

# UEFI & EDK II Training

Continuous Integration (CI) Unit Test Framework for  
Developer Validation

[tianocore.org](https://tianocore.org)

See also [LabGuide.md](#) for Copy & Paste examples in labs

# LESSON OBJECTIVE

- ✿ What is the Unit Test Framework and how does it work?
- ✿ How is Unit Test code included with Continuous Integration (CI)?
- ✿ Steps for adding a unit test with the Unit Test Framework

# WHAT IS THE UNIT TEST FRAMEWORK?

How does the Unit Test Framework work?

# EDK II Open Source Unit Test Framework

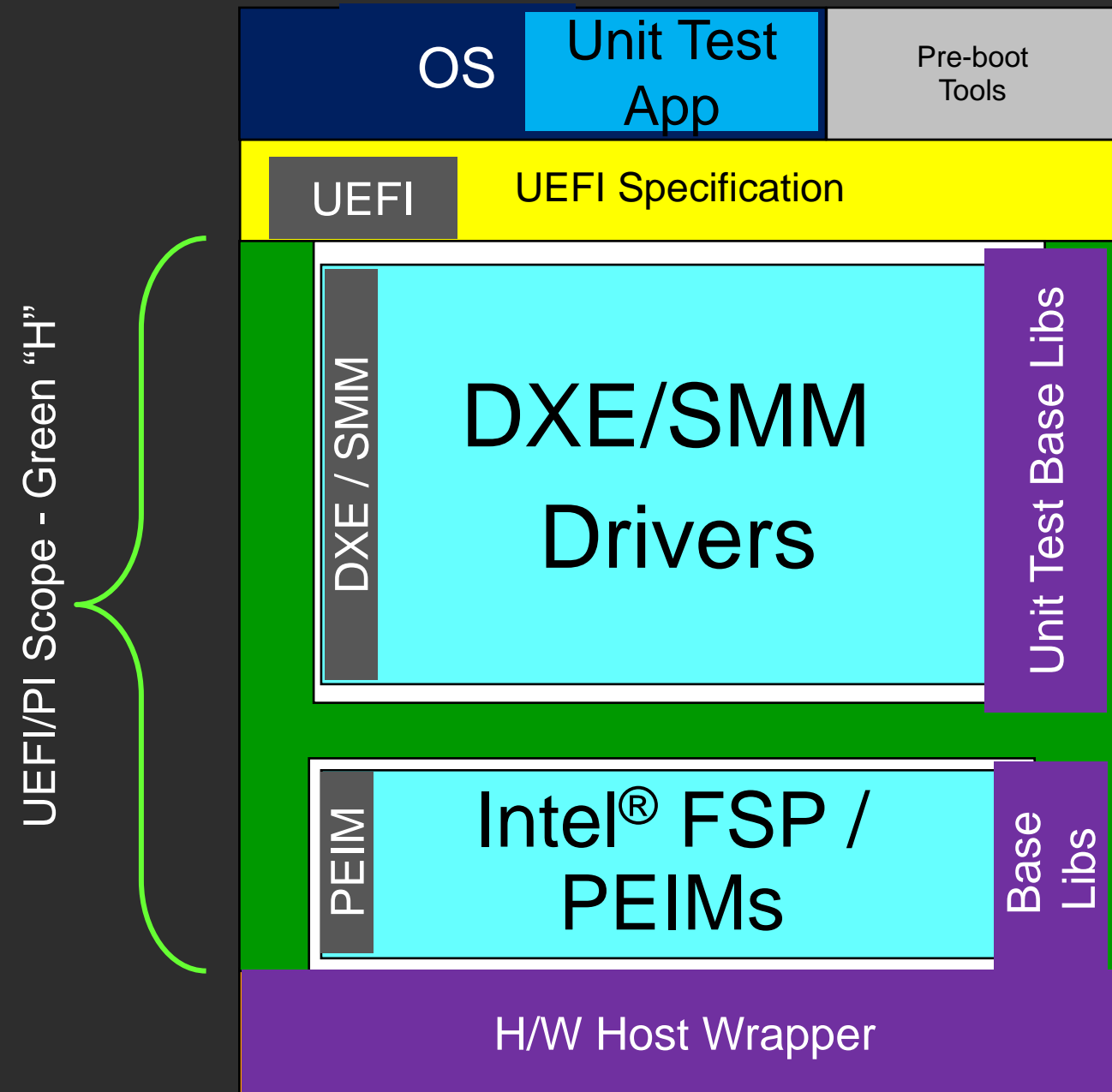
## Overview

- Prior Efforts
  - Component of Host Based Firmware Analyzer ([HBFA](#)) (open sourced by Intel mid 2019)
  - Microsoft UEFI Shell only unit test solution
- Collaboration between Intel and Microsoft\*
- Future expansion plans to include Code Coverage ([Gcov](#))



## Benefits




- Supports Host Environments for Developers & CI Agents
- Supports Target Environments (PEI, DXE, SMM, UEFI Shell)
- Focused on low level testing of interfaces, libraries, and modules
- Includes [cmocka](#) to support mocked interfaces
- Generates standard XML [JUnit](#) reports

# EDK II Unit Test Framework

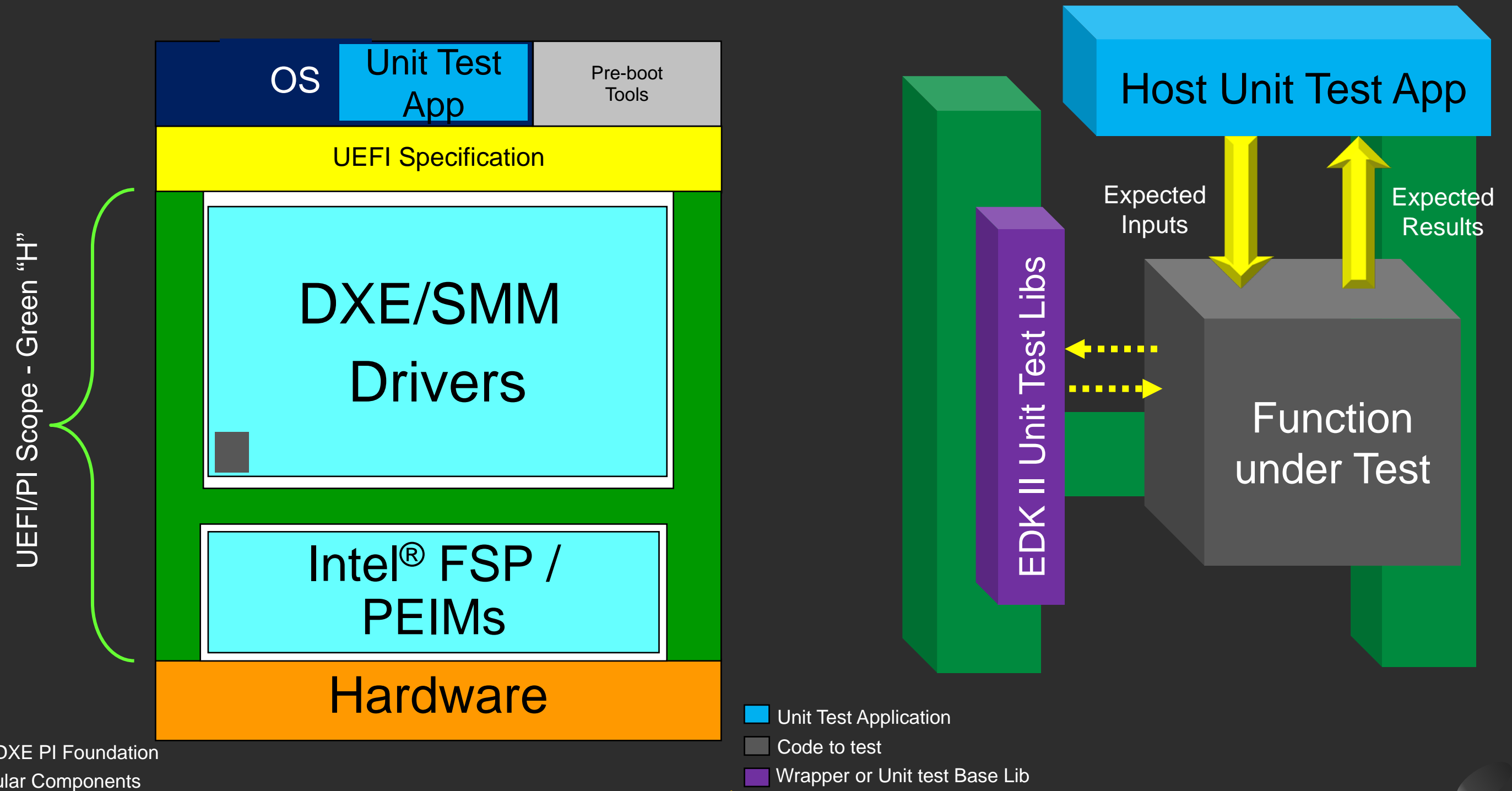


- Unit Test Framework Template for Host, DXE, SMM and PEI testing
- Unit Test Framework wraps code around interfaces
- Unit Test Base Libs used to interface against real hardware
- Host and Target based unit testing available

 PEI/DXE PI Foundation  
 Modular Components

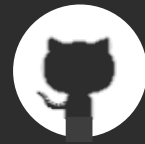
 Unit Test Application  
 Code to test  
 Wrapper or Unit test Base Lib

