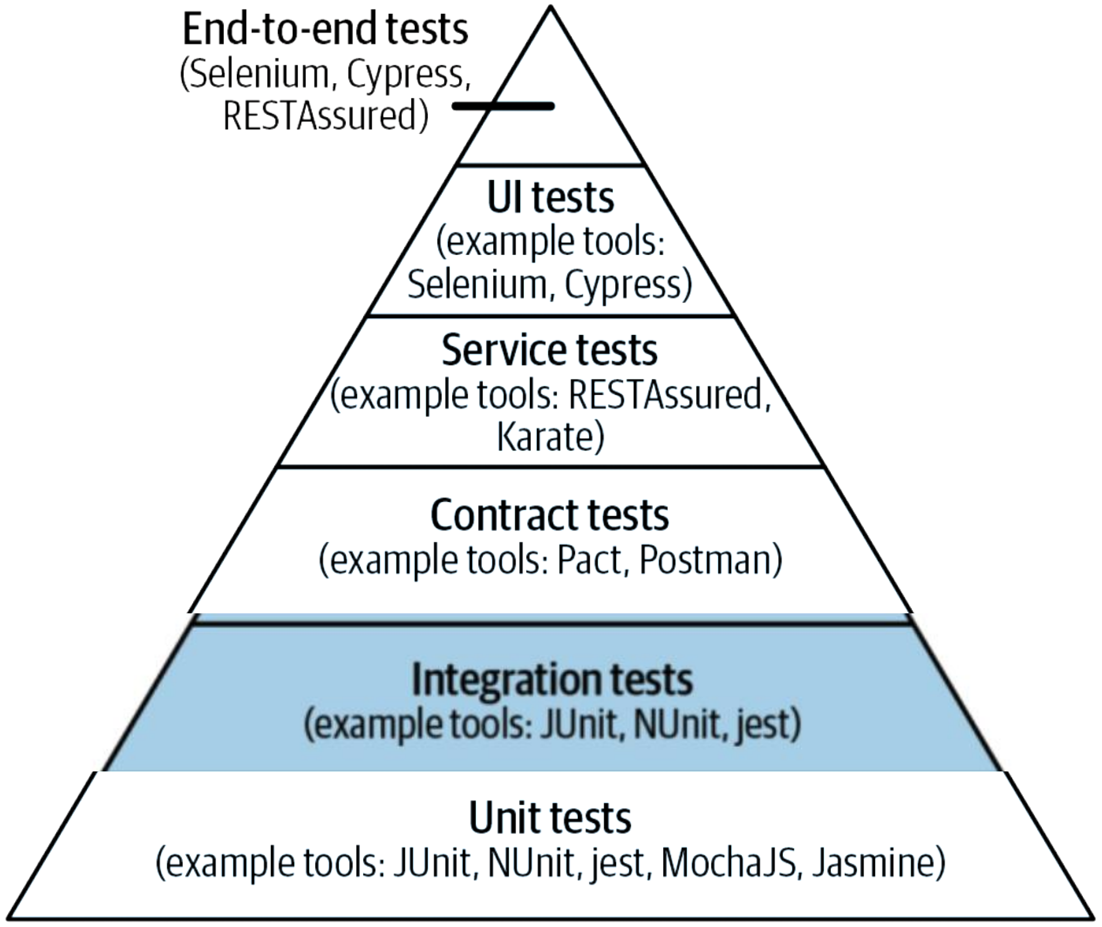
Unit Testing

- Validate smallest portion of app's functionality
- Typically done in automated and continuous testing
- Eg. Verify method behaviour in a class
- Example: return_order_total(item_prices) has the following tests:
  - Return correct totals after negative value inside item_prices due to discount
  - Return when item_prices empty
  - Return properly rounded total amount with fixed decimals
  - Etc
- Fastest to run


End-to-end tests
(Selenium, Cypress, RESTAssured)
UI tests
(example tools: Selenium, Cypress)
Service tests
(example tools: RESTAssured, Karate)
Contract tests
(example tools: Pact, Postman)
Integration tests
(example tools: JUnit, NUnit, jest)
Unit tests
(example tools: JUnit, NUnit, jest, MochaJS, Jasmine)

Integration Testing

- Focus on positive/negative integration flows, not detailed end-to-end functionality
- Small as unit tests
- Example: in ecommerce app, order service integrate with internal components such as ecommerce UI, database, external services
- Tests for each service to verify wheter it can properly communicate one another.



