17-723: Designing Large-scale Software Systems

Design for Testability

Tobias Dürschmid
(partially inspired by a lecture by Claire Le Goues)

This Lecture - Testability

- How to Design Testable Software?
- How to Test Quality Attributes?
- How to Increase Test Coverage?
- How to Tailor Testing to Different Domains?

Case Study: Netflix!

Carnegie Mellon University

How to **Design Testable Software?**

Designing Large-scale Software Systems – Design for Testability

Carnegie Mellon University

Definition of Testability

Testing is nice; testability is better.
…because testing won't make bad code good, and you can't test well if the code itself is untestable.

The degree to which the **functionality** and **quality attributes** of a system (or component) can be **assessed** via run-time observation. (i.e., How hard is it to write effective tests?)

Testability = Controllability + Observability

Controllability

How easy is it to **provide** a program or component with the **needed inputs**, in terms of values, operations, and behaviors, and bringing it into the desired **state** that should be tested.

Testability = **<u>Controllability</u>** + **Observability**

Carnegie Mellon University

Observability

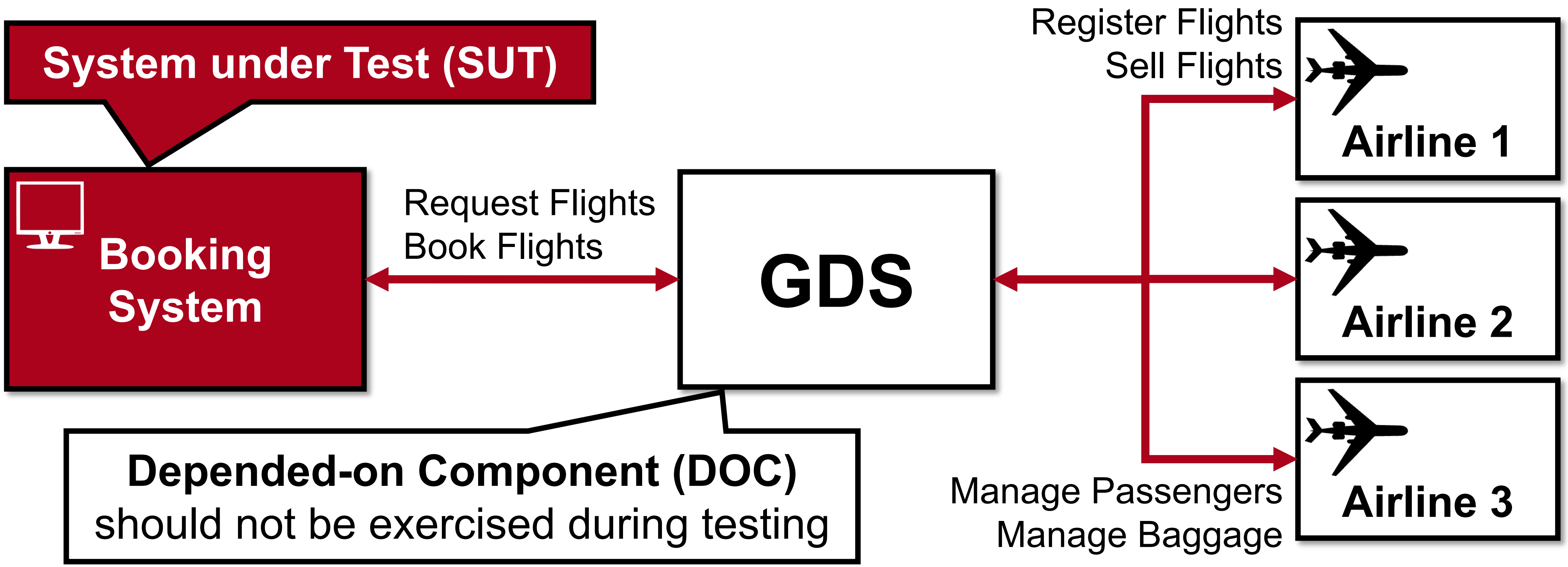
How easy is it to **observe** the **behavior** of a program or component in terms of its outputs, quality attributes, effects on the environment, and other hardware and software components.

Testability = Controllability + Observability

Carnegie Mellon University

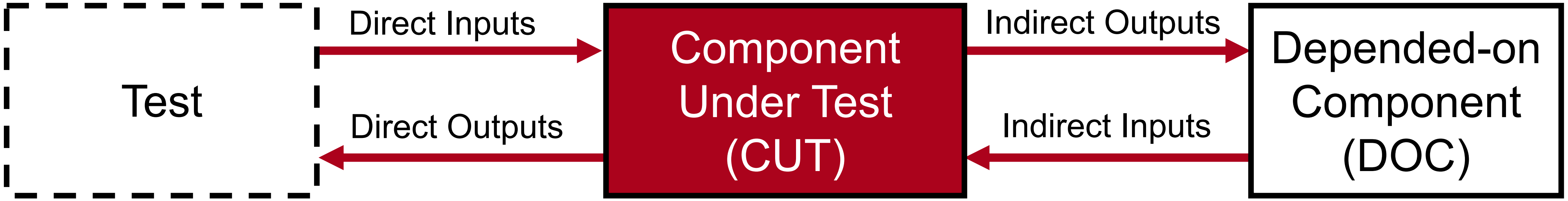
Question: What makes testing a booking system that relies on GDS difficult?