# EDK II Unit Test Framework



# Where is the Unit Test Framework Source?

## GITHUB



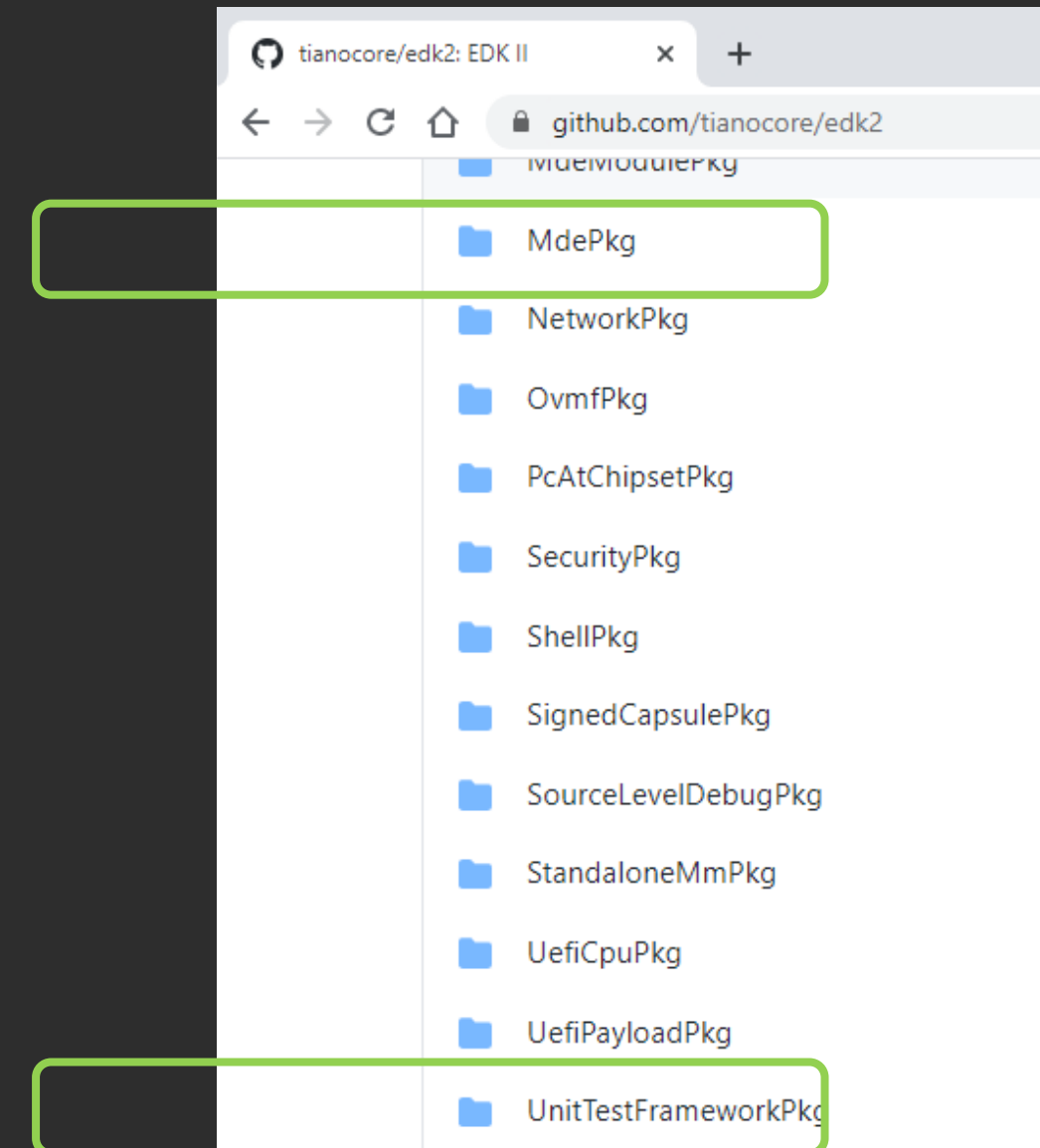
EDK II Repo - <https://github.com/tianocore/edk2>

Library Class to include:

[MdePkg/Include/Library/UnitTestLib.h](#)

Unit Test Framework Package:

[UnitTestFrameworkPkg](#)



# Directories and Files

## EDK II Open Source Unit Test Framework

```
MdePkg/Include/Library/UnitTestLib.h
UnitTestFrameworkPkg/
  PrivateInclude/
  Library/
    CmockaLib/
    Posix/
      DebugLibPosix/
      MemoryAllocationLibPosix/
    UnitTestBootLibNull/
    UnitTestBootLibUsbClass/
    UnitTestDebugAssertLib/
    UnitTestLib/
    UnitTestPersistenceLibNull/
    UnitTestPersistenceLibSimpleFileSystem/
    UnitTestResultReportLib/
```

- ← Library to include for framework interfaces
- ← Package to include for definitions
- ← Private Include

### Unit Test Framework Package Libraries

- ← Cmocka lib for cmocka functions
  - Posix Libraries for:
    - ← DebugLib
    - ← and memory allocation
- ← Null Library for Boot
- ← Boot Library for USB Class
- ← Replacement for DebugAssert() in DebugLib
- ← Unit Test Library
- ← Null Library for Persistence Test
- ← Library Persistence for Simple File System
- ← Library for Reporting Results



# How Does the Unit Test Framework Work?

The `UnitTestLib` Class provides functions for creating unit test suites and test cases for the functions under test.

Library Function	Description
<code>InitUnitTestFramework()</code>	A Unit Test Framework is registered
<code>CreateUnitTestSuite()</code>	A Unit Test Suite is registered
<code>AddTestCase()</code>	Each individual test case is added to the test suite
<code>RunAllTestSuites()</code>	Run all the tests in the test suite
<code>FreeUnitTestFramework()</code>	Free Registered test suite framework
<code>SaveFrameworkState()</code>	Save registered test suites through a reset

# How Does the Unit Testing Work?

- Incorporate changed code into the Unit Test Framework sample templates using the UnitTestLib Class
- The unit test must test paths in the changed code with a deterministic and expected desired result
- The unit test will return **UNIT\_TEST\_PASSED** for passing test.
- The unit test will return a failure from an **UT\_ASSERT\_...** test.

Assert tests returns failure on failed conditions:

- `UT_ASSERT_TRUE(Expression)`
- `UT_ASSERT_FALSE(Expression)`
- `UT_ASSERT_EQUAL(ValueA, ValueB)`
- `UT_ASSERT_MEM_EQUAL(BufferA, BufferB, Length)`
- `UT_ASSERT_NOT_EQUAL(ValueA, ValueB)`
- `UT_ASSERT_NOT_EFI_ERROR(Status)`
- `UT_ASSERT_STATUS_EQUAL(Status, Expected)`
- `UT_ASSERT_NOT_NULL(Pointer)`

# Example: Super Simple Function to Unit Test

Sample function to test:

```
UINTN NumberPlusOne(  
    IN UINTN Number  
)  
{  
    UINTN ReturnNumber = 0;  
    if (Number < 100){  
        ReturnNumber = Number + 1;  
    }  
    return (ReturnNumber); // result  
}
```