End-to-end tests
(Selenium, Cypress, RESTAssured)
UI tests
(example tools: Selenium, Cypress)
Service tests
(example tools: RESTAssured, Karate)
Contract tests
(example tools: Pact, Postman)
Integration tests
(example tools: JUnit, NUnit, jest)
Unit tests
(example tools: JUnit, NUnit, jest, MochaJS, Jasmine)

Figure 3-2. Test pyramid for a service-oriented web application

Contract Testing

- Simply verify the contract structure
- Contract agreed so the stubs* can be worked on during development
- Contract test don't have to check exact data, but focus on the contract structure
- Tests for each contract if there changes, the dependent service can be notified and changed.

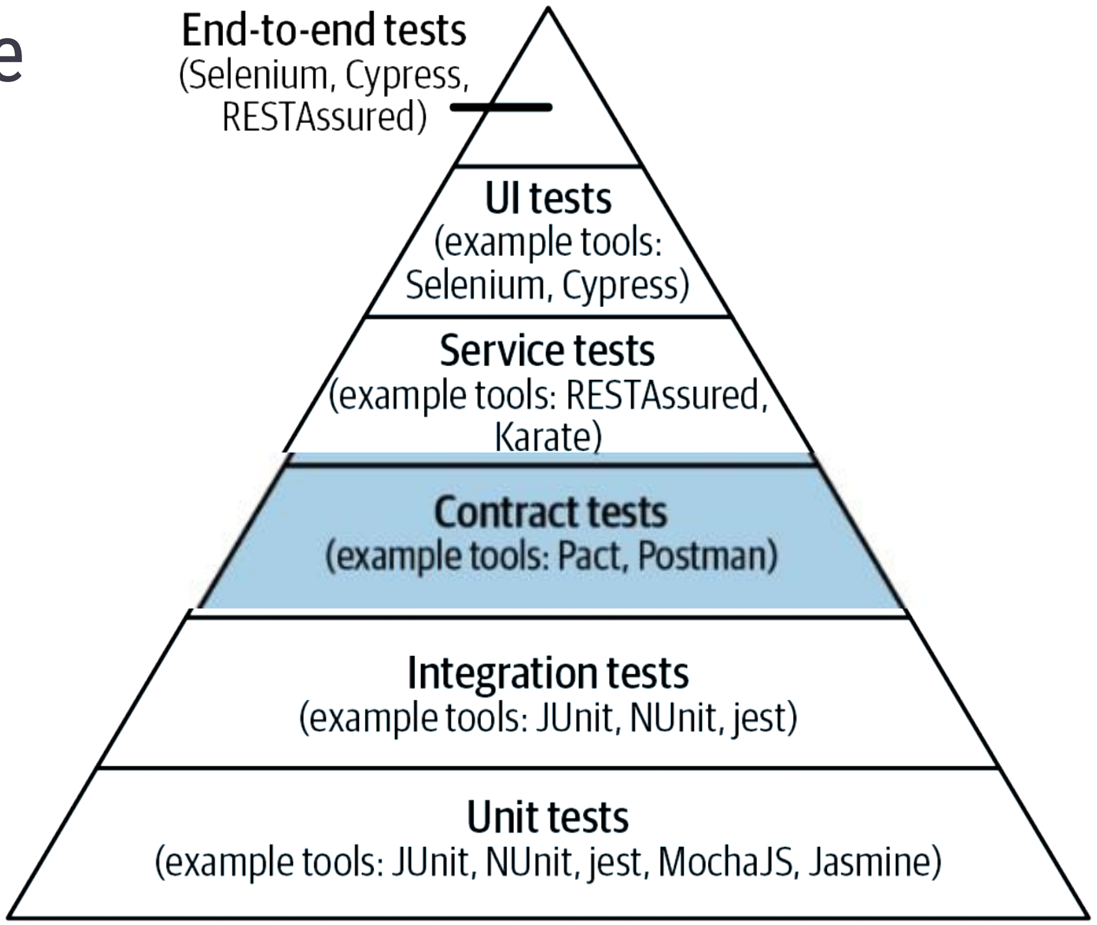
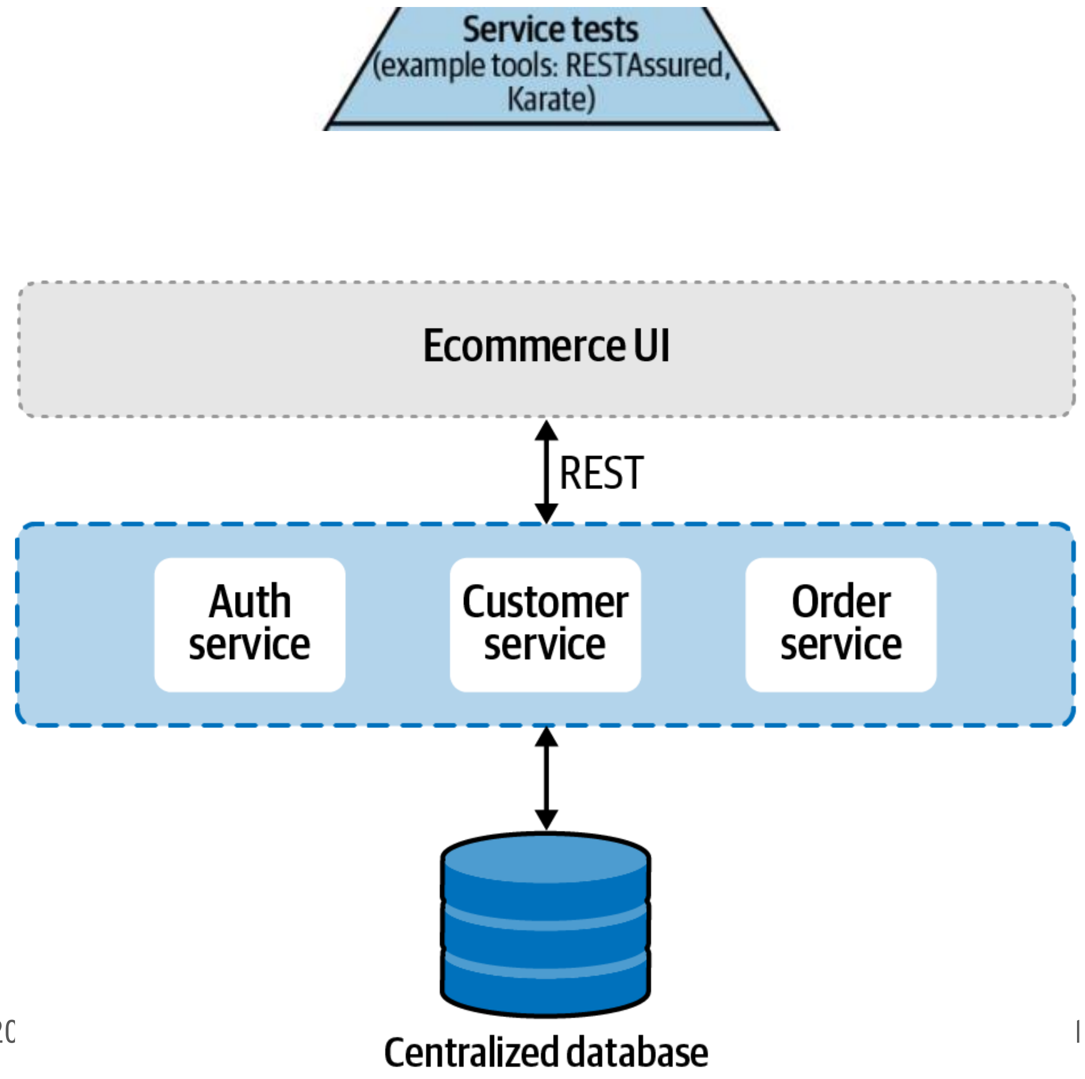

