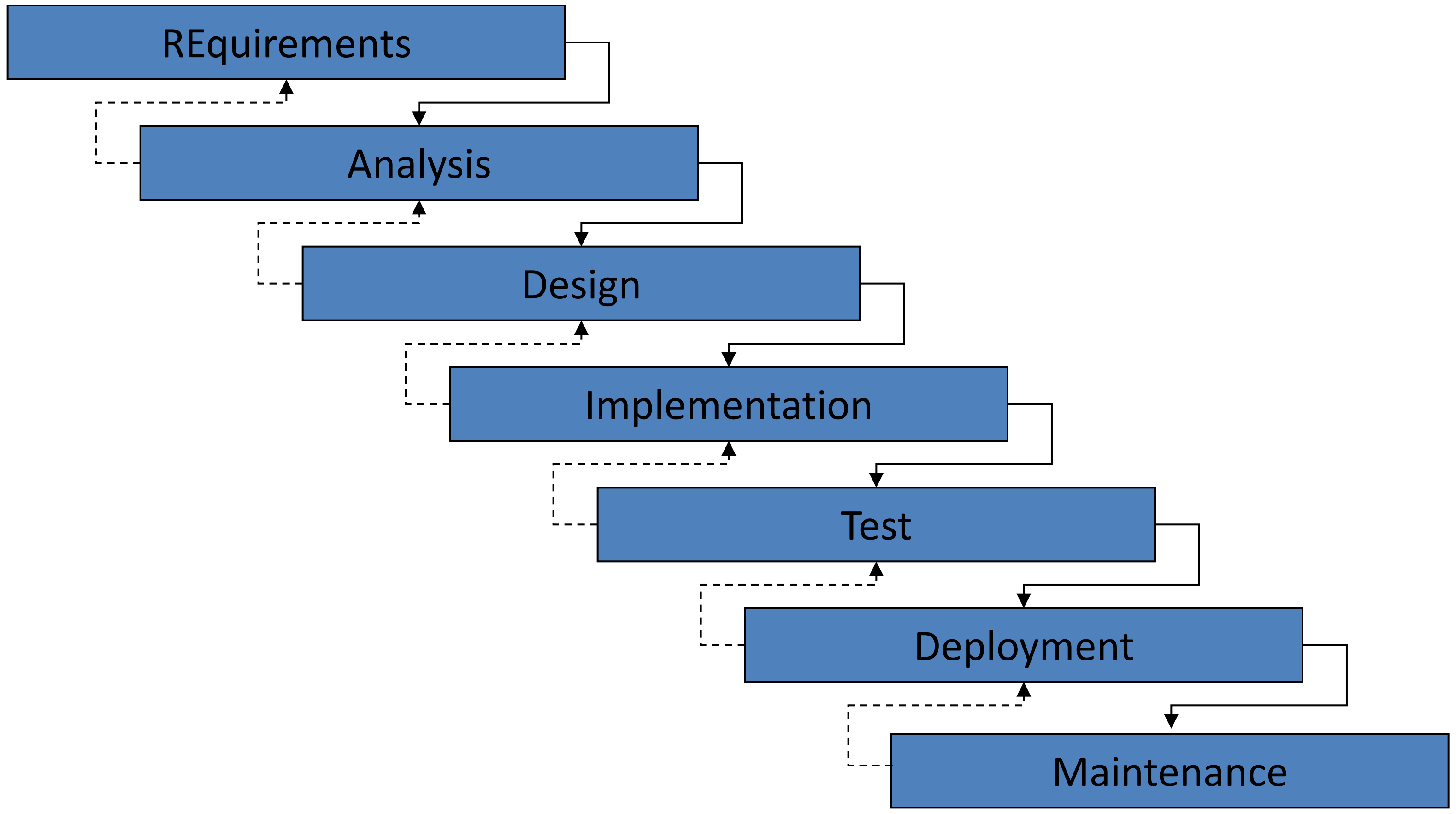
Tests - JUnit

Development stages (classical model)



Quick Prototyping

- Short development cycles
- Quick *feature* integration and demonstration
- Partial re-evaluation of the objectives in each iteration
- Several current approaches are based on these principles

Extreme Programming

- Tests first:
  - Forces a clear definition of the class
  - Allows continuous test during development

- Details:
  - Chapter 16 [Eckel 2002]

Tests

- *Test Driven Development (TDD):*
  - No line of code is written unless a test fails if it is absent
  - Eliminate redundancy
  - Regression tests: Reaply tests whenever code changes.

JUnit

- Library / Platform (not a standard, … yet), to ease test implementation in Java
- Authors: Erich Gamma e Kent Beck.
- More info: http://junit.org.

JUnit annotations

`@Test`

Method contains a test

`@Before`
`@After`

Methods (public void and with no parameters) to execute always before / after a class test

`@BeforeClass`
`@AfterClass`

Methods (public void and with no parameters) to execute always before / after each test method

`@ignore`

Method to be ignored

`@Test(expected= …Exception.class)`

Should fail throwing the expected exception

`@Test(timeout=100)`

Should fail if not finished within the timeout (100ms)

Main methods and classes

- `static` methods in class class `Assert`: `assertTrue, assertFalse, assertEquals, assertSame, assertArrayEquals, assertNull, assertNotNull`
- `static` method in class `Assume` (check pre-conditions)
- Implementer of interface MethodRule (e.g. TimeOut allows a timer for a test-class)
- To start tests:

```
public static void main(String args[]) {
    org.junit.runner.JUnitCore.main("TestX");
}
```

TestX is the name of the class containing the tests. Usually done automatically by the IDE.

Example

```
@Before
public void setUp() throws Exception {
    isa = new InstructionSetArchitecture("testarch.xml");
    isa.load("testprogram.asm");
}

@Test
public void testGetRegisters() {
    assertNonNull(isa);
    assertNonNull(isa.getRegisters());
    assertEquals(isa.getRegisters().size(), 6);
}

@Test (expected= IllegalRegisterAddressed.class)
public void testRegisterBankFail() throws IllegalRegisterAddressed {
    assertNonNull(isa.getRegisters().getRegisterByName("R5"));
}
```

Good practice: tests

- Each test unit should test highly related classes, typically one test per class
- Create tests before the tested class
- Always check limit situations (null references, empty Strings, limit numerical values, etc.)

Good practice: tests

- Tests should stick to the public interface
- Avoid changing the non-private interface of a class after the first version
- If interface is changed review all tests and documentation
- When changing review the invariant conditions

References

- Y. Daniel Liang, *Introduction to Java Programming*, 7.ª edição, Prentice-Hall, 2008.

Summary

- Tests
- JUnit