Agile Software Development

Produced by

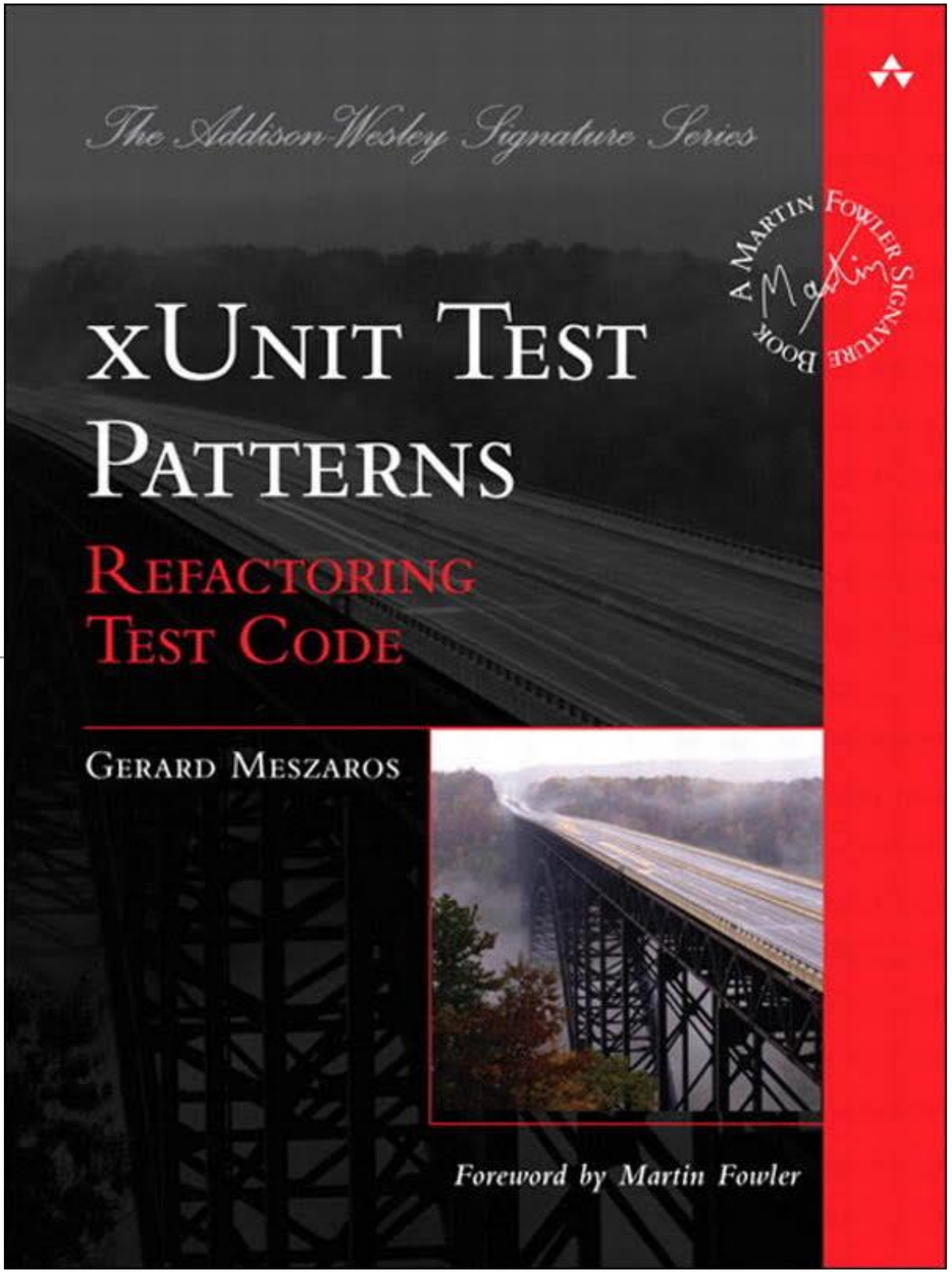
Eamonn de Leastar (edeleastar@wit.ie)

