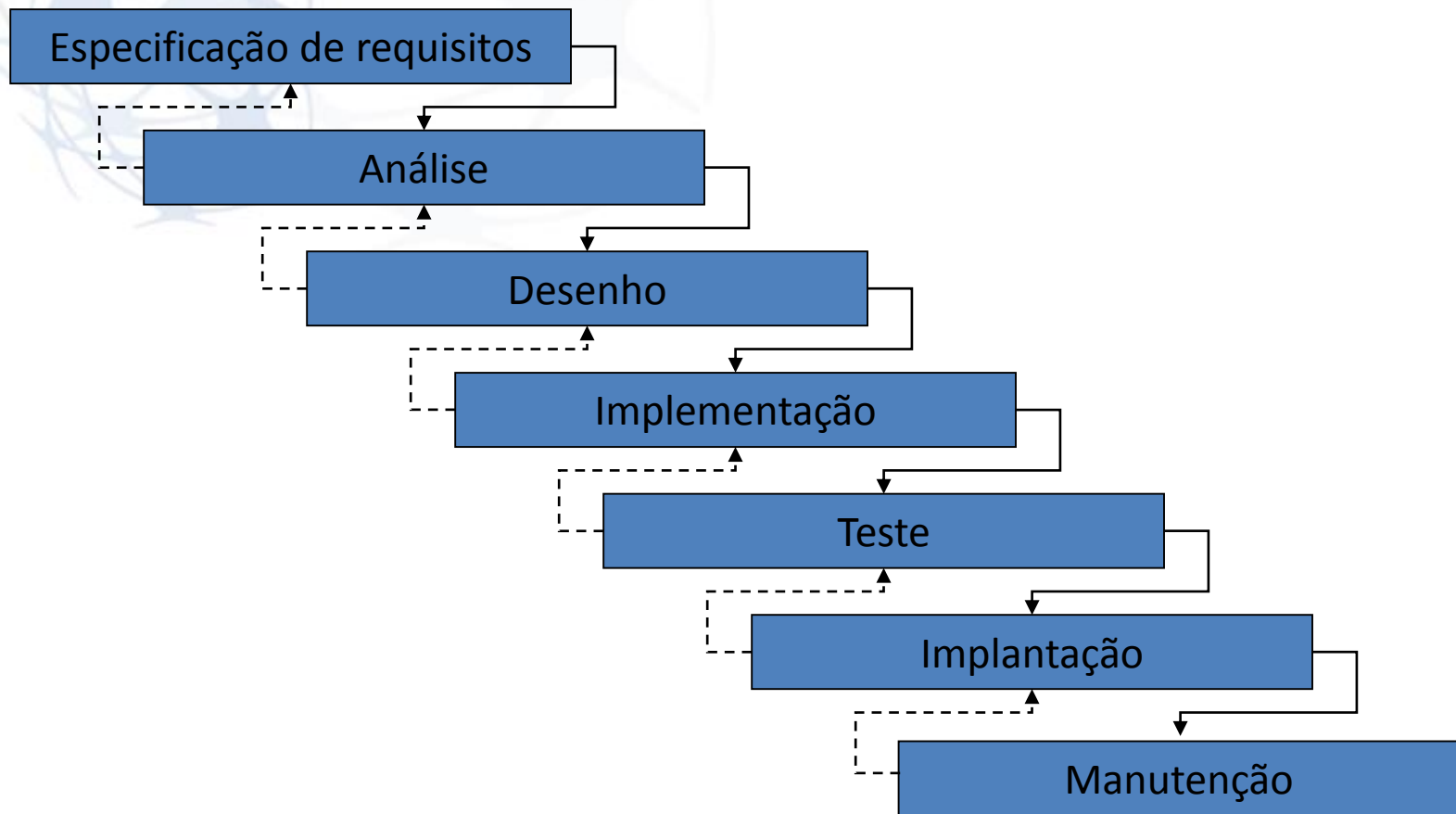


Testes

Introdução à JUnit

Etapas de Desenvolvimento (modelo clássico)



Prototipagem rápida

- Ciclos de desenvolvimento completos de curta duração
- Integração rápida de *features* no protótipo e demonstração
- Re-avaliação parcial dos objetivos e desenho em cada iteração
- Várias abordagens atuais têm por base alguns destes princípios

Extreme Programming

- Escrever os testes primeiro:
 - Força uma definição clara da classe
 - Permite o teste contínuo ao longo do desenvolvimento
- Mais detalhes:
 - Capítulo 16 [Eckel 2002]

Testes

- *Test Driven Development (TDD):*
 - Não escreva uma linha de código a não ser que isso vá corrigir um erro num teste
 - Elimine redundâncias
 - Testes de regressão (*regression testing, regressive tests*) Re-aplicação de testes sempre que há uma alteração ao código.

JUnit

- Biblioteca / plataforma (não standard, ... ainda), para facilitar a escrita de testes em Java
- Autores: Erich Gamma e Kent Beck.
- Mais informações: <http://junit.org>.

Principais anotações suportadas (JUnit)

`@Test`

Método que contém um teste

`@Before`

`@After`

Métodos (public void) a executar sempre antes / depois de um teste da classe

`@BeforeClass`

`@AfterClass`

Métodos (public static void, e sem argumentos) a executar uma vez antes / depois dos restantes testes da classe

`@ignore`

Método a ignorar nos testes

`@Test(expected= ...Exception.class)`

Método que deve falhar lançando a exceção indicada

`@Test(timeout=100)`

Método que deve falhar caso não tenha um resultado ao fim de 100ms

Principais métodos e classes

- Métodos static da classe Assert (verificar condições que devem ser verdadeiras para que o teste tenha sucesso): assertTrue, assertFalse, assertEquals, assertSame, assertArrayEquals, assertNull, assertNotNull
- Métodos static da classe Assume (verificar pré-condições)
- Classes que implementam o interface MethodRule (por exemplo Timeout que permite definir um temporizador para uma classe de testes)
- Para iniciar os testes:

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

TestX é o nome da classe que contém os testes.

Exemplo

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

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

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

Boas práticas: testes

- Para cada unidade (classe ou conjunto de classes intimamente ligadas) crie um caso de teste JUnit
- Crie os testes antes mesmo de começar a desenhar a unidade: pensar nos testes ajuda a perceber melhor que interface deve ter a unidade
- Teste sempre as situações limite e casos especiais (referências nulas, cadeias de caracteres vazias, valores numéricos no limite, etc.)

Boas práticas: testes

- Lembre-se que os testes se devem basear apenas na interface não privada da unidade (testes de caixa preta)
- Evite alterar a interface não privada quando fizer alterações a uma unidade
- Se precisar de alterar a interface não privada de uma unidade, actualize todos testes e reveja todos os contratos e respectiva documentação
- Quando alterar a implementação de uma unidade, reveja o seu invariante e execute todos os seus testes

Mais informação / Referências

- Y. Daniel Liang, *Introduction to Java Programming*, 7.^a edição, Prentice-Hall, 2008.
- Tutoriais sobre JUnit (procure Tutorials JUnit 4)
 - <http://www.vogella.com/tutorials/JUnit/article.html>
 - <http://www.tutorialspoint.com/junit/>
 - <http://java.dzone.com/articles/junit-tutorial-beginners>
 - <http://courses.cs.washington.edu/courses/cse143/11wi/eclipse-tutorial/junit.shtml>
 - <https://netbeans.org/kb/docs/java/junit-intro.html>
 - <http://www.mkyong.com/tutorials/junit-tutorials/>

Sumário

- Testes
- JUnit