“SPLMT-TE: A Software Product Lines System Test Case Tool”

By

Crescencio Rodrigues Lima Neto

M.Sc. Dissertation

RECIFE, AUGUST/2011
Crescencio Rodrigues Lima Neto

“SPLMT-TE: A Software Product Lines System Test Case Tool”

A M.Sc. Dissertation presented to the Center for Informatics of Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

Advisor: Silvio Romero de Lemos Meira
Co-Advisor: Eduardo Santana de Almeida

RECIFE, AUGUST/2011
I dedicate this dissertation to myself and all my family, who gave me all necessary support to get here.
Acknowledgements

First and foremost, I would like to thank my wonderful family and friends. Especially to my father, without his advice I would never get this far; my mother, besides the distance, she always stood by me; my philosophical brother, Marcos Júnior; and my unbelievable happy sister, Natália. I love them and appreciate the efforts they have put into giving me the conditions to finish this work. I would like to express my heartiest thanks to my wife, Cecília for her understanding, endless patience and for giving me the most important presents of my life, my children: Caio and Júlia.

I would like to grateful acknowledge Dr. Eduardo Almeida. He provided me with many helpful suggestions and encouragement during the course of this work as well as the challenging research that lies behind it. I also wish to express my appreciation to Dr. Silvio Meira, for accepting me as M.Sc. Student. Special gratitude goes to the rest of the teaching staff of the CIn/UFPE, for their valuable classes.

The results of this dissertation could not be achieved without the support of the Reuse in Software Engineering (RiSE) Labs. My gratitude to Ivan Machado and Paulo Anselmo for their advises and support in this long journey. I would like to thank all my friends from RiSE (Hugo, Ivonei, Iuri, Jonatas, Leandro, Luanna, Raphael, Simoni, Tassio, Thiago, Vinicius, Yguarata)

I cordially thank my friends and colleagues that I have met during my journey in Recife since 2006. I want to express my deep thanks to my foster brothers Emmanuel, Filipe, Dihego, and my foster mother aunt Dida. Aunts Edna, Angela, Rita, Ceia, Erica, and Monica, grandma Creuza, Clara, uncles Eugênio, and Valmir. I thank my housemates in Recife, Keite, Nivaldo, Ricson, Afonso, Vinícius, Diegos, friends from C.E.S.A.R. and Informatics center at UFPE for making me feel like home.

Thanks for my aunts Vânia and Irani, grandma Nozinha my uncles Paulo and Ivan. Moreover, I like to dedicate this dissertation for Grandma Nora, grandpas Crescencio and Manoel, uncles Ires and Iracsson, Adriana and Ceia, examples of life, force, and perseverance. They will be always in my mind.

There were a lot of people that directly or indirectly participated with me along my Master path. However, it will not be possible to remember of all of them now. My excuses for the ones I forgot.

My sincere thanks are extended for the National Institute of Science and Technology for Software Engineering (INES), for partially support this dissertation.

Also, I would like to thank to Dr. David M. Weiss and Dr. John D. McGregor for helping at the experiment definition and their suggestions during our discussions improved the quality of my work.

Finally, I would like to thank God for giving me the wisdom to perform this work.
Veja!
Não diga que a canção está perdida
Tenha em fé em Deus, tenha fé na vida
Tente outra vez!...

Beba!
Pois a água viva ainda está na fonte
Você tem dois pés para cruzar a ponte
Nada acabou!

Tente!
Levante sua mão sedenta e recomece a andar
Não pense que a cabeça aguentar se você parar
Há uma voz que canta, uma voz que dança, uma voz que gira
Bailando no ar

Queira!
Basta ser sincero e desejar profundo
Você será capaz de sacudir o mundo
Tente outra vez!

Tente!
E não diga que a vitória está perdida
Se é de batalhas que se vive a vida
Tente outra vez!...

—RAUL SEIXAS (Tente Outra Vez)
Atualmente a decisão de trabalhar, ou não, com Linhas de Produtos de Software (LPS) se tornou um requisito obrigatório para o planejamento estratégico das empresas que trabalham com domínio específico. LPS possibilita que as organizações alcancem reduções significativas nos custos de desenvolvimento e manutenção, melhorias quantitativas na produtividade, qualidade e satisfação do cliente.

Por outro lado, os pontos negativos em adotar LPS são demanda extra de investimentos para criar os artefatos reusáveis, fazer mudanças organizacionais, etc. Além disso, teste é mais complicado e crítico em linhas de produtos do que em sistemas simples. Porém, continua sendo a forma mais efetiva para garantia de qualidade em LPS.

Por isso, aprender a escolher as ferramentas certas para teste em LPS é um benefício que contribui para redução de alguns desses problemas enfrentados pelas empresas. Apesar do crescente número de ferramentas disponíveis, teste em LPS ainda necessita de ferramentas que apoiem o nível de teste de sistema, gerenciando a variabilidade dos artefatos de teste.

Neste contexto, este trabalho apresenta uma ferramenta de teste de linhas de produtos de software para construir testes de sistema a partir dos casos de uso que endereçam desafios para teste em LPS identificados na revisão literária. A ferramenta foi desenvolvida com o intuito de reduzir o esforço necessário para realizar as atividades de teste no ambiente de LPS.

Além disso, esta dissertação apresenta um estudo exploratório sistemático que tem como objetivo investigar o estado da arte em relação a ferramentas de teste, sintetizando as evidências disponíveis e identificar lacunas entre as ferramentas, disponíveis na literatura. Este trabalho também apresenta um estudo experimental controlado para avaliar a eficácia da ferramenta proposta.

**Palavras-chave:** Linhas de Produtos, Teste de Software, Reuso de Software, Processo de Software, Ferramentas de Teste, Ferramentas de Teste para Linhas de Produtos.
Nowadays, the decision whether to work with Software Product Lines (SPL) or not becomes a requirement for the strategic planning of companies working in specific domain. SPL can enable organizations to achieve significant reductions in terms of development and maintenance cost, remarkable quantitative improvements in productivity, quality, and customer satisfaction.

On the other hand, the drawbacks of SPL adoption are the demand of extra investments to build reusable assets, make changes at the organization, testing is more critical and complex, etc. Furthermore, testing is more critical and complex for product lines than traditional single software systems. However, still the most effective way for quality assurance in SPL.

Thus, learning how to choose the ideal tools to test a SPL is beneficial for companies in order to lessen some of these problems. Even though the ascending number of available software engineering testing tools, SPL testing lacks specific tools able of supporting the system testing level of the SPL testing process and managing the variability of test assets.

In this context, this work presents a software product lines testing tool to build system tests from use cases that addresses challenges for SPL testing identified in the literature review. The tool intended to provide a way for organizations to save effort when performing testing activities in a SPL environment.

