



Testes de Software

1

AULA 01 – INTRODUÇÃO A TESTES DE SOFTWARE

joao.augusto@ifsc.edu.br

Conteúdo Programático do Curso

2

- Introdução a Testes de Software
- Técnicas de Testes de Software
- Elaboração e Execução dos Testes
- Aula prática com Ferramentas de Testes Unitários, Funcionais, Carga, Cobertura e Aceitação
- Aula prática com Ferramentas de Gerência dos Testes e Gerência dos Defeitos
- Discussão sobre melhoria no processo de desenvolvimento e teste de software do NTI

Conteúdo Programático – Aula 01

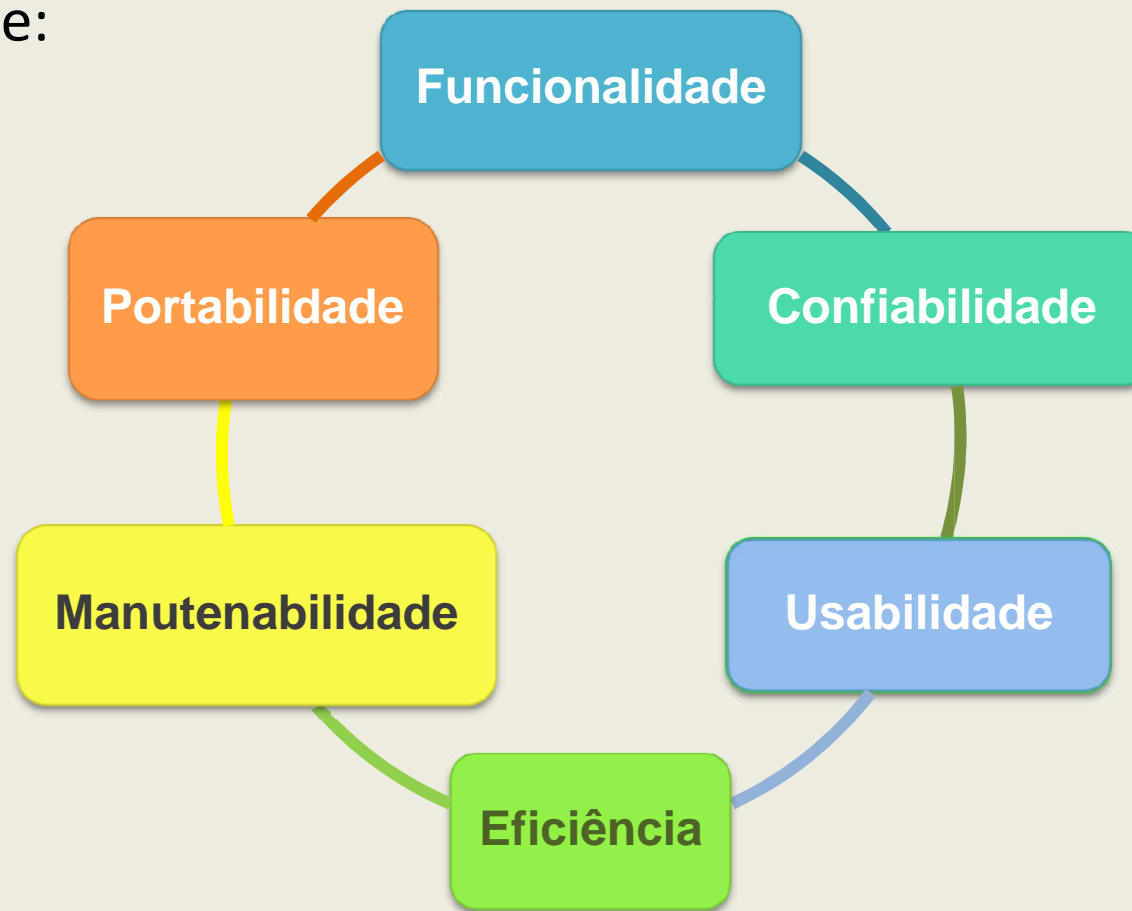
3

- Como adquirir qualidade em um software?
- O que é teste de software?
- Por que testar é necessário?
- Confiabilidade do Software x Defeitos
- Tipos de Teste de Software
- Níveis de Teste de Software
- Testes Funcionais x Testes de Unidade
- A Equipe de Testes
- Processo de Testes x Processo de Desenvolvimento
- Quando usar ferramentas de teste de software?
- Quando os testes devem ser automatizados?
- Considerações Finais
- Referências

Como adquirir qualidade em um software?

4

- A Norma ISO 9126 define as seguintes características para qualidade:



O que é teste de software?

5

- Os testes são realizados com a intenção de descobrir **erros e defeitos** em um sistema. [Myres, 2004]
- Os testes de software podem ser usados para mostrar a presença de defeitos, mas **nunca** para mostrar a **ausência** deles. [Dijkstra, 1972]
- Os testes de software servem para medir a confiabilidade de um sistema: à medida que **poucos defeitos** são encontrados em um determinado tempo, o software é considerado **mais confiável**.

Por que testar é necessário?

6

- Para assegurar que as necessidades dos usuários estejam sendo atendidas.
- Porque é provável que o software possua defeitos.
- Desenvolvedor já alocado para outro projeto teria que resolver muitos bugs de projetos anteriores em produção.
- Porque falhas podem custar muito caro.
- Para avaliar a qualidade do software.

Erro, Defeito e Falha

7

- **Erro:** é uma ação humana que produz um resultado incorreto.
- **Defeito:** A manifestação de um erro no software.
📖 Também conhecido como Bug
- **Falha:** quando o sistema se comporta de forma inesperada devido ao defeito.