Testing a GDS-dependent **Booking System**



Carnegie Mellon University

Indirect Inputs & Indirect Outputs
Make Testing more Difficult

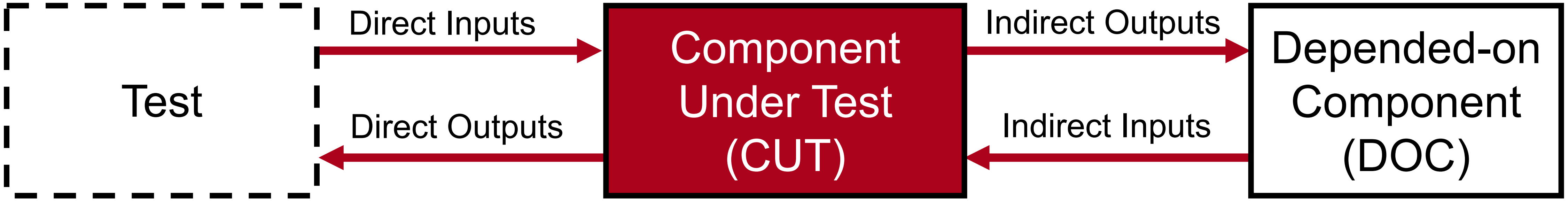


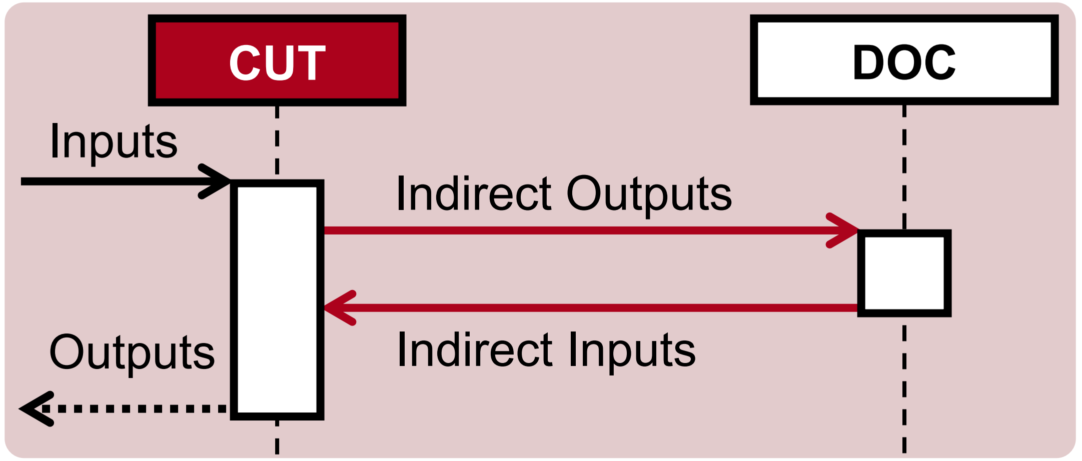
Injecting **indirect inputs** is a **Controllability** challenge

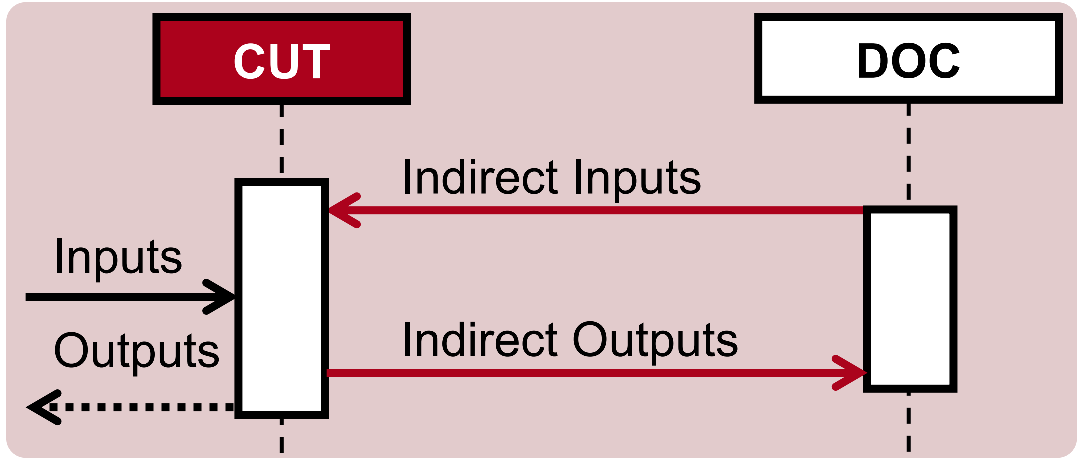
Verifying **indirect outputs** is an **Observability** challenge

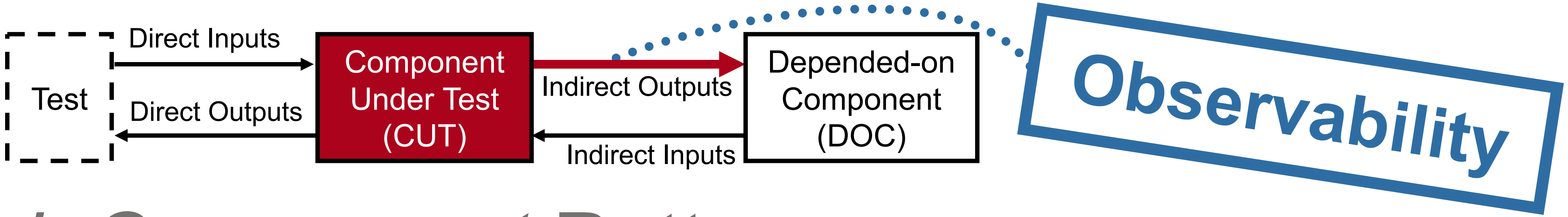
Carnegie Mellon University

Indirect Inputs & Indirect Outputs
Can be Ordered in Many Different Sequences







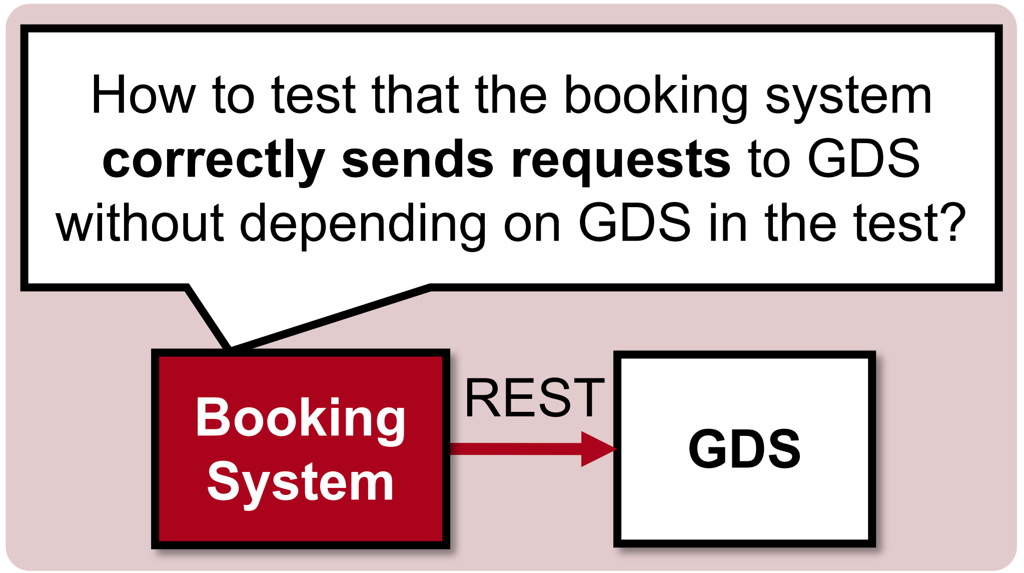


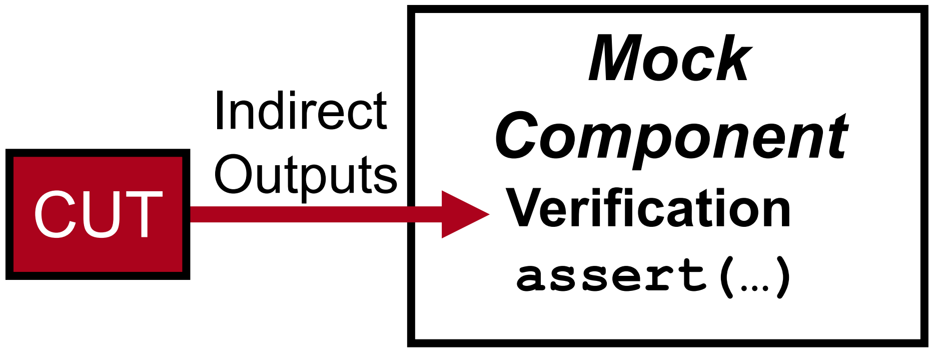
Mock Component Pattern

Problem: How to observe indirect outputs sent to separate DOCs?