Moreover, this dissertation introduces a systematic scoping study that aims to investigate the state-of-the-art regarding to testing tools, synthesize available evidence, and identify gaps among the tools, available in the literature. This work also presents a controlled experimental study to evaluate the proposed tool effectiveness.

**Keywords:** Software Testing, Software Product Lines, Software Reuse, Software Process, Testing Tools, Software Product Lines Testing Tools.
# Table of Contents

## List of Figures

## List of Tables

## List of Acronyms

1. **Introduction** 1
   1.1 Motivation 2
   1.2 Scope 4
      1.2.1 Context 4
   1.3 Out of Scope 6
   1.4 Statement of the Contributions 7
   1.5 Organization of this Dissertation 7

2. **Foundations on Software Product Lines and Testing** 9
   2.1 Software Product Lines (SPL) 10
      2.1.1 SPL Essential Activities 11
         Core Asset Development 11
         Product Development 13
         Management 13
   2.2 Software Testing 14
      2.2.1 Test Levels 15
         Unit Testing 15
         Integration Testing 15
         System Testing 15
         Acceptance Testing 16
      2.2.2 Overview on Software Testing Tools 16
         Application Test Tools 17
         Web Test Tools 18
         Test Management Tools 20
         API Test Tools 20
         Communications Test Tools 20
   2.3 Testing in Software Product Lines 21
      2.3.1 Testing in core asset development 21
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2.2 Non-Functional Requirements</td>
<td>51</td>
</tr>
<tr>
<td>4.2.3 Functionalities Priority</td>
<td>52</td>
</tr>
<tr>
<td>4.3 Tool Architecture Overview</td>
<td>53</td>
</tr>
<tr>
<td>4.3.1 Metamodel for managing variability</td>
<td>54</td>
</tr>
<tr>
<td>4.4 Technologies</td>
<td>55</td>
</tr>
<tr>
<td>4.5 Implementation</td>
<td>57</td>
</tr>
<tr>
<td>4.6 Software Product Lines System Test Case Tool</td>
<td>58</td>
</tr>
<tr>
<td>4.6.1 SPLMT-TE Steps and activities</td>
<td>60</td>
</tr>
<tr>
<td>4.7 SPLMT-TE in Action</td>
<td>62</td>
</tr>
<tr>
<td>4.8 Chapter Summary</td>
<td>64</td>
</tr>
<tr>
<td>5 Experimental Study</td>
<td>68</td>
</tr>
<tr>
<td>5.1 Definition</td>
<td>69</td>
</tr>
<tr>
<td>5.2 Planning</td>
<td>70</td>
</tr>
<tr>
<td>5.2.1 Context Selection</td>
<td>70</td>
</tr>
<tr>
<td>5.2.2 Hypothesis Formulation</td>
<td>71</td>
</tr>
<tr>
<td>Null Hypothesis ($H_{0}$)</td>
<td>71</td>
</tr>
<tr>
<td>Alternative Hypothesis ($H_{1}$)</td>
<td>72</td>
</tr>
<tr>
<td>5.2.3 Selection of subjects</td>
<td>72</td>
</tr>
<tr>
<td>5.2.4 Experiment design</td>
<td>73</td>
</tr>
<tr>
<td>Design Type</td>
<td>73</td>
</tr>
<tr>
<td>5.2.5 Instrumentation</td>
<td>73</td>
</tr>
<tr>
<td>5.2.6 Training</td>
<td>75</td>
</tr>
<tr>
<td>5.3 The Experimental Study Project</td>
<td>76</td>
</tr>
<tr>
<td>5.4 Operation</td>
<td>78</td>
</tr>
<tr>
<td>5.4.1 Pilot Study</td>
<td>79</td>
</tr>
<tr>
<td>Preparation</td>
<td>79</td>
</tr>
<tr>
<td>Execution</td>
<td>79</td>
</tr>
<tr>
<td>Data Validation</td>
<td>81</td>
</tr>
<tr>
<td>Lessons Learned</td>
<td>81</td>
</tr>
<tr>
<td>5.4.2 Experiment</td>
<td>82</td>
</tr>
<tr>
<td>Preparation</td>
<td>82</td>
</tr>
<tr>
<td>Execution</td>
<td>82</td>
</tr>
<tr>
<td>Data Validation</td>
<td>84</td>
</tr>
<tr>
<td>5.5 Analysis and Interpretation</td>
<td>84</td>
</tr>
<tr>
<td>5.5.1 Descriptive Statistic</td>
<td>84</td>
</tr>
<tr>
<td>Chapter</td>
<td>Section</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>5.5.2</td>
<td>Hypotheses Testing</td>
</tr>
<tr>
<td>5.5.3</td>
<td>Qualitative analysis</td>
</tr>
<tr>
<td>5.6</td>
<td>Validity Evaluation</td>
</tr>
<tr>
<td></td>
<td>5.6.1 Conclusion Validity</td>
</tr>
<tr>
<td></td>
<td>5.6.2 Internal Validity</td>
</tr>
<tr>
<td></td>
<td>5.6.3 Construct Validity</td>
</tr>
<tr>
<td></td>
<td>5.6.4 External Validity</td>
</tr>
<tr>
<td>5.7</td>
<td>Lessons Learned</td>
</tr>
<tr>
<td>5.8</td>
<td>Chapter Summary</td>
</tr>
<tr>
<td>6</td>
<td>Conclusion</td>
</tr>
<tr>
<td></td>
<td>6.1 Research Contributions</td>
</tr>
<tr>
<td></td>
<td>6.1.1 Scoping Study</td>
</tr>
<tr>
<td></td>
<td>6.1.2 SPLMT-TE</td>
</tr>
<tr>
<td></td>
<td>6.1.3 Experimental study</td>
</tr>
<tr>
<td></td>
<td>6.2 Related Work</td>
</tr>
<tr>
<td></td>
<td>6.3 Future Work</td>
</tr>
<tr>
<td></td>
<td>6.4 Academic Contributions</td>
</tr>
<tr>
<td></td>
<td>6.5 Concluding Remarks</td>
</tr>
</tbody>
</table>