Department of Computing, Maths & Physics
Waterford Institute of Technology

http://www.wit.ie

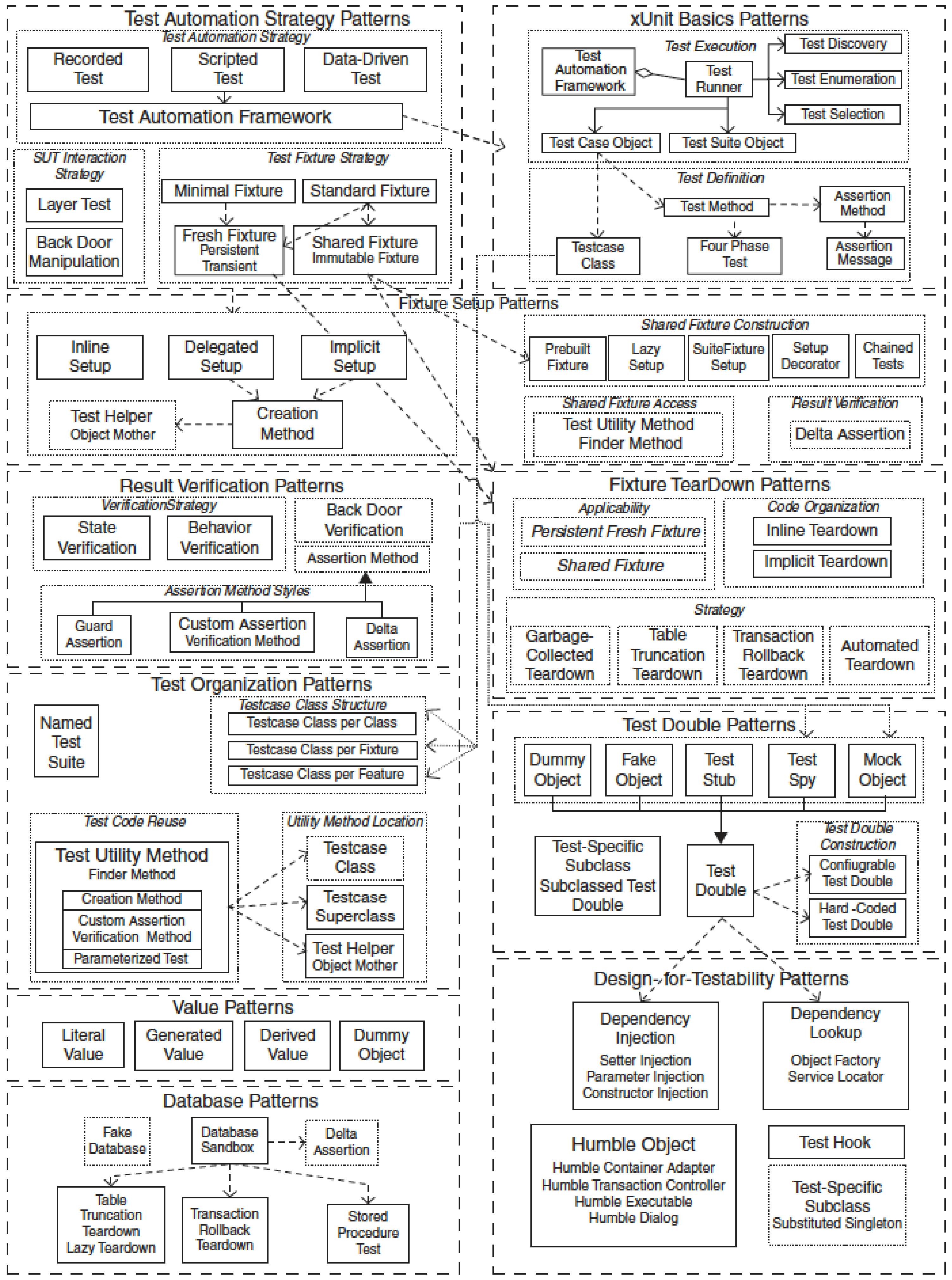
http://elearning.wit.ie

http://xunitpatterns.com/


The Addison-Wesley Signature Series
A Martin Fowler Signature Book
xUnit Test Patterns
Refactoring Test Code
Gerard Meszaros
Foreword by Martin Fowler

