



**Yrkes
Akademin**
Vi hjälper dig att lyckas!

Software Testing

Programming and Development of Embedded Systems

Software Testing

❖ What is Software Testing?

- The process to evaluate functionalities of a software in order to
 - Find whether the developed software fulfils the specified requirements or not
 - Identify the defects to ensure that the software is defect free

❖ Why is Software Testing Important?

- Cost effectiveness
 - Identifying and fixing a bug early cost much less to fix it later
 - Nissan recalled 1.23 million cars in order to fix a problem with the backup cameras (2019)
- Customer Satisfaction
 - Customer satisfaction is the ultimate goal of any business
 - Software testing improves the user experience of an application

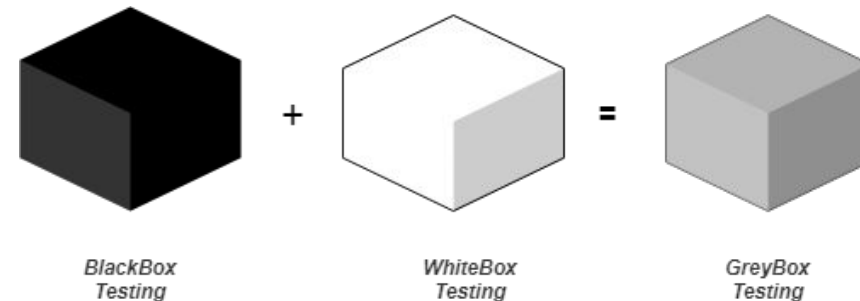
Software Testing

❖ Why is Software Testing Important?

- Safety and Security
 - Testing helps in removing vulnerabilities in the product
 - China Airlines Airbus A300 crashed due to a software bug, killing 264 lives (1994)
- Product Quality
 - Testing helps to deliver high quality products to clients according to requirements

❖ Testing Methods:

- White Box Testing
- Black Box Testing
- Grey Box Testing
 - It is a combination of white-box testing and black-box testing
 - It is primarily used in Integration Testing



Black Box Testing

- ❖ It is also known as **Behavioral** testing
- ❖ The internal structure/design/implementation of the component/system is unknown
- ❖ The higher the level, bigger and more complex the box, the more black-box testing is used
- ❖ It can be used in the functional and non-functional testings

- ❖ Some Techniques:

- Boundary Value Analysis

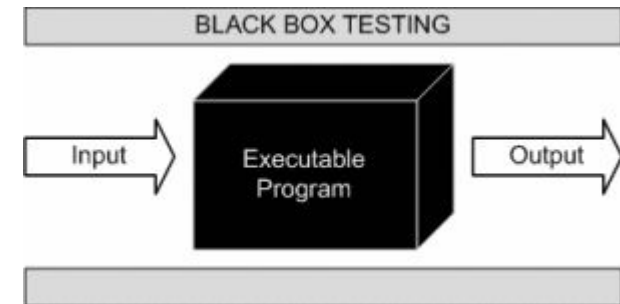
- Involves the determination of boundaries for input values
 - Using values that are at the boundaries and just inside/outside of the boundaries as test data

- Equivalence Partitioning

- Dividing input values into valid and invalid partitions and using them as the test data

- Cause-Effect Graphing:

- Involves identifying the cases that produce a **Cause-Effect Graph**, and generating test cases



Black Box Testing

❖ Black Box Testing can be used to find errors in

- Incorrect or missing functions, Interface errors and errors in data structures or external database
- Behavior or performance errors, and initialization and termination errors

❖ Black Box Testing Advantages

- Tests are done from a user's point of view
- Tests can be done by a body independent of developers
- Tester does not need to know programming languages or how the software has been implemented
- Test cases can be designed as soon as the specifications are complete



[What is Black Box Testing?](#)

❖ Black Box Testing Disadvantages

- Only a small number of possible inputs can be tested and many program paths will be left untested
- Without clear specifications, which is the case in many projects, test cases will be difficult to design
- Tests can be redundant if the software designer/developer has already run a test case

White Box Testing

- ❖ The internal structure/design/implementation of the component/system is known
- ❖ Testing is based on an analysis of the internal structure of the component or system
- ❖ White Box Testing Advantages
 - Testing is more complete, with the possibility of covering most paths
 - Code optimization by finding hidden errors
 - Testing can be started at an earlier stage
 - Test cases can be easily automated
- ❖ White Box Testing Disadvantages
 - It can be quite complex and expensive (highly skilled resources are required)
 - Test script maintenance can be a burden if the implementation changes too frequently
 - Since it is closely tied to the application under test, tools for every kind of implementation/platform may not be available



White Box Testing

❖ White Box Testing Techniques

- Statement Coverage: Every possible statement in the code should be tested at least once during the testing
- Branch Coverage: It checks every possible path (if-else and other conditional loops) of a software

❖ What do you verify in White Box Testing?