Bibliography 102

Appendices 112

A Journals and Conferences 113

A.1 List of Journals 113
A.2 List of Conferences 114

B Experimental Study Instruments 115

B.1 Background Questionnaire 116
B.2 Consent form 118
B.3 Test Case Creation Form 119
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.4</td>
<td>Test Case Execution Form</td>
<td>120</td>
</tr>
<tr>
<td>B.5</td>
<td>Error Reporting Form</td>
<td>121</td>
</tr>
<tr>
<td>B.6</td>
<td>Automatic Test Case Creation Report Form</td>
<td>122</td>
</tr>
<tr>
<td>B.7</td>
<td>Feedback Questionnaire A</td>
<td>123</td>
</tr>
<tr>
<td>B.8</td>
<td>Feedback Questionnaire B</td>
<td>126</td>
</tr>
<tr>
<td>B.9</td>
<td>Feedback Questionnaire C</td>
<td>128</td>
</tr>
<tr>
<td>B.10</td>
<td>Feedback Questionnaire D</td>
<td>129</td>
</tr>
<tr>
<td>C</td>
<td>Pilot Study Information</td>
<td>131</td>
</tr>
<tr>
<td>C.1</td>
<td>Execution</td>
<td>131</td>
</tr>
<tr>
<td>C.2</td>
<td>Analysis and Interpretation</td>
<td>132</td>
</tr>
<tr>
<td>Figure</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>1.1</td>
<td>RiSE Labs Influences</td>
<td>5</td>
</tr>
<tr>
<td>1.2</td>
<td>RiSE Labs Projects</td>
<td>6</td>
</tr>
<tr>
<td>2.1</td>
<td>Essential Product Line Activities</td>
<td>11</td>
</tr>
<tr>
<td>2.2</td>
<td>Core Asset Development</td>
<td>12</td>
</tr>
<tr>
<td>2.3</td>
<td>Product Development</td>
<td>13</td>
</tr>
<tr>
<td>2.4</td>
<td>Types of System Test</td>
<td>16</td>
</tr>
<tr>
<td>3.1</td>
<td>The Systematic Mapping Process</td>
<td>26</td>
</tr>
<tr>
<td>3.2</td>
<td>Study Selection Summary</td>
<td>30</td>
</tr>
<tr>
<td>3.3</td>
<td>Distribution of primary studies</td>
<td>32</td>
</tr>
<tr>
<td>3.4</td>
<td>Amount of Studies vs. sources</td>
<td>32</td>
</tr>
<tr>
<td>3.5</td>
<td>Tools that support the Testing process</td>
<td>38</td>
</tr>
<tr>
<td>3.6</td>
<td>Where the tools were developed and used</td>
<td>40</td>
</tr>
<tr>
<td>3.7</td>
<td>Tool’s classification according to Testing Level</td>
<td>43</td>
</tr>
<tr>
<td>3.8</td>
<td>Selected Single System Test Tools timeline</td>
<td>44</td>
</tr>
<tr>
<td>3.9</td>
<td>Selected SPL Test Tools timeline</td>
<td>44</td>
</tr>
<tr>
<td>4.1</td>
<td>Test Plan Management</td>
<td>51</td>
</tr>
<tr>
<td>4.2</td>
<td>Proposed architecture</td>
<td>55</td>
</tr>
<tr>
<td>4.3</td>
<td>The tests model excerpt</td>
<td>56</td>
</tr>
<tr>
<td>4.4</td>
<td>Proposed Low Level Architecture</td>
<td>57</td>
</tr>
<tr>
<td>4.5</td>
<td>Proposal</td>
<td>59</td>
</tr>
<tr>
<td>4.6</td>
<td>Test Case Composition</td>
<td>60</td>
</tr>
<tr>
<td>4.7</td>
<td>Using of the Test Case Document</td>
<td>61</td>
</tr>
<tr>
<td>4.8</td>
<td>Tool’s Execution Flow</td>
<td>61</td>
</tr>
<tr>
<td>4.9</td>
<td>Activities Phases Supported by the proposal</td>
<td>62</td>
</tr>
<tr>
<td>4.10</td>
<td>Tool in Action: Test Case</td>
<td>63</td>
</tr>
<tr>
<td>4.11</td>
<td>Tool in Action: Test Creation</td>
<td>65</td>
</tr>
<tr>
<td>4.12</td>
<td>Alert Message</td>
<td>66</td>
</tr>
<tr>
<td>4.13</td>
<td>Tool in Action: Test Suite Creation</td>
<td>66</td>
</tr>
<tr>
<td>4.14</td>
<td>Tool in Action: Test Plan Creation</td>
<td>67</td>
</tr>
<tr>
<td>5.1</td>
<td>Experiment Design</td>
<td>74</td>
</tr>
<tr>
<td>5.2</td>
<td>Arcade Games in Execution</td>
<td>77</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>-------------------------------------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>5.3 Feature Model from Arcade Game</td>
<td>78</td>
<td></td>
</tr>
<tr>
<td>5.4 NotepadSPL in execution</td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>5.5 Feature Model from NotepadSPL</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>5.6 Boxplot with Designed Test Cases per Group</td>
<td>87</td>
<td></td>
</tr>
<tr>
<td>5.7 Boxplot with Efficiency in Test Case Design</td>
<td>88</td>
<td></td>
</tr>
<tr>
<td>5.8 Boxplot with the Efficiency in Test Cases Execution</td>
<td>89</td>
<td></td>
</tr>
<tr>
<td>5.9 Boxplot with the Errors Found</td>
<td>89</td>
<td></td>
</tr>
<tr>
<td>5.10 Boxplot with the Test Case Effectiveness</td>
<td>91</td>
<td></td>
</tr>
</tbody>
</table>
List of Tables

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>List of Research Strings</td>
<td>29</td>
</tr>
<tr>
<td>3.2</td>
<td>Number of tools selected in each category</td>
<td>30</td>
</tr>
<tr>
<td>3.3</td>
<td>Testing Process Support</td>
<td>39</td>
</tr>
<tr>
<td>3.4</td>
<td>Where the Tools were developed and used</td>
<td>41</td>
</tr>
<tr>
<td>3.5</td>
<td>Tool’s classification according to Testing Level</td>
<td>42</td>
</tr>
<tr>
<td>4.1</td>
<td>Functional requirements of each system testing tool</td>
<td>51</td>
</tr>
<tr>
<td>4.2</td>
<td>Summary of requirements</td>
<td>53</td>
</tr>
<tr>
<td>5.1</td>
<td>Experiment Training and Execution Agenda</td>
<td>83</td>
</tr>
<tr>
<td>5.2</td>
<td>Subjects from the Experiment divided into groups</td>
<td>84</td>
</tr>
<tr>
<td>5.3</td>
<td>Subjects’ Profile from Groups 1 and 2 - Experiment</td>
<td>85</td>
</tr>
<tr>
<td>5.4</td>
<td>Final grouping of Subjects from the Experiment</td>
<td>86</td>
</tr>
<tr>
<td>5.5</td>
<td>Experiment information</td>
<td>86</td>
</tr>
<tr>
<td>5.6</td>
<td>Designed Test Cases</td>
<td>86</td>
</tr>
<tr>
<td>5.7</td>
<td>Efficiency in Test Case Design</td>
<td>87</td>
</tr>
<tr>
<td>5.8</td>
<td>Efficiency in Test Cases Execution</td>
<td>88</td>
</tr>
<tr>
<td>5.9</td>
<td>Number of Errors Found</td>
<td>89</td>
</tr>
<tr>
<td>5.10</td>
<td>Test Case Effectiveness</td>
<td>90</td>
</tr>
<tr>
<td>A.1</td>
<td>List of Journals</td>
<td>113</td>
</tr>
<tr>
<td>A.2</td>
<td>List of Conferences</td>
<td>114</td>
</tr>
<tr>
<td>B.1</td>
<td>Consent Form</td>
<td>118</td>
</tr>
<tr>
<td>B.2</td>
<td>Test Case Creation Form</td>
<td>119</td>
</tr>
<tr>
<td>B.3</td>
<td>Test Case Execution Form</td>
<td>120</td>
</tr>
<tr>
<td>B.4</td>
<td>Error Reporting Form</td>
<td>121</td>
</tr>
<tr>
<td>B.5</td>
<td>Test Case Creation Report Form</td>
<td>122</td>
</tr>
<tr>
<td>C.1</td>
<td>Pilot Study Training and Execution Agenda</td>
<td>131</td>
</tr>
<tr>
<td>C.2</td>
<td>Subjects of the pilot divided into groups</td>
<td>132</td>
</tr>
<tr>
<td>C.3</td>
<td>Subjects’ Profile from Groups 1 and 2 - Pilot</td>
<td>133</td>
</tr>
<tr>
<td>C.4</td>
<td>Final grouping of Subjects from the Pilot</td>
<td>134</td>
</tr>
<tr>
<td>C.5</td>
<td>Pilot Study information</td>
<td>134</td>
</tr>
<tr>
<td>C.6</td>
<td>Designed Test Cases - Pilot</td>
<td>134</td>
</tr>
</tbody>
</table>
C.7 Efficiency in Test Case Design - Pilot ................. 134
C.8 Efficiency in Test Case Execution - Pilot ............... 135
C.9 Test Case Effectiveness - Pilot .......................... 135
List of Acronyms

