Scaffolding in software testing refers to **all the temporary code and structures** required to enable a specific module of a system to be tested in isolation.[1]

It is the supporting framework—composed of **Drivers**, **Stubs**, and **Mocks**—that allows a testing tool to provide inputs to the code being tested and capture the outputs, while replacing all external dependencies.

🏗️ What is Scaffolding in Testing?

Scaffolding is any code written **for testing purposes only** that is not part of the final application's business logic.[2] Its primary goal is to **isolate** the **Module Under Test (MUT)** from the rest of the system to ensure that test failures are caused by bugs in the MUT, not in its dependencies.[3]

Component	Role in Scaffolding
Driver	The code that **calls** the MUT, passing in test data and initiating the execution.
Stub/Mock/Fake	The temporary code that **replaces** external dependencies (like database calls, network requests, or other modules) that the MUT relies on.
Oracle	The **assertion logic** that compares the actual output of the MUT to the expected result.

In this context, **scaffolding** is the blanket term for the collection of all these pieces (the test runner, the test methods, the drivers, and the stubs) that collectively create the testing environment.

🛠️ How to Create Scaffolding for a System

Creating scaffolding is the process of setting up the environment for unit or integration testing.[4] This process is generally automated using testing frameworks, but the underlying steps involve replacing dependencies and writing test execution logic.

1. Identify Dependencies and Isolate the MUT

The first step is to identify the **Module Under Test (MUT)** and all the other classes, functions, or services it calls.

- **Example MUT:** A class method UserService.createUser(data).

- **Dependencies:** The DatabaseRepository.save(user) method, and the EmailService.sendWelcome(user) method.

2. Implement Test Doubles (Stubs/Mocks/Fakes)

For each dependency identified in Step 1, a **Test Double** (a generic term for Stubs, Mocks, and Fakes) must be created to control the dependency's behavior. This ensures the test is fast and predictable, and doesn't rely on external factors like an actual database or email server being available.

Dependency	Test Double Type	Action
DatabaseRepository.save()	**Stub/Mock**	Implement a substitute method that does **nothing** (i.e., doesn't hit the DB) but returns a **fixed success value** (e.g., True or a mock object).
EmailService.sendWelcome()	**Mock**	Implement a substitute that allows the test to **verify** if this method was called *exactly once* with the correct parameters, without actually sending an email.

3. Write the Test Driver and Oracle

The test driver is the actual test method that orchestrates the execution.[5]

- **Setup:** Use the testing framework's tools to **inject** the Stubs/Mocks into the MUT, replacing the real dependencies.
- **Execution (Driver):** Call the MUT method (UserService.createUser(testData)).
- **Verification (Oracle):**
    - Assert that the MUT returned the **expected value**.
    - Assert that the Mocks were **called correctly** (e.g., check that EmailService.sendWelcome() was called with the new user object).

Many modern frameworks, such as JUnit (Java), Jest (JavaScript), or the built-in unittest module in Python, automate much of this scaffolding with specialized syntax for creating and injecting mocks, making the process cleaner.

Creating scaffolding for unit tests involves setting up the **temporary structure** that allows developers to isolate and test individual components of their code, as explained in this video about software development and code automation.[6] What Is Scaffolding In Software Development?