- Internal security holes
- The flow of specific inputs through the code
- Expected output
- The functionality of conditional loops
- Testing of each statement, object, and function on an individual basis
- Broken or poorly structured paths in the coding processes



[What is White Box Testing? Tutorial with Examples](#)

Testing Levels

❖ Software Testing Levels

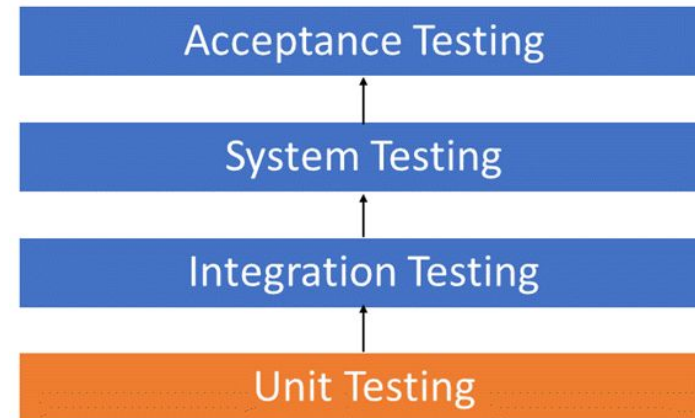
- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing

❖ Types of Software Testing

- Functional Testing
- Non-Functional Testing
- Maintenance (Regression and Maintenance)
- And etc.



[Types of Software Testing: 100 Examples of Different Testing Types](#)

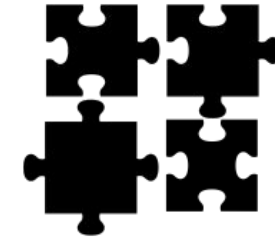


Unit Testing

- ❖ The first level of testing; Individual units/components of a software are tested
- ❖ The purpose is to validate that each unit of the software performs as designed
- ❖ A unit is the smallest testable part of a software (e.g. functions, components and etc.)
- ❖ It is performed using the **White Box Testing** method
- ❖ It is normally performed by software developers themselves
- ❖ Unit Testing Benefits
 - It increases confidence in changing/maintaining code (they are run every time any code is changed)
 - Codes are more reusable. In order to make unit testing possible, codes need to be modular.
 - Development is faster (short term and long term)
 - Writing tests takes lesser time than the time it takes to run the tests
 - It takes lesser time to find and fix bugs during unit testing than system or acceptance testing
 - Debugging is easy. When a test fails, only the latest changes need to be debugged



[What is Unit Testing? - Software Testing Tutorial](#)

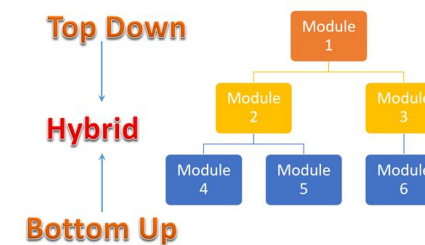
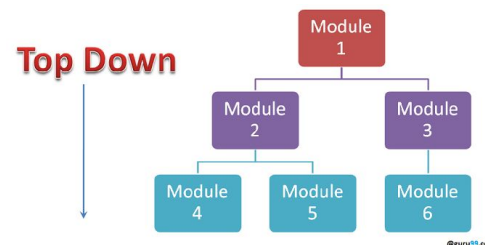
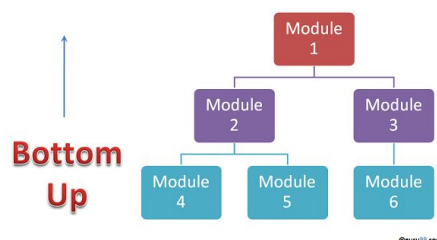
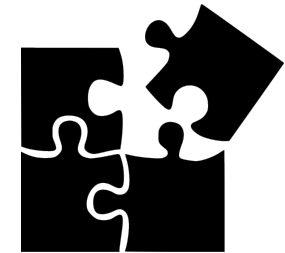