C.E.S.A.R. Recife Center For Advanced Studies and Systems
IDE Integrated Development Environment
GQM Goal Question Metric
GUI Graphical User Interface
PDF Portable Document Format
RISE Reuse in Software Engineering Labs
RiPLE RiSE process for Product Line Engineering
SEI Software Engineering Institute
SPL Software Product Lines
SPLMT Software Product Lines Management Tool
SPLT Software Product Lines Testing
SPLTT Software Product Lines Testing Tools
SSTT Single System Testing Tools
TDD Test Driven Development
UFBA Federal University of Bahia
UFPE Federal University of Pernambuco
XML Extensible Markup Language
Introduction

Never lose the child-like wonder.
It’s just too important.
It’s what drives us.
—RANDY PAUSCH (The Last Lecture)

Software testing is known by the high cost of implementation. However, it is inevitable to achieve quality without this activity. According to Graham (1993), omitting all testing would be more productive, cheaper, and faster to deliver the software but it will almost certain lead to disaster.

The demand for software with better quality has motivated the creation of methods, techniques, processes, and tools, which permit that such quality can be achieved. In addition, researchers are investigating different aspects of the testing area (Tevanlinna et al., 2004), in order to reduce the cost without loss of quality.

Another way of reducing development and maintenance costs is using SPL practices, which have been applied in industry for improving software productivity, time-to-market, and quality improvements (Clements and Northrop, 2001; van der Linden et al., 2007). Many of the benefits expected from these practices are based on the assumption that the investment in setting up a SPL pays of later when products are created. However, in order to take advantage of this assumption, organizations need to optimize the development and quality evaluation of core assets and products that brings the maximum business value (Abrahão et al., 2011). Thus, testing still the most effective way for quality assurance, is more critical and complex for product lines than traditional single software systems. (Kolb and Muthig, 2003).

The product lines approach requires a carefully planned testing process that can be
1.1 Motivation

Easily adapted and used in different domains. Currently, there is a gap between the overall product lines engineering processes and the assumed testing processes using the practical testing methods, because it is not clear how the processes and test assets are aligned to test a product lines in a cost-effective manner, while preserving quality. (Tevanlinna et al., 2004)

Product lines exploit the commonalities and variability of reusable assets across different products. Management of testing becomes very difficult if testers cannot tell the difference between assets that belong to the domain and assets that belong to a product. Nowadays, the most evident and perhaps most important question is how to handle and represent variability (Jaring and Bosch, 2002). Due to the large scale and huge complexity of today’s software-intensive systems, tool support is a key success factor in Software Product Lines Engineering (Clements and Northrop, 2001) and Software Product Lines Testing.

In this dissertation, the focus is on studying the state-of-the-art in software testing tools for software product lines and providing tool support for the system testing level of the Software Product Lines Testing (SPLT) process, always searching for maximizing the benefits of systematic reuse. For this reason, product lines testing tools are implemented to reduce the effort during the testing phase.

The remainder of this chapter describes the focus of this dissertation and starts by presenting its motivation in Section 1.1 and a clear definition of the problem in Section 1.2, while Section 1.3 describes some related aspects that are not directly addressed by this work. Section 1.4 presents the main contributions and, finally, Section 1.5 describes how this dissertation is organized.

1.1 Motivation

Aiming to improve the quality, some organizations have used testing tools to manage, store and handle the tests. According to Nakagawa et al. (2007), tools availability make testing a more systematic activity and minimizes the costs, the time consumed, as well as the errors caused by human intervention.

Craig and Jaskiel (2002) presented information from a survey conducted at the Rational ASQ Conference1, in 1997. According to it, 28% of respondents said they did not use automated testing tools due to the lack of management support or budget; 18% answered that adequate tools were not available; 13% reported their current testing effort

1http://wcqi.asq.org/
was too disorganized to use automated tools; 7% affirmed their current manual testing was adequate; 5% asserted they did not know that tools were available; and 0% said they did not see any benefit to using tools. The authors did not detail the remaining 29%.

Thus, finding an effective and efficient software testing tool could be a lifesaver for a project or organization. Nevertheless, there is no testing tool suitable for all possible systems (Yang et al., 2006). In order to choose the right tool, it is important that the requirements be formulated and prioritized for the selection and use of a proper testing tool.

During SPLT phases, the management of testing becomes very difficult if testers cannot tell the difference between assets that belong to the domain and assets that belong to a product. Commonality and variability in product lines require rich traceability, from requirements to implementation and test assets, encompassing the whole development life cycle (McGregor, 2001).

Integrating test environments and the test asset repository is a cumbersome work, that is even worse due to the lack of automated support tool, which leads to be a manually performed task. There is a need for specific testing tools that should help to manage the reusable testing assets and automate the execution of tests and the analysis of their results as well (Tevanlinna et al., 2004).

The Reuse in Software Engineering Labs (RiSE) developed several tools for software reuse such as BART (Santos et al., 2006), CORE (Burégio et al., 2007), LIFT (Brito, 2007) and ToolDay (Lisboa et al., 2007). The earlier research of the RiSE Labs are dedicated to study SPL, based in the experience gained during industrial SPL project development, the Software Product Lines Management Tool (SPLMT) was implemented (Cavalcanti et al., 2011) to coordinate SPL activities, by managing different SPL phases and their responsible, and to maintain the traceability and variability among different artifacts.