Determine the “Correct” results to add test cases

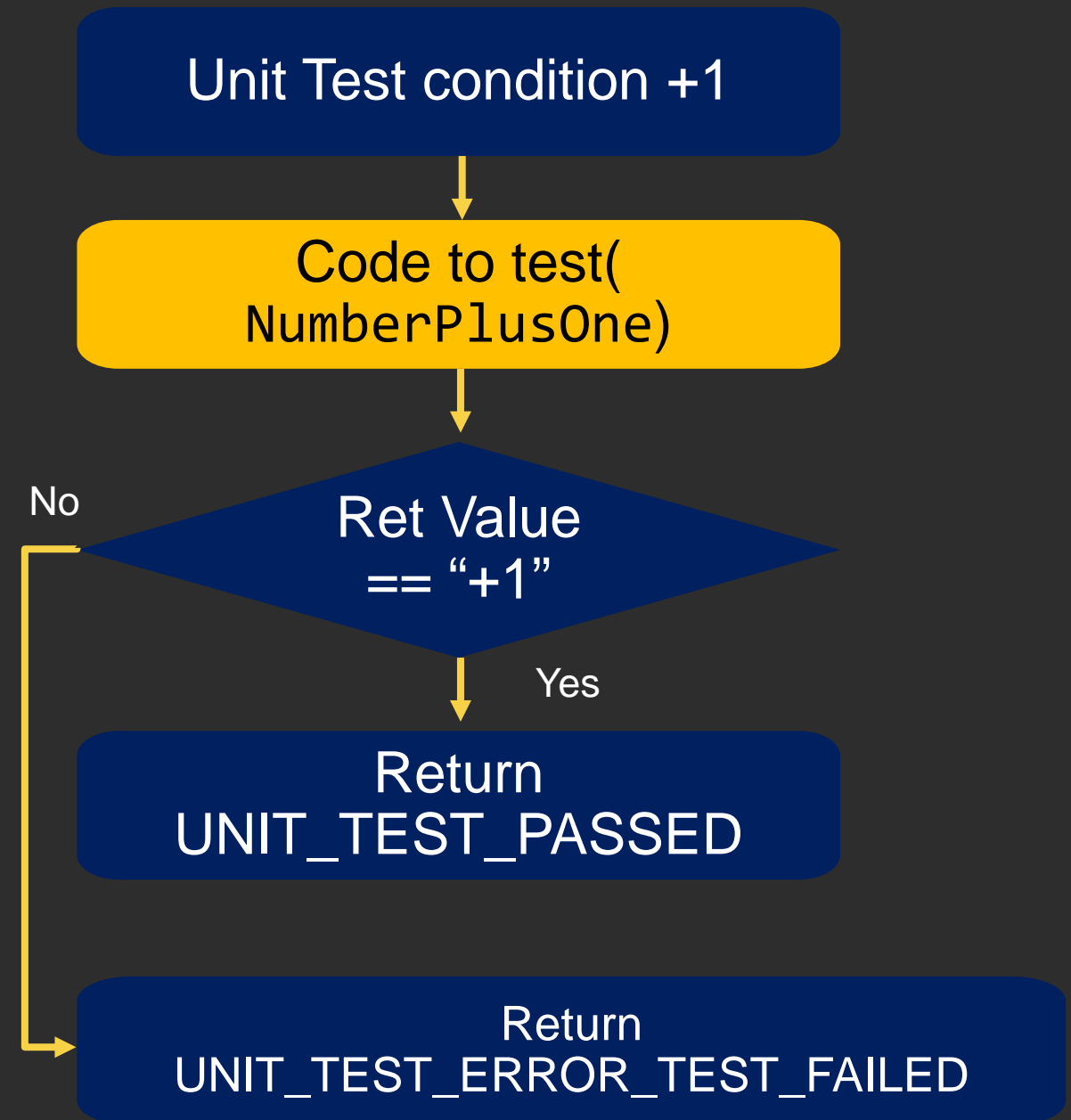
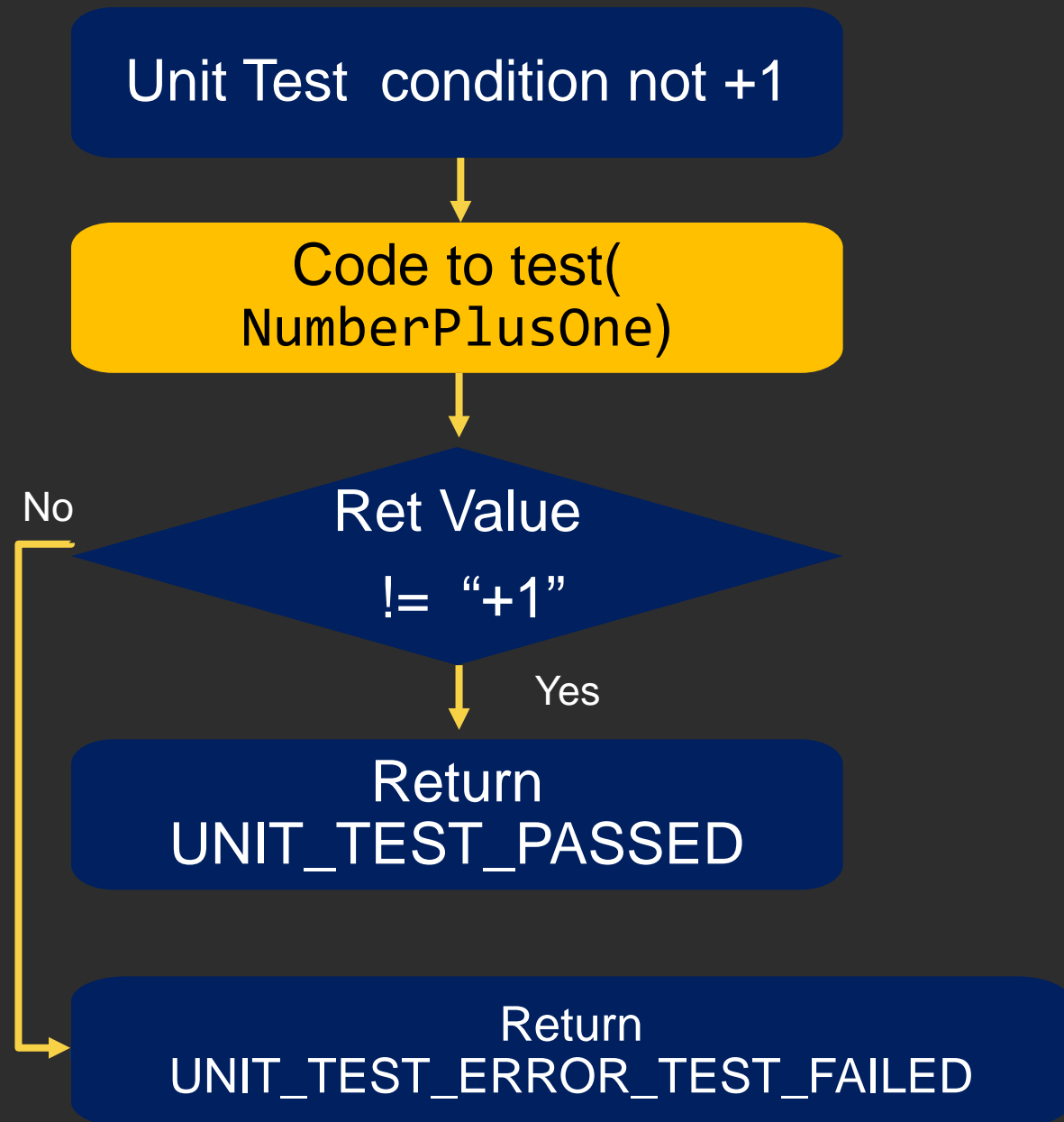
Observations:

1. If we pass a value of less than “100” the return value should be Number +1.
2. If we pass a value greater or equal than “100” then the return will be “0”

Result:

Two test cases can be done to unit test this function

# Unit Tests Determined Flow for All Results



# Example Test Case Functions

```
UNIT_TEST_STATUS
EFIAPI
IsNumberNotPlus1(
IN UNIT_TEST_CONTEXT Context
)
{
    UINTN RetNumber;

    RetNumber = NumberPlusOne(100);
    UT_ASSERT_EQUAL (RetNumber, 0);
    return UNIT_TEST_PASSED;
}
```

```
UNIT_TEST_STATUS
EFIAPI
IsNumberPlus1(
IN UNIT_TEST_CONTEXT Context
)
{
    UINTN RetNumber;

    RetNumber = NumberPlusOne(1);
    UT_ASSERT_EQUAL (RetNumber, 2);
    return UNIT_TEST_PASSED;
}
```

# Example Test Case Function Combined

```
UNIT_TEST_STATUS
EFIAPI
IsNumberCorrect(
IN UNIT_TEST_CONTEXT Context
)
{
    UINTN RetNumber;
    // Check for != +1 or 0 condition
    RetNumber = NumberPlusOne(100);
    UT_ASSERT_EQUAL (RetNumber, 0);

    // Check for is 2 condition
    RetNumber = NumberPlusOne(1);
    UT_ASSERT_EQUAL (RetNumber, 2);

    return UNIT_TEST_PASSED;
}
```

This test case combined both example deterministic results into one test case

If either result fails, the UT\_ASSERT will return a failure on the testing.

# Example Output From Stuart CI Build

In your immediate output, any build failures will be highlighted. You can see these here as "WARNING" and "ERROR" messages

```
SECTION - Building UnitTestFrameworkPkg Package
PROGRESS - --Running UnitTestFrameworkPkg: Host Unit Test . . .NOOPT --
PROGRESS - Start time: 2020-08-07 10:22:59.833725
PROGRESS - Setting up the Environment
PROGRESS - Running Pre Build
PROGRESS - Running Build NOOPT
PROGRESS - Running Post Build
SECTION - Run Host based Unit Tests
SUBSECTION - Testing for architecture: X64
WARNING - CheckNumberPlusOneUnitTestHost.exe Test Failed
WARNING - Test Description - UT_ASSERT_EQUAL(RetNumber:2, 1:1)
<filepath/CheckNumberPlusOneUnitTest.c:106: error: Failure!
ERROR - Plugin Failed: Host-Based Unit Test Runner returned 1
CRITICAL - Post Build failed
PROGRESS - End time: 2020-08-07 10:23:13.86 Total time Elapsed: 0:00:14
ERROR - --->Test Failed: Host Unit Test Compiler Plugin NOOPT returned 1
ERROR - Overall Build Status: Error
PROGRESS - There were 1 failures out of 1 attempts
SECTION - Summary
ERROR - Error
```

# Example Output From Stuart CI Build - Passing

Example with all  
unit tests  
passing

```
SECTION - Init SDE
SECTION - Loading Plugins
SECTION - Start Invocable Tool
SECTION - Getting Environment
SECTION - Loading plugins
SECTION - Building UnitTestFrameworkPkg Package
PROGRESS - --Running UnitTestFrameworkPkg: Host Unit Test Plugin NOOPT --
PROGRESS - Start time: 2020-08-07 12:42:01.690900
PROGRESS - Setting up the Environment
PROGRESS - Running Pre Build
PROGRESS - Running Build NOOPT
PROGRESS - Running Post Build
SECTION - Run Host based Unit Tests
SUBSECTION - Testing for architecture: X64
PROGRESS - End time: 2020-08-07 12:42:14.190751 Total time Elapsed: 0:00:12
PROGRESS - --->Test Success: Host Unit Test Compiler Plugin NOOPT
PROGRESS - Overall Build Status: Success
SECTION - Summary
PROGRESS - Success
```



# Example Test Result Output

The host application can be run manually to get more details

Example: Output Running All 3 Unit Tests:

Start with “[ RUN ]”

If Test Pass end with “[ OK]”

```
Sample Check Number Plus One Unit Test v0.1
```

```
-----  
----- RUNNING ALL TEST SUITES -----  
-----
```

```
-----  
RUNNING TEST SUITE: Simple Number Tests  
-----
```

```
[=====] Running 3 test(s).
```

```
[ RUN ] Test IsNumberPlus1 - produce 1+1=2
```

```
[ OK ] Test IsNumberPlus1 - produce 1+1=2
```

```
[ RUN ] Test IsNumberNotPlus1 - Not produce number +1
```

```
[ OK ] Test IsNumberNotPlus1 - Not produce number +1
```

```
[ RUN ] Test IsNumberCorrect w/ results in one test
```

```
[ OK ] Test IsNumberCorrect w/ results in one test
```

```
[=====] 3 test(s) run.
```

```
[ PASSED ] 3 test(s).
```

# Unit Test Framework Logging

## UnitTestLib Class Log Output

```
UT_LOG_ERROR(Format, ...)
```

```
UT_LOG_WARNING(Format, ...)
```

```
UT_LOG_INFO(Format, ...)
```

```
UT_LOG_VERBOSE(Format, ...)
```

Log Description records current executing test case

- ← ERROR message in the test framework
- ← Warning message in the test framework log
- ← Information message in the test framework log
- ← Verbose message in the test framework log

# Example Test Result Output with LOG Data

Output displays  
intermediate test  
information

```

Sample Check Number Plus One Unit Test v0.1
-----
-----          RUNNING ALL TEST SUITES          -----
-----
-----
RUNNING TEST SUITE: Simple Number Tests
-----
[=====] Running 3 test(s).
[ RUN      ] Test IsNumberPlus1 - produce 1+1=2
[      OK  ] Test IsNumberPlus1 - produce 1+1=2
[ RUN      ] Test IsNumberNotPlus1 - Not produce number +1
[      OK  ] Test IsNumberNotPlus1 - Not produce number +1
[ RUN      ] Test IsNumberCorrect w/ results in one test
UnitTest: Test-3 - Test IsNumberCorrect w/ results in one test
Log Output Start
[INFO]      Number is: 100 returned was (0)
[INFO]      Number is:  1 returned was (2)
Log Output End
[      OK  ] Test IsNumberCorrect w/ results in one test
[=====] 3 test(s) run.
[ PASSED  ] 3 test(s).

```

# Results From a Unit Test Run on Non-Host

```

-----
----- UNIT TEST FRAMEWORK RESULTS -----
-----
////////////////////////////////////
SUITE: Simple Number Tests
PACKAGE: Sample.Test
////////////////////////////////////
*****

CLASS NAME: Test-1
TEST:      Test IsNumberPlus1 - produce 1+1=2
STATUS:    PASSED
FAILURE:   NO FAILURE
FAILURE MESSAGE:

*****
*
*****

CLASS NAME: Test-2
TEST:      Test IsNumberNotPlus1 - Not produce number +1
STATUS:    PASSED
FAILURE:   NO FAILURE
FAILURE MESSAGE:

```

```

*****

CLASS NAME: Test-3
TEST:      Test IsNumberCorrect w/ results in one test
STATUS:    PASSED
FAILURE:   NO FAILURE
FAILURE MESSAGE:

LOG:
[INFO]      Number is: 100 returned was (0)
[INFO]      Number is: 1 returned was (2)
*****
*
+++++++
Suite Stats
Passed:  3  (100%)
Failed:  0  (0%)
Not Run: 0  (0%)
+++++++
=====
Total Stats
Passed:  3  (100%)
Failed:  0  (0%)
Not Run: 0  (0%)
=====

```

```

*****

```

```

=====

```

\*

Non-Host test results will go to the Debug Serial output or to the Console

# Samples of Unit Test Framework

Sample Unit Test

BaseSafeIntLib Unit Test

BaseLib Unit Test

DxeResetSystemLib Unit Test

## DSC Files to Build Sample Tests

- Host DSC file for Sample Unit Test
- Host DSC file for both BaseSafeIntLib and BaseLib Unit Test
- Host DSC file for DxeResetSystemLib Unit Test

Include Host DSC file for Host Unit Test Framework for CI

Include Target DSC file for including test framework into the Platform Package DSC file



# HOW IS UNIT TEST CODE INCLUDED WITH CI?

Tool rules for including unit test code to build & run with Stuart Continuous Integration

# Prerequisites for Stuart CI Build

- Windows 10:
  - Visual Studio VS2017 or VS2019
  - Windows SDK (for rc)
  - Windows WDK (for Capsules)
- Ubuntu 18.04 or Fedora
  - GCC5 or greater
- Python 3.7.x or greater on Path
- Git on Path

Download / Clone the required EDK II and other Repos

Example tianocore.org - edk2 :

```
$ git clone  
https://github.com/tianocore/edk2.git
```

# Preparation Before Building for CI locally

Install the pip requirements<sup>1</sup>

```
$ pip install --upgrade -r pip-requirements.txt
```

Get the code dependencies (done only when submodules change)

```
$ stuart_setup -c .pytool\CISettings.py TOOL_CHAIN_TAG=<Your TAG>
```

Update other dependencies (done on new command prompt)

```
$ stuart_update -c .pytool\CISettings.py TOOL_CHAIN_TAG=<Your TAG>
```

Build the BaseTools (done only when BaseTools change and first time)

```
$ python BaseTools\Edk2ToolsBuild.py -t <Your TAG>
```

Note that the "<Your tag>" is one of prerequisite compilers: VS2017, VS2019 or GCC5

<sup>1</sup>. May need proxy behind firewall ( --proxy <http://proxy-chain.intel.com:911>)



# Continuous Integration Host Module Test w/ PyTool – Build, Run & Verify Test Capabilities

## Inclusion Tests

Scans all INF files from a package for host based unit tests

All INF files must be listed in [Components] section of DSC where:

- MODULE\_TYPE = HOST\_APPLICATION
- Or Library instances that support HOST\_APPLICATION

## Compilation and Run Tests

Uses the Target = NOOPT

Include Host-based test in the Package YAML file:

**NamePkg.ci.yaml** contains:

```
"HostUnitTestCompilerPlugin": {  
  "DscPath": "Test/NamePkgHostTest.dsc"  
},
```

See [PyTool/Readme.md](#)

# Configuring the Host Based DSC

Any Host Based DSC for a package Pkg will be located:

```
<PackageName>Pkg/Test/<PackageName>PkgHostTest.dsc
```

To add automated host-based unit test building to a new package, create a similar DSC.

- The new DSC should make sure to have the NOOPT BUILD\_TARGET and should include the line:

```
!include UnitTestFrameworkPkg/UnitTestFrameworkPkgHost.dsc.inc
```

All of the modules that are included in the Components section of this DSC should be `MODULE_TYPE = HOST_APPLICATION`.

See [.pytool Readme.md Host Unit Test](#)

# Unit Test Location Layout Rules

Host - Library, Protocol, PPI,  
GUID Interface

- Scoped to the parent package  
MdePkg/Test/UnitTest/[interface<sup>1</sup>]/

Host - Library, Driver

- Scoped to the implementation directory itself in a UnitTest Dir  
<Package>Pkg/DriverXDxe/UnitTest

Host - Functionality or Feature

- Should be located in the package-level Tests dir under a HostFuncTest sub-dir  
<Package>Pkg/Test/HostFuncTest/Feature

Non-Host  
(PEI/DXE/SMM/Shell)

- Should be located with the package-level  
Pkg/Test/[Shell/Dxe/Smm/Pei]Test directory.

Link to Layout Rules : [UnitTestFrameworkPkg Readme.md](#)

<sup>1</sup> where [interface] is Library, Protocol, Ppi, or Guid

# Example Package Directory Tree

```

<PackageName>Pkg /
  ComponentY/
    ComponentY.inf
    ComponentY.c
    UnitTest /
      ComponentYHostUnitTest.inf
      ComponentYHostUnitTest.c
      Driver

  Library /
    . . .
    SpecificLibDxe
      UnitTest /
        SpecificLibDxeHostUnitTest.c
        SpecificLibDxeHostUnitTest.inf
      Library

  Test / # Host Based Test Apps
  
```

```

Test / # Cont.
  <Package>PkgHostTest.dsc
  UnitTest /
    InterfaceX /
      InterfaceXHostUnitTest.inf
      InterfaceXPeiUnitTest.inf
      InterfaceXDxeUnitTest.inf
      InterfaceXSmmUnitTest.inf
      InterfaceXShellUnitTest.inf
      InterfaceXUnitTest.c
      Host & Non-Host By INF

    GeneralPurposeLib /
      GeneralPurposeLibTest.c
      GeneralPurposeLibHostUnitTest.inf

  <Package>Pkg.dsc
  