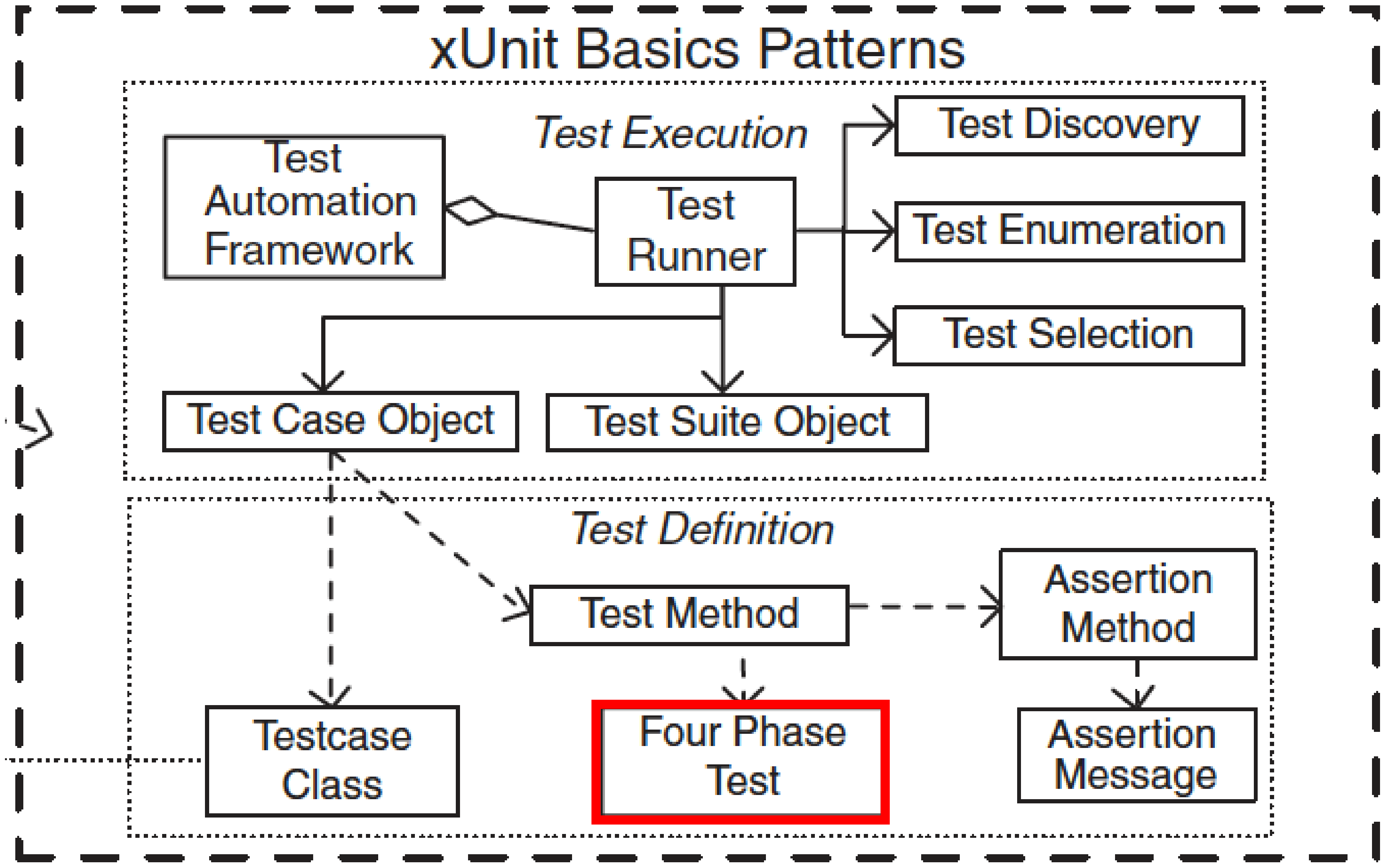
Unit Test Patterns

- Comprehensive and exhaustive catalog of test Patterns covering:
  - Test Automation
  - xUnit Basics
  - Fixture Setup (& teardown)
  - Result Verification
  - Test Organisation
  - Test Doubles
  - Value
  - Database
  - Design-for-Testability



xUnit Basics

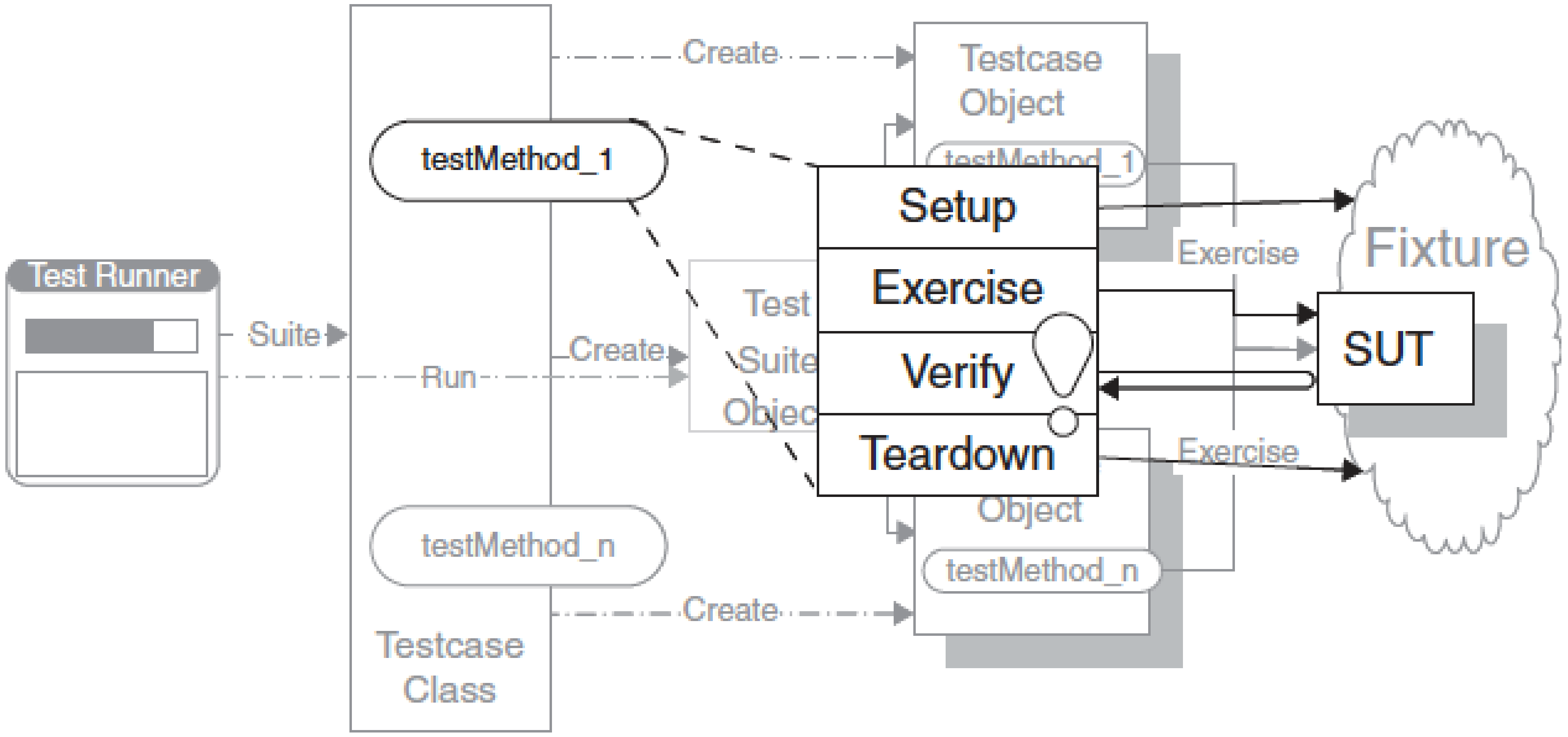
- Features of the xUnit framework.
- x indicates the programming language.
- Largely implemented by JUnit - and automatically integrated into:
  - IDE (Eclipse)
  - Build System (maven)



Four Phase Test

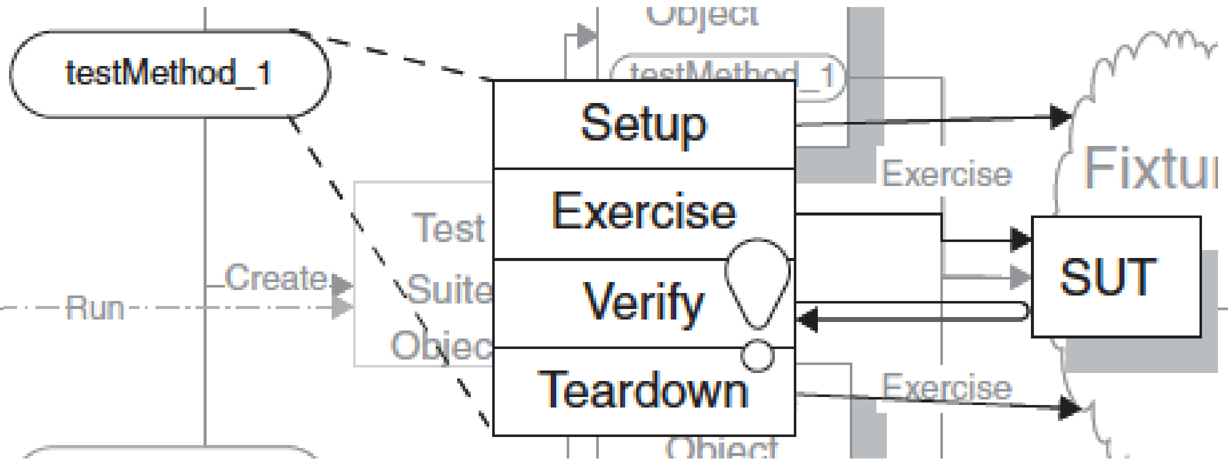
How do we structure our test logic to make what we are testing obvious?

We structure each test with four distinct parts executed in sequence.



SUT = System Under Test

How it works



- SETUP: In the first phase, we set up the test fixture (the “before” picture) that is required for the SUT to exhibit the expected behavior as well as anything you need to put in place to be able to observe the actual outcome.

- EXERCISE: In the second phase, we interact with the SUT.

- VERIFY: In the third phase, we do whatever is necessary to determine whether the expected outcome has been obtained.

- TEARDOWN: In the fourth phase, we tear down the test fixture to put the world back into the state in which we found it.

Example

```
@Test
public void testXMLSerializer() throws Exception
{
    String datastoreFile = "testdatastore.xml";
    deleteFile (datastoreFile);

    Serializer serializer = new XMLSerializer(new File (datastoreFile));

    pacemaker = new PacemakerAPI(serializer);
    populate(pacemaker);
    pacemaker.store();

    PacemakerAPI pacemaker2 =  new PacemakerAPI(serializer);
    pacemaker2.load();

    assertEquals (pacemaker.getUsers().size(), pacemaker2.getUsers().size());
    for (User user : pacemaker.getUsers())
    {
        Collection<User> users = pacemaker2.getUsers();
        System.out.println("User to search for:");
        System.out.println(user);
        System.out.println("Collection");
        System.out.println(users);
        assertTrue (users.contains(user));
    }
    deleteFile (datastoreFile);
}
```

Example

```
@Test
public void testXMLSerializer() throws Exception
{
```

Phase 1 (setup)

```
    String datastoreFile = "testdatastore.xml";
    deleteFile (datastoreFile);

    Serializer serializer = new XMLSerializer(new File (datastoreFile));

    pacemaker = new PacemakerAPI(serializer);
    populate(pacemaker);
```

Phase 2(exercise)

```
    pacemaker.store();

    PacemakerAPI pacemaker2 =  new PacemakerAPI(serializer);
    pacemaker2.load();
```

Phase 3 (verify)

```
    assertEquals (pacemaker.getUsers().size(), pacemaker2.getUsers().size());
    for (User user : pacemaker.getUsers())
    {
        Collection<User> users = pacemaker2.getUsers();
        System.out.println("User to search for:");
        System.out.println(user);
        System.out.println("Collection");
        System.out.println(users);
        assertTrue (users.contains(user));
    }
```

Phase 4 (teardown)

```
    deleteFile (datastoreFile);
}
```

Example (xtend)

Phase 1 (setup)

Phase 2 (exercise)

Phase 3 (verify)

```
@Test def void testXMLSerializer()
{
  pacemaker.store
  val pacemaker2 =  new PacemakerAPI(serializer)

  pacemaker2.load

  pacemaker.users.forEach [assertTrue(pacemaker2.users.contains(it))]
}
```

Phase 4 (in teardown)

Why we do this

- The test reader must be able to quickly determine what behavior the test is verifying.
- It can be very confusing when various behaviors of the SUT are being invoked—some to set up the pre-test state (fixture) of the SUT, others to exercise the SUT, and yet others to verify the post-test state of the SUT.
- Clearly identifying the four phases makes the intent of the test much easier to see.
- Avoid the temptation to test as much functionality as possible in a single Test Method because that can result in Obscure Tests.
- It is preferable to have many small Single-Condition Test Methods.

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit