6 Estudo de Caso

Nesse capítulo será apresentada a utilização no sistema A do processo de teste descrito nesta dissertação. Serão apresentadas as dificuldades e as vantagens encontradas durante a execução do processo em um sistema já em produção. Para a mesma funcionalidade utilizada pelo novo processo, foi realizado um processo de teste manual com intuito de comparar os dois.

6.1.Processo de Teste

A funcionalidade do sistema XPTO que foi testada é a inserção de uma seção de duto. Essa funcionalidade é bem simples, sendo responsável por inserir uma seção de duto de forma adequada. A interface que permite essa inserção está apresentada na figura 57.

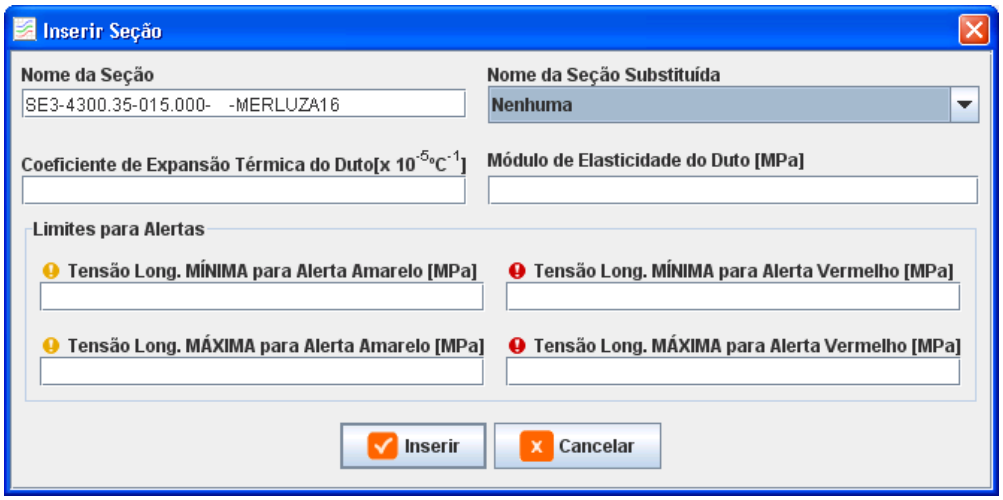
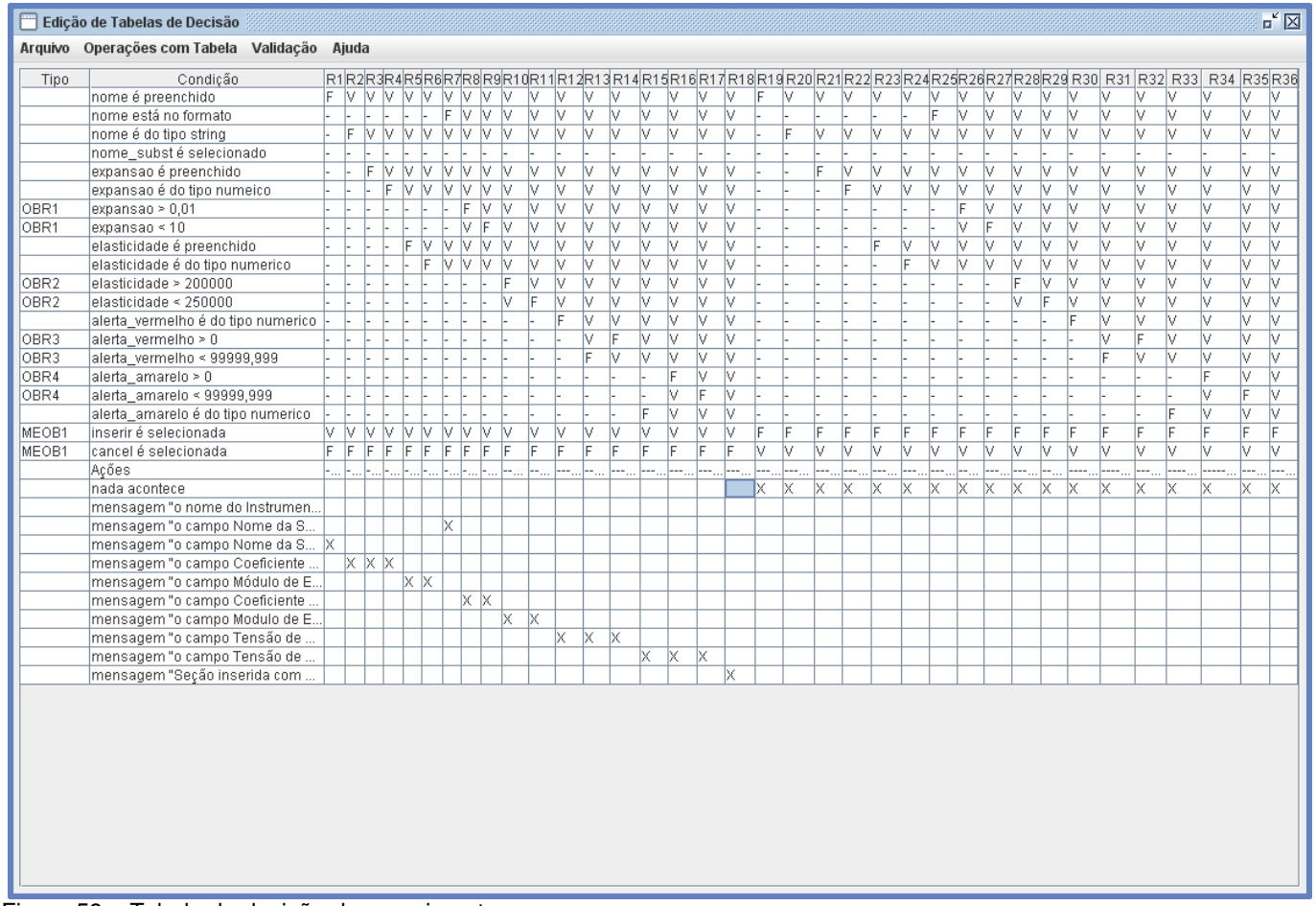


Figura 57 – Interface do sistema a ser testado.

Essa funcionalidade foi escolhida por apresentar uma variação de componentes gráficos que deveriam ser validados no experimento, esses componentes são: campo texto, campo numérico e *combo box*. Além disso, essa funcionalidade irá gerar uma tabela de decisão pequena que servirá como

comprovação que as características de uma interface comum de cadastro podem ser facilmente descrita em uma tabela de decisão.

O processo de teste é iniciado com a leitura de uma especificação da funcionalidade, sendo criada uma tabela de decisão associada à interface apresentada na figura 58. Essa tabela inclui todas as condições referentes a interface a ser testada e a combinação de valores que as condições podem ter. Nessa primeira etapa a principal dificuldade foi que, como acontece em grande parte dos projetos, a especificação não estava atualizada para a nova versão do sistema. Então, além de ler a especificação, foi necessário executar o sistema para conseguir criar a tabela de decisão. Mesmo não tendo a especificação dos requisitos, isso não impede que a tabela de decisão seja construída, já pode ser utilizado outras técnicas como observação de um usuário ou entrevista com o desenvolvedor. Outra dificuldade com relação à descrição da tabela de decisão, é que no documento de especificação não havia informações sobre a prioridade dos campos, ou seja, se a interface tiver dois campos preenchidos erroneamente, qual mensagem a ser apresentada, a do primeiro erro, a do segundo, ou de ambos. Esse problema fez com que a tabela de decisão tivesse que ser modificada algumas vezes e isto só era percebido ao executar o script de teste. O lado positivo é que, como a geração dos scripts é realizada automaticamente a partir da tabela de decisão, o esforço de modificar a tabela de decisão não causava nenhum esforço adicional nas outras etapas.



Tipo	Condição	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16	R17	R18	R19	R20	R21	R22	R23	R24	R25	R26	R27	R28	R29	R30	R31	R32	R33	R34	R35	R36
	nome é preenchido	F	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	F	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V
	nome está no formato	-	-	-	-	-	-	F	V	V	V	V	V	V	V	V	V	V	V	-	-	-	-	-	-	F	V	V	V	V	V	V	V	V	V	V	V
	nome é do tipo string	-	F	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	-	F	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V
	nome_subst é selecionado	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	expansao é preenchido	-	-	F	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	-	-	F	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V
	expansao é do tipo numeico	-	-	-	F	V	V	V	V	V	V	V	V	V	V	V	V	V	V	-	-	-	F	V	V	V	V	V	V	V	V	V	V	V	V	V	V
OBR1	expansao > 0,01	-	-	-	-	-	-	-	F	V	V	V	V	V	V	V	V	V	V	-	-	-	-	-	-	-	F	V	V	V	V	V	V	V	V	V	V
OBR1	expansao < 10	-	-	-	-	-	-	-	V	F	V	V	V	V	V	V	V	V	V	-	-	-	-	-	-	-	V	F	V	V	V	V	V	V	V	V	V
	elasticidade é preenchido	-	-	-	-	F	V	V	V	V	V	V	V	V	V	V	V	V	V	-	-	-	-	F	V	V	V	V	V	V	V	V	V	V	V	V	V
	elasticidade é do tipo numerico	-	-	-	-	-	F	V	V	V	V	V	V	V	V	V	V	V	V	-	-	-	-	-	F	V	V	V	V	V	V	V	V	V	V	V	V
OBR2	elasticidade > 200000	-	-	-	-	-	-	-	-	F	V	V	V	V	V	V	V	V	V	-	-	-	-	-	-	-	-	F	V	V	V	V	V	V	V	V	V
OBR2	elasticidade < 250000	-	-	-	-	-	-	-	-	-	V	F	V	V	V	V	V	V	V	-	-	-	-	-	-	-	-	V	F	V	V	V	V	V	V	V	V
	alerta_vermelho é do tipo numerico	-	-	-	-	-	-	-	-	-	-	-	F	V	V	V	V	V	V	-	-	-	-	-	-	-	-	-	-	F	V	V	V	V	V	V	V
OBR3	alerta_vermelho > 0	-	-	-	-	-	-	-	-	-	-	-	-	V	F	V	V	V	V	-	-	-	-	-	-	-	-	-	-	-	V	F	V	V	V	V	V
OBR3	alerta_vermelho < 99999,999	-	-	-	-	-	-	-	-	-	-	-	-	F	V	V	V	V	V	-	-	-	-	-	-	-	-	-	-	-	F	V	V	V	V	V	V
OBR4	alerta_amarelo > 0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	F	V	V	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	F	V	V
OBR4	alerta_amarelo < 99999,999	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	V	F	V	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	V	F	V
	alerta_amarelo é do tipo numerico	-	-	-	-	-	-	-	-	-	-	-	-	-	-	F	V	V	V	-	-	-	-	-	-	-	-	-	-	-	-	-	-	F	V	V	V
MEOB1	inserir é selecionada	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
MEOB1	cancel é selecionada	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V
	Ações	-...	-...	-...	-...	-...	-...	-...	-...	-...	--...	--...	---...	--...	---...	---...	---...	---...	---...	---...	---...	---...	---...	---...	---...	---...	---...	---...	---...	---...	---...	---...	---...	----...	----...	----...	---...	---...
	nada acontece																			X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
	mensagem "o nome do Instrumen...																																				
	mensagem "o campo Nome da S...							X																														
	mensagem "o campo Nome da S...	X																																			
	mensagem "o campo Coeficiente ...		X	X	X																																
	mensagem "o campo Módulo de E...					X	X																														
	mensagem "o campo Coeficiente ...								X	X																											
	mensagem "o campo Modulo de E...										X	X																									
	mensagem "o campo Tensão de ...												X	X	X																						
	mensagem "o campo Tensão de ...															X	X	X																			
	mensagem "Seção inserida com ...																		X																		

Figura 58 – Tabela de decisão do experimento

Seguindo o processo, depois de criar e validar a tabela de decisão são gerados os casos de teste e, a seguir, o script de teste. Na figura 59 está apresentado um caso de teste e, na figura 60, três casos de teste contidos no script.

```
<Dados>
 <casoteste>
<CasoTeste>
    <campos>
     <Campo>
      <componente>Campo Texto</componente>
      <nomecampo>nome</nomecampo>
      <valor>349</valor>
     </Campo>
     <Campo>
      <componente>Campo Texto</componente>
      <nomecampo>expansao</nomecampo>
      <valor>5115030</valor>
     </Campo>
     <Campo>
      <componente>Campo Texto</componente>
      <nomecampo>elasticidade</nomecampo>
      <valor></valor>
     </Campo>
     <Campo>
      <componente>Campo Texto</componente>
      <nomecampo>alerta_amarelo</nomecampo>
      <valor></valor>
     </Campo>
     <Campo>
      <componente>Campo Texto</componente>
      <nomecampo>alerta_vermelho</nomecampo>
      <valor></valor>
     </Campo>
     <Campo>
      <componente>Campo Texto</componente>
      <nomecampo>alertamin_verm</nomecampo>
      <valor></valor>
     </Campo>
     <Campo>
      <componente>Campo Texto</componente>
      <nomecampo>alertamin_amarelo</nomecampo>
      <valor></valor>
     </Campo>
     <Campo>
      <componente>Botao</componente>
      <nomecampo>inserir</nomecampo>
      <valor></valor>
     </Campo>
    </campos>
    <oraculos>
     <Oraculo>
      <nome>mensagem “O campo Coeficiente de Expansão Térmica do Duto deve
conter valores entre 0,01 e 10”</nome>
     </Oraculo>
    </oraculos>
  </CasoTeste>
```

Figura 59 – Arquivo XML com dados do teste

```
	@Test public void caso_teste_13() {
		window.textBox("nome").enterText("499");
		window.textBox("expansao").enterText("4");
		window.textBox("elasticidade").enterText("200000");
		window.textBox("alerta_amarelo").enterText("");
		window.textBox("alerta_vermelho").enterText("");
		window.textBox("alertamin_verm").enterText("");
		window.textBox("alertamin_amarelo").enterText("");
		window.button("inserir").click();
		window.optionPane().requireMessage("O campo Módulo de Elasticidade do
Duto [MPa] deve conter valores entre 200.000 e 250.000");
	}

	@Test public void caso_teste_14() {
		window.textBox("nome").enterText("907");
		window.textBox("expansao").enterText("3");
		window.textBox("elasticidade").enterText("4510921");
		window.textBox("alerta_amarelo").enterText("");
		window.textBox("alerta_vermelho").enterText("");
		window.textBox("alertamin_verm").enterText("");
		window.textBox("alertamin_amarelo").enterText("");
		window.button("inserir").click();
		window.optionPane().requireMessage("O campo Módulo de Elasticidade do
Duto [MPa]  deve conter valores entre 200.000 e 250.000");
	}

	@Test public void caso_teste_15() {
		window.textBox("nome").enterText("378");
		window.textBox("expansao").enterText("7");
		window.textBox("elasticidade").enterText("250000");
		window.textBox("alerta_amarelo").enterText("");
		window.textBox("alerta_vermelho").enterText("");
		window.textBox("alertamin_verm").enterText("");
		window.textBox("alertamin_amarelo").enterText("");
		window.button("inserir").click();
		window.optionPane().requireMessage("O campo Módulo de Elasticidade do
Duto [MPa] deve conter valores entre 200.000 e 250.000");
	}

	@Test public void caso_teste_16() {
		window.textBox("nome").enterText("804");
		window.textBox("expansao").enterText("9");
		window.textBox("elasticidade").enterText("208772");
		window.textBox("alerta_amarelo").enterText("");
		window.textBox("alerta_vermelho").enterText("S");
		window.textBox("alertamin_verm").enterText("");
		window.textBox("alertamin_amarelo").enterText("");
		window.button("inserir").click();
		window.optionPane().requireMessage("Seção inserida com sucesso");
	}
```

Figura 60- Casos de teste do script gerado

Ao executar pela primeira vez o script gerado foi encontrada a principal dificuldade de realizar o processo. O script precisa de uma instância da interface que será testada. Inicialmente a idéia era instanciar a interface dentro do script e assim conseguir executá-lo normalmente. Ao verificar que a interface a ser testada precisava de alguns objetos antes da sua instanciação, foi necessário investigar como esses objetos eram instanciados e fazer a instanciação de todos eles para depois conseguir instanciar a interface desejada. O problema é que alguns objetos precisavam de outros objetos e isso foi virando uma solução impraticável. Ao examinar o problema, foi verificado que a melhor solução era executar o sistema normalmente e, quando este instanciasse a interface desejada, ele passaria para o script de teste a instância da interface e, a partir daí, o sistema iria executar o teste.

Seguindo este raciocínio foi encontrado no código do sistema o ponto em que é instanciada a interface. Nesse ponto foram inseridas algumas linhas de código para poder atribuir o nome para os elementos gráficos e também foi passado para o script de teste o objeto correspondente a interface a ser testada. A atribuição dos nomes foi necessária já que os desenvolvedores do sistema não tinham como padrão de programação colocar esses nomes.

Ao executar o sistema, quando tentava executar o script de teste era lançado o erro "***Cannot call method from the event dispatcher thread***". ***Event dispatcher thread*** é um tipo de *thread* de Java utilizada para executar interfaces descritas através da biblioteca *AWT*. Esse erro é causado por uma limitação do framework utilizado, que não permite executar os métodos, se o teste estiver sendo executado por uma *thread* deste tipo.

Então foi necessário ver outra solução para o problema. Desta vez, o script de teste foi modificado para que este pudesse ser uma *thread,* e assim pudesse ser executado após a instanciação da interface pelo sistema. Depois que o sistema instancia a interface a ser testada ele cria um objeto do tipo do *script* de teste e o executa. Assim foi possível executar os testes gerados para a interface desejada. As modificações necessárias para transformar o script de teste em uma *thread* e conseguir chamar sua execução por dentro do sistema em teste ocorreram somente na parte inicial do script e estão apresentadas na figura 61

```
public class MassaExperimento extends Thread{
	private static FrameFixture window;
	private static MassaExperimento instancia = null;

	public MassaExperimento()
	{
		instancia = this;
	}
	public void run()
	{
		Class c;
		try {
			c = Class.forName("experimento.MassaExperimento");

			Result result = JUnitCore.runClasses( c );

			List <Failure> failures = result.getFailures();

			for ( Failure f : failures ) {
				System.out.println( f.getTestHeader() );
				System.out.println( f.getMessage() );
				System.out.println( f.getException() );
				System.out.println( f.getTrace() );
			}

			System.exit(0);
		}catch (ClassNotFoundException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}

	private static JFrame janela;

	public static void setJanela(JFrame j)
	{
		janela = j;
	}

	@Before public void setUp() {
		window = new FrameFixture(janela);
		window.show(); /* shows the frame to test */
		window.textBox("alertamin_amarelo").deleteText();
	}

	@After public void tearDown() {
		window.cleanUp();
	}
```

Figura 61 – Parte inicial do script alterado

Com a execução dos casos de teste foi possível encontrar três defeitos. Esses erros ocorreram nos casos de teste 13, 15 e 16. O primeiro e o segundo erro ocorrem quando são colocados os valores limites para o campo **"Módulo de Elasticidade"** e o último é quando é inserido um valor do tipo *string* no campo **"Tensão Long. MÁXIMA para Alerta Vermelho"** que é um campo numérico.

Como esta funcionalidade não foi alterada recentemente, encontrar 3 defeitos foi um resultado bom porque essa funcionalidade já está sendo utilizada em produção e ninguém ainda tinha os encontrado.

6.2.Experimento do Teste Manual

Foram, realizadas comparações entre o novo processo e um processo de teste totalmente manual. É importante ressaltar que o novo processo não irá substituir completamente o teste manual, já que este teste manual foca em aspectos como usabilidade e qualidade da interface que não são vistos pelo novo processo. . Entretanto, a parte que é testada pelo novo processo também pode ser testada através de um processo manual. Os dois tipos de teste, manual e automatizado, são importantes em qualquer sistema.

6.2.1.Primeiro Experimento

No primeiro experimento, para ser possível a comparação entre os processos foram levadas em consideração duas métricas: cobertura e defeitos encontrados. Para a cobertura, foi utilizado o framework EclEmma (http://www.eclemma.org/), um *plugin* para o IDE Eclispe que gera um relatório da porcentagem de código executada de cada classe durante a utilização do sistema. Para a última métrica foi verificado quantos defeitos foram encontrados pelos dois processos.

Para a realização do teste manual deste primeiro experimento, foi pedido para um analista testar a mesma funcionalidade descrita na seção anterior até quando achasse que estava suficiente. O levantamento de quais casos de teste deveriam ser realizados foi feito empiricamente, tendo o auxilio apenas do sistema e da especificação do caso de uso da funcionalidade. Esse analista já trabalha há 2 anos com testes. Nesse experimento não foi pedido ao analista que fizesse a especificação dos casos de testes, apenas realizasse a execução do sistema.

Para a cobertura, o framework EclEmma gera um relatório da porcentagem do código executado considerando três tipos de componentes: método, bloco e linha. A tabela 6.1 apresenta as porcentagens de código executado das classes importantes para a funcionalidade dividido por componente para os dois processos, sendo PM o processo manual e o NP o novo processo.

Classes	Método		Bloco		Linha	
	PM (%)	NP (%)	PM (%)	NP (%)	PM (%)	NP (%)
GRDialogAction	67%	67%	77%	77%	78%	78%
ElementAction	83%	83%	53%	69%	62%	72%
GRAbstractPopup	44%	44%	48%	50%	43%	45%
PlacePipePopup	100%	100%	100%	100%	100%	100%
GRProjectTree	50%	50%	52%	52%	57%	57%
PlacePipe	28%	28%	38%	38%	32%	32%
PipeSection	15%	33%	25%	39%	22%	25%

Tabela 2 – Comparação entre a cobertura no primeiro experimento

Os valores da cobertura dos dois métodos não apresentaram diferenças significativas, uma das razões disso acontecer é a simplicidade da funcionalidade testada.

Nesse primeiro experimento a última métrica foi a que apresentou a maior diferença entre os dois processos. O processo manual encontrou um único defeito na interface. Esse defeito era que o campo *código* permitia inserir espaços entre os números. Já o novo processo, encontrou 3 defeitos que ocorrem com o correto preenchimento da interface como foi explicado na seção anterior.

6.2.2. Segundo Experimento

Ao finalizar este primeiro experimento verificamos que seria importante comparar também a etapa de planejamento dos testes, já que o novo processo também engloba essa atividade. Então foi realizado um segundo experimento com intuito de comparar o teste manual e o novo processo, só que desta vez o processo do teste manual seria diferente.

O processo do teste manual foi dividido em duas etapas: especificação e execução dos testes.

- Na primeira etapa foi pedido ao analista de teste que especificasse todos os casos de teste que julgasse necessário utilizando para isso a especificação do caso de uso e a interface do sistema. A descrição foi realizada utilizando a ferramenta IBM TestManager [TESTMANAGER]. Essa descrição inclui: nome do caso de teste, descrição do teste e passo a passo para a execução do teste.
- A segunda etapa era a execução dos casos de testes descritos e verificação de defeitos no sistema. Esse experimento foi realizado por outro analista,

isso para que não houvesse nenhuma influencia no resultado por conhecimento prévio do sistema.

Antes da avaliação das métricas, foi realizada uma verificação que quais e quantos casos de teste foram descritos pelo processo manual. Foram especificados 14 casos de testes pelo processo de teste manual e 36 casos de teste pelo novo processo através do uso da tabela de decisão. A grande diferença entre as duas especificações é que no teste manual cada campo dificilmente é testado com diversas possibilidades de valores. Por exemplo, para testar um campo número que tem uma restrição de limite, no processo manual foi especificado apenas um caso de teste de falha para este campo enquanto na tabela de decisão foram descritos quatro casos de teste.

Nesse segundo experimento foram utilizadas três métricas para comparação entre os processos, são elas: esforço, defeitos encontrados e cobertura.

Para primeira métrica foi feito a contagem de tempo separadamente para as duas etapas: especificação e execução. Na primeira etapa, o processo manual gastou duas horas e trinta minutos para a especificação enquanto o novo processo realizou essa etapa em 30 minutos. Na segunda etapa, a de execução, o processo manual demorou 1 hora e 30 enquanto o novo processo demorou 20 minutos para a adaptação do código para testar e 10 minutos da realização do teste. No total, o processo manual demorou 4 horas e o novo processo demorou 1 hora.

Com relação a segunda métrica, o novo processo encontrou três defeitos como foi dito no experimento anterior. Já o processo manual, encontrou 4 melhorias para o sistema, mas os defeitos encontrados pelo novo processo não foram descobertos. Um exemplo de melhoria descrita foi que ao tentar digitar um caractere no campo código, não é dado nenhum aviso ao usuário que essa é uma ação indesejada, o sistema simplesmente não preenche o campo com os caracteres digitados. Desta métrica, pode-se comprovar que os dois tipos de teste são importantes, mas que o novo processo encontra defeitos que não são sempre identificados pelos testes manuais.

A tabela 3 mostra a cobertura realizada pelos dois tipos de processo e percebe-se que a diferença entre os dois processos não é significativa, pelo mesmo motivo comentado no experimento anterior.

Classes	Método		Bloco		Linha	
	PM (%)	NP (%)	PM (%)	NP (%)	PM (%)	NP (%)
GRDialogAction	67%	67%	77%	77%	78%	78%
ElementAction	92%	83%	74%	69%	78%	72%
GRAbstractPopup	44%s	44%	48%	50%	43%	45%
PlacePipePopup	100%	100%	100%	100%	100%	100%
GRProjectTree	50%	50%	52%	52%	57%	57%
PlacePipe	28%	28%	38%	38%	32%	32%
PipeSection	36%	33%	47%	39%	42%	36%

Tabela 3 – Comparação entre a cobertura no segundo experimento

Ao avaliar a tabela 3 e com intuito de melhorar a geração dos casos de teste, foram investigadas quais as partes do código que não foram percorridas pelo teste automatizado. Na classe `ElementAction`, não foram gerados testes para duas exceções: uma de banco de dados e outra de passagem ilegal de parâmetro. Para o primeiro ponto ainda não são gerados testes desse tipo, mas existe a possibilidade dessa geração. No segundo caso, passagem ilegal de parâmetros, é difícil gerar um teste para passar nesse ponto uma vez que a passagem do parâmetro é influenciada pelo arco de chamadas realizado. Quanto mais longo este arco, considerando a sua origem na interface com o usuário, mais difícil, ou mesmo impossível, é a geração de dados que produzam valores ilegais. Nas classes `GRAbstractPopup`, `GRProjectTree` não foram percorridas as linhas que faziam a construção da interface, uma vez que isso é feito antes da instanciação da interface, que é onde o teste inicia. Na classe `PlacePipe` existem diversas outras funcionalidades em relação ao elemento testado. O teste realizado era só para a inserção da seção de duto, mas ainda é disponibilizada pelo sistema a alteração e remoção do elemento. Já na classe `PipeSectionInfo` existiam diversas *geters* e *seters* que não forma percorridos. Por último na classe `PipeSection` ocorreram problemas similares aos descritos nas classes `PlacePice` e `PipeSectionInfo`. Em parte as deficiências de cobertura encontradas devem-se ao fato de se medi-la considerando apenas os casos de testes gerados pela ferramenta considerando determinado contexto, e na verdade deveriam ser adicionados casos de teste manuais para situações muito específicas.

6.2.3.Conclusão dos Experimentos

Com base nos dois experimentos, é possível verificar que um dos pontos fortes do processo, auxiliado pela tabela de decisão, é testar os campos com valores diversos, o que normalmente não é feito e nem garantido pelo teste

manual. Outra vantagem do processo que não é evidenciada no experimento é a automação, ou seja, é possível realizar o teste diversas vezes e sempre atingindo a mesma cobertura, o que usualmente não ocorre para o teste manual, pois não existe controle de quais testes foram realizados e um teste manual completo tende a ter um custo proibitivo. Essa vantagem torna-se mais relevante quando o tempo gasto para a repetição dos testes no novo processo é baixa, quando comparado com o teste manual, o que foi mostrado no segundo experimento. Outra vantagem do processo é a fácil visualização dos casos de testes descritos através do uso da tabela de decisão e a geração automática dos casos de teste.

Um ponto fraco é a dificuldade de avaliar alguns detalhes de comportamento do sistema que não ficam claros através da especificação do caso de uso, mas que são importantíssimas para a construção da tabela de decisão. Um desses pontos é a comunicação com o banco de dados, por exemplo, como validar se existe algum tratamento para duplicação de informação.