```

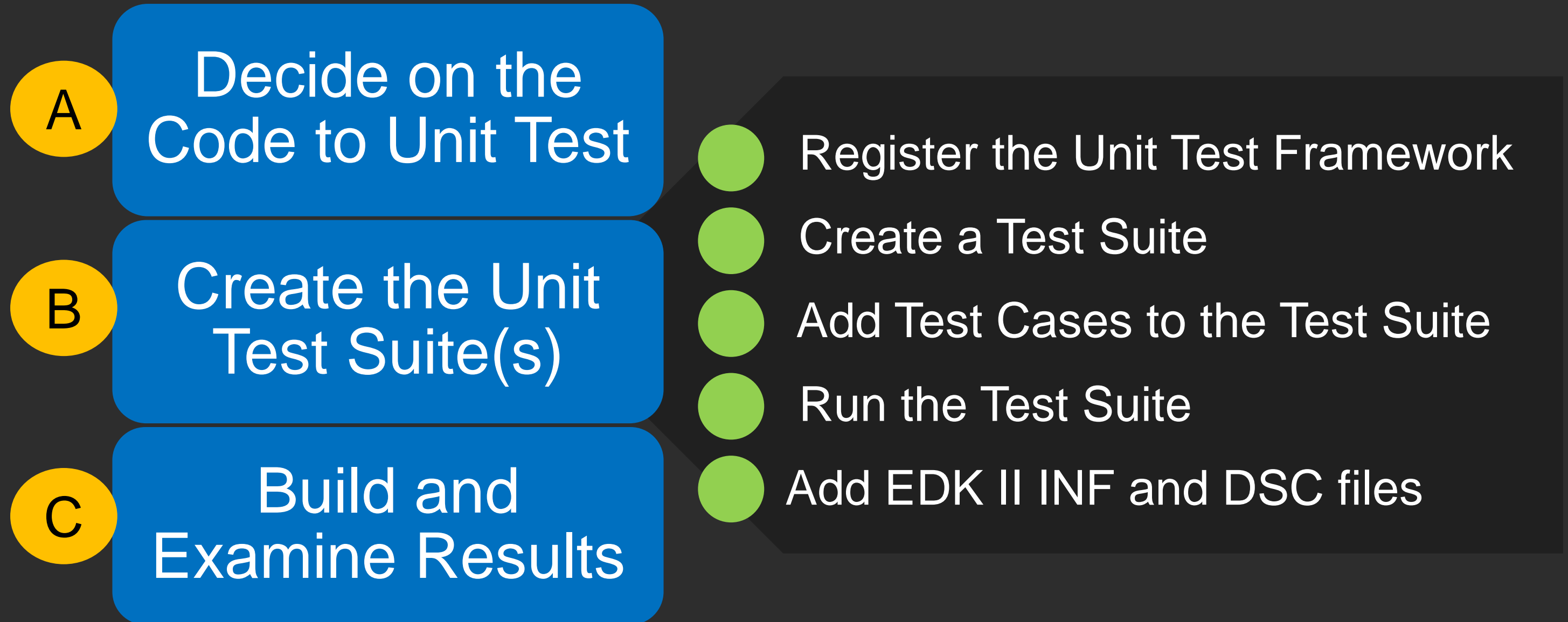
Feature

Host &  
Non-Host  
By INF

# STEPS FOR ADDING A UNIT TEST

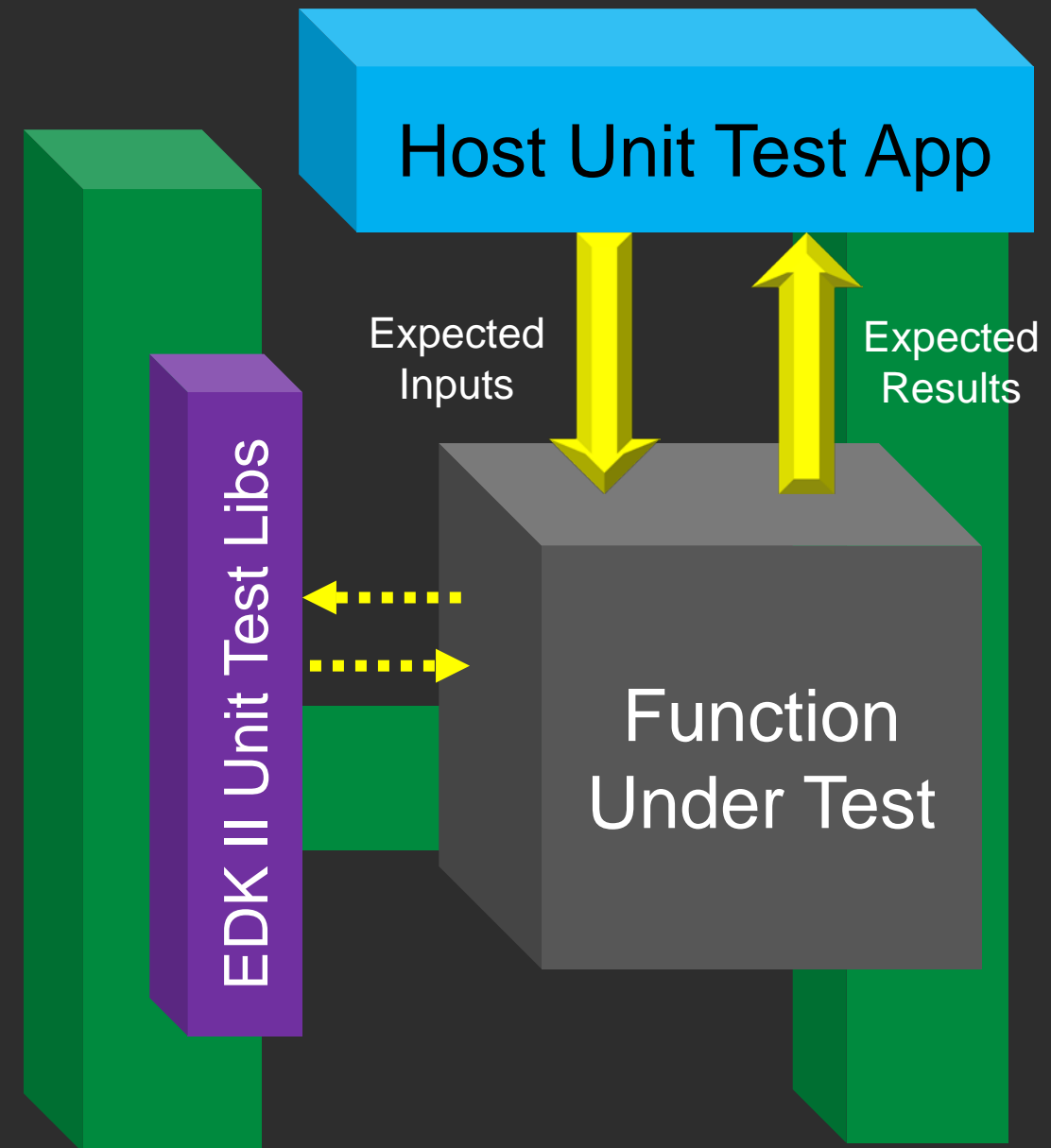
What are the steps for adding a unit test in the Unit Test Framework?

# Steps for Setting up Test Cases and Test Suites



# Decide on the Code to Unit Test A

- Function under test needs to work well with “White Box” testing  
Internal structure/ design/ implementation are known to the developer
- May need to break down test cases into reasonable test functions
- Test cases should validate all possible “good” & “bad” expected results
- Plans to add Code Coverage ([Gcov](#)) to help determine code paths
- Determine possibilities for Host-based Testing vs. testing on Target



# Host Versus Non-Host Based Test Cases

## Host

- Supports CI agent environments
- Independent of OS
- Supports Standard POSIX C
- Cmocka capabilities
- Set Target to “NOOPT”
- Specific Host DSC file include for Host Package DSC file for testing with CI

## Non-Host PEI, DXE, SMM, ...

- Pass/Fail Output logging to Debug Serial port
- Separate Entry Points for DXE or PEI or Other
- Only cursory build validation so far
- Specific Target DSC file include for Platform Package DSC file for Non-Host testing

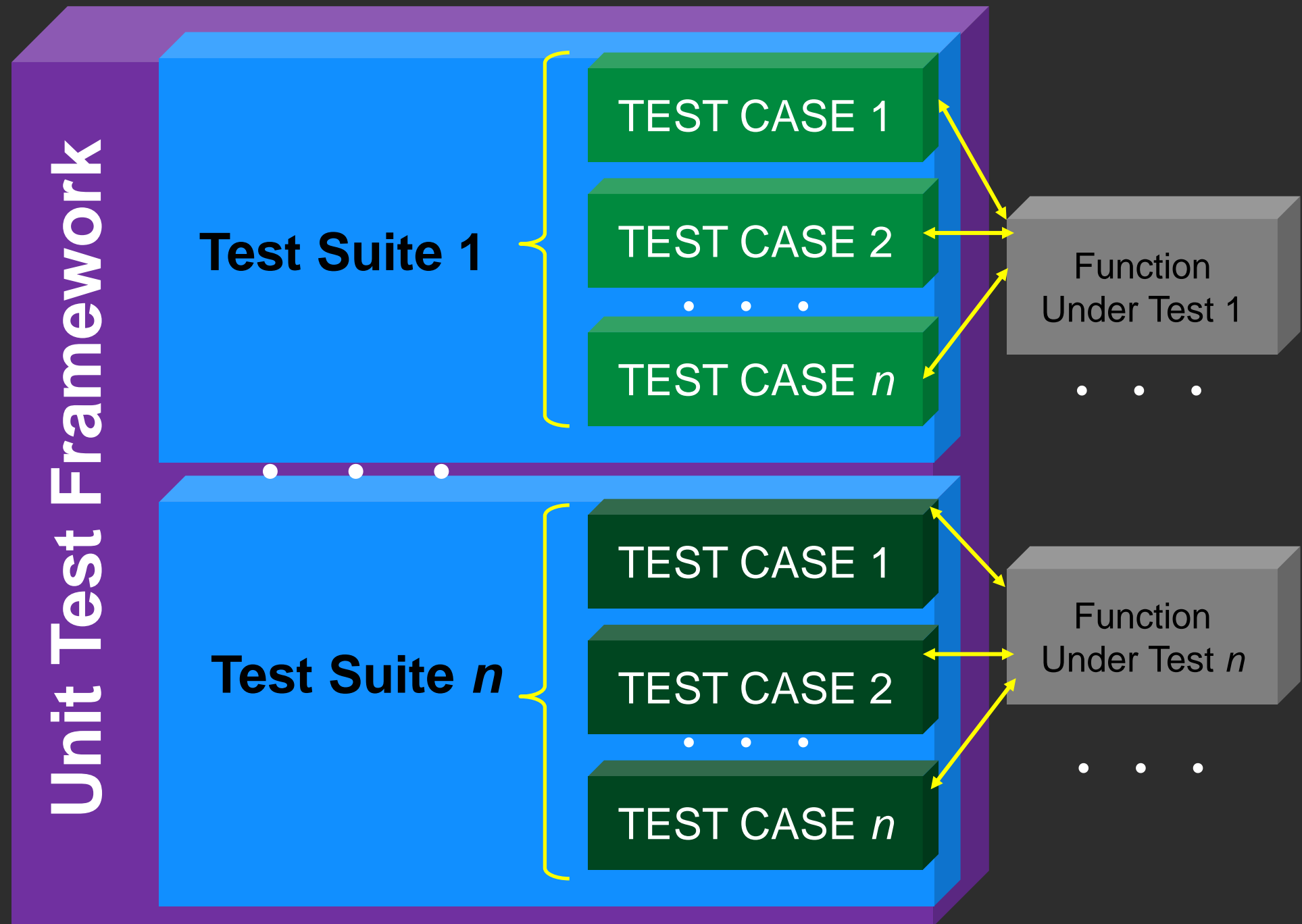
## Both

Requires a separate .INF for different environments (Host, DXE, DXE SMM & PEI)



# Create and Add a Unit Test Suite(s) B

- Register the Framework
- Create Test Suite 1
- Add Test Cases 1-  $n$
- Create Test Suite  $n$
- Add Test Cases 1-  $n$
- Run the Tests



# Register the Unit Test Framework 1

## Register with InitUnitTestFramework()

```
InitUnitTestFramework (&Framework, UNIT_TEST_NAME, gEfiCallerBaseName,  
UNIT_TEST_VERSION);
```

Where:

Framework	UNIT_TEST_FRAMEWORK_HANDLE
UNIT_TEST_NAME	Constant Define user defined string
gEfiCallerBaseName	Base name from EDK II build
UNIT_TEST_VERSION	Constant of user defined sting version number

This call will register the unit test framework structure (giving it a handle)

## Create a test suite with CreateUnitTestSuite()

```
CreateUnitTestSuite (&SuiteHandle, Framework, "Test Title", "Test.Name", NULL, NULL);
```

Where:

SuiteHandle	UNIT_TEST_SUITE_HANDLE
Framework	UNIT_TEST_FRAMEWORK_HANDLE (same as Init)
"Test Title"	User friendly ASCII String for Suite Title
"Test.Name"	User friendly ASCII String for Suite Name (no spaces)
NULL	Optional Setup Function runs before suite
NULL	Optional Teardown Function runs after suite

This call will register the test suite framework structure (giving it a handle)

## Create a test case with AddTestCase ()

```
AddTestCase (SuiteHandle, "Test Description", "short.name", UnitTestFunction, NULL, NULL, NULL);
```

### Where:

SuiteHandle	UNIT_TEST_SUITE_HANDLE (same as Create for Suite)
"Test Description"	User friendly ASCII String for Test Case Description
"Test.Name"	User friendly ASCII String for Test Case Name (no spaces)
UnitTestFunction	Unit Test Function
NULL	Optional Prerequisite Function runs before test function
NULL	Optional Clean-up Function runs after test function
NULL	UNIT_TEST_CONTEXT - Optional Pointer to Context

- This call will add a test case to the test suite framework structure
- Multiple Test cases can be added to the same test suite

# Example Test Case Function 3

The `UnitTestFunction` Shows that it will make a call to the Function Under Test with the expected results

Call to `UT_ASSERT_EQUAL` compares the Output Value returned with a known expected value

If the returned Output Value is not equal to the Expected Value, the test will return a failure.

```
UNIT_TEST_STATUS
EFIAPI
UnitTestFunction(
IN UNIT_TEST_CONTEXT Context
)
{
    // Local Variables InputParm, OutValue;

    FunctionUnderTest(InputParm, &OutValue)
    UT_ASSERT_EQUAL (OutValue,
EXPECTED_VALUE);
    return UNIT_TEST_PASSED;
}
```

Run the test all the test case suites with RunAllTestSuites ( )

```
Status = RunAllTestSuites (Framework);
```

Where:

Framework           UNIT\_TEST\_FRAMEWORK\_HANDLE (same as the InitUnitTestFramework )

This call will run all test cases for all test suites registered

# Initialize and Create Unit Test Suite “Main()”

```

EFI_STATUS
EFIAPI
UefiTestMain (
    VOID
)
{
    EFI_STATUS          Status;
    UNIT_TEST_FRAMEWORK_HANDLE Framework;
    UNIT_TEST_SUITE_HANDLE SuiteHandle;

    Framework = NULL;
    //Start setting up the test framework
    1 Status = InitUnitTestFramework (&Framework,
        UNIT_TEST_NAME, gEfiCallerBaseName,
        UNIT_TEST_VERSION);
    // Check if (EFI_ERROR (Status)) and goto EXIT on error

    // Populate the Unit Test Suite.
    2 Status = CreateUnitTestSuite (&SuiteHandle,
        Framework, "Test Suite Title", "Suite.Test",
        NULL, NULL);

    // Check if (EFI_ERROR (Status)) and goto EXIT on error

```

```

// Add test cases

    3 AddTestCase (SuiteHandle,
        " Test Description ",
        "Test.one", UnitTestFunction, NULL, NULL, NULL);

    // . . . Add more test cases to this test suite.

    // . . . Create more Unit Test Suite and
    // . . . Add more test cases to this test suite.

    // Execute the tests.
    4 Status = RunAllTestSuites (Framework);

EXIT:
    if (Framework) {
        FreeUnitTestFramework (Framework);
    }
    return Status;
}

```

# Example of Unit Test “C” file #Include & #define Statements

```
/** @file  
File Header  
**/
```

```
#include <PiPei.h>  
#include <Uefi.h>  
#include <Library/UefiLib.h>  
#include <Library/DebugLib.h>  
#include <Library/PrintLib.h>  
#include <Library/UnitTestLib.h>  
// Other Includes
```

← Required Library Class

```
#define UNIT_TEST_NAME "Sample Unit Test"  
#define UNIT_TEST_VERSION "0.1"
```

← Preferred Defines used for Test Reports



# Example: Unit Test Suite Entry Points

```
/**
 * Standard PEIM entry point for target based unit test
 * execution from PEI.
 */
EFI_STATUS
EFIAPI
PeiEntryPoint (
    IN EFI_PEI_FILE_HANDLE      FileHandle,
    IN CONST EFI_PEI_SERVICES  **PeiServices
)
{
    return UefiTestMain ();
}
```

PEI

```
/**
 * Standard POSIX C entry point for host based
 * unit test execution.
 */

int
main (
    int argc,
    char *argv[]
)
{
    return UefiTestMain ();
}
```

HOST

```
/**
 * Standard UEFI entry point for target based unit test
 */
EFI_STATUS
EFIAPI
DxeEntryPoint (
    IN EFI_HANDLE      ImageHandle,
    IN EFI_SYSTEM_TABLE *SystemTable
)
{
    return UefiTestMain ();
}
```

DXE / UEFI

...  
Another Entry Point for a  
different environment

Other

# Example INF file for Host Unit Test 5

```
## @file
[Defines]
  INF_VERSION      = 0x00010005
  BASE_NAME        = SampleUnitTestHost
  FILE_GUID        = <get a new GUID>
  MODULE_TYPE      = HOST_APPLICATION
  VERSION_STRING   = 1.0
#
#  VALID_ARCHITECTURES  = IA32 X64
#
[Sources]
  SampleUnitTest.c
  < . . . Other sources >

[Packages]
  MdePkg/MdePkg.dec
  < . . . Other packages >
```

“Test”  
required in  
name

```
[LibraryClasses]
  BaseLib
  DebugLib
  UnitTestLib
  < . . . Other Libraries >

[Pcd]
  gEfiMdePkgTokenSpaceGuid.PcdDebugPropertyMask
  < . . . Other Pcds >
```

UnitTestLib  
Required

Creates the executable:  
SampleUnitTestHost

For Library Classes INF Files:  
LIBRARY\_CLASS =  
NameLib|HOST\_APPLICATION

# Example INF file for Non-Host Target 5

The Sections: Sources, Packages, LibraryClasses, and Pcd remain the same

Note, always get a **NEW** FILE\_GUID for each INF file.

## Differences:

### INF for PEI:

```
[Defines]
  BASE_NAME      = SampleUnitTestPei
  MODULE_TYPE    = PEIM
  ENTRY_POINT    = PeiEntryPoint
[Depex]
  gEfiPeiMemoryDiscoveredPpiGuid
```

### INF for DXE:

```
[Defines]
  BASE_NAME      = SampleUnitTestDxe
  MODULE_TYPE    = DXE_DRIVER
  ENTRY_POINT    = DxeEntryPoint
[Depex]
  TRUE
```

### INF for SMM:

```
[Defines]
  BASE_NAME      = SampleUnitTestSmm
  MODULE_TYPE    = DXE_SMM_DRIVER
  ENTRY_POINT    = DxeEntryPoint
[Depex]
  gEfiSmmCpuProtocolGuid
```

... There maybe others

# Example DSC File for Host Unit Test 5

```
[Defines]
  PLATFORM_NAME           = NamePkgHostTest
  PLATFORM_GUID           = <GUID>
  PLATFORM_VERSION        = 0.1
  DSC_SPECIFICATION       = 0x00010005
  OUTPUT_DIRECTORY       = Build/NamePkg/HostTest
  SUPPORTED_ARCHITECTURES = IA32|X64
  BUILD_TARGETS           = NOOPT
  SKUID_IDENTIFIER       = DEFAULT
```

```
!include UnitTestFrameworkPkg/UnitTestFrameworkPkgHost.dsc.inc
```

```
[PcdsPatchableInModule]
  gEfiMdePkgTokenSpaceGuid.PcdDebugPropertyMask|0x17
```

The DSC should have the NOOPT BUILD\_TARGET See [.pytool Readme.md Host Unit Test](#)  
Link to [UnitTestFrameworkPkgHost.dsc.inc](#)