Integration Testing

- ❖ It is a level of software testing where individual units are combined and tested as a group
- ❖ The purpose of this level of testing is to expose faults in the interaction between integrated units.
- ❖ Any of Black Box, White Box and Gray Box Testing methods can be used
- ❖ It is done by developers themselves or independent testers
- ❖ Integration Testing Approaches
 - **Big Bang:** Here all components are integrated together at once and then tested
 - **Incremental:** Testing is done by joining two or more modules that are logically related
 - Bottom-up Integration, Top-down Integration and Hybrid/ Sandwich Integration

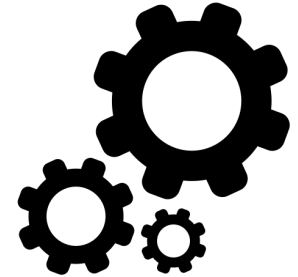


[What is Integration Testing?](#)



System Testing

- ❖ The process of testing a fully integrated system to verify that it meets specified requirements
- ❖ Usually, Black Box Testing method is used
- ❖ Independent testers perform System Testing
- ❖ Some Types of System Testing
 - Usability Testing: mainly focuses on the user's ease to use the application
 - Load Testing: is necessary to know that the product will work under real-life loads
 - Regression Testing: ensures that changes to the software have not caused new defect
 - Functional Testing: involves trying to find any possible missing functional requirements
 - Hardware/Software Testing: focuses on the interactions between the hardware and software during system testing
 - And etc.



[What is System Testing? Software Testing Tutorial](#)

Acceptance Testing

- ❖ Formal testing with respect to user needs, requirements, and business processes to:
 - Determine whether or not a system satisfies the acceptance criteria of the customer
 - Enable the customer or other authorized entities to determine whether or not to accept the system
- ❖ Usually, Black Box Testing is used in Acceptance Testing
- ❖ Who performs the Acceptance Testing?
 - Internal Acceptance Testing
 - Is performed by members of the organization who are not directly involved in the project
 - External Acceptance Testing: Is performed by people who are not employees of the organization
 - User Acceptance Testing: is performed by the end users of the software
 - They can be customers themselves or the customers' customers
 - Customer Acceptance Testing
 - Is performed by the customer of the organization that developed the software

- 1 {
 - Developers have included features on their "own" understanding
- 2 {
 - Requirements changes "not communicated" effectively to the developers

Test-Driven Development - TDD

- ❖ Test-Driven Development is a technique for building software incrementally
 - Small steps can help form better components
- ❖ New automated unit test is followed immediately by production code satisfying the test
 - Healthy code growth, components are less prone of bugs
- ❖ Passing tests means the software is doing its job
 - Peace of mind, if you trust the tests you can have confidence in the code
 - Fewer bugs, ensuring components are well tested can help reduce bugs
- ❖ Failing tests means the software is not done
- ❖ Well tested code helps the software quality, maintainability and reliability
- ❖ Refactor friendly, tests ensure behaviour stays the same

Test-Driven Development - TDD

- ❖ Refactoring should make components easier to understand
 - Cleaner and more readable code
- ❖ Test automation is key to TDD
 - Testing is automatically performed by machine over and over.
 - No need for manual testing.
- ❖ TDD improves the functional and structural quality of the code
- ❖ Three rules of TDD
 - Write production code only to pass a failing unit test.
 - Write no more of a unit test than sufficient to fail.
 - Compilation failures are failures
 - Write no more production code than necessary to pass the one failing unit test.

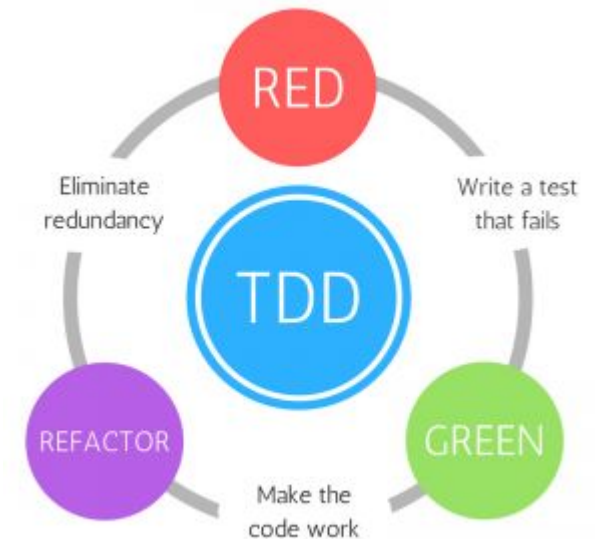
Test-Driven Development - TDD

❖ The steps of the TDD cycle

- Add a small test
- Run all the tests and see the new one fails, maybe not even compile
- Make the small changes needed to pass the failing test
- Run all the tests and see the new one passes
- Refactor to remove duplication and improve expressiveness

❖ Unit tests

- Isolated, test cases should be kept to one component at a time
- Small, expected behaviour should be broken down into minimal tests
- Fast, it should be quick and easy to run your unit tests
- Everywhere, local developer machine or remote building environment



Test-Driven Development - TDD

❖ General testing guidelines

- Tests should
 - Be easy to follow
 - Help to maintain healthiness of the code
 - Tell us when something is changed
 - Be derived from requirements
 - Enforce a specific behaviour
 - Encourage refactoring

- Different test cases require different test types
 - Evaluate the best test type for your tests
 - Tests can derive into multiple test types

Test-Driven Development - TDD

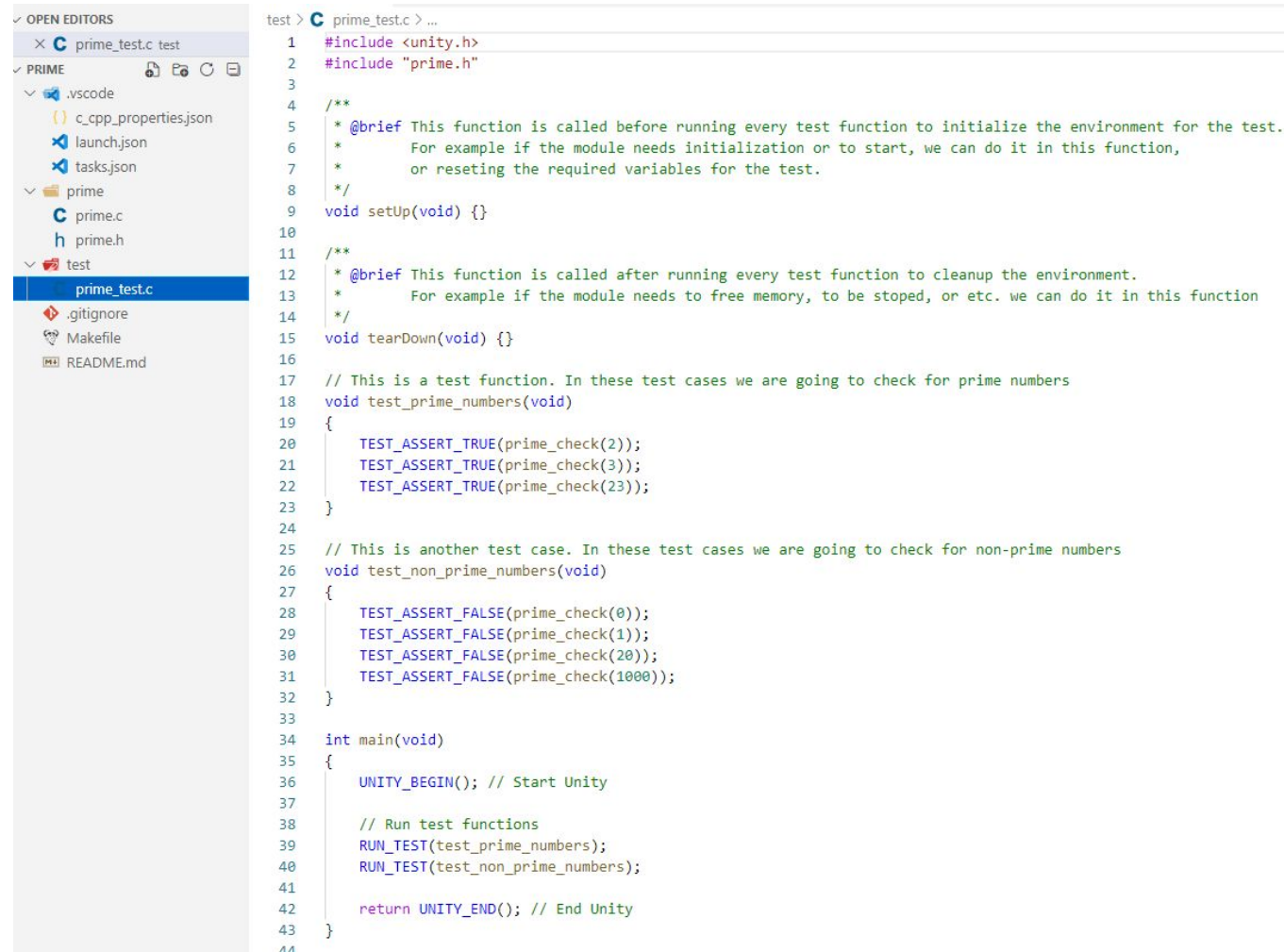
❖ TDD Benefits

- Fewer bugs: Small and large logic errors are found quickly during development
- Less debug time: Having fewer bugs means less debug time.
- Fewer side effect defects: When new code violates a constraint or assumption, the tests scream
- Documentation
 - Well-structured tests become a form of executable documentation
- Improved design
 - A good design is a testable design.
 - You get an early warning of design problems if tests cannot be written for the code change
- Progress monitor
 - The tests keep track of exactly what is working and how much work is done

Unity

- ❖ [Unity](#) is a portable unit testing framework written in standard C and supports testing of embedded systems. [Unity Test](#)