For this reason, we preferred to extend the SPLMT instead of start the creation of a new tool. Based on the SPLT gaps identified by the RiSE Labs (Neto et al., 2011a,b) we developed the test module of the SPLMT named SPLMT-TE.

The focus of this work is proposing a software product lines testing tool that can support the system testing level of the SPL testing process. The necessity of SPL testing tools is better explained and characterized in Chapter 3, through a study which examines the factors that cause it and how it impacts on the software development. Furthermore, the other challenges are further detailed on Chapter 3.
1.2 Scope

The goal of this dissertation can be stated as follows:

This work defines the requirements, design and implementation of a software product lines system test case tool, aiming at the creation and management of test assets, which contributes for the reduction of effort during the system testing level of the SPL testing process.

In order to reduce the effort during the use of the SPL testing process, it was developed the SPLMT-TE. The remainder of this section describes the context where it was developed and the outline of the proposal.

The proposed solution consists of a Web based application that enables test engineers involved with test assets creation and management to perform such tasks more effectively. The tool aims to support the test planning and test asset design activities from the SPLT process.

1.2.1 Context

This dissertation is part of the RiSE labs (Almeida et al., 2004), formerly called RiSE Project, whose goal is to develop a robust framework for software reuse in order to enable the adoption of a reuse program. However, it is influenced by a series of areas, such as software measurement, architecture, quality, environments and tools, and so on, in order to achieve its goal. The influence areas are depicted in Figure 1.1.

Based on these areas, the RiSE Labs is divided in several projects, as shown in Figure 1.2. As it can be seen, this framework embraces several different projects related to software reuse and software engineering. They are:

- **RiSE Framework:** Involves reuse processes (Almeida, 2007), component certification (Alvaro, 2009) and reuse adoption process (Garcia, 2010).

- **RiSE Tools:** Research focused on software reuse tools, such as the Admire Environment (Mascena, 2006), the Basic Asset Retrieval Tool (B.A.R.T) (Santos et al., 2006), which was enhanced with folksonomy mechanisms (Vanderlei et al., 2007), semantic layer (Durao, 2008), facets (Mendes, 2008) and data mining (Martins et al., 2008), the Legacy InFormation retrieval Tool (LIFT) (Brito, 2007), the Reuse Repository System (CORE) (Burégio et al., 2007, 2008), and the Tool for Domain Analysis (ToolDay) (Lisboa, 2008);
1.2. SCOPE

**RiPLE**: Stands for RiSE Product Line Engineering Process and aims at developing a methodology for Software Product Lines (Filho et al., 2008), composed of scoping (de Moraes, 2010), requirements engineering (Neiva, 2009), design (Filho et al., 2009; Cavalcanti et al., 2011), implementation, tests (Machado, 2010; Neto, 2010), and evolution management (Oliveira, 2009);

**RiPLE-TE**: Part of the RiPLE, which encompasses activities for the test discipline. The RiPLE-TE provides a systematic way to handle testing in SPL projects, including variability management concerns (Machado, 2010; Machado et al., 2011).

**SOPLE**: Development of a methodology for Software Product Lines based on services (Medeiros, 2010; Cavalcanti, 2010; Ribeiro, 2010);

**MATRIX**: Investigates the area of measurement in reuse and its impact on quality and productivity;

**BTT**: Research focused on tools for detection of duplicate bug reports, such as in Cavalcanti et al. (2008); Cavalcanti (2010). Thus, this work is part of the BTT research group;
• **Exploratory Research**: Investigates new research directions in software engineering and its impact on reuse;

• **CX-Ray**: Focused on understanding the Recife Center For Advanced Studies and Systems (C.E.S.A.R.)\(^2\), and its processes and practices in software development.

This dissertation is part of the RiSE process for Product Line Engineering (RiPLE) project and its goal is to provide a tool for support the testing process with the goal of effort reduction.

![RiSE Labs Projects](image)

**Figure 1.2** RiSE Labs Projects

### 1.3 Out of Scope

As the proposed tool is part of a broader context, a set of related aspects will be left out of its scope. Thus, the following issues are not directly addressed by this work:

- **Impact on other levels of the testing process.** Our solution concerns only with the system testing level of the SPL testing process. Thus, the main concern is on how this level can be improved by the proposed tool. For this reason, it is out of scope the improvement of other levels (Unit, integration, and acceptance testing).

- **Type of users.** Initially, the subjects of this work can be test analysts, test managers, test designers, test architects and testers. However, it is out of scope to provide a tool that supports users such as developers and other stakeholders with some technical background in software development.

\(^2\)http://www.cesar.org.br/
1.4 Statement of the Contributions

As a result of the work presented in this dissertation, the following contributions can be highlighted:

- A **scoping study on testing tools**, which can provide research community with the state-of-the-art in the field, comprising current and relevant information, extracted from a formalized evaluation, analysis, and synthesis process.

- The **requirements, design, and implementation of the SPLMT-TE tool**, developed within the context of the RiPLE-TE, which encompasses activities for the test discipline. The proposed tool provides support for the SPL testing process. It also includes the creation and management of test assets.

- The **definition, planning, operation and analysis of an experimental study**, aimed at evaluation the proposed solution. It was conducted a pilot study and then the experiment to have extensive feedbacks on the adoption of the tool through observing their outcomes. It was set up and thus documented in a way that enable opportunities for replication

In addition to the contributions mentioned, a paper presenting the findings of this dissertation was published:


1.5 Organization of this Dissertation

The reminder of this dissertation is organized as follows:

- **Chapter 2** reviews the essential topics used in this work: Software Product Lines and Software Testing;

- **Chapter 3** presents a scoping study that investigates the state-of-the-art on single system and software product lines testing tools, performed with the objective of mapping how this area is currently supported by the tools;
1.5. ORGANIZATION OF THIS DISSERTATION

- **Chapter 4** describes the proposed tool discussing the requirements, used technologies, design and the implementation aspects;

- **Chapter 5** reports the experiments performed in order to evaluate the proposed tool. It details the purpose of the experiments and discuss the context in which they took place;

- **Chapter 6** concludes this dissertation by summarizing the findings and proposing future enhancements to the solution, along with some concluding remarks;

- **Appendix A** presents the information sources from where primary studies were extracted, to serve as basis for the scoping study;

- **Appendix B** presents the questionnaire and forms applied in the experiment.
Foundations on Software Product Lines and Testing

Don’t complain; just work harder.
—RANDY PAUSCH (The Last Lecture)

Over the years, the software development process rises to a high level of maturity and complexity. The competitiveness among companies leads to a lot of investment in the area. As a result of this progress, the software starts to become invisible to the end user, making complex tasks more simpler. In order to achieve such evolution, Product line engineering has gained increasing attention, allowing the production of a variety of products in a short period of time coupled with cost reduction through.