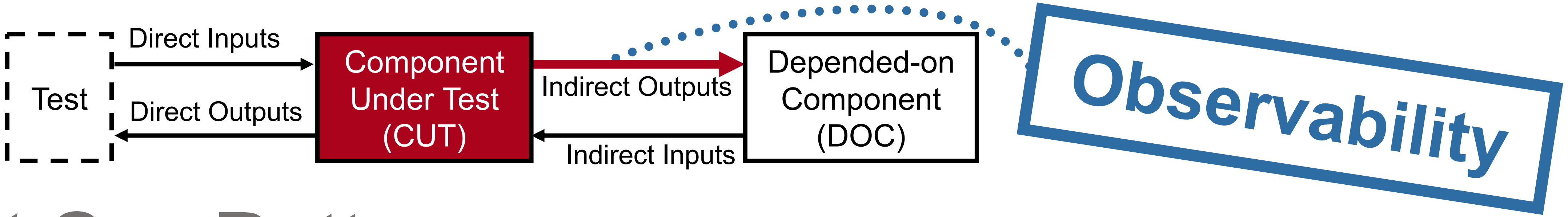
Context: The connector between CUT and DOC cannot easily be intercepted.

Solution: Create a ***Mock Component*** that **replaces** the **DOC** and only **verifies the indirect outputs**


How to test that the booking system **correctly sends requests** to GDS without depending on GDS in the test?
Booking System
REST
GDS



More Details here: http://xunitpatterns.com/Mock%20Object.html



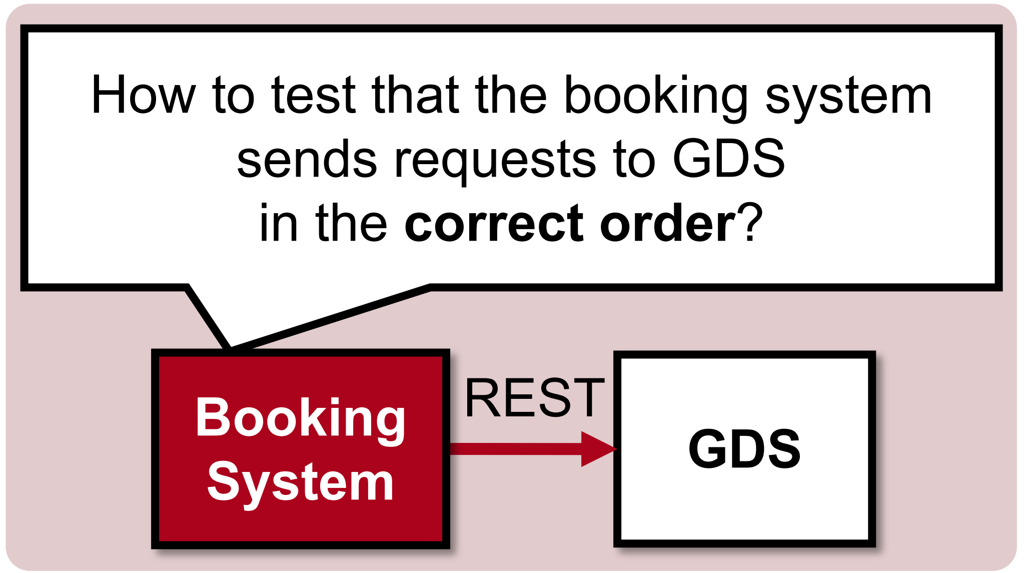
Test Spy Pattern

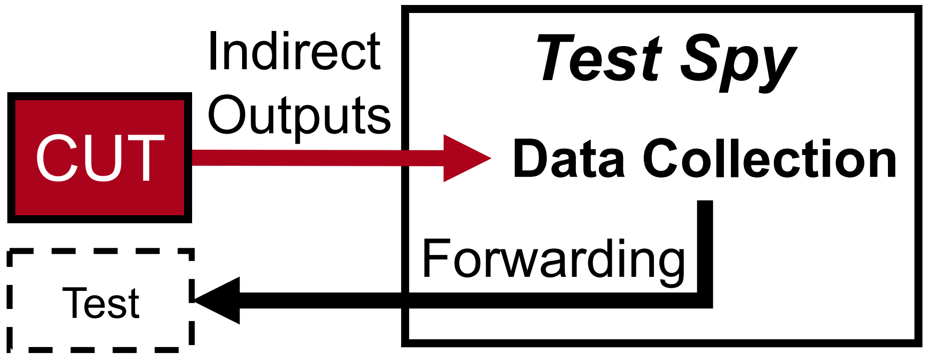
Problem: How to observe indirect outputs sent to separate DOCs?

Context: The connector between CUT and DOC cannot easily be intercepted.

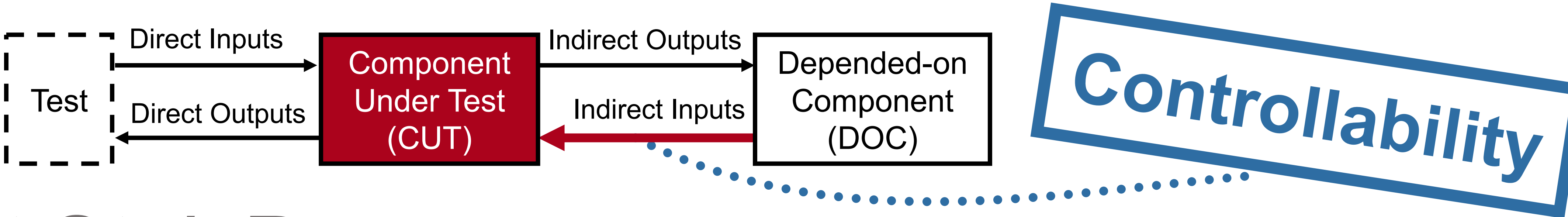
Solution: Create a ***Test Spy*** component that **replaces** the **DOC** and forwards the indirect outputs to the test.