- ❖ Unity uses assertions to test the actual values and the expectations
- ❖ Assertions are statements of what we expect to be true about the code under test.
 - E.g. `TEST_ASSERT_EQUAL_UINT8(expected, actual);`
 - Checks if the expected value and the actual value are equal or not
 - If they are not equal, it means the test is failed
- ❖ Unity is used to test modules. Means that the code under test shall be module(s).
 - A module is a source(.c) and a header(.h) file.
 - To test a module you also need to create a test file (.c) and include unity.h
 - The test cases are implemented in this file.

Unity - Example



The screenshot shows a Visual Studio Code editor with a project named 'PRIME'. The file explorer on the left shows the project structure, including files like .vscode, c_cpp_properties.json, launch.json, tasks.json, prime, prime.c, prime.h, test, prime_test.c, .gitignore, Makefile, and README.md. The 'prime_test.c' file is selected and its content is displayed in the editor. The code is a C program using the Unity testing framework. It includes 'unity.h' and 'prime.h'. It defines two test functions: 'test_prime_numbers' and 'test_non_prime_numbers'. The 'test_prime_numbers' function uses 'TEST_ASSERT_TRUE' to verify that 'prime_check' returns true for prime numbers (2, 3, 23). The 'test_non_prime_numbers' function uses 'TEST_ASSERT_FALSE' to verify that 'prime_check' returns false for non-prime numbers (0, 1, 20, 1000). The 'main' function starts the Unity test runner, runs the test functions, and ends the test runner.