# Example DSC File for Host Unit Test 5

```
[Components]
#
# Build HOST_APPLICATION that tests the SampleUnitTest
#
NamePkg/Test/UnitTest/Sample/SampleUnitTest/SampleUnitTestHost.inf

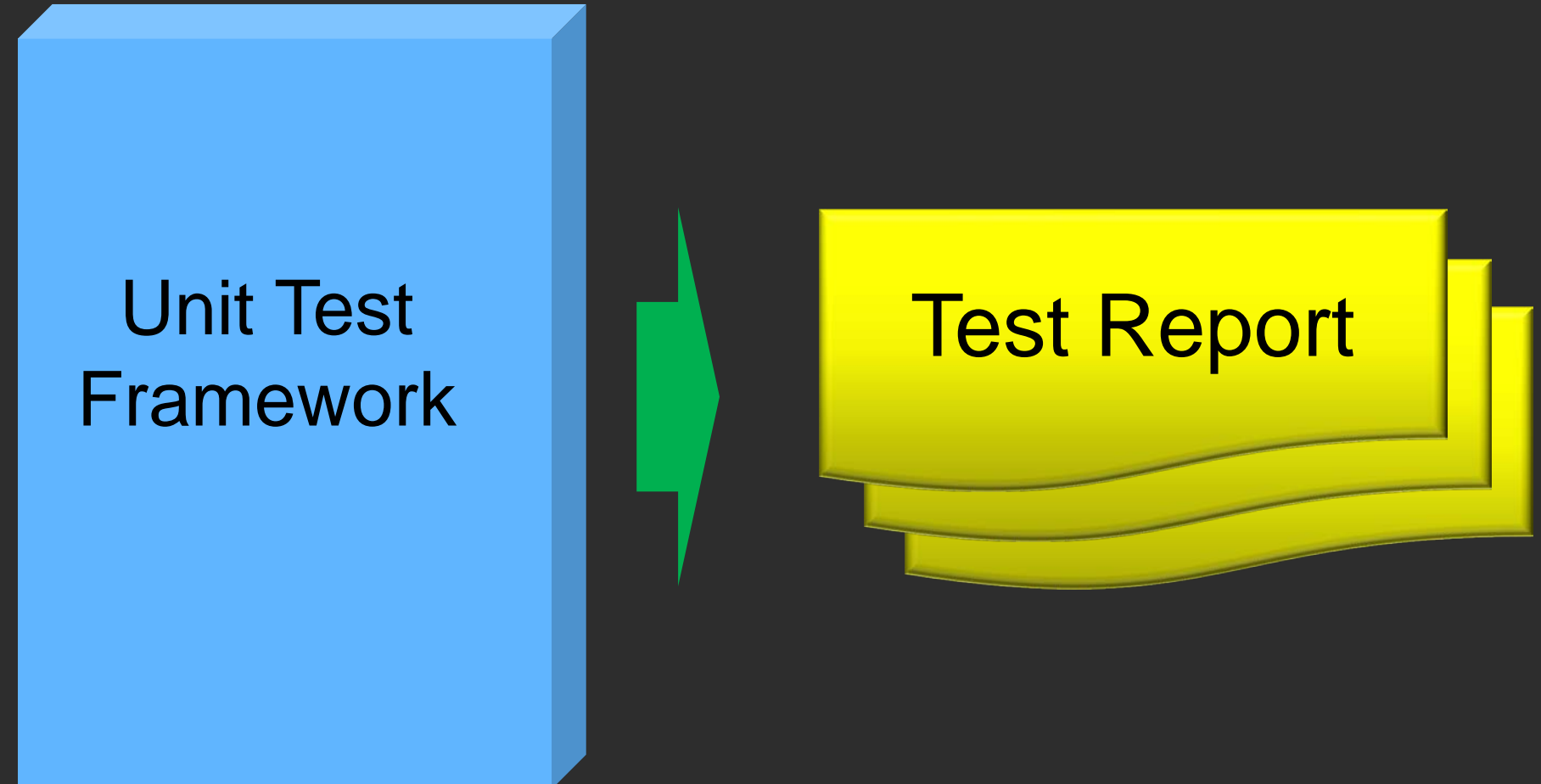
# Other Components
```

All of the modules that are included in the Components section of this DSC have:  
MODULE\_TYPE = HOST\_APPLICATION

Or For a library class have:

LIBRARY\_CLASS = NameLib|HOST\_APPLICATION

# Build and Examine Results



# Build/Run CI Locally Then Examine Results

## Continuous Integration (CI) Build

```
stuart_ci_build  
  -c .pytool/CISettings.py  
  TOOL_CHAIN_TAG=VS2019 -t NOOPT -p  
  UnitTestFrameworkPkg -a X64
```

Use pytools to setup the JUNIT XML output format

Parse the XML for failures and show the return status in the logs from the pytools (pass == 0 or fail == 1)

## OUTPUT

```
SECTION - Init SDE  
SECTION - Loading Plugins  
SECTION - Start Invocable Tool  
SECTION - Getting Environment  
SECTION - Loading plugins  
SECTION - Building UnitTestFrameworkPkg Package  
PROGRESS - --Running UnitTestFrameworkPkg: Host Unit  
           Test Compiler Plugin NOOPT --  
PROGRESS - Start time: 2020-07-08 14:20:08.036407  
PROGRESS - Setting up the Environment  
PROGRESS - Running Pre Build  
PROGRESS - Running Build NOOPT  
PROGRESS - Running Post Build  
SECTION - Run Host based Unit Tests  
SUBSECTION - Testing for architecture: X64  
PROGRESS - End time: 2020-07-08 14:21:40.247474  
PROGRESS - --->Test Success: Host Unit Test Compiler  
PROGRESS - Overall Build Status: Success  
SECTION - Summary  
PROGRESS - Success
```

# Examine Pass / Fail Reports - Manually

Run the created Host executable.

```
SampleUnitTestHost
```

Each test case will return a  
**UNIT\_TEST\_PASSED** if the test case was  
successful

Otherwise, on a test case assertion the unit  
test will return

**UNIT\_TEST\_ERROR\_TEST\_FAILED**

**PASS:** Report contains all the test case  
results all passing

```
[      OK ] String from test case
```

## Summary

```
===== # test(s) run.
[  PASSED  ] # test(s).
```

**FAIL:** If there are any tests with a failure

```
[  ERROR   ] --- Condition of Failure
[   LINE   ] --- File and line of Failure
```

## Summary

```
===== # test(s) run.
[  PASSED  ] # test(s)
[  FAILED  ] # test(s), listed below:
[  FAILED  ] TestName w/ results in # test
```



Most Host Based Unit Tests can be debugged on the host OS with software debugger:

- GDB or Visual Studio or others

- Use `CpuBreakpoint()`;

Some of the UnitTestFramework (mostly Cmocka) sets its own flags that cause some symbols and exception captures harder to debug.

- Use the build switch: `BLD*_UNIT_TESTING_DEBUG=TRUE`

Example:

```
stuart_ci_build -c .pytool/CISettings.py TOOL_CHAIN_TAG=VS2019 -t NOOPT -p -a X64
  UnitTestFrameworkPkg BLD*_UNIT_TESTING_DEBUG=TRUE
```

# NEXT STEPS

Example using MtrrLib in the UefiCpuPkg and next steps

- Review the `UnitTestFrameworkPkg SampleUnitTest` and unit test code examples on next slide
- Review the example using `MtrrLib` in the `UefiCpuPkg` [Link](#)
  - Determine how the `CPUID` and `Read/Write MSR` are done through the mock in the host unit test environment
- Complete the Unit Test Framework Lab Guide: [Link](#)
  - Step by step guide for the Stuart CI build and run the Sample Unit Test from `UnitTestFrameworkPkg`
  - Create a Host Unit Test Framework for a simple function
  - Add a UEFI Shell Unit Test Framework using the `EmulatorPkg`.

## Unit Test Framework Package Overview

– [Link](#)

Continuous Integration (CI) Configuring  
for Unit Tests – [Link](#)

## Code Examples of Unit Test Cases

- [Sample Unit Test](#)
  - [BaseSafeIntLib Unit Test](#)
  - [BaseLib Unit Test](#)
  - [DxeResetSystemLib Unit Test](#)
  - [MtrrLibUnitTest](#)
- 
- Cmocka Edk II Unit Test ChefCook  
example: [link](#)

- ✿ What is the Unit Test Framework and how does it work?
- ✿ How is Unit Test code included with Continuous Integration (CI)?
- ✿ Steps for adding a unit test with the Unit Test Framework

# Questions?



# Return to Main Training Page



Return to Training Table of contents for next presentation [link](#)







# ACKNOWLEDGEMENTS

Redistribution and use in source (original document form) and 'compiled' forms (converted to PDF, epub, HTML and other formats) with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code (original document form) must retain the above copyright notice, this list of conditions and the following disclaimer as the first lines of this file unmodified.

Redistributions in compiled form (transformed to other DTDs, converted to PDF, epub, HTML and other formats) must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS DOCUMENTATION IS PROVIDED BY TIANOCORE PROJECT "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL TIANOCORE PROJECT BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS DOCUMENTATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright (c) 2021-2022, Intel Corporation. All rights reserved.

# EXAMPLE USING CMOCKA

# How Does Cmocka work?

- The function under test requires a real-time call to another function
- Using Cmocka the unit test case will prime a “Mock” of the real-time function that the function under test would call
- Then each test case can prime the “Mock” real-time function to return deterministic values when the function under test calls it.
- The unit test case can then have a Pass / Fail decision-based on the values primed to the “Mock” real-time function
- The github [Cmocka](#) library is included as submodules from the EDK II UnitTestFrameworkPkg
- Cmocka [Tutorial](#)