Erro, Defeito e Falha

8



Uma pessoa
comete um
erro...

...que cria um
defeito no
software...



...que pode
causar uma
falha na
operação.



Conceitos Básicos de Teste

Artefatos de Teste

- todo o conjunto de documentação gerado pelo processo de teste de software.

Caso de Teste

- é composto por um conjunto de entradas, por passos de execução e um resultado esperado.

Roteiro de Teste

- É composto por um conjunto de casos de teste definidos para uma determinada especificação.

Conceitos Básicos de Teste

10

Requisitos

- regras de negócio do sistema.

Testar

- descobrir falhas através da execução do sistema.

Bug

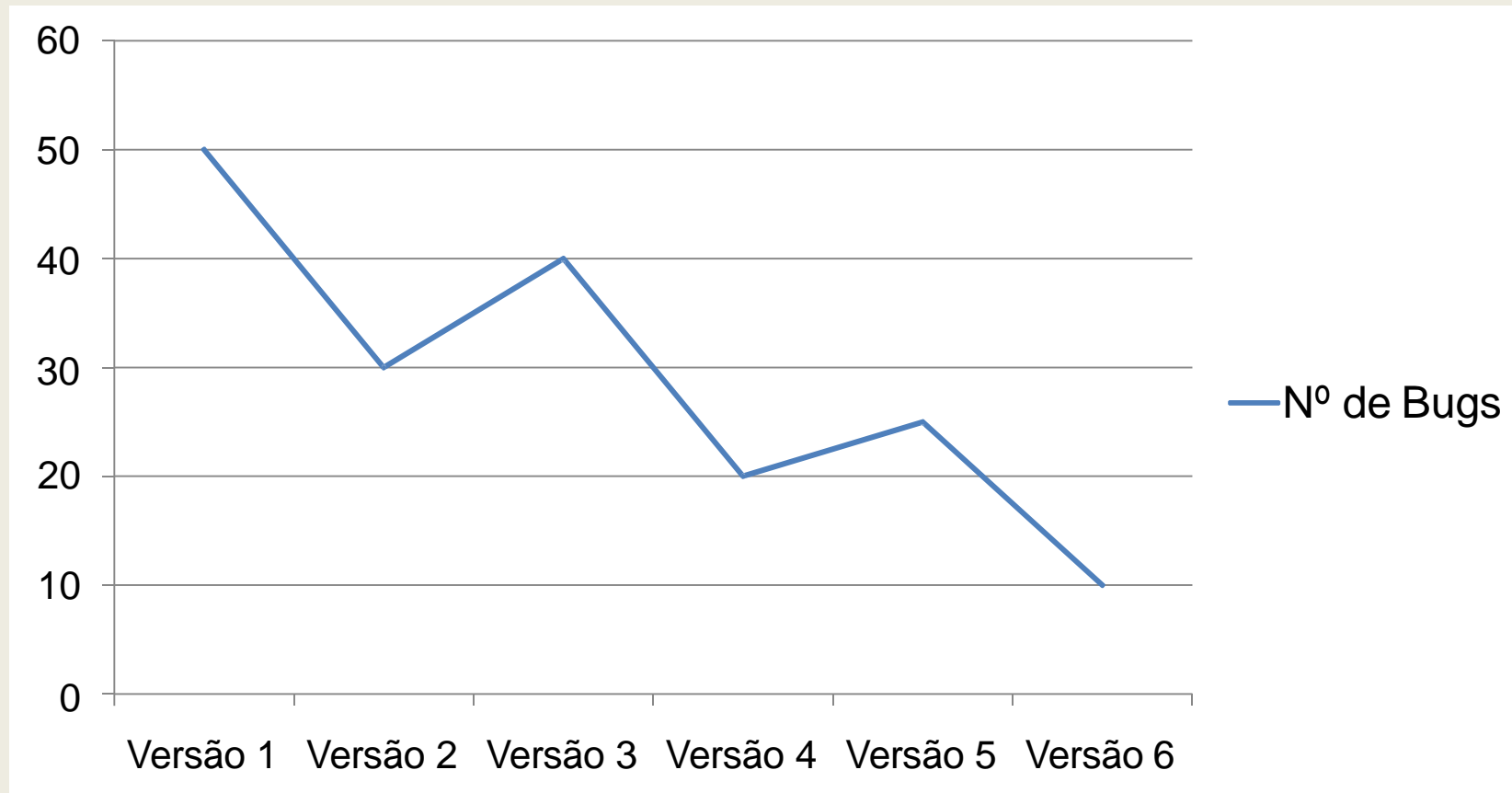
- é um defeito encontrado no sistema em execução.

Confiabilidade do Software

Confiabilidade do Software é a probabilidade que o software não causará uma falha no sistema por um tempo especificado, sob condições determinadas.