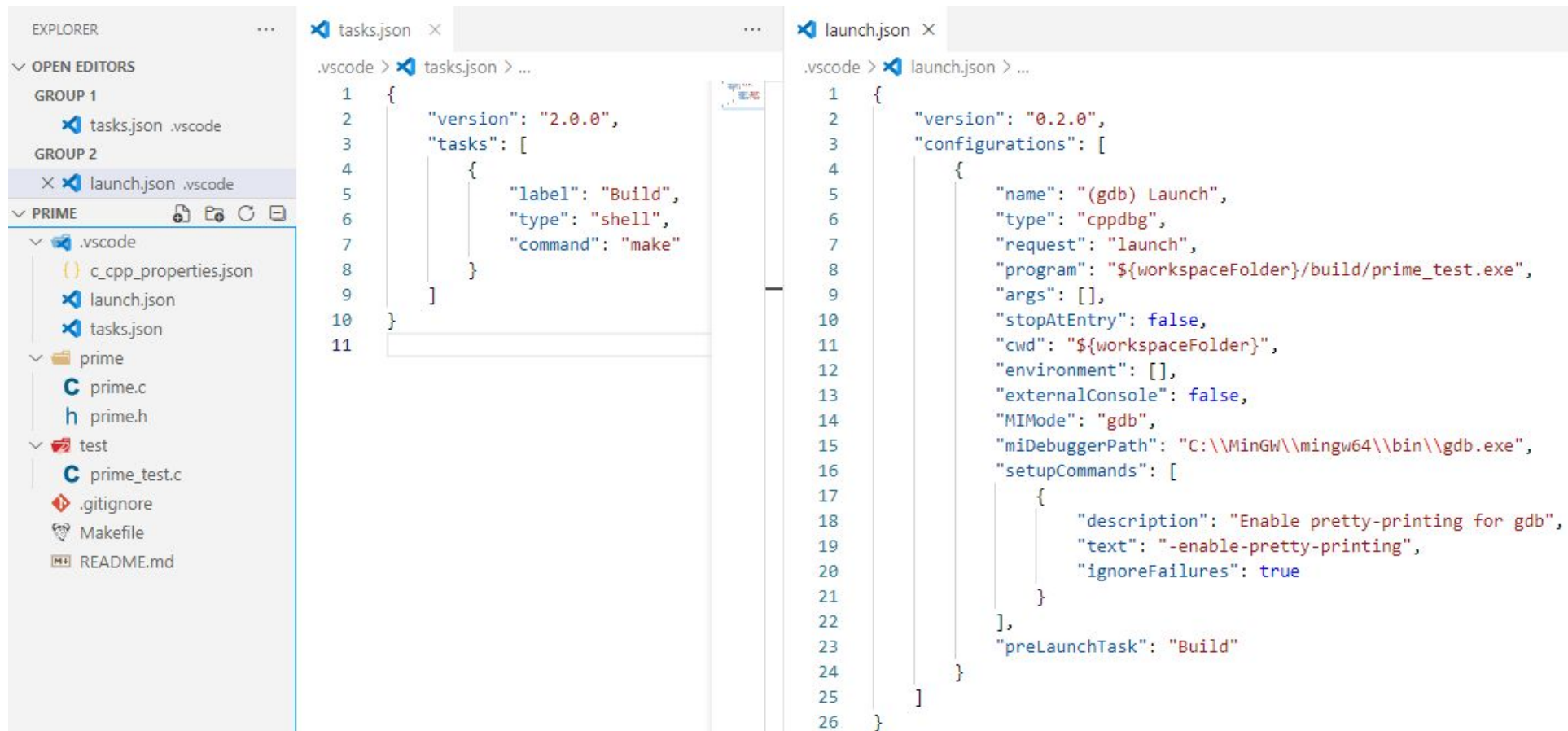
```
test > C prime_test.c > ...
1  #include <unity.h>
2  #include "prime.h"
3
4  /**
5   * @brief This function is called before running every test function to initialize the environment for the test.
6   *        For example if the module needs initialization or to start, we can do it in this function,
7   *        or resetting the required variables for the test.
8   */
9  void setUp(void) {}
10
11 /**
12  * @brief This function is called after running every test function to cleanup the environment.
13  *        For example if the module needs to free memory, to be stopped, or etc. we can do it in this function
14  */
15 void tearDown(void) {}
16
17 // This is a test function. In these test cases we are going to check for prime numbers
18 void test_prime_numbers(void)
19 {
20     TEST_ASSERT_TRUE(prime_check(2));
21     TEST_ASSERT_TRUE(prime_check(3));
22     TEST_ASSERT_TRUE(prime_check(23));
23 }
24
25 // This is another test case. In these test cases we are going to check for non-prime numbers
26 void test_non_prime_numbers(void)
27 {
28     TEST_ASSERT_FALSE(prime_check(0));
29     TEST_ASSERT_FALSE(prime_check(1));
30     TEST_ASSERT_FALSE(prime_check(20));
31     TEST_ASSERT_FALSE(prime_check(1000));
32 }
33
34 int main(void)
35 {
36     UNITY_BEGIN(); // Start Unity
37
38     // Run test functions
39     RUN_TEST(test_prime_numbers);
40     RUN_TEST(test_non_prime_numbers);
41
42     return UNITY_END(); // End Unity
43 }
44
```