End-to-end tests
(Selenium, Cypress, RESTAssured)
UI tests
(example tools: Selenium, Cypress)
Service tests
(example tools: RESTAssured, Karate)
Contract tests
(example tools: Pact, Postman)
Integration tests
(example tools: JUnit, NUnit, jest)
Unit tests
(example tools: JUnit, NUnit, jest, MochaJS, Jasmine)

Figure 3-2. Test pyramid for a service-oriented web application

Service/API Testing

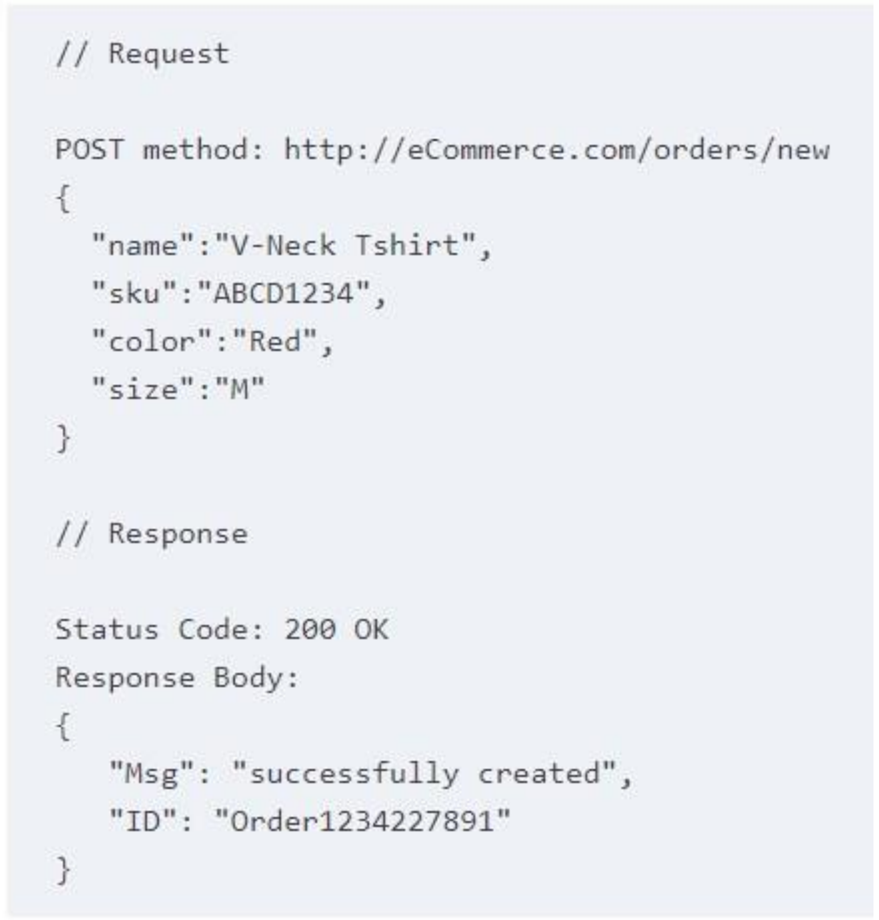
- An application programming interface, or API, provides a way for systems to interact with each other.
- APIs essentially abstract away the underlying complexities of a system and simplify the exchange of information over the network as XML, JSON, or plain text using the HTTP protocol.
- A web service is a component that serves an independent purpose within an application. For example, the order service in our sample ecommerce application might have the responsibility of managing (creating, updating, and deleting) orders, while the customer service is responsible for maintaining the customers' details.
- Every service need to have service tests for all its endpoints
- Example:
  - Verify only authenticated user can create new order
  - Verify that the correct HTTP status codes are returned for positive and negative inputs



Protocols for APIs

- To standardize information exchange, protocols like SOAP and specifications like REST were invented. These days RESTful APIs are more prevalent than SOAP, and even the legacy systems that use SOAP are being rewritten with REST specifications.

Example 2-1. Sample REST request and response



```
// Request

POST method: http://eCommerce.com/orders/new
{
  "name":"V-Neck Tshirt",
  "sku":"ABCD1234",
  "color":"Red",
  "size":"M"
}

// Response

Status Code: 200 OK
Response Body:
{
   "Msg": "successfully created",
   "ID": "Order1234227891"
}
```

Example of API test Framework: REST Assured

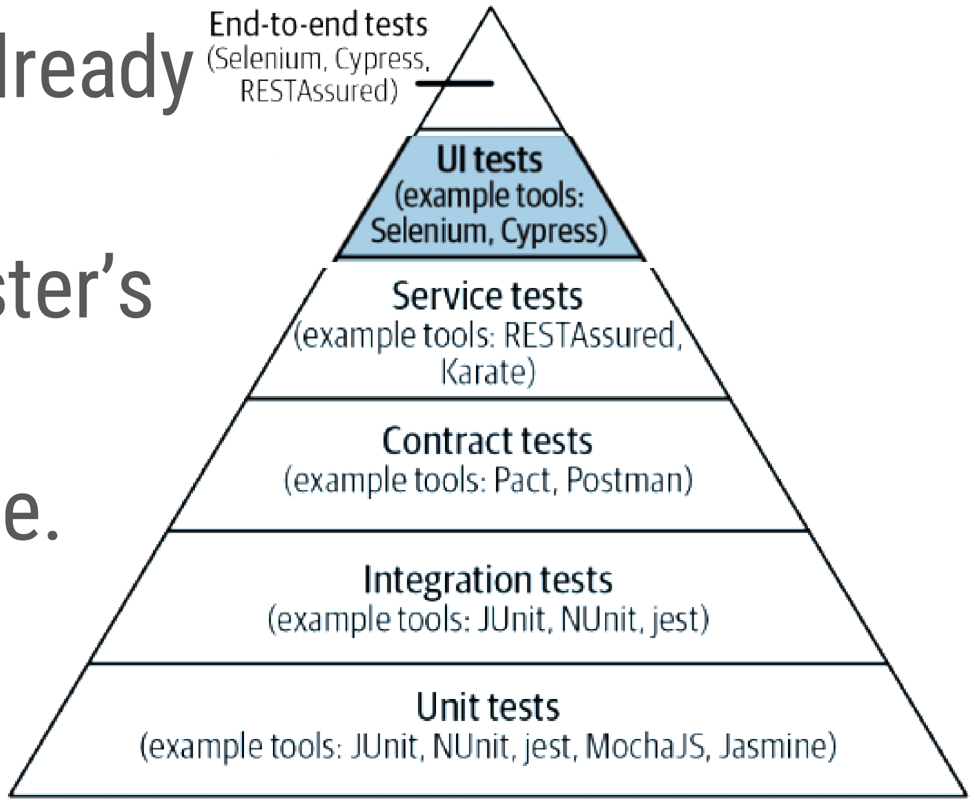
- REST Assured (rest-assured.io)

```
GET: https://eCommerce.com/items

Response:

Status Code: 200
[
    {
        "SKU": "984058981",
        "Color": "Green",
        "Size": "M"
    }
]
```

```java
// ItemsTest.java

package apitests;

import org.testng.annotations.Test;

import static io.restassured.RestAssured.given;

public class ItemsTest {

    @Test
    public void verifyGetItemsEndpointReturnsSuccessStatusCode(){
        given().
                when().
                get("http://localhost:1000/items").
                then().
                assertThat().statusCode(200);
    }
}
```