Confiabilidade do Software

12



O custo de um defeito

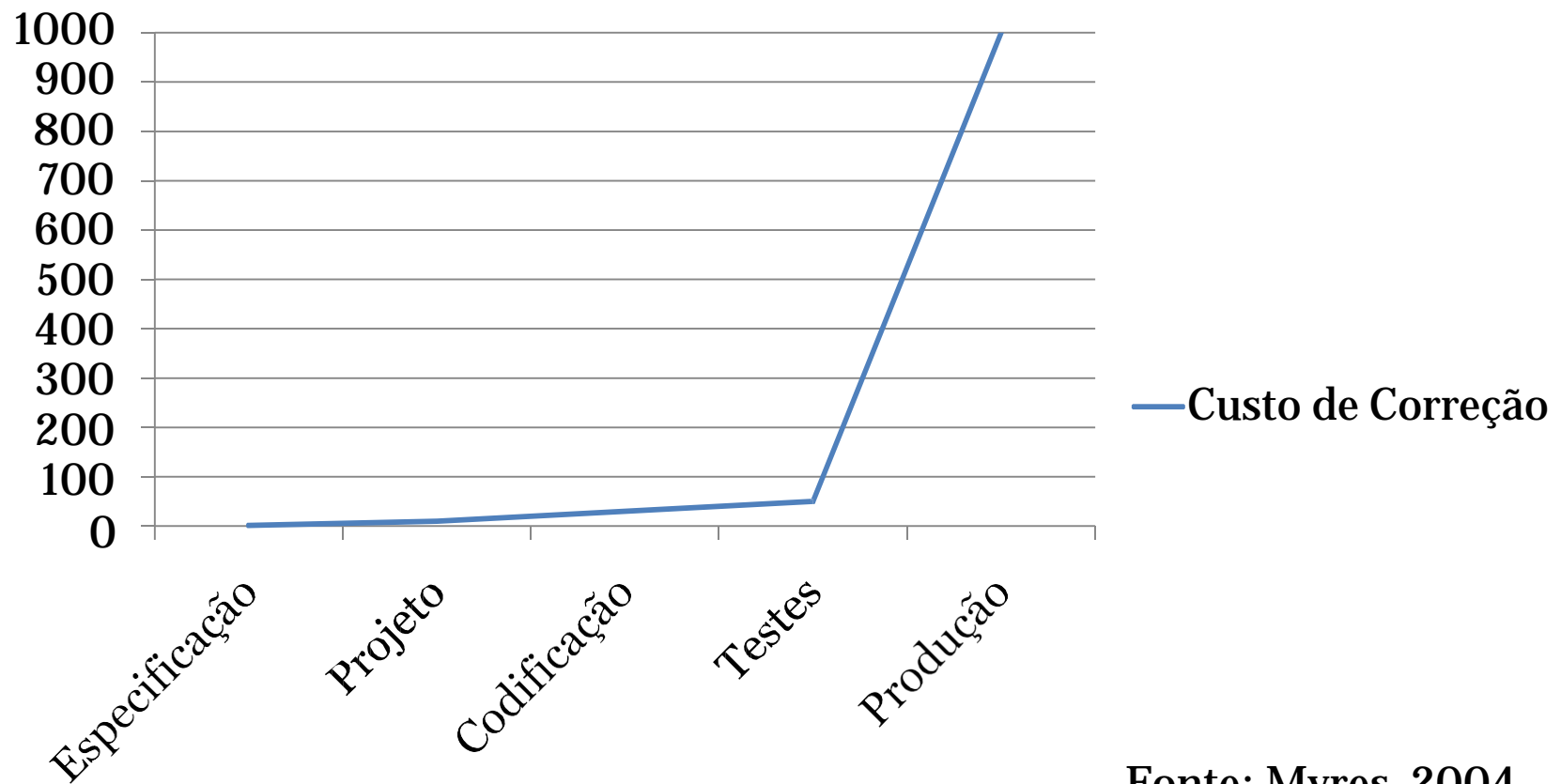


O **custo** da correção de um defeito tende a ser cada vez **maior** quanto **mais tarde** ele for descoberto. [Myres, 2004]

O custo de um defeito



14



Fonte: Myres, 2004



Desastres causados por erros em softwares

15

- Em 1996 - Um software com uma exceção não tratada foi responsável pela explosão do **foguete Ariane-5**, quando a 40 seg após a iniciação da seqüência de vôo, o foguete se desviou de sua rota, partiu e **explodiu**, tendo um prejuízo de 500 milhões de dólares.





Desastres causados por erros em softwares

16

- Em 2000 - Erro de cálculo no **sistema de radioterapia**, que era utilizado para controlar a emissão de radiação em tratamentos de câncer **matou 8 pessoas e causou queimaduras graves em outras 20.**



Mitos sobre os Testes

17

O testador é inimigo do desenvolvedor.

Os testadores devem ser os desenvolvedores menos qualificados.

O sistema está pronto quando o desenvolvedor termina de codificar.

Um programador consegue testar eficientemente o próprio código.