GDB

- ❖ Debugging is the process of finding and resolving problems within a program using a debugger.
- ❖ **GDB** is the [GNU Project debugger](#) which can be used to debug programs written in C language.
- ❖ To compile your program for debugging you need to use **-g**. E.g. **`gcc -g main.c -o main`**
- ❖ You can use **gdb** to debug your program in [command line](#) or in Visual studio code.
- ❖ To use gdb & gcc to build, debug and run in visual studio code
 - Click on the run icon and then click on the **Run and Debug** button
 - Select C++(GDB/LLDB) in the opened list. And then Select **Default Configuration**
 - A json file (**launch.json**) is opened. Then set the path to your program: "**program**": "**path/to/your/program**"
 - Then set the path of the debugger(gdb): "**miDebuggerPath**": "**C:\\MinGW\\mingw64\\bin\\gdb.exe**"
 - To make a task to compile & run your program click on **Terminal > Configure Tasks..** in the main menu.
 - Then click on **Create tasks.json file from template** and then **Others Examples to run an arbitrary external command**.
 - The **tasks.json** file is created. Change the label from **echo** to **Build**
 - In the **command** write the shell commands to compile and run your program.
 - Then open **launch.json** and add "**preLaunchTask**": "**Build**" to the configurations.

Visual Studio Code - GDB

❖ An example of launch.json and tasks.json



The screenshot shows the Visual Studio Code interface with three panels. The left panel is the Explorer, showing the file structure of a project named 'PRIME'. The middle panel shows the 'tasks.json' file, and the right panel shows the 'launch.json' file.

Explorer Panel:

- OPEN EDITORS
 - GROUP 1
 - tasks.json .vscode
 - GROUP 2
 - launch.json .vscode
- PRIME
 - .vscode
 - c_cpp_properties.json
 - launch.json
 - tasks.json
 - prime
 - prime.c
 - prime.h
 - test
 - prime_test.c
 - .gitignore
 - Makefile
 - README.md

tasks.json:

```
1 {
2   "version": "2.0.0",
3   "tasks": [
4     {
5       "label": "Build",
6       "type": "shell",
7       "command": "make"
8     }
9   ]
10 }
11
```

launch.json:

```
1 {
2   "version": "0.2.0",
3   "configurations": [
4     {
5       "name": "(gdb) Launch",
6       "type": "cppdbg",
7       "request": "launch",
8       "program": "${workspaceFolder}/build/prime_test.exe",
9       "args": [],
10      "stopAtEntry": false,
11      "cwd": "${workspaceFolder}",
12      "environment": [],
13      "externalConsole": false,
14      "MIMode": "gdb",
15      "miDebuggerPath": "C:\\MinGW\\mingw64\\bin\\gdb.exe",
16      "setupCommands": [
17        {
18          "description": "Enable pretty-printing for gdb",
19          "text": "-enable-pretty-printing",
20          "ignoreFailures": true
21        }
22      ],
23      "preLaunchTask": "Build"
24    }
25  ]
26 }
```

Exercise 12 - TDD

Using TDD develop
a linked list module
of unique and sorted
elements.

The module name
shall be **list** and here
you have the header
file of the module.

```
#ifndef LIST_H
#define LIST_H

#include <stddef.h>
#include <stdbool.h>

// This function is used to insert data in the list. It returns true if value is unique and inserted successfully; otherwise false
bool list_insert(int data);

// This function returns number of the nodes in the list.
size_t list_available(void);

// This function is used to search for a value in the list. It returns 0 if data is not in the list; otherwise the position of the data in the list.
size_t list_find(int data);

// This function is used to get the data stored in the nth node. n is the nth element. n shall be a valid position and greater than 0. It returns
// false if n is not valid; otherwise true. The data stored in the nth node will be stored in the variable whose address is passed as ptr.
bool list_get_data(size_t n, int *ptr);

// This function is used to change data in the list. It returns false if old_data does not exist or new_data already exists; otherwise true
bool list_edit(int old_data, int new_data);

// This function is used to delete an element in the list. It returns false if data does not exist; otherwise true
bool list_delete(int value);

// This function is used to destroy the list and free the allocated memory.
void list_clear(void);

#endif /* LIST_H */
```


Software Testing

❖ Some useful links

- [Software Testing Tutorial: Free Course](#)
- [Software Testing Fundamentals](#)
- [The Three Laws of TDD](#)
- [What Is Software Testing?](#)
- [The Three Laws of TDD \(Featuring Kotlin\)](#)
- [How TDD is related to the quality of code](#)
- [What is Test Driven Development?](#)
- [What is Behavior Driven Development?](#)
- [TDD vs BDD. Giants of Test Go Head-to-Head \(1\)](#)
- [TDD vs BDD. The Giants of Test Go Head-to-Head \(2\)](#)
- [TDD vs BDD - Your free CHEAT SHEET](#)
- [TDD and BDD - What Are The Key Differences?](#)