UI functional test

- Macro-level test that Mimic the user actions
- Validate all critical user flows
- Avoid validating same details that is already covered in lower level
- Kept as separate codebase, part of tester’s purview.
- Take longer to run and tend to be brittle.


End-to-end tests
(Selenium, Cypress, RESTAssured)
UI tests
(example tools: Selenium, Cypress)
Service tests
(example tools: RESTAssured, Karate)
Contract tests
(example tools: Pact, Postman)
Integration tests
(example tools: JUnit, NUnit, jest)
Unit tests
(example tools: JUnit, NUnit, jest, MochaJS, Jasmine)

End to End test

- Validate entire breadth of domain workflow, including downstream systems
- Intent to check if all the components are integrated properly end to end
- Can be implemented as few tests that will activate all components
- Combination of UI, service, and DB testing tool to cover entire flow
- Take longest time to run

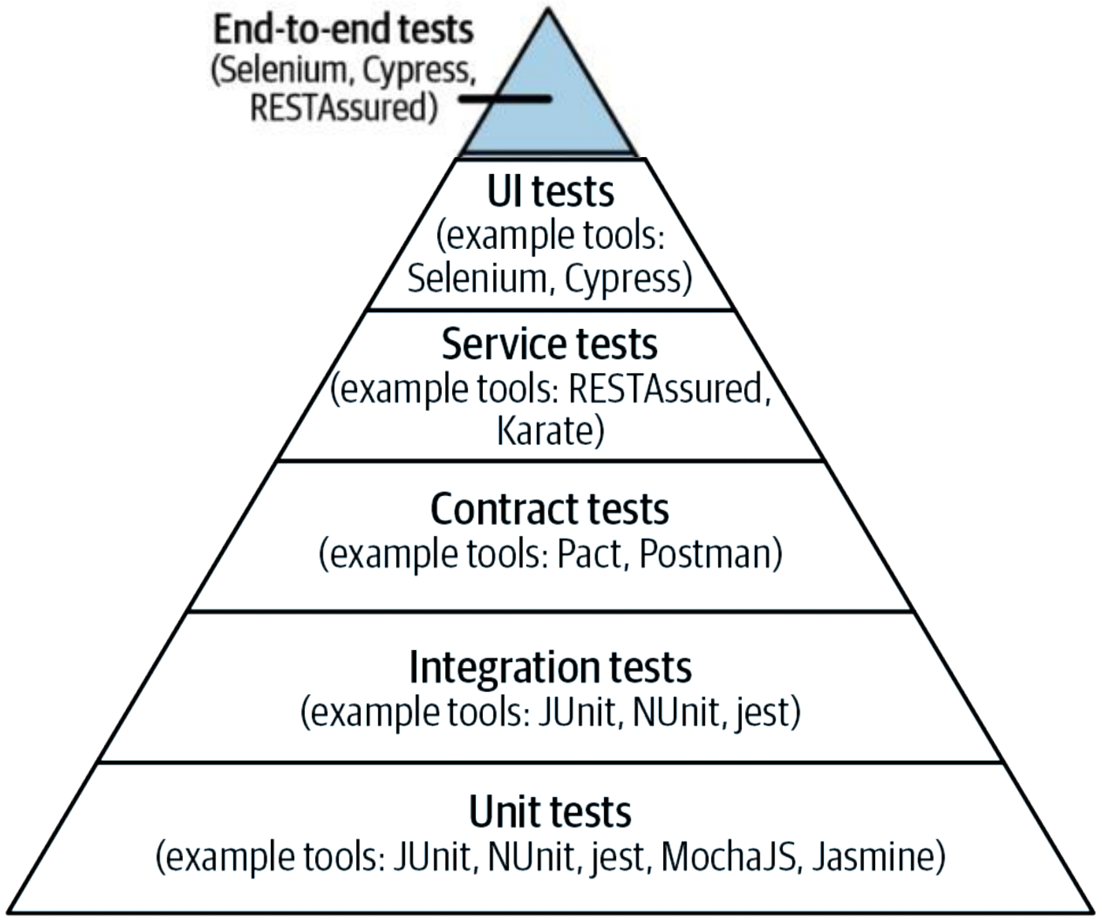
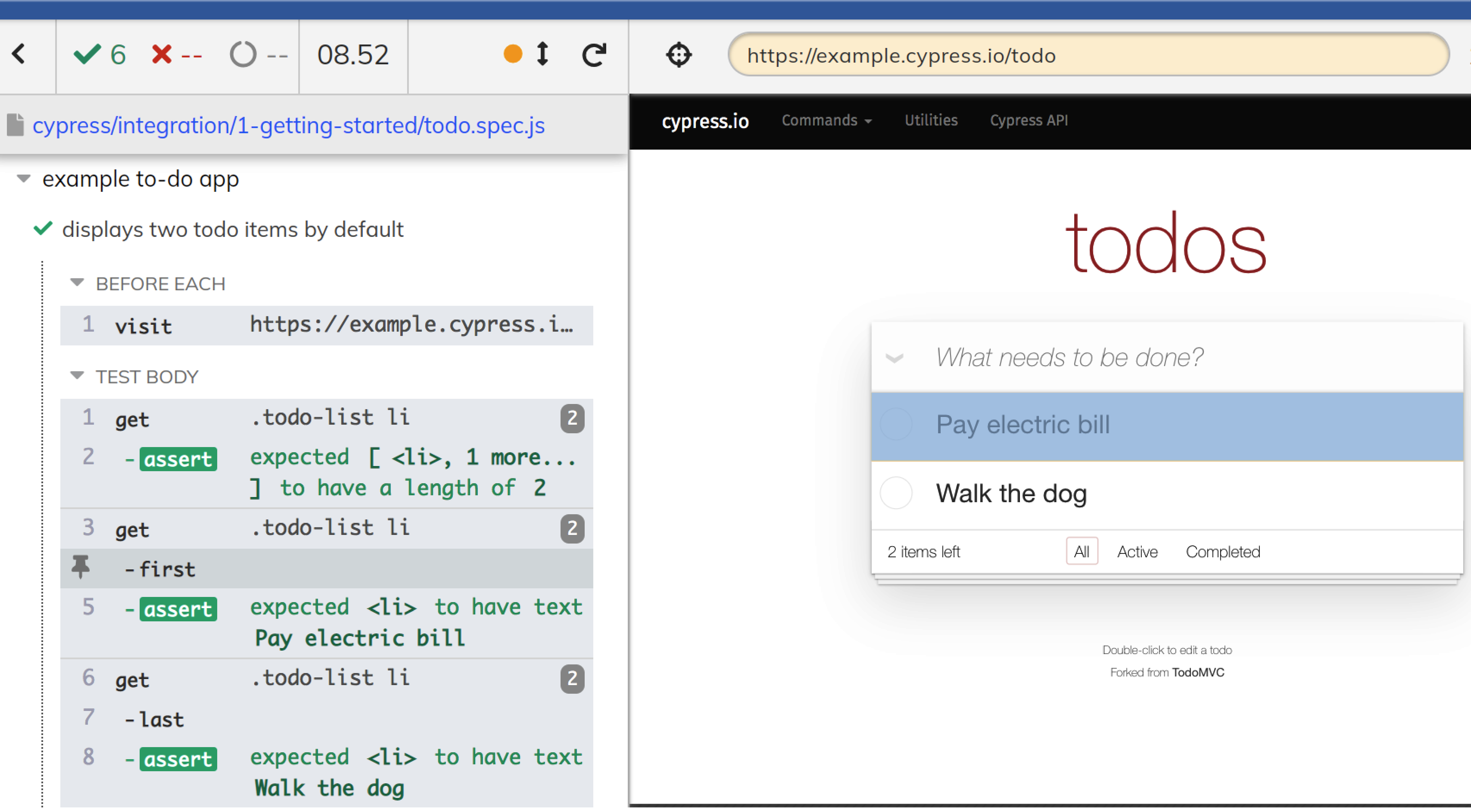