# Example: Waiter to Return the Correct Dish



1. Customer  
Orders  
Hotdog

4. Hotdog  
Served



Waiter

2. Order is  
Hotdog

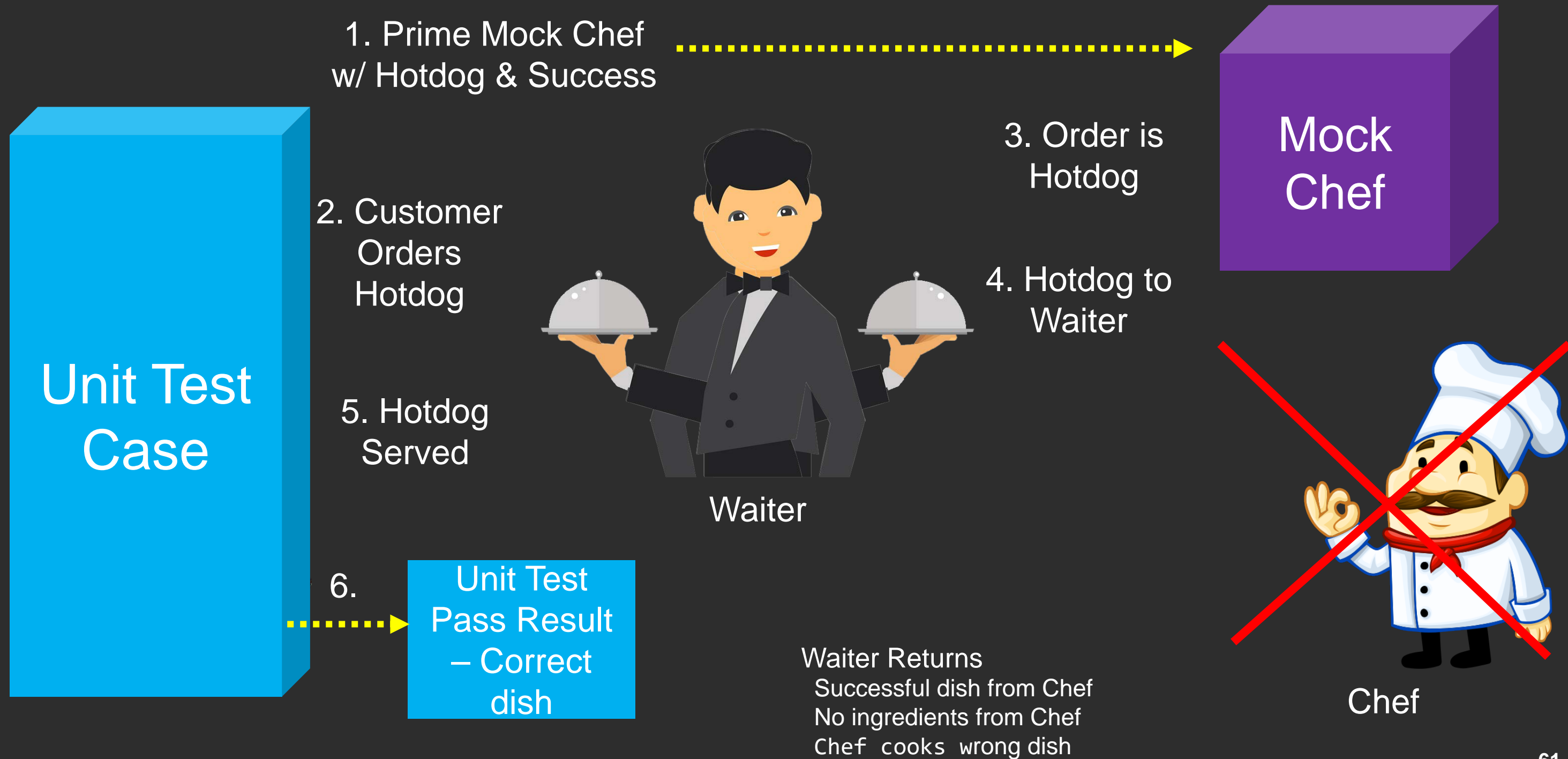
3. Hotdog  
to Waiter

Waiter Returns  
Successful dish from Chef  
No ingredients from Chef  
Chef cooks wrong dish

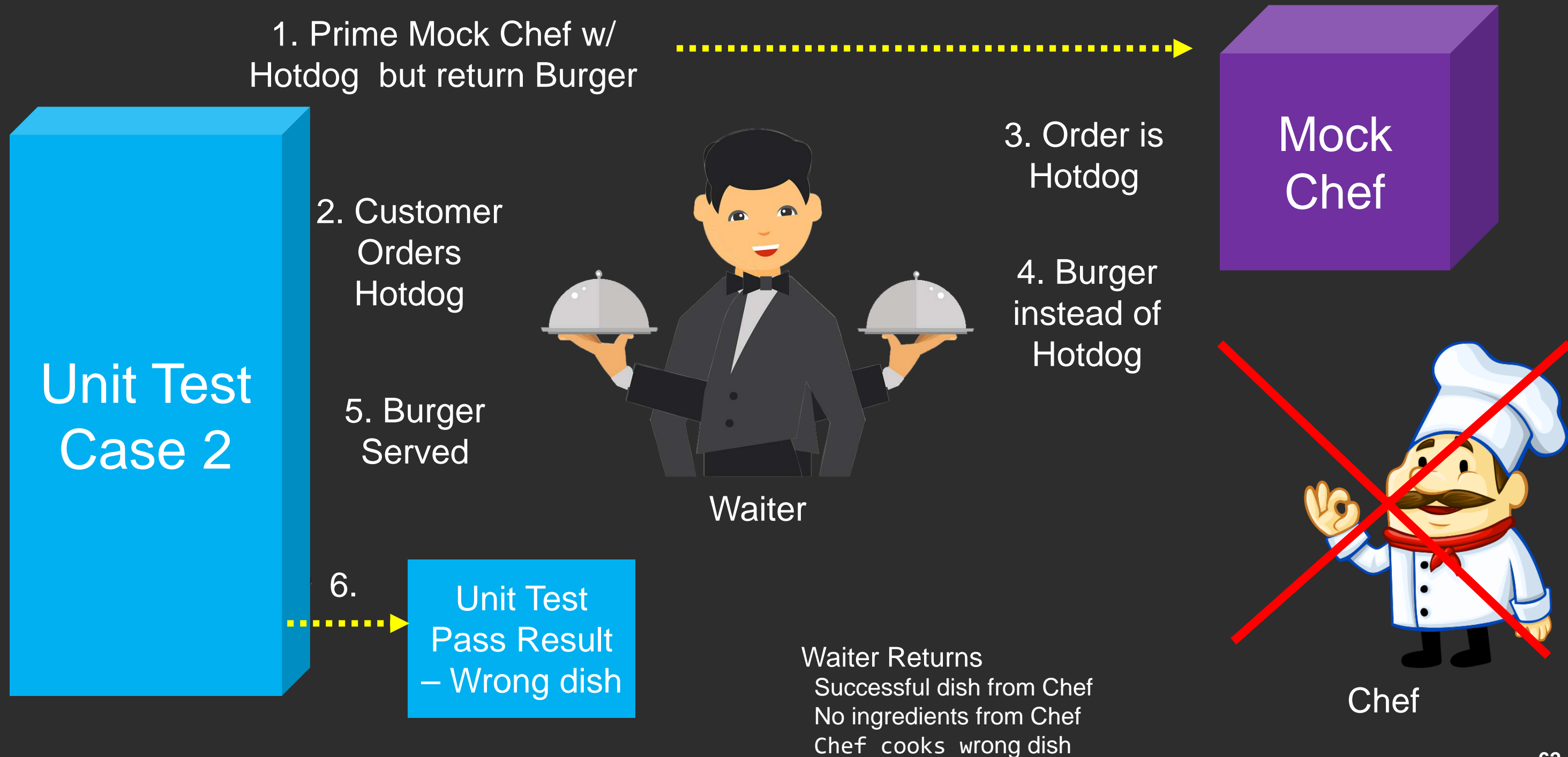


Chef

# Example: Waiter to Return the Correct Dish



# Example: Waiter to Return the Wrong Dish



# Using Cmocka Functions

## Return Values

`will_return(_mock, value)`

Values passed to `will_return()` are added to a queue

Successive call to `_mock()` removes a return value from the queue

## Input Parameters

`expect_*`

Store expected values for mock function parameters queued

`check_expected()`

Used in `_mock()` function to fill parameters for mocked function

Checks for validity and will signal failure if invalid



# Example: WaiterProcess call to ChefCook

```
STATIC INTN WaiterProcess(  
    IN CONST CHAR16 *order,  
    OUT CHAR16 **dish)  
{  
    INTN rv;  
  
    rv = ChefCook(order, dish);  
    if (rv != 0) {  
        return -1;  
    }  
  
    // Check if we received the dish we wanted  
    if (StrCmp(order, *dish) != 0) {  
        FreePool(*dish);  
        *dish = NULL;  
        return -2;  
    }  
    return 0;  
}
```

WaiterProcess is function to test

This is the function to test it calls ChefCook()

- 0 - success
- 1 - kitchen failed
- 2 - kitchen succeeded, but cooked a different food

Cmocka Edk II Unit Test ChefCook example: [link](#)

Cmocka example Chef\_wrap : [link](#)



# Mock Function ChefCook()

```
INTN __wrap_ChefCook(
    IN CONST CHAR16 *order,
    OUT CHAR16 **dish_out)
{
    BOOLEAN has_ingredients;
    BOOLEAN knows_dish;
    CHAR16 *dish;
    EFI_STATUS Status;

    check_expected_ptr(order);

    knows_dish = mock_type(BOOLEAN);
    if (knows_dish == FALSE) {
        return -1;
    }

    has_ingredients = mock_type(BOOLEAN);
    if (has_ingredients == FALSE) {
        return -2;
    }
}
```

1

2

3

```
    dish = mock_ptr_type(CHAR16 *);

    if (*dish_out == NULL) {
        *dish_out =
            (CHAR16*)AllocateZeroPool(MAX_DISH_SIZE
                * sizeof(CHAR16));
    }
    Status = StrCpyS(*dish_out, (MAX_DISH_SIZE *
        sizeof(CHAR16)) , dish);
    if (EFI_ERROR(Status)) {
        DEBUG((DEBUG_INFO, "Status = %r\n", Status));
        return -1;
    }

    return mock_type(INTN);
}
```

4

5

# Example Test Case - Correct Dish Served

```
UNIT_TEST_STATUS
EFIAPI
TestOrderHotdog(
    IN UNIT_TEST_CONTEXT Context
)
{
    INTN rv;
    CHAR16 *dish = NULL;
    // We expect the chef to receive an order for
    // a hotdog
    1 expect_string(__wrap_ChefCook, order, L"hotdog");

    // And we tell the test chef that he knows how
    // to cook a hotdog and has the ingredients
    2 will_return(__wrap_ChefCook, TRUE);
    3 will_return(__wrap_ChefCook, TRUE);

    // The result will be a hotdog and the cooking
    // process will succeed
    4 will_return(__wrap_ChefCook,
        cast_ptr_to_largest_integral_type(L"hotdog"));
    5 will_return(__wrap_ChefCook, 0);
}
```

```
// Test the waiter process - function to test
rv = WaiterProcess(L"hotdog", &dish);

// We expect the cook to succeed cooking
// the hotdog
UT_ASSERT_EQUAL(rv, 0);

// And actually, receive one
UT_ASSERT_MEM_EQUAL(dish, L"hotdog",
    sizeof(L"hotdog"));
if (dish != NULL) {
    FreePool(dish);
}
return UNIT_TEST_PASSED;
}
```