Teste e Depuração de Sistemas Web

Prof. Fellipe Aleixo (fellipe.aleixo@ifrn.edu.br)

Testando Aplicações Web

- É bem mais complicado testar interfaces Web do que classes de objetos
- Aplicações WEB não permitem verificar facilmente se o conteúdo presente é o esperado
- Informações importantes que precisam ser avaliadas encontram-se misturadas com tags HTML

Testando Aplicações Web

- O layout das aplicações web mudam com frequência
- É preciso construir testes que não precisem ser reescritos toda vez que o layout das páginas mudem

Testando Aplicações Web

- Alguns problemas que dificultam a criação de testes automáticos para aplicações web:
  - Resubmissão de um formulário (por exemplo, o usuário submete um pedido de compra, enquanto o servidor ainda está processando a primeira submissão)
  - Implicações do uso do botão "Voltar" (muitas vezes, objetos perdem seus valores quando mudam de escopo)
  - Segurança, ataques do tipo DoS (Denial of Service)
  - Implicações do browser (valores existentes na memória cache do browser sendo usados sem o conhecimento do usuário)

Testando Aplicações Web

- Aplicações WEB são difíceis de testar pois dependem da comunicação com o *container*
- Para testar aplicações WEB precisamos fazer:
  - testes de unidade (white-box): testar pedaços de código;
  - testes funcionais (black-box): testar as funcionalidades do sistema do ponto de vista do usuário;
  - testes de integração: testar a integração com o container.

Testando as Camadas de Forma Isolada

- **Camada de Apresentação**
  - Os testes consistem em verificar os elementos da interface de uma página web.
  - Ex:
    - Fonte, layout de tela, cores, resolução gráfica, links, gramática, estética global, ortografia, precisão dos campos, valores padrão, etc.

Testando as Camadas de Forma Isolada

- Testes da Camada de Apresentação
  - Teste 1: Validar o Código
  - Teste 2: Verificar Links Ativos
  - Teste 3: Checar texto alternativo
  - Teste 4: Verificar acessibilidade
  - Teste 5: Verificar Usabilidade
  - Teste 6: Verificar formulários
  - Teste 7: Verificar entrada de dados inválidos
  - Teste 8: Redimensionar o navegador
  - Teste 9: Examinar os objetos de página da Web
  - Teste 10: Verificar o acesso não autorizado
  - Teste 11: Alterar configurações do navegador

Testando as Camadas de Forma Isolada

- Utilização de validadores
  - Conformidade aos padrões
    - W3C - http://validator.w3.org
  - Acessibilidade
    - HERA - http://www.sidar.org/hera/
    - AChecker - http://achecker.ca/checker

Testando as Camadas de Forma Isolada

- Teste de interface de rede
  - Teste 12: alterar a velocidade de acesso à rede
- Testes dos componentes do lado do cliente
  - Teste 13: Conferir Plug-Ins
  - Teste 14: Verificar o Ambiente

Testando as Camadas de Forma Isolada

- **Camada de Negócio**
  - Os testes da camada de negócios tem como objetivo encontrar erros na lógica do negócio
  - Pode ser utilizadas as técnicas de testes caixa branca e caixa preta
  - Os testes vão detectar erros nos requisitos de desempenho do aplicativo, aquisição de dados e processamento de transações e autenticação de usuários

Testando as Camadas de Forma Isolada

- Teste dos componentes do lado do servidor:
  - Teste 15: Teste de Carga
  - Teste 16: Teste de Stress
  - Teste 17: Teste de Performance

Testando as Camadas de Forma Isolada

- **Camada de Acesso a Dados**
  - Os testes da camada de dados procuram por erros da seguinte categoria: Tempo de resposta, integridade dos dados, tolerância a falha e recuperação de condições de erro
  - Teste 18: Teste de Volume

Tipos de Teste

- Cada tipo de teste tem foco em um objetivo particular, ele define o alvo do teste, que pode ser o teste de uma funcionalidade, a ser realizada pelo software ou uma característica não-funcional

Tipos de Teste

- Teste Funcional
  - Testa as funções, requisitos funcionais e casos de uso
- Teste de Segurança
  - Se dados e sistemas são acessados apenas pelos possíveis atores
- Teste de Configuração
  - Se o software funciona no hardware e plataformas propostas
- Teste de Performance
  - Fluxo de execução, acesso a dados, chamadas ao sistema, tempo de resposta são monitorados para a identificação de gargalos na performance e processos ineficientes

Tipos de Teste

- Teste de Carga
  - Avalia performance do software sobre condições normais de uso. Exemplo: número de transações por unidade de tempo, tempo de resposta, usuários simultâneos, etc.
- Teste de Volume
  - Como o sistema trabalha com grande volume de dados, transações, usuários, periféricos, por um longo período de tempo
  - Trabalha com o volume máximo requerido