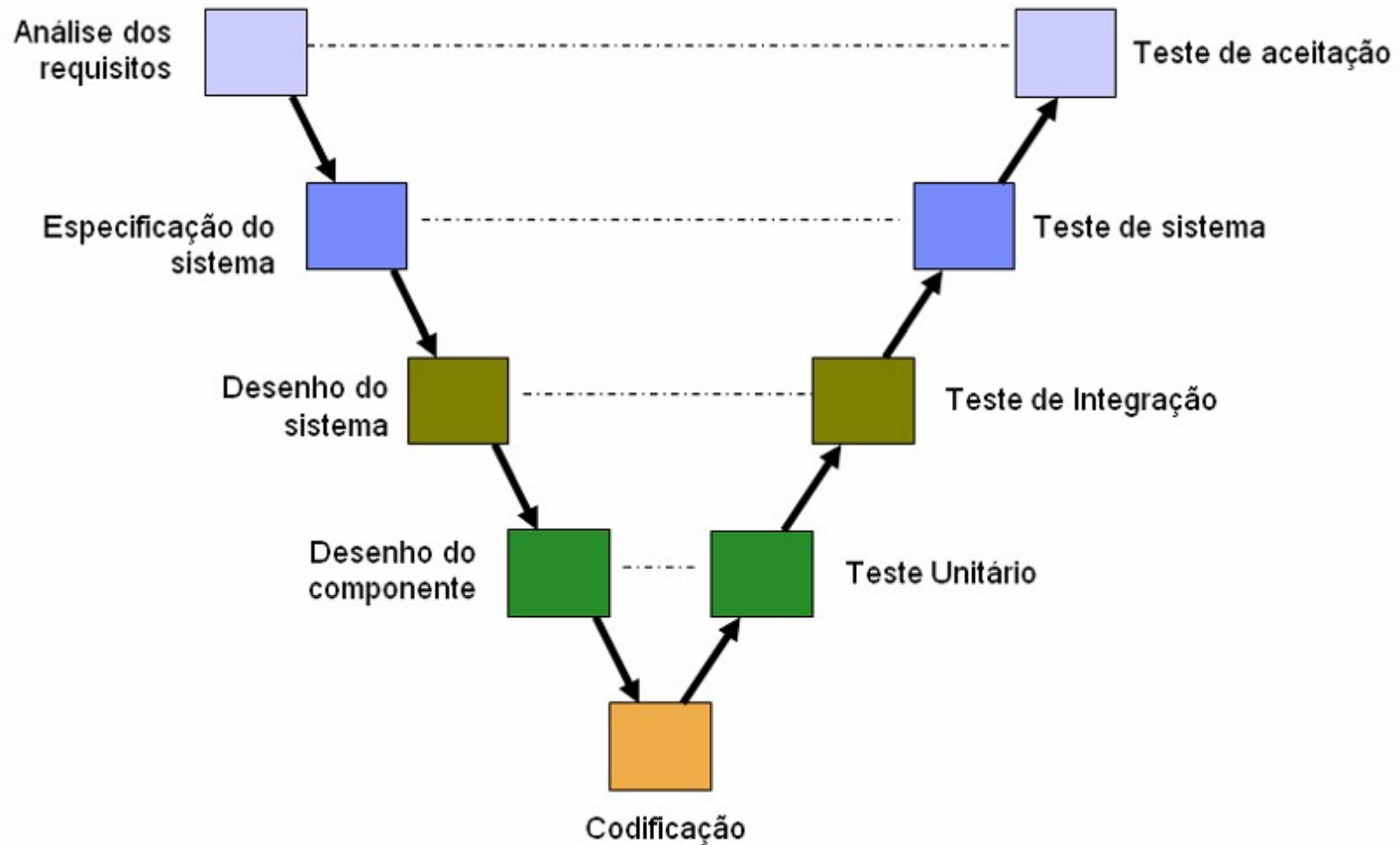
Tipos de Teste de Software

- Testes de Caixa-Branca (Estrutural)
 - Testes de Unidade
 - Teste de Integração
- Testes de Caixa-Preta (Funcional)
 - Testes Funcionais
 - Testes de Aceitação
 - Testes Exploratórios
- Testes de Caixa-Cinza
 - Testes de Regressão
 - Testes de Cobertura



Níveis de Teste de Software

19



Níveis de Teste de Software

20

Atributos	Nível dos Testes			
	Testes Unitários	Testes de Integração	Testes de Sistema	Testes de Aceitação
Escopo	<i>Unidades</i>	<i>Conjunto de unidades agrupadas</i>	<i>Sistema todo</i>	<i>Sistema todo</i>
Equipe	<i>Desenvolvedores</i>	<i>Desenvolvedores e Analistas de Sistema</i>	<i>Analista de Testes e Testadores</i>	<i>Analista de Testes, Testadores e Usuários</i>
Origem dos dados	<i>Criação manual</i>	<i>Criação manual</i>	<i>Criação automática / dados reais</i>	<i>Dados reais</i>
Volume dos dados	<i>Pequeno</i>	<i>Pequeno</i>	<i>Grande</i>	<i>Grande</i>
Interfaces	<i>Não existem</i>	<i>Não existem</i>	<i>Simuladas / Reais</i>	<i>Reais</i>
Ambientes	<i>Desenvolvimento</i>	<i>Desenvolvimento</i>	<i>Testes</i>	<i>Testes / Produção</i>

- **Testes de Unidade**

- ▮ Teste estrutural ou Caixa-branca
- ▮ Teste realizado em uma unidade ou componente para verificar sua corretude
 - × Ex.: Teste para uma classe ou métodos do sistema.
- ▮ Realizado pelo **desenvolvedor** que codificou o componente
- ▮ Para Java, existe a ferramenta **JUnit**
- ▮ Realizado de forma **automática**

Níveis de Teste de Software



- Exemplo de Teste de Unidade

The screenshot shows an IDE with two main panes. The left pane displays the JUnit test results, and the right pane shows the source code for the test.

JUnit Test Results (Left Pane):

- Finished after 0,069 seconds
- Runs: 4/4
- Errors: 1
- Failures: 0
- test.CalculadoraTest [Runner: JUnit 4] (0,018 s)
 - testAdicao (0,003 s)
 - testSubtracao (0,002 s)
 - testMultiplicacao (0,003 s)
 - testDivisao (0,010 s) - **Failed**

Failure Trace (Bottom Left):

```
java.lang.ArithmeticException: / by zero
at code.Calculadora.divisao(Calculadora.java:22)
at test.CalculadoraTest.testDivisao(CalculadoraTest.java:40)
```

Source Code (Right Pane):

```
package test;

import junit.framework.Assert;
import org.junit.Before;
import org.junit.Test;
import code.Calculadora;

public class CalculadoraTest {

    private Calculadora calculadora;

    @Before
    public void setUp() throws Exception {
        calculadora = new Calculadora();
    }

    @Test
    public void testAdicao() {
        Assert.assertEquals(1, calculadora.adicao(1, 0));
        Assert.assertEquals(-11, calculadora.adicao(-1, -10));
    }
}
```

- **Testes de Integração**

- ▣ Teste estrutural ou Caixa-branca

- ▣ Tem a finalidade de verificar se ao juntar vários componentes do sistema, se eles se comunicam corretamente.

- ▣ A interface entre as unidades é testada

- **Testes de Integração**

- ▣ Realizado pelos **desenvolvedores** ou analistas de sistema para testar um **módulo do sistema**.
- ▣ Utiliza 'Stubs' para simular módulos que ainda não foram implementados, mas que se comunicam com o módulo a ser testado.
- ▣ Realizado de forma **automática**

- **Testes de Sistema**

- Teste funcional ou Caixa-preta

- Tem por objetivo verificar se o sistema está em conformidade com a especificação de requisitos

- Realizado pelo **testador**, o qual tem acesso apenas a interface do sistema.

● Testes de Sistema

- ▣ O testador não faz parte da equipe de desenvolvimento.
- ▣ Os testes geralmente são baseados em **roteiros de teste** criados a partir da **especificação**.
- ▣ Pode ser realizado de forma **manual** ou **automática**.
- ▣ Existe várias ferramentas, como: Selenium, Watir, Badboy etc

- **Testes de Aceitação**

- Teste funcional ou Caixa-preta

- Tem por objetivo verificar se o sistema está em conformidade com os requisitos esperados pelo cliente

- Realizado pelo **cliente** ou pelo **testador**, desde que este possua um **checklist** feito pelo cliente do que é esperado que haja no sistema.

- Realizado no **ambiente de homologação**

● Testes de Aceitação

- ▣ O sistema é utilizado para capacitação dos usuários de forma que eles validem todos os requisitos do sistema

- ▣ Realizado de forma **manual** ou **automática**

- ▣ Existe a ferramenta EasyAccept.

- ▣ Teste realizado pelo usuário pode ser:
 - × **Teste Alfa:** em um ambiente de homologação.
 - × **Teste Beta:** em um ambiente de produção.

Tipos de Teste de Software

- **Testes Exploratórios**

- Teste funcional ou Caixa-preta

- Realizado por **testadores com experiência**

- Quando não há muita documentação sobre o sistema

- Realizado de forma **manual**

- Os defeitos encontrados são reportados à medida que eles ocorrem

Tipos de Teste de Software

- **Testes de Regressão**

- Teste funcional ou estrutural Caixa-cinza

- À medida que uma nova versão do sistema é liberada, novos bugs podem ser incluídos no sistema

- Tem a finalidade de **realizar novamente testes** em um sistema já testado

- Realizado pelo testador

- Pode ser realizado de forma **manual** ou **automática**

Tipos de Teste de Software

- **Testes de Cobertura**

- Teste funcional ou estrutural Caixa-cinza

- **Estrutural:** Tem a finalidade de identificar se os testes realizados no sistema abrangem pelo menos 95% do código produzido.

- **Funcional:** Os roteiros de teste abrangem 100% das funcionalidades do sistema, ou seja, possui pelo menos 1 caso de teste para cada regra de negócio.

Testes Funcionais x Testes de Unidade

32

Funcionais

- Teste de Caixa-Preta
- Manual ou Automático
- Testador diferente do desenvolvedor faz os testes
- Testador acessa o sistema via interface do usuário
- Testes verificam se o sistema **faz o que deve fazer** e **não faz o que não deve fazer**

Unitário

- Teste de Caixa-Branca
- Automático
- **Desenvolvedor faz os testes**
- Testador tem acesso ao código
- Testes verificam a corretude de cada unidade (método, classe) de forma independente

A Equipe de Testes

- Gerente de teste

- ▣ Lidera a equipe de teste.

- ▣ Comunicação entre a equipe de teste e de desenvolvimento.

- ▣ Planeja os testes, define estratégias, etc.



A Equipe de Testes

34

- Arquiteto de teste
 - ▣ Conhece os requisitos do sistema.
 - ▣ Elabora os roteiros de teste.



A Equipe de Testes

- Testador


- É criativo ao executar os testes.
- Tem noções de programação.
- É objetivo ao descrever um erro.
- É perfeccionista.





O Testador

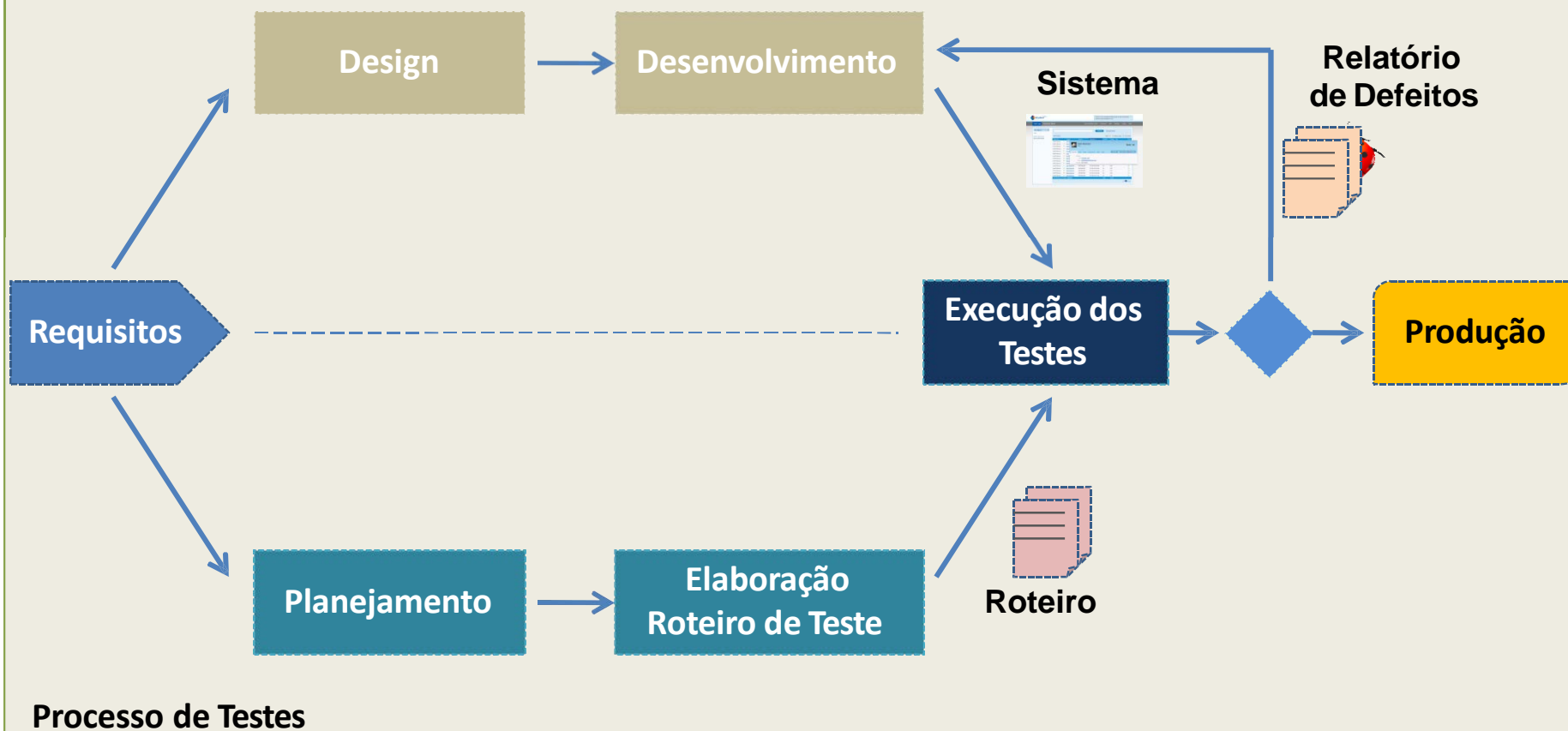
36

- 
1. Não deve testar seu próprio programa.
 2. Não deve duvidar que um erro existe.
 3. Deve ter cuidado para não reportar falsos bugs.
 4. O testador não é inimigo do desenvolvedor.
 5. O testador deve saber se comunicar com o desenvolvedor.
 6. Os bugs descritos por ele devem ser baseados em fatos.
 7. Um bom testador é aquele que encontra muitos bugs!

Processo de Testes x Processo de Desenvolvimento

37

Processo de Desenvolvimento



Quando usar ferramentas de teste de software?

- Quando há apenas 1 testador para o projeto
 - Deve realizar apenas testes manuais.
 - Não há necessidade de ferramentas para criar os roteiros de teste.
 - Ferramenta apenas para reportar os defeitos. (**Redmine**)
- Quando há uma equipe de teste
 - Ferramenta para gerenciar a equipe. (**Redmine**)
 - Ferramenta para criar os roteiros de teste. (**TestLink**)
 - Ferramenta para reportar e gerenciar os defeitos. (**Redmine**)
 - Automatizar os testes para auxiliar nos testes de regressão. (**Selenium**)

Quando os testes devem ser automatizados?

39

- Frequência de execução dos testes.
- As funcionalidades são testadas mais de uma vez.
- Baixo esforço para automatizar (equipe experiente).
- Ferramentas de automação relevantes para a realidade do projeto.
- Dificuldade de executar os testes manualmente.



Implantando Testes de Software

40

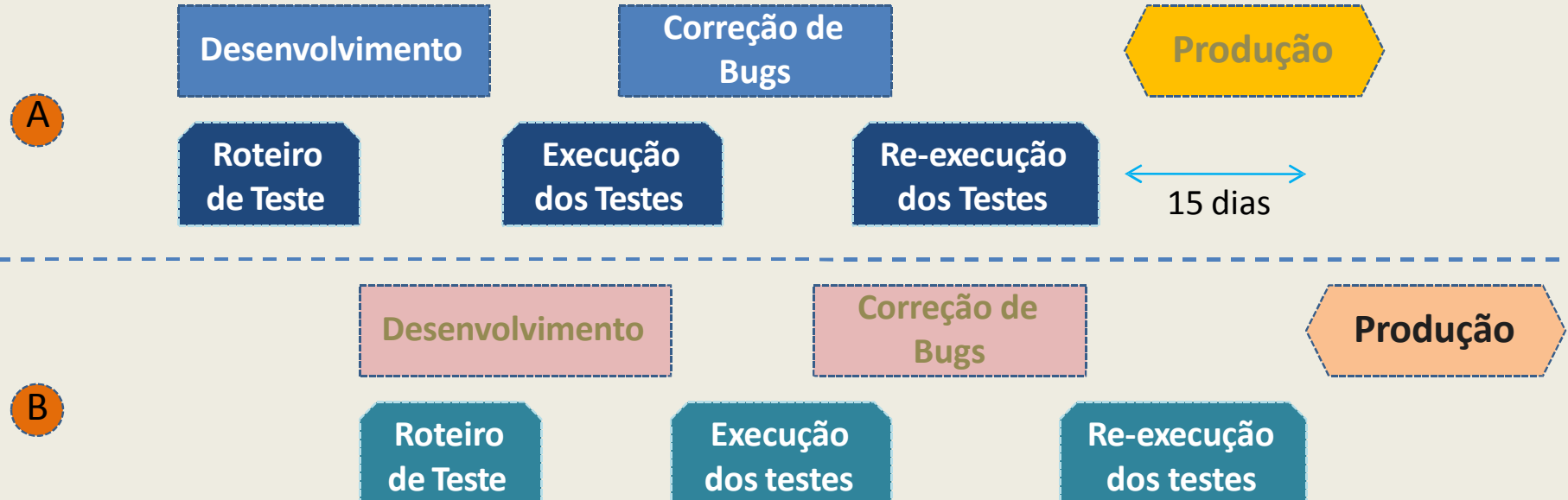
- Montar uma equipe de teste.
 - ▣ O tamanho da equipe depende da quantidade de projetos a serem testados.
- Qualificar a equipe de testes com treinamentos.



Implantando Testes de Software

41

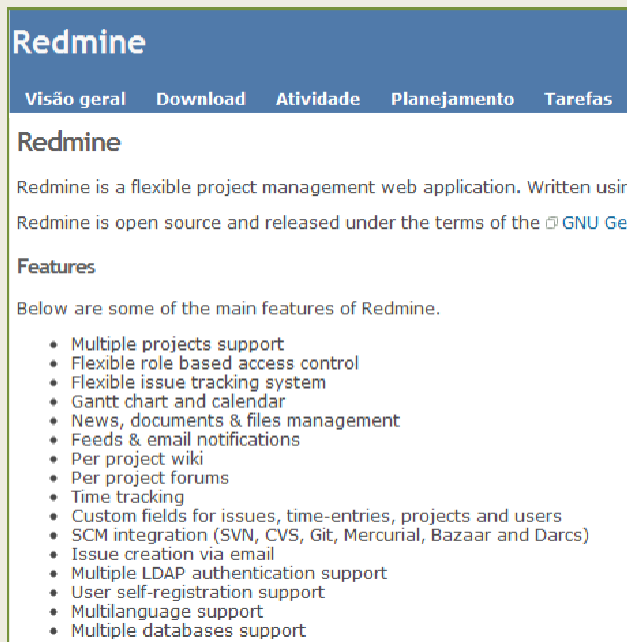
- Quando a equipe de teste é pequena e precisa testar 2 projetos.
- Então, deve-se planejar a entrega de cada projeto em datas diferentes.



Implantando Testes de Software

42

- Iniciar com testes manuais.
- Usar ferramenta para **gerenciar os defeitos** [Redmine] e para gerenciar os testes [TestLink].



Redmine

Visão geral Download Atividade Planejamento Tarefas

Redmine

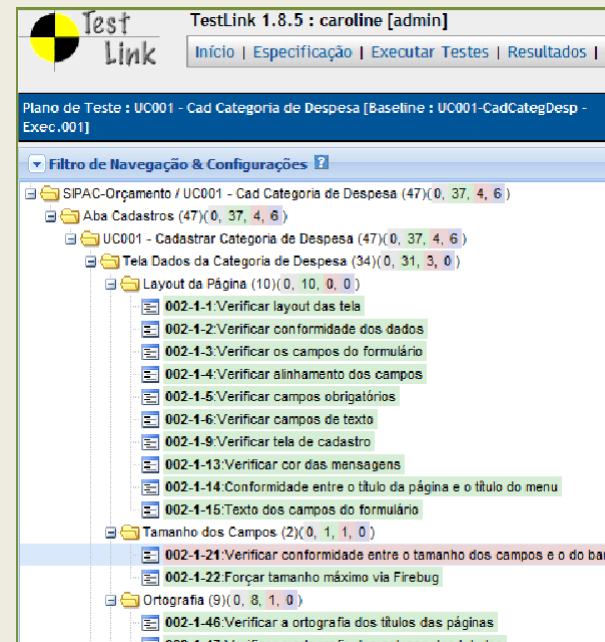
Redmine is a flexible project management web application. Written using Ruby on Rails framework, it is open source and released under the terms of the GNU General Public License.

Features

Below are some of the main features of Redmine.

- Multiple projects support
- Flexible role based access control
- Flexible issue tracking system
- Gantt chart and calendar
- News, documents & files management
- Feeds & email notifications
- Per project wiki
- Per project forums
- Time tracking
- Custom fields for issues, time-entries, projects and users
- SCM integration (SVN, CVS, Git, Mercurial, Bazaar and Darcs)
- Issue creation via email
- Multiple LDAP authentication support
- User self-registration support
- Multilanguage support
- Multiple databases support

Redmine



TestLink 1.8.5 : caroline [admin]

Início | Especificação | Executar Testes | Resultados | Usar

Plano de Teste : UC001 - Cad Categoria de Despesa [Baseline : UC001-CadCategDesp - Exec.001]

Filtro de Navegação & Configurações ?

- [-] SIPAC-Orçamento / UC001 - Cad Categoria de Despesa (47)(0, 37, 4, 6)
- [-] Aba Cadastros (47)(0, 37, 4, 6)
- [-] UC001 - Cadastrar Categoria de Despesa (47)(0, 37, 4, 6)
- [-] Tela Dados da Categoria de Despesa (34)(0, 31, 3, 0)
- [-] Layout da Página (10)(0, 10, 0, 0)
- [-] 002-1-1:Verificar layout das tela
- [-] 002-1-2:Verificar conformidade dos dados
- [-] 002-1-3:Verificar os campos do formulário
- [-] 002-1-4:Verificar alinhamento dos campos
- [-] 002-1-5:Verificar campos obrigatórios
- [-] 002-1-6:Verificar campos de texto
- [-] 002-1-9:Verificar tela de cadastro
- [-] 002-1-13:Verificar cor das mensagens
- [-] 002-1-14:Conformidade entre o título da página e o título do menu
- [-] 002-1-15:Texto dos campos do formulário
- [-] Tamanho dos Campos (2)(0, 1, 1, 0)
- [-] 002-1-21:Verificar conformidade entre o tamanho dos campos e o do banco
- [-] 002-1-22:Forçar tamanho máximo via Firebug
- [-] Ortografia (9)(0, 8, 1, 0)
- [-] 002-1-46:Verificar a ortografia dos títulos das páginas
- [-] 002-1-47:Verificar a ortografia das colunas das tabelas

TestLink

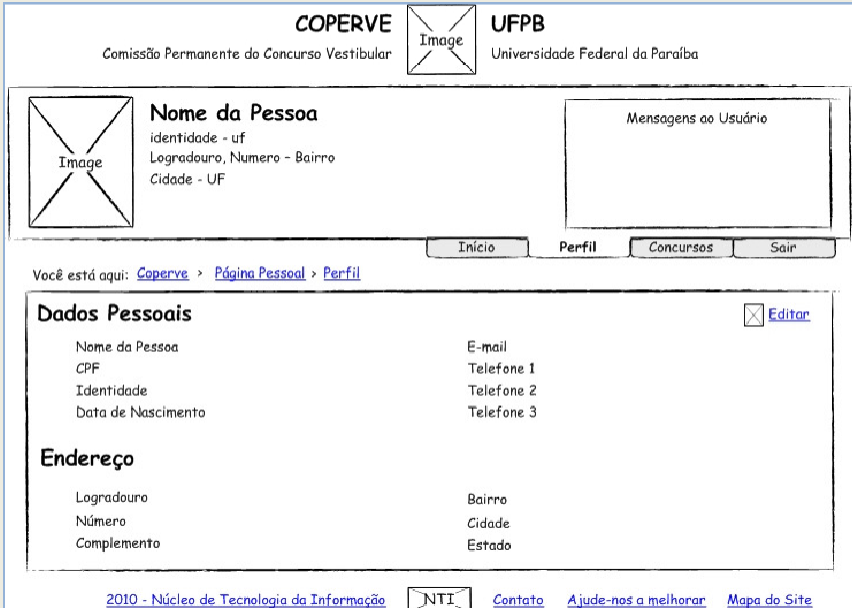
Implantando Testes de Software

43

- Sistemas devem possuir uma **especificação básica** necessária para criar os roteiros de teste e um protótipo de cada tela.
- Os **protótipos** de tela devem ser validados pelo cliente antes do início do desenvolvimento.

UC01.	Consulta do local de prova
1. Descrição	Um candidato inscrito no vestibular deverá poder consultar o local de prova na data pré-determinada.
2. Atores	Candidatos inscritos para o vestibular
3. Pré-condições	<ol style="list-style-type: none">1. Computador com acesso a internet.2. Sistema disponível.3. Usuário ter conseguido efetuar o login com sucesso.4. O período para consulta do local de prova esteja vigente.
4. Fluxo de Eventos	<p>4.1. Fluxo Principal</p> <ol style="list-style-type: none">1. O usuário se loga no sistema.2. O usuário visualiza a página principal do sistema com os links para os locais de prova dos concursos que ele se inscreveu. [A.01] [E.01]3. O usuário clica no link do concurso desejado a obter a informação.4. O usuário será redirecionado para uma página que mostra as informações do concurso selecionado, do candidato e do local de prova(Consultar wireframe referente a local de prova) e uma mensagem informando o que é permitido levar no dia da prova [E.02] [E.03]5. O usuário visualizará um botão no qual ele poderá imprimir a página. [E.04]

Especificação



COPERVE UFPB
Comissão Permanente do Concurso Vestibular Universidade Federal da Paraíba

Nome da Pessoa
identidade - uf
Logradouro, Numero - Bairro
Cidade - UF

Mensagens ao Usuário

Início Perfil Concursos Sair

Você está aqui: [Coperve](#) > [Página Pessoal](#) > [Perfil](#)

Dados Pessoais [Editar](#)

Nome da Pessoa	E-mail
CPF	Telefone 1
Identidade	Telefone 2
Data de Nascimento	Telefone 3

Endereço

Logradouro	Bairro
Número	Cidade
Complemento	Estado

2010 - Núcleo de Tecnologia da Informação [Contato](#) [Ajude-nos a melhorar](#) [Mapa do Site](#)

Protótipo

Considerações Finais

- Software com testes melhoram a credibilidade do setor de informática
 - Usuário mais satisfeito
- Desenvolvedor perderá menos tempo resolvendo bugs de sistemas em produção, enquanto está alocado em outro projeto
 - Desenvolvedor mais satisfeito e motivado
- Sistema só deve ser colocado em produção após aprovação da equipe de testes.

Considerações Finais

- A equipe de teste é parte da equipe de desenvolvimento.
- Cronogramas devem levar em consideração os testes.
- Processo de Teste deve ser integrado ao processo de desenvolvimento
- Testar reduz riscos do negócio

Referências

- [AllBusiness] AllBusiness - Site: <http://www.allbusiness.com/technology/computer-software/210053-1.html> Acessado em Maio/2011.
- [Delamaro, 2007] Delamaro, M., Maldonado, J. C., Jino, M. “Introdução ao Teste de Software”. Ed. Elsevier, Rio de Janeiro, 2007.
- [Dijkstra, 1972] Dijkstra, E. W. "The Humble Programmer". *Communications of the ACM* **15** (10): 859–866, 1972.
- [GTSW] Grupo de Testadores de Software - <http://gtsw.blogspot.com> Acessado em Maio/2011.
- [Molinari, 2008] Molinari, L. “Testes Funcionais de Software”. Ed. Visual Books. Florianópolis, 2008.
- [Myres, 2004] Myres , G. F. “The Art of Software Testing”. Ed. John Wiley & Sons, Inc. New Jersey, 2004.