More Details here: http://xunitpatterns.com/Test%20Spy.html


How to test that the booking system sends requests to GDS in the **correct order**?
Booking System
REST
GDS



Carnegie Mellon University

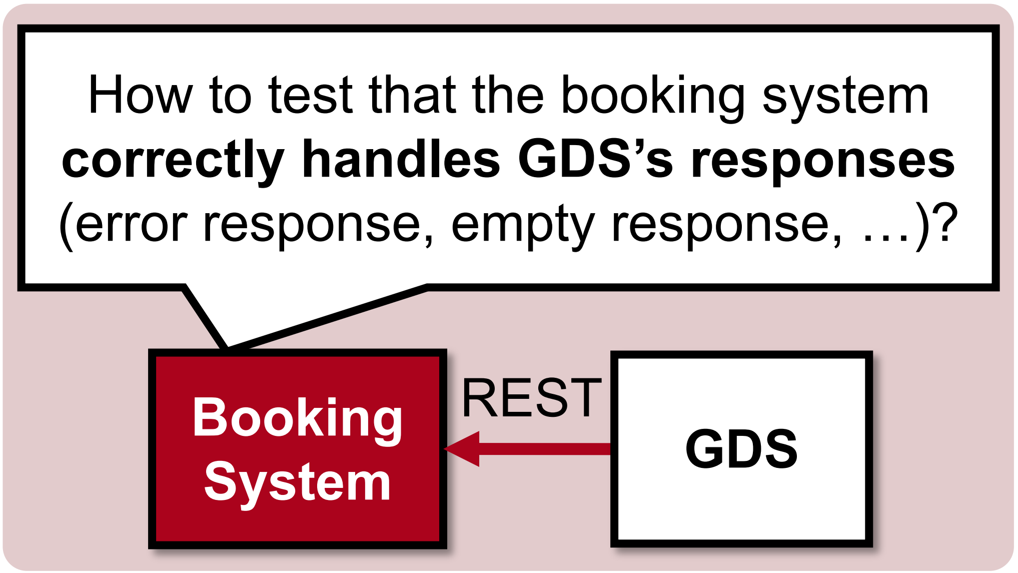


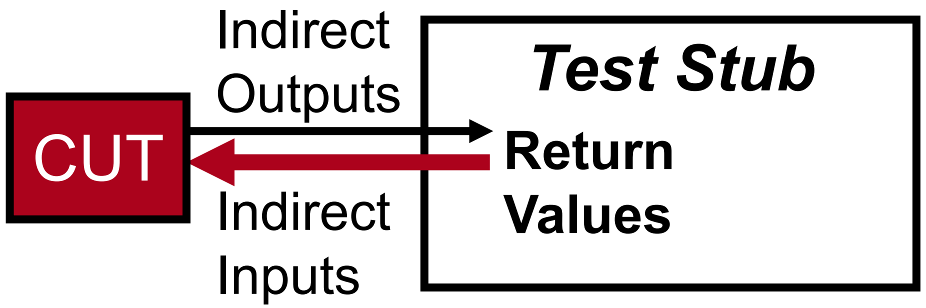
Test Stub Pattern

Problem: How to control indirect inputs sent from separate DOCs?

Context: The connector between CUT and DOC cannot easily be intercepted.

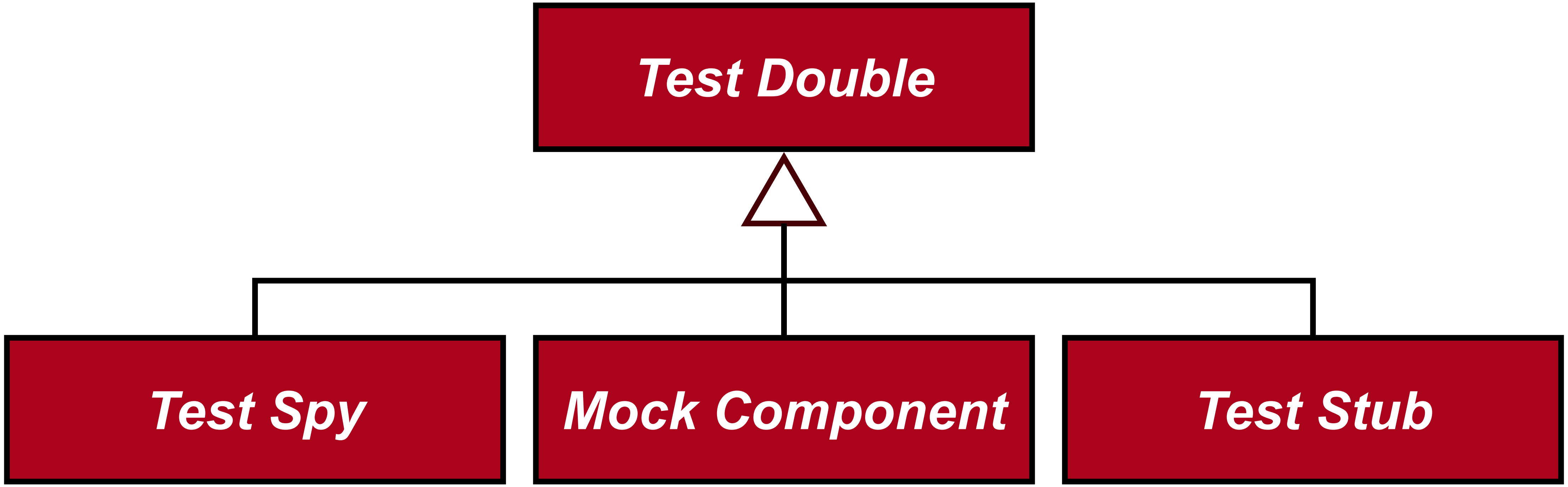
Solution: Create a ***Test Stub*** component that **replaces** the **DOC** and sends the desired inputs to the CUT.


How to test that the booking system **correctly handles GDS's responses** (error response, empty response, …)?
Booking System
REST
GDS



More Details here: http://xunitpatterns.com/Test%20Stub.html

Test Spies, *Mock Components*, and *Test Stubs* are all unified under the term ***Test Doubles***



Design Principle for Testability: **Apply the SOLID Principles**

Single Responsibility Principle (SRP)

→ **Smaller pieces** of functionality are **easier to test**.

Interface Segregation Principle (ISP)

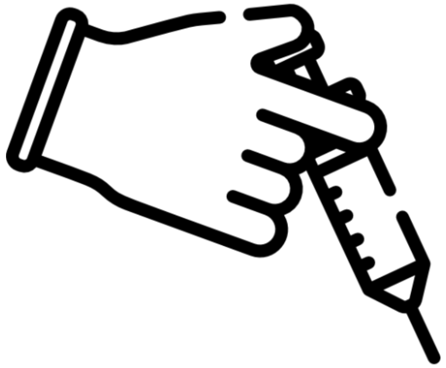
→ **Small interfaces** reduce the effort to **create *test doubles***.

Dependency Inversion Principle (DIP)

→ **Low coupling** makes it easier to **inject *test doubles***.

Carnegie Mellon University

Controllability Checklist



- Can you manipulate **configuration** settings easily during testing?
- Is it easy to inject test **inputs** and test **data** into your software?
- Is it easy to create and insert **test stubs**?
- Are **cyclic dependencies** minimized to allow isolated deployment?
- Are **simulators** or **emulators** available for environment behavior?

Observability Checklist

- ✓ Can all **component states** be accessed by your tests?
- ✓ Can you detect any **change** in the component state?
- ✓ Can you detect & read **messages** sent between components?
- ✓ Is it easy to create and insert ***test doubles***?
- ✓ Are **logs** generated for all critical events, errors, and warnings?

Carnegie Mellon University

How to **Test Quality Attributes?**

Designing Large-scale Software Systems – Design for Testability

Carnegie Mellon University

Recall – Quality Attribute Specifications

Scenario

Controllability

Measure

Observability

In-Class Activity: How can we *control* the scenario and *observe* the measure?

Testing Reliability

Specifying Reliability Requirements:

Scenario

1. The **functionality** that should be reliable
2. Considered deviations from normal conditions

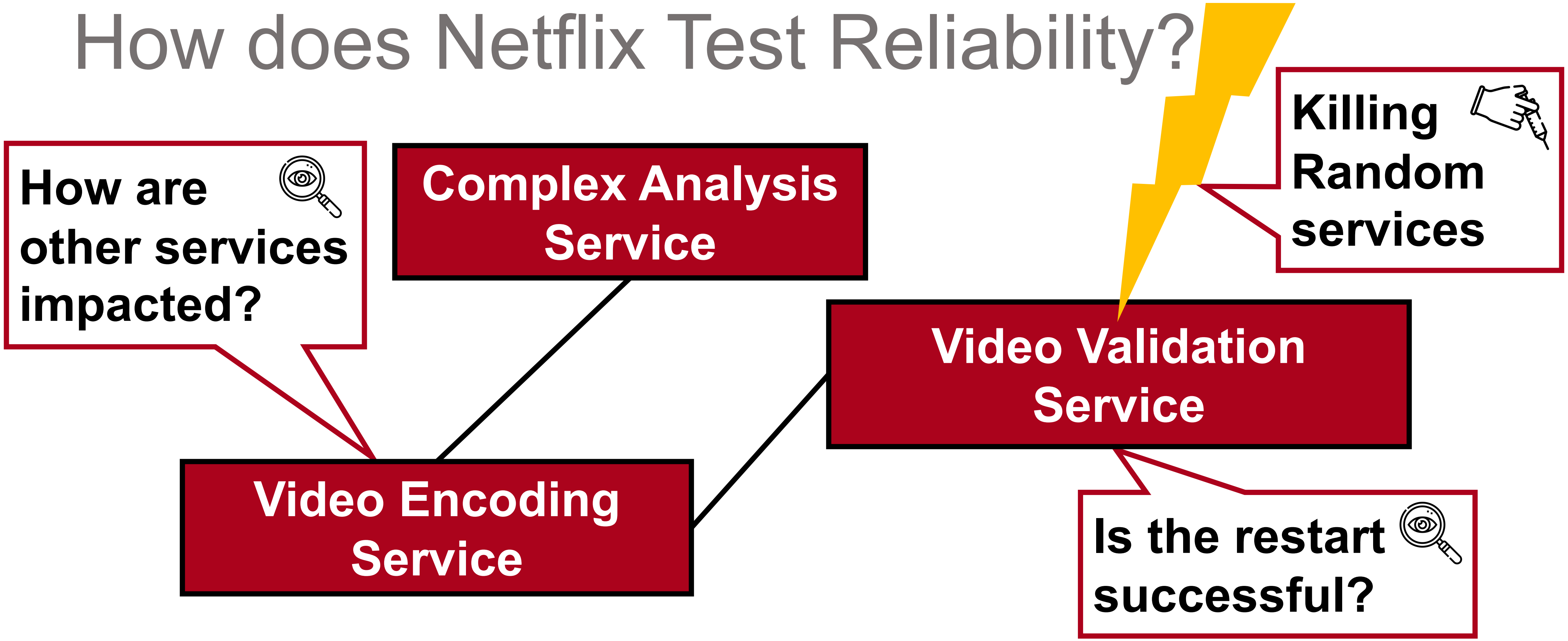
Measure

The percentage of deviations that preserve the functionality

Carnegie Mellon University

In-Class Activity: How can we generalize these insights beyond Netflix?

How does Netflix Test Reliability?



Carnegie Mellon University

Testing Reliability

Controllability

- **Test Stubs inject deviations**
  - Injecting faults
  - Simulating environment changes

Observability

- Check whether **functionality** is **preserved**
  - Functional assertions
  - **Ping/Echo** or **Heartbeat**

Testing Performance

<u>Specifying Performance Requirements:</u>

Scenario

1. Arrival of an **event** (e.g., request)
2. System's **response**

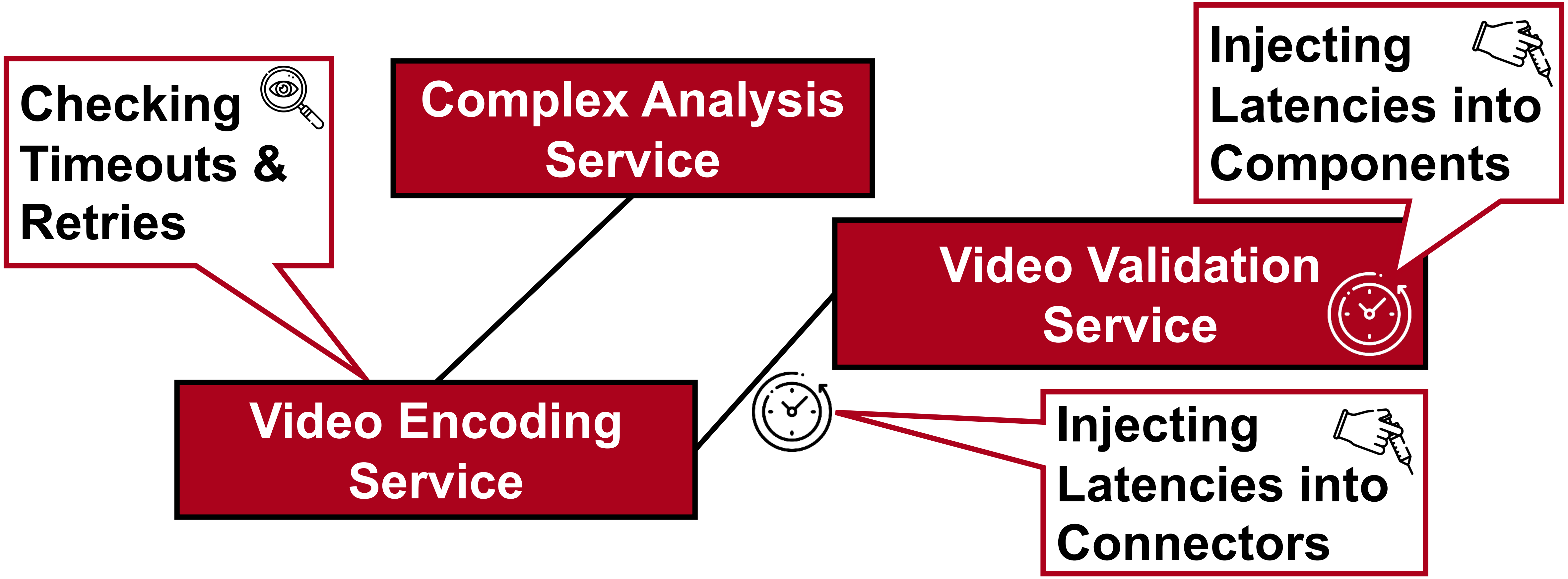
Measure

1. Average / minimum / maximum
2. Latency / deadline / throughput / jitter / miss rate

Carnegie Mellon University

In-Class Activity: How can we generalize these insights beyond Netflix?

How does Netflix Test Performance?


Checking Timeouts & Retries
Complex Analysis Service
Injecting Latencies into Components
Video Validation Service
Video Encoding Service
Injecting Latencies into Connectors

Carnegie Mellon University

Testing Performance

Controllability

- **Inject** the **request**
  - **Test Stubs** inject indirect requests
- **Test Stubs** inject latencies to analyze their impact
- **Stress Test**: Create high load

Observability

- **Measure execution times**
  - **Test Spies** measure latencies for **indirect outputs**
- Identify bottlenecks

Carnegie Mellon University

<u>In-Class Activity</u>: How can we *control* the scenario and *observe* the measure?

Testing Security

<u>Specifying Security Requirements:</u>

Scenario

1. The functionality that should be preserved
2. The type of attack

Measure

How does the system respond to the attack

(prevented, time to detect / repair, …)

Carnegie Mellon University

Testing Security

Controllability

- **Simulate the attack** (e.g., injection of malicious inputs, unauthorized access, …)

Observability

- Check whether functionality is preserved
- Measure detection / repair times

Carnegie Mellon University

In-Class Activity: How can we test availability without waiting for a long duration?

Testing Availability

Specifying Availability Requirements:

Scenario

1. The **functionality** that should be available
2. The **operating conditions**

Measure

Percent of **uptime** / time to repair / time to detect and/or recover from partial unavailability

Carnegie Mellon University

Testing Availability

Controllability

- Injecting faults
- Create **high-load situations**

Observability

- **Check when Components are Responsive**
  - **Ping/Echo** or **Heartbeat**
- **Extrapolate from data points**

Carnegie Mellon University

How to **Increase Test Coverage?**

Designing Large-scale Software Systems – Design for Testability

Carnegie Mellon University

Question: Why / when do we need unit tests?

Exhaustive Testing is Impossible
We need to **Find the Right Balance**

Testing Effort

Test **Critical Functionality** First

System- and and **Integration**-Tests cover a lot of code with less effort

Confidence

Complex Parts need more tests

Confidence requires **More Assertions** rather than just covering more lines of code

Carnegie Mellon University

Question: Why can't we have ONLY monkey tests?

Monkey Testing / Random Testing

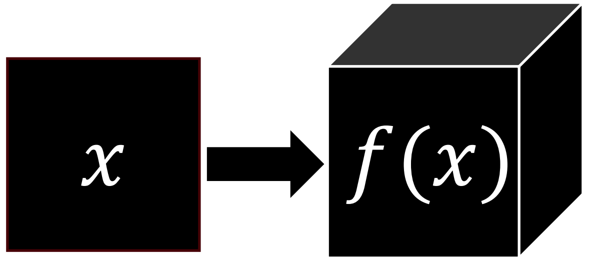
- **Problem:** Specifying many input-output relationships is **too costly**
- **Context:** A good foundation of **traditional tests** exist

Randomly Trigger Possible System Events

Sample from the whole input space, try breaking the system, avoid repetition

Check for Crashes and Undesired States

Assertions in the code, monitoring component states, observe long latencies



Metamorphic Testing

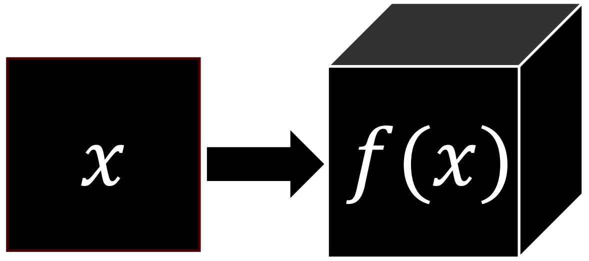
- **Problem:** Specifying many input-output relationships is **too costly**
- **Context:** A good foundation of **traditional tests** exist

Test Random Inputs for the CUT

Sample from the component's whole input space, avoid repetition

Verify Metamorphic Relations that should hold for Every Input

Essential properties of the provided functionality



Examples of Metamorphic Relations

Math App

$\sin(\pi - x) = \sin(x)$

$\min(a, b) = \min(b, a)$

Financial App

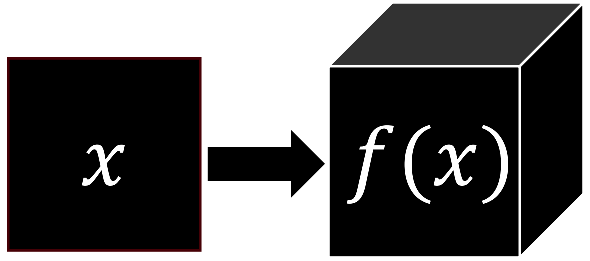
$\text{tax}(\text{income}) < \text{tax}(\text{income} + 1000)$

$x = \text{USDtoEURO}(\text{EUROtoUSD}(x))$

Computer Vision Component

$\text{objRecognition}(\text{img}) = \text{objRecognition}(\text{img} + \text{minorNoise})$

Carnegie Mellon University



Examples of Metamorphic Relations

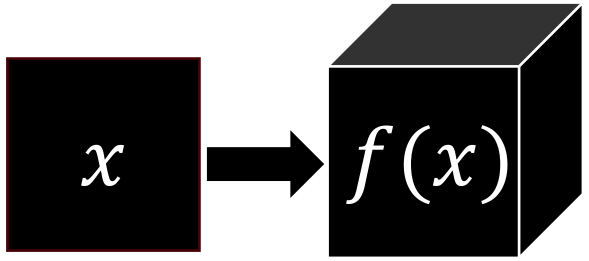
Interactive Applications

User **changes** the numbers in the table ⇒ numbers in other **views change**

Online Shops

Filtering by the price range or star rating returns a **subset** of the previous list

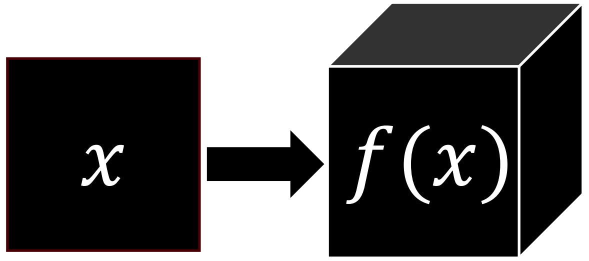
In-Class Activity: Describe Metamorphic Relations that might be observed in Netflix



Metamorphic Relations In Web Apps

- Two searches for films with the same query should return the same results regardless of the user profile (order might vary)
- After a user completed watching a movie it should not appear in their recommendations anymore

Question: Why can't we have ONLY metamorphic tests?



Metamorphic Testing

- **Problem:** Specifying many input-output relationships is **too costly**
- **Context:** A good foundation of **traditional tests** exist

Test Random Inputs for the CUT

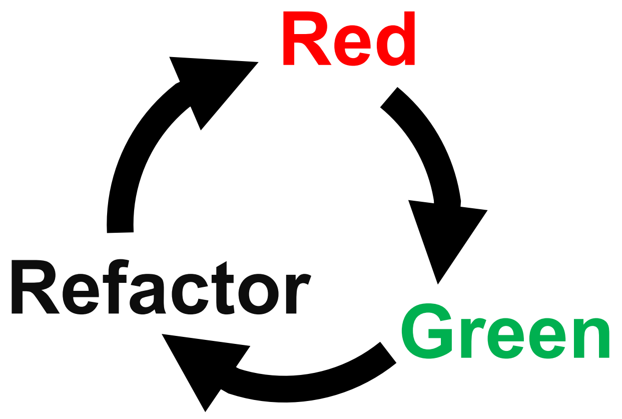
Sample from the component's whole input space, avoid repetition

Verify Metamorphic Relations that should hold for Every Input

Essential properties of the provided functionality

Carnegie Mellon University

Question: Why is it not enough to have **only unit tests**?



Test-First Programming / **Test-Driven Development (TDD)**

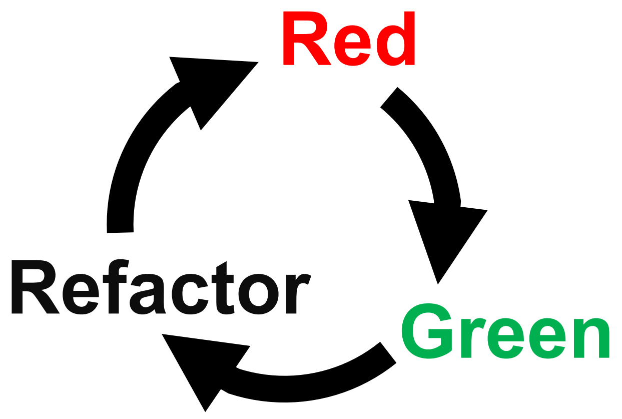
Red	For your new requirement write a **small test that fails**, and perhaps doesn't even compile at first
Green	Make the **test pass** with **minimal coding effort**, potentially using simplifying shortcuts in the process
Refactor	Make the design more **elegant,** cleaner, and potentially **faster** while **keeping the functionality**

More in on this in "*Test Driven Development: By Example*" by Kent Beck

Carnegie Mellon University

Question: (How) does this work for **quality attribute testing**?

Write Tests Before Implementation!
Test-Driven Development (TDD)



- Guarantees **testability** and very **high coverage** of unit tests
- Leads to more **modular** design due to focus on loosely coupled design
- Finding bugs earlier **saves time**
- Helps to keep **focused** on the current task
- Iterative approach does not work well for **extremely complex behavior**

Carnegie Mellon University

How to **Tailor Testing to Different Domains?**

Designing Large-scale Software Systems – Design for Testability

Question: What challenges with Controllability & Observability do we face?

Case Study: Web Apps

Controllability

How to **simulate user input** (e.g., clicking buttons, entering text, waiting for page to load, …)?

Observability

How to **verify the output** (e.g., text on the webpage, element is visible, …)?

Carnegie Mellon University

Controllability

End-to-end Web Testing Frameworks

```
await page.goto('https://playwright.dev/');

await page.getByRole('textbox').fill('example
value');

const getStarted = page.getByRole('link',
{ name: 'Submit' });
await getStarted.click();
```

Page Navigation

Entering Text

Clicking a Link

See more detailed here: https://playwright.dev/docs/writing-tests

Carnegie
Mellon
University

Observability

End-to-end Web Testing Frameworks

```
await expect(page.getByText('Welcome')).toBeVisible();

// At least one of the two elements is visible, possibly both.
await expect(
page.getByRole('button', { name: 'Sign in' })
.or(page.getByRole('button', { name: 'Sign up' }))
.first()
).toBeVisible();

const locator = page.locator('.title');
await expect(locator).toContainText('substring');
await expect(locator).toContainText(/\d messages/);
```

Assertion for Visibility

Assertion for Text Content

See more detailed here: https://playwright.dev/docs/writing-tests

Question: What quality attributes do we want to test? What challenges with Controllability & Observability do we face?

Case Study: Mars Helicopter

Carnegie Mellon University

Case Study: Mars Helicopter

Controllability

How to simulate **lower gravity**?

How to simulate **thinner atmosphere** and different atmospheric **composition**?

Carnegie Mellon University

Testing Robotics Systems

- Simulation can find some bugs, but is often not enough
- Huge space of potential inputs and environment conditions
- Stubbing computer vision components is especially challenging
- Record & replay of events can help minimize testing effort

Testing Mobile Apps

- Monkey testing is very popular
- Android has higher **device-heterogeneity**, leading to challenges with **controllability** of code that depends on hardware (e.g., GPU)
- Simulators are available to test software off-device
- Google offers Cloud Testing on actual devices

How does **Testability** Relate to **Changeability**?

