



IIT Madras BSc Degree

Copyright and terms of use

IIT Madras is the sole owner of the content available in this portal - onlinedegree.iitm.ac.in and the content is copyrighted to IIT Madras.

- Learners may download copyrighted material for their use for the purpose of the online program only.
- Except as otherwise expressly permitted under copyright law, no use other than for the purpose of the online program is permitted.
- No copying, redistribution, retransmission, publication or exploitation, commercial or otherwise of material will be permitted without the express permission of IIT Madras.
- Learner acknowledges that he/she does not acquire any ownership rights by downloading copyrighted material.
- Learners may not modify, publish, transmit, participate in the transfer or sale, create derivative works, or in any way exploit, any of the content, in whole or in part.

Testing

Application Testing

- Why?
- What?
- When?
- How?
- Pytest

Why?

Does *something* work *as intended*

- Requirements - specifications
- Respond correctly to inputs
- Respond within reasonable time
- Installation and environment
- Usability and Correctness

Static vs. Dynamic

Static Testing:

- Code review, correctness proofs

Dynamic Testing:

- Functional tests
- Apply suitable inputs

White-box testing

- Detailed knowledge of implementation
- Can examine internal variables, counters
- Tests can be created based on knowledge of internal structure
- Pro:
 - More detailed information available, better tests
- Con:
 - Can lead to focusing on less important parts because code is known
 - Does not encourage clean abstraction
 - Too much information?

Black-box testing

- Only interfaces are available, not the actual code
- Tests based on how it would look from outside
- Pro:
 - Closer to real usage scenario
 - Encourages (enforces) clean abstraction of interface
- Con:
 - May miss corner cases that would have been obvious if internal structure was known
 - Debugging is harder - even if it failed, why did it fail?

Grey-box testing

- Hybrid approach between white-box and black-box
- Enforce interface as far as possible
- Internal structure mainly used for debugging, examining variables etc.

Regressions

- Maintain series of tests starting from basic development of code
 - Each test is for some specific feature or set of features
- **Regression:** loss of functionality introduced by some change in the code
- Future modifications to code should not break existing code
- Sometimes necessary
 - Update tests
 - Update API versions etc.

Coverage

- How much of the code is covered
 - Every line is executed at least once - 100% code coverage
 - Does not guarantee “correctness” in all conditions
 - There may be more complex paths or other conditions that can cause failure
- Branch coverage, condition coverage, function coverage ...

Example

```
int foo (int x, int y)
{
    int z = 0;
    if ((x > 0) && (y > 0))
    {
        z = x;
    }
    return z;
}
```

Src: Wikipedia

Example

```
int foo (int x, int y)
{
    int z = 0;
    if ((x > 0) && (y > 0))
    {
        z = x;
    }
    return z;
}
```

Src: Wikipedia

Function coverage

- Test invokes foo() at least once

Example

```
int foo (int x, int y)
{
    int z = 0;
    if ((x > 0) && (y > 0))
    {
        z = x;
    }
    return z;
}
```

Src: Wikipedia

Statement coverage

- Example: foo(1,1)
 - All statements in code will be executed

Example

```
int foo (int x, int y)
{
    int z = 0;
    if ((x > 0) && (y > 0))
    {
        z = x;
    }
    return z;
}
```

Src: Wikipedia

Branch coverage

- At least two tests needed:
- foo(1,1)
 - Branch taken
- foo(1,0)
 - Branch not taken

Example

```
int foo (int x, int y)
{
    int z = 0;
    if ((x > 0) && (y > 0))
    {
        z = x;
    }
    return z;
}
```

Src: Wikipedia

Condition coverage

- At least two tests needed:
- foo(0,1)
 - First condition fails, second succeeds
- foo(1,0)
 - First condition succeeds, second fails
- Note: does not guarantee branch coverage

Summary

- Requirements specified by user
- Creating suitable tests can itself be challenging
- How much knowledge of the code internals should the tester have?
- Separation of concerns:
 - ideally tester should be able to generate test cases based only on spec and without knowing code
- Code coverage useful metric
 - Does not guarantee all scenarios actually tested!