The decision of adopting software product lines is based, in general, on economic considerations. Some of reasons that strengthen this option are: support for large-scale reuse, improvements of the software development process, cost reduction, reduction of time to market, and so on.

On the other hand, the drawbacks of software product lines adoption are the demand of extra investments to build reusable assets, make changes at the organization, etc. It also consumes some time to break-even. Once achieved the goals and overcome the obstacles, a product line provides benefits such as: reduction of maintenance costs, quality assurance, more reliable and secure systems, etc. (van der Linden et al., 2007)

This Chapter describes important concepts relevant to this research. It is organized as follows: Section 2.1 introduces Software Product Lines ideas; Software Testing is introduced in Section 2.2; Section 2.3 sketches the relationship between the previously addressed concepts; and, finally, the chapter Summary is presented in Section 2.4
2.1 Software Product Lines (SPL)

The first and foremost difference between single system development and software product line engineering is the change of context: product lines instead of an individual system. This change suggests a modification of strategy: build a family of products instead of only one application.

According to Clements and Northrop (2001):

A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.

Because of the changes in the software development paradigm, is important to discuss in more details it. Thus:

…set of software-intensive systems…
The product line is a set of products offered by producer to customers, which provides the production for a mass market cheaply than individual product creation. Both types of products can be identified in the software domain (Pohl et al., 2005).

…sharing a common, managed set of features…
The variability must be managed throughout software product line engineering. There are three main types. Commonality which identifying ahead of time common features and implementing these for reuse. Variability which is a characteristic that may be common to some product, but not all. Product-specific which is a characteristic that may be part of only one product (van der Linden et al., 2007).

…common set of core assets…
A “core” asset is anything used to produce multiple products (source code, software architecture, test infrastructure, test cases, test data, production plans, etc). The assets are designed to handle the range of variability defined in the product lines scope, and each one is accompanied by an attached process.

…a prescribed way.
A production strategy aligns the business goals with the development of core assets
2.1. SOFTWARE PRODUCT LINES (SPL)

and products. A production plan describes the way in which each product needs to be produced.

2.1.1 SPL Essential Activities

Software Product Lines Engineering involves three essential activities, as shown in Figure 2.1: core asset development (CAD) activity that does not directly aim at developing a product, but rather aims to develop assets to be further reused in other activities, product development (PD) activity which takes advantage of existing, reusable assets, and finally, management activity which includes technical and organizational management (van der Linden et al., 2007). Each of these activities is essential, as is the mixture of all three. The rotating arrows indicate not only that core assets are used to develop product but also that revisions of existing core assets or even new core assets might, and most often do, evolve out of product development. There is also a strong feedback loop between the core assets and the products. Each activity is described below:

![Figure 2.1 Essential Product Line Activities (Northrop, 2002)]

**Core Asset Development**

Also known as domain engineering, is the life cycle that results in the common assets that in conjunction compose the product line’s platform (van der Linden et al., 2007). It involves the evolution of the assets in response to product feedback, new market needs,
and so on (Clements and Northrop, 2001). Figure 2.2 shows the core asset development activity along with its outputs and influential contextual factors.

This activity is iterative, the rotating arrows in Figure 2.2 suggest that its inputs and outputs affect each other. This context influences the way in which the core assets are produced. For example, to expand the product lines, the scope may admit new classes of systems to examine possible sources of legacy assets. The production constraint may lead to restrictions on the product line architecture. This restriction will determine which preexisting assets are candidates for reuse or mining (Northrop, 2002).

The core asset development is divided in five subprocesses: (i) domain requirements encompasses all activities for eliciting and documenting the common and variable requirements of the product line, (ii) domain design encompasses all activities for defining the reference architecture of the product line, (iii) domain realization deals with the detailed design and the implementation of reusable software components, (iv) domain testing is responsible for the validation and verification of reusable components, and (v) evolution management deals with the economic aspects of the software product line and in particular with the market strategy (Pohl et al., 2005).

![Figure 2.2 Core Asset Development](Northrop, 2002)
2.1. SOFTWARE PRODUCT LINES (SPL)

Product Development

Also known as *application engineering*, the main goal of this activity is to create individual products by reusing the core assets previously developed. The CAD outputs (product lines scope, core assets and production plan), in conjunction with the requirements for individual products are the main inputs for PD activity. Figure 2.3 illustrates the product development activity along with its outputs and influential contextual factors.

The product development is divided in four subprocesses: (i) *application requirements* encompasses all activities for developing the application requirements specification, (ii) *application design* encompasses the activities for producing the application architecture, (iii) *application realization* creates the considered application, and finally (iv) *application testing* comprises the activities necessary to validate and verify an application against its specification (Pohl et al., 2005).

The rotating arrows in Figure 2.3 indicate iteration and involved relationships. This activity has an obligation to give feedback on any problems or deficiencies encountered with the core assets, in order to keep the core asset base in accordance with the products.

![Figure 2.3 Product Development (Northrop, 2002)](image)

Management

The management is divided in technical and organizational levels, where technical management is responsible for requirement control and the coordination between core asset
and product development and organizational management is responsible for managerial and organizational activities.

The common set of assets and the plan for how they are used to build product do not just materialize without planning, and they certainly do not come free. They require organizational foresight, investment, planning and direction. They require strategic thinking that looks beyond a single product. The disciplined use of the assets to build products does not just happen either. Management must direct, track, and enforce the use of the assets. Software product lines are as much about business practices as they are about technical practices (Clements and Northrop, 2001).

2.2 Software Testing

The motivation for testing software development projects is concerned to reduce: the risk of unplanned development expense, the risk of project failure, or the most common, the introduction of involuntary errors.

According to Graham (1993):

People and computers are basically incompatible. It is not possible for human beings to write software, which is always perfectly defect-free. Making errors is a result of human creativity in a completely intolerant environment. Hence errors are inevitable in software, but can have serious effects in live operation. The problem with testing is that we did not plan to put the errors in, so it is all too easy to forget to plan to look them and take them out.

Defects in a product can come from any phase. An essential condition should be that every phase of software development (requirements, design, coding, and so on) should catch and correct defects at that phase, without letting this defects escape to the next stage. When the error is found in a phase closer to the end-user, the cost of maintenance will be more expensive. According to Graham (1993), testing is the only way to assess the quality of software in its actual environment. Testing is also a way to achieve and preserve quality.