High Changeability Leads to High Testability

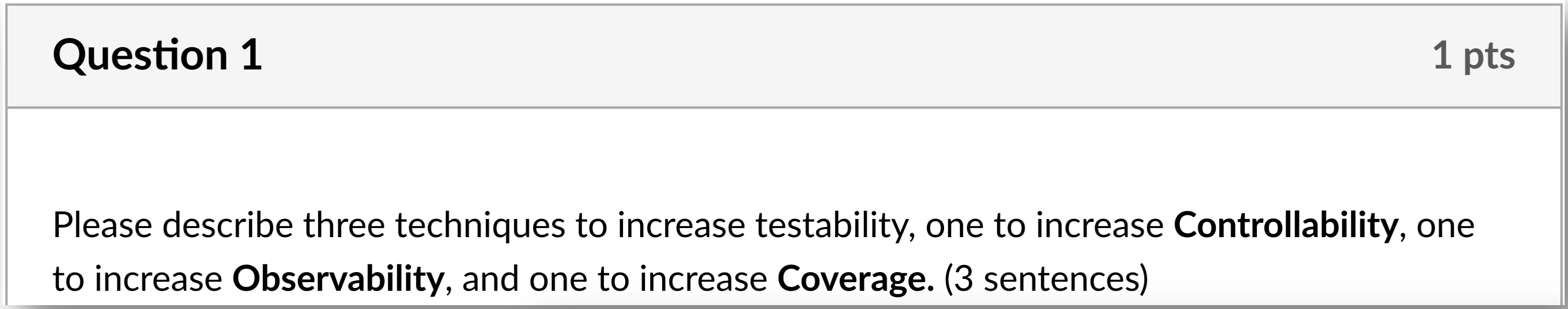
Modular design makes it easier to write tests, due to **fewer dependencies**, **simpler interfaces**, and better support for ***test doubles***.

High Testability Leads to High Changeability

Having many and good tests makes it easier to **change code without fearing** to introduce **bugs**

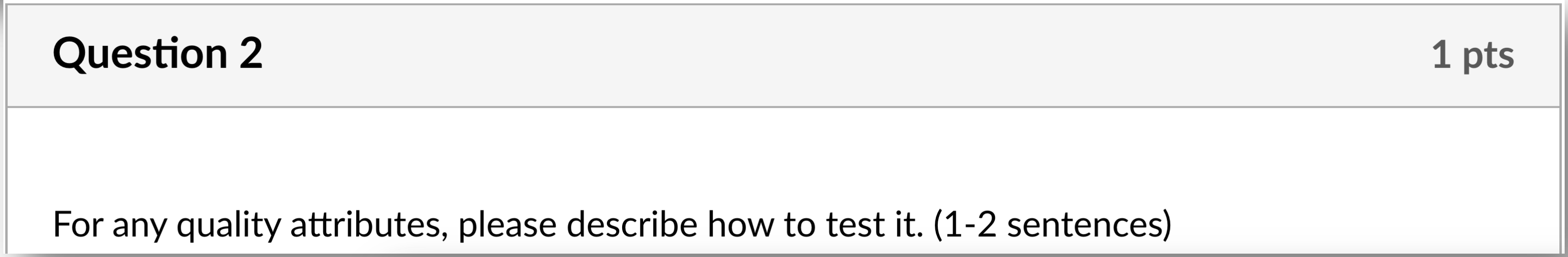
Carnegie Mellon University

Please Complete the Exit Ticket in Canvas!



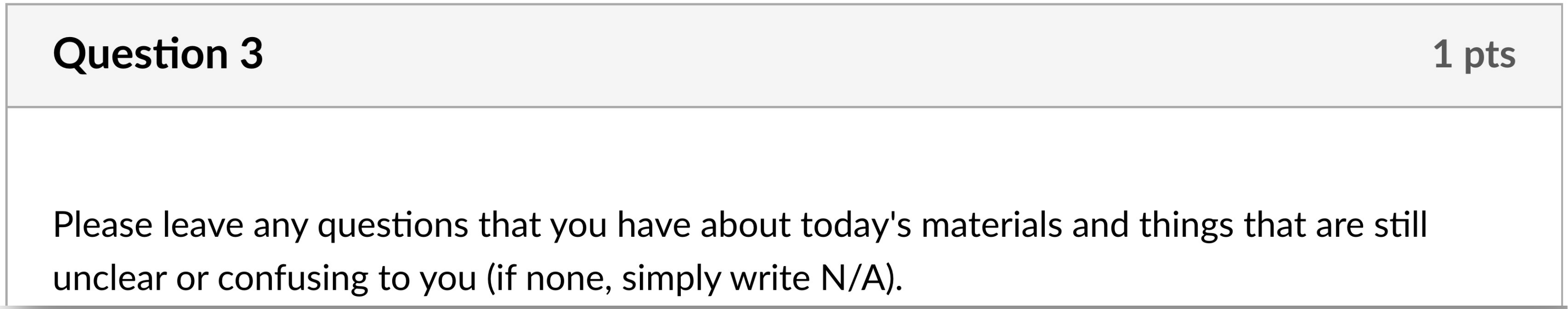
Question 1 1 pts

Please describe three techniques to increase testability, one to increase **Controllability**, one to increase **Observability**, and one to increase **Coverage.** (3 sentences)



Question 2 1 pts

For any quality attributes, please describe how to test it. (1-2 sentences)

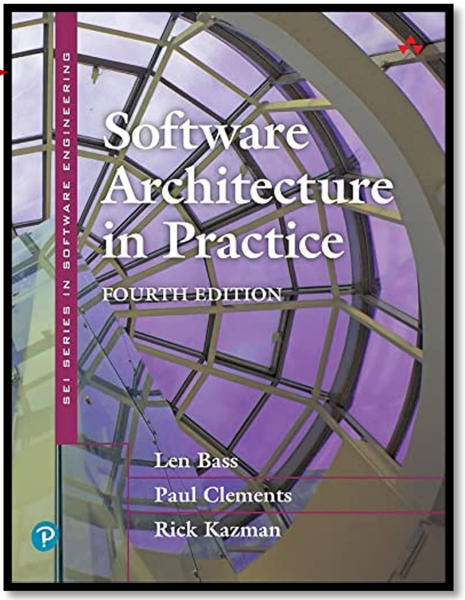


Question 3 1 pts

Please leave any questions that you have about today's materials and things that are still unclear or confusing to you (if none, simply write N/A).

Carnegie Mellon University

More on testability in "Software Architecture in Practice" chapter 12


SEI SERIES IN SOFTWARE ENGINEERING
Software Architecture in Practice
FOURTH EDITION
Len Bass
Paul Clements
Rick Kazman

Summary

- SOLID principle help to design easier testable software
- Controllability can be increased via *Test Stubs*
- Observability can be increased via *Mock Components and Test Spies*
- Coverage can be increased via Monkey Testing and Metamorphic Testing
- TDD helps to reach high coverage of unit tests while creating modular software

Credits: These slide use images from Flaticon.com (Creators: Freepik, kliwir-art, leremy)