End-to-end tests (Selenium, Cypress, RESTAssured)
UI tests (example tools: Selenium, Cypress)
Service tests (example tools: RESTAssured, Karate)
Contract tests (example tools: Pact, Postman)
Integration tests (example tools: JUnit, NUnit, jest)
Unit tests (example tools: JUnit, NUnit, jest, MochaJS, Jasmine)

Figure 3-2. Test pyramid for a service-oriented web application

Example of E2E Test framework: Cypress

- Cypress’s architecture is such that it doesn’t execute commands over the network like Selenium does but executes them within the same run-loop as the application. This makes it much faster.
- Cypress is bundled with all the tools necessary to write end-to-end UI automation tests, so you don’t need to set up additional tools like TestNG, Cucumber, etc. It incorporates existing, proven tools to do their respective tasks. For example, by default Cypress uses Mocha as a testing framework and Chai for assertions.
- Since Cypress is embedded within the application, it enables the creation of varied test cases such as stubbing application functions, simulating server-down scenarios by altering requests, setting up predefined application states, and more.
- Cypress addresses test flakiness due to incorrect adoption of wait strategies by automatically waiting for a page to load and elements to be visible or clickable.
- Cypress makes debugging test failures much simpler as it provides screenshots, logs, and videos for every command that the test has executed. It also allows you to inspect errors on the application page in their predefined state as part of the test flow using Chrome DevTools.



cypress/integration/1-getting-started/todo.spec.js

example to-do app

✓ displays two todo items by default

BEFORE EACH

```
1 visit       https://example.cypress.i…
```

TEST BODY

```
1 get         .todo-list li                                  2
2 - assert    expected [ <li>, 1 more... ] to have a length of 2
3 get         .todo-list li                                  2
  - first
5 - assert    expected <li> to have text Pay electric bill
6 get         .todo-list li                                  2
7 - last
8 - assert    expected <li> to have text Walk the dog
```

todos

What needs to be done?

Pay electric bill

Walk the dog

2 items left   All   Active   Completed

Double-click to edit a todo

Forked from TodoMVC

Example of E2E Test framework: Cypress

```
// page-objects/login-page.js

/// <reference types="cypress" />

export class LoginPage {

  login(email, password){
    cy.get('[id=user_email]').type(email)
    cy.get('[id=user_password]').type(password)
    cy.get('.submitPara > .gr-button').click()
  }
}

// page-objects/home-page.js

/// <reference types="cypress" />

export class HomePage {

  getTitle(){
    return cy.title()
  }
}
```

```
// integration/eCommerce-e2e-tests/login_tests.spec.js

/// <reference types="cypress" />

import {LoginPage} from '../../page-objects/login-page'
import {HomePage} from '../../page-objects/home-page'

describe('example to-do app', () => {
   const loginPage = new LoginPage()
   const homePage = new HomePage()

   beforeEach(() => {
     cy.visit('https://example.com')
   })

   it('should log in and land on home page', () => {
     loginPage.login('example@gmail.com', 'Admin123')
     homePage.getTitle().should('have.string', 'Home Page')
   })
})
```

- The bundled Mocha test framework
  - it()
- bundled with Chai assertion library
  - should() etc