According to Beizer (1990), Graham (1993), Harrold (2000), Yang et al. (2006), and Bertolino et al. (2007) testing phase can consume fifty percent, or even more, of the total cost of software development. All researchers agree that is an expensive and laborious phase of the software process. Hence, testing tools were among the first software tools to be developed (Sommerville, 2008).
2.2. SOFTWARE TESTING

2.2.1 Test Levels

In order to enhance quality, testing should start at the beginning of a project. It should occur at different places and times during the development life cycle, from the requirements analysis until the final delivery of the application to the custumer. Hence, it is necessary to organize the testing phase into activities that should be achieved in each distinct software development activity. The testing levels are described as follow:

Unit Testing

According to the IEEE Standard Glossary of Software Engineering Terminology, Unit Testing is the testing of individual hardware or software units or groups of related units (IEEE, 1990). It is designed to validate the units created by the implementation phase and is the “lowest” level of testing. Unit testing has as goal the capability to guarantee that each individual software unit is working according to its specification. Unit test is crucial since it evaluates a small and simple portion of a software facilitating the location and detection of errors. It should be performed independently of the other units by an independent team (Collard and Burnstein, 2002).

Integration Testing

It is a testing in which software components, hardware components, or both are combined and tested to evaluate the interaction between them (IEEE, 1990). The integration testing level is designed to assess whether the interfaces between units have consistent assumptions and communicate properly. It should be performed only if the units that have been passed through unit test were approved.

System Testing

Testing conducted on a complete, integrated system to evaluate the system’s compliance with its specified requirements (IEEE, 1990). It should compare the system to its original objectives. Functional behavior and quality requirements such as security, performance and reliability should be evaluated. Figure 2.4 presents several type of system test.

- Functional Tests are responsible for evaluating if the behavior of the system complies with the requirements specification.

- Stress and Load Tests try to break the system, finding scenarios that force until the limit.
2.2. SOFTWARE TESTING

- **Security Tests** insure that the system is safe and secure.

- **Configuration Tests** allow developers and testers evaluate the system performance and availability when modifications happen at the hardware and configuration.

- **Performance Tests** aims to test non-functional requirements such as memory use, response time and delays, etc.

- **Recovery Tests** determine if the system could recover to a well-known state, without loose any previous transaction.

**Acceptance Testing**

Formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system (IEEE, 1990). The acceptance tests are based on requirements, user manual or even system tests. In this level, the software must be executed under the real-world conditions on operational hardware and software.

**2.2.2 Overview on Software Testing Tools**

Software testing tools are used to verify and validate software, helping in the detection of errors. Many failures can be avoided if the defect is detected and resolved during
the development phase. Testing tools now offer a range of facilities and their use can significantly reduce the costs of testing (Sommerville, 2008).

There are different kinds of testing tools for different purposes. We analyzed some of the most common tools to gather information about the area. At least one tool of each category is described next.

**Application Test Tools**

Application test tools are used to test software depending on the activity for which it was designed. Some application packages offer considerable computing power by focusing on a single task; others, called integrated software, offer somewhat less power but include several applications and each one must be tested properly. During this work, we analyzed 70 application test tools organized as follows:

- **Source Test Tools**: groups everything related with source code (code coverage, static analysis and metric, identification of errors during compilation).

- **Functional Test Tools**: validate the functions by feeding them input and examining the output. These tools verify a program by checking it against design documents or specifications. We analyzed 32 functional test tools. For example PETA\(^1\) is a platform for automated functional testing of distributed software systems such as client-server or service oriented architectures (SOA) on the integration or system layer. PETA contains Eclipse\(^2\) based tools for test case development, automated test execution and result analysis.

- **Performance Test Tools**: cover a broad range of engineering or functional evaluations where a material, product, system, or person is not specified by detailed material or component specifications: rather, emphasis is on the final measurable performance characteristics. Performance Test Tools are used to determine the speed or effectiveness of the application.

This process can involve quantitative tests, such as measuring the response time or the number of MIPS (millions of instructions per second) at which a system functions. Qualitative attributes such as reliability, scalability and interoperability may also be evaluated. There tools for stress test considered as the performance category. We analyzed 12 Performance test tools. For example, Silk Performer\(^3\)

---

2.2. SOFTWARE TESTING

offers a cost-effective, enterprise-class tool for software application performance and load testing, delivering automated software load, stress and performance testing in an open and sharable model.

- **Java Test Tools**: groups every tool related with Java technology. Java Test tools ensure that the problems in source code are addressed during the development. Java Test tools also ensure fully functional, highly intelligent test classes that understand the application and test the source files thoroughly. Java test tool must recognize errors conditions such as Memory leaks, Race conditions, and Thread locks, causing the application to slow down or even crash.

We analyzed 26 Java Test Tools. For example Junit\(^4\): a unit testing framework for the Java programming language. JUnit has been important in the development of test-driven development, and is one of a family of unit testing frameworks collectively known as xUnit that originated with SUnit.

- **Embedded Test Tools**: are responsible for test embedded (real time) systems. Embedded test tools allow detection of timing issues when designed to replicate the system timing as closely as possible using system clocks instead of separate (usually low-speed) test clocks. Another feature of embedded test tools, known as fault insertion, allows software groups to verify that faults within a system actually invoke the correct recovery and diagnosis routines.

- **Database Test Tools**: focus on tools that test the referential integrity of a database, the database security including database permissions and privileges, the interfaces to the database such as GUIs, Java, and web services, the Application programming interfaces (APIs) such as ODBC\(^5\), JDBC\(^6\), and OLEDB\(^7\), the data-format integrity and the database and datamart loads through specialized load tools.

**Web Test Tools**

Web Test Tools are responsible for testing applications that are accessed over a network such as the Internet or an intranet. Web applications are popular due to the ubiquity of web browsers, and the convenience of using a web browser as a client. Due to this,

---

\(^4\)http://www.junit.org/

\(^5\)Open Data Base Connectivity

\(^6\)Java Database Connectivity

\(^7\)Object Linking and Embedding, Database
is extremely necessary to test the web applications in different browsers and operational systems.

The ability to update and maintain web applications without distributing and installing software on potentially thousands of client computers is a key reason for their popularity, as is the inherent support for cross-platform compatibility. Common web applications include webmail, online retail sales, online auctions, wikis and many other functions to be tested in many categories. In our work, we analyzed 69 web test tools organized as follows:

- **Link and HTML Test Tools** and **On-line Link and HTML Test Services**: consist of verifying if the connections with hyperlinks are correct and the identification of the HTML and services related are working.

- **Functional Test Tools**: is similar to functional test tools for applications, but the focus changes to web systems and its particularities. We analyzed 38 functional Test Tools for web applications, such as: Selenium[^8]: a suite of tools that supports rapid development of test automation for web-based applications. Selenium provides a rich set of testing functions specifically geared to the needs of testing of a web application.

- **Security Test Tools**: try to determine that an information system protects data and maintains functionality as intended. The six basic security concepts (Lehtinen et al., 2006) that need to be covered by security test tools are: confidentiality, integrity, authentication, availability, authorization and non-repudiation. We found only one Security Test Tool, QAInspect[^9] from HP, conducts and manages functional testing and website security testing from a single platform without the need for specialized security knowledge. HP QAInspect finds and prioritizes website security vulnerabilities in a web application and presents detailed information and remediation advice for each vulnerability.