Tipos de Teste

- Teste de Instalação
  - Se o software instala conforme planejado em diferentes hadwares/softwares e sobre condições diferentes
- Teste de Integridade
  - Testa robustez (resistência à falhas), conformidade com a linguagem, sintaxe e uso dos recursos
- Teste de Stress
  - Testa condições anormais, como por exemplo: picos excessivos de carga em pequenos períodos

Tipos de Teste

- Teste de Usabilidade
  - Foca em fatores humanos, estética, consistência da interface, “help”, “wizards”, documentação do usuário, treinamento, etc.
- Teste de Regressão
  - Repetição dos testes de um componente para verificar se modificações não causaram efeitos indesejáveis e se certificar que o componente ainda atende aos requisitos especificados
- Teste de Sanidade
  - Representa um conjunto de casos de teste que estabelece que o sistema é estável e todas as funcionalidades principais estão presentes e trabalham sob condições normais

Elaboração dos Testes

- **Cenário de Teste**: É o caminho a ser seguido ou a situação a ser testada
- **Caso de Teste**: É o cenário a ser executado para verificar se o que foi especificado está devidamente implementado

Elaboração dos Testes

- Aplicação Bancária
  - Requisito: Transferência Bancária
  - Cenário:
    - Consultar o saldo da conta origem
    - Consultar o saldo da conta destino
    - Transferir valor da conta origem para conta destino
      - Casos de teste:
        - valor <= saldo
        - valor > saldo
        - valor < 0 (zero)
    - Consultar novo saldo das contas

Testes no PLAY

Testando a sua Aplicação

- As classes de teste ficam na pasta "test" da sua aplicação
- Os testes são executados a partir do console do Play
  - "test" → para executar todos os testes
  - "test-only namespace.ClasseTeste" → para uma classe de teste
- Padrão para o teste de aplicações com o Play é o Junit
  - http://junit.org
  - https://www.youtube.com/watch?v=eSaeSMm_SeA

Utilizando JUnit

- Exemplo de teste JUnit:

```
package test;
import org.junit.*;
import play.mvc.*;
import play.test.*;
import play.libs.F.*;
import static play.test.Helpers.*;
import static org.fest.assertions.Assertions.*;

public class SimpleTest {

    @Test
    public void simpleCheck() {
        int a = 1 + 1;
        assertThat(a).isEqualTo(2);
    }
}
```

Testando com Aplicações "falsas"

- Se o código a ser testado depender um aplicação "rodando"

```
@Test
public void findById() {
  running(fakeApplication(), new Runnable() {
    public void run() {
      Computer macintosh = Computer.find.byId(211);
      assertThat(macintosh.name).isEqualTo("Macintosh");
      assertThat(formatted(macintosh.introduced)).isEqualTo("1984-01-24");
    }
  });
}
```

  - Passar detalhes adicionais para a criação da máquina "falsa" fakeApplication(inMemoryDatabase())

Escrevendo Testes Funcionais

- Testando um *template*:

```
@Test
public void renderTemplate() {
  Content html = views.html.index.render("Coco");
  assertThat(contentType(html)).isEqualTo("text/html");
  assertThat(contentAsString(html)).contains("Coco");
}
```

  - http://www.playframework.com/documentation/2.0.1/api/java/play/test/Helpers.html

Escrevendo Testes Funcionais

- Testando os seus controladores:

```
@Test
public void callIndex() {
    Result result = callAction(
      controllers.routes.ref.Application.index("Kiki")
    );
    assertThat(status(result)).isEqualTo(OK);
    assertThat(contentType(result)).isEqualTo("text/html");
    assertThat(charset(result)).isEqualTo("utf-8");
    assertThat(contentAsString(result)).contains("Hello Kiki");
}
```

Escrevendo Testes Funcionais

- Testando o "roteador":

```
@Test
public void badRoute() {
  Result result = routeAndCall(fakeRequest(GET, "/xx/Kiki"));
  assertThat(result).isNull();
}
```

Escrevendo Testes Funcionais

- Utilizando um servidor HTTP:

```
@Test
public void testInServer() {
  running(testServer(3333), new Runnable() {
      public void run() {
         assertThat(
           WS.url("http://localhost:3333").get().get().getStatus()
         ).isEqualTo(OK);
      }
  });
}
```

Escrevendo Testes Funcionais

- Testando através de um navegador Web:
  1. Utilizando a API FluentLenium
     - https://github.com/FluentLenium/FluentLenium

```
@Test
public void runInBrowser() {
    running(testServer(3333), HTMLUNIT, new Callback<TestBrowser>() {
        public void invoke(TestBrowser browser) {
           browser.goTo("http://localhost:3333");
           assertThat(browser.$("#title").getTexts().get(0)).isEqualTo("Hello Guest");
           browser.$("a").click();
           assertThat(browser.url()).isEqualTo("http://localhost:3333/Coco");
           assertThat(browser.$("#title", 0).getText()).isEqualTo("Hello Coco");
        }
    });
}
```

Selenium IDE

- Testando através de um navegador Web:
  1. Com o Selenium IDE
- Grava uma sequência de ações em um site Web e as executa na forma de teste
- https://www.youtube.com/watch?v=bZA_auUmoDo