- **Performance Test Tools** and **Performance Test Services** is similar to performance test tools for applications, but the focus changes to web systems and its particularities. We analyzed 30 performance test tools and performance test services Test Tools for web applications. For instance, SiteStress[^10] which runs a series of tests, and reviews the results meaning. There are two services available,

[^9]: [http://goo.gl/0xj1J](http://goo.gl/0xj1J)
2.2. SOFTWARE TESTING

Full-Service Load Testing solution that includes a dedicated analyst who is assigned for scripting, running tests and providing analysis and recommendations, and On-Demand Load Testing allow the user to run tests at any time, night or day.

Test Management Tools

Test management tools are used to structure automated tests and manual test processes, and manage multiple environments. Quality assurance teams use these types of tools as a single application for managing test cases, environments, automated tests, defects and project tasks. Test management tools streamline the testing process and offer quick access to data analysis, collaborative tools and communication across multiple project teams. Tracking of bugs, defects and project tasks are done within one application, simplifying processes and saving teams time and money.

We analysed 8 Test Management Tools such as, HP Quality Center\(^{11}\) from Hewlett-Packard, which offers a single, scalable, Web-based application that supports all aspects of application quality test management and delivery. It enables a consistent, repeatable process for: Requirements management, Test planning, scheduling and execution, Release and cycle management, and finally Defect management. Rational Quality Manager\(^{12}\) from IBM is a web-based centralized test management environment for business, system and IT decision makers and quality professionals who seek a collaborative and customizable solution for test planning, workflow control, tracking and metrics reporting how project decisions and deliverable impact and align with business objectives.

API Test Tools

API (application programming interface) testing tools test the application using public and private APIs. It focus on the validation of interfaces.

Communications Test Tools

Communications Test Tools group the tools that test network communication between applications, network performance, multiple concurrency, etc.

\(^{11}\)http://goo.gl/hlOU  
\(^{12}\)http://www.ibm.com/software/awdtools/rqm/
2.3 Testing in Software Product Lines

Testing in a product line company examines the core asset software, the product-specific software, the interaction between them, and ultimately the completed products. Responsibilities for testing in a product line organization may be distributed differently than in a single system development effort (Clements and Northrop, 2001).

Also, unlike single system development projects, testing is an activity (and producer of artifacts) whose outputs is reused across a multitude of products. It utilizes activities from the validation of the initial requirements model to verify activities executed by customers to accept a product. Planning is necessary to take full advantage of the benefits that reuse brings.

According to McGregor (2001):

The testing activities vary in scope from encompassing the entire product line to focusing on a specific product down to examining an individual component that is one part of a product. These activities address added dimensions of complexity beyond those in a typical development project.

Testing is a fundamental component of the product line assets development, addressed in the production plan. The same existent reuse opportunities from assets created to support the testing process can be used to create assets for development (McGregor, 2001).

2.3.1 Testing in core asset development

The number of variation points and possible values for each variation make the testing of all possible products a tremendous task and testing all variants of core assets is usually an impractical activity.

Another point to consider is that the core assets are created and tested by the core-asset teams. Individual products are composed by specialized versions of these assets usually tested by the product team that creates the product. When different products are using the same asset (or set of assets) do not need to retest everything, but rather use the results achieved before.

Unit Testing

The core asset developers are responsible for unit Testing. The unit tests are reused across versions of the product line component set and among specific products. The
software produced for the tests is structured to connect the units in the production with the test software, allowing the traceability between test and the production software units (McGregor, 2001).

Integration Testing

Integration is a shared responsibility between core-assets builders and the product builders. At some point, units will be integrated among themselves in order to build a specific product that will be tested by the product team. This integration and integration testing continue constantly until the units form a completed product. A number of variants may be produced for use at each variation point. However, this makes it impossible to test the integration of all possible combinations of all variations (McGregor, 2001).

Complete integration and system testing in CAD is usually not yet specified (Tevanlinna et al., 2004). Try to test the different combinations of components leads to exponential growth of tested configurations (McGregor, 2001). It is hard to decide on how much we can depend on the domain testing. According to Tevanlinna et al. (2004) it does not seem to be clear where testing belongs in the overall SPL process.

2.3.2 Testing in product development

Testing a product requires a different perspective from testing individual assets. The focus changes from correctness and completeness to conformance. Each product must reach external requirements (McGregor, 2001).

Product development testing is performed to ensure that an application is of sufficient quality. Although the core assets were tested, this does not mean that they should not be tested again. The expensive nature of the product line scale leads to the reuse of tests, due to the redundant testing of the common assets.

For those products requirements that are equal to core assets requirements, core assets tests can be repeated. Other tests will contain variation points that must be bound to match the product. For these specific product requirements, new tests must be developed and executed.

The role of the product tester encompasses the testing of single applications. This results at the use of the core asset testing strategy and the reusable test assets provided by the core asset test team. Towards the other product development roles, the product tester has the normal roles of executing integration and system tests (van der Linden et al., 2007).
2.4. CHAPTER SUMMARY

2.3.3 SPL Testing Tools

A software product line is a large system, with large volume of source code, and testing tools can aid developers and testers to scan through the large volume of product lines source code and reveal unreliable program features to the human program. Testing tools can support testing large size of software product lines to achieve its goals (Edwin, 2007).

Product lines and single system testing consist of different activities; consequently they require comprehensive and careful attention to most if not all of the areas in software product lines testing process. Therefore, tools support individual tasks in product lines practice areas. Tools can help software product line suppliers to develop specific products right from requirements to architecture to component, unit and system testing (Edwin, 2007).

An important factor in SPLT is automated testing and the use of testing tools. This decreases the effort when reusing test assets and makes the complex testing process more manageable (Tevanlinna et al., 2004). Test case creation is a time consuming activity in software product lines development. This effort can be reduced by creating test cases starting at product line level, and then extend them to specific products. Moreover, the application of these test cases is another challenging task that is time consuming (Edwin, 2007).

There are numerous mature testing tools available. Despite the seeming abundance of available testing tools SPL testing lacks efficient tool support. The problem with testing tools is that few of them directly support SPL (Tevanlinna et al., 2004). According to Edwin (2007), SPL not only require automated tool support, but require robust and specific kind of tool support. Taking into consideration, core assets used for building products have to express the variation and commonality among products assets.

2.4 Chapter Summary

In this chapter, we introduced briefly, the area of software product lines, software testing, software testing tools and then discussed how these concepts can interact, since they are fundamental to understand the purpose of the work described in this dissertation.

Next chapter presents the research process that we used to map the tools found at the literature. Some applications used by companies and developed for the industry with commercial proposes were analyzed. We also investigated tools published in academic projects.
A Scoping Study on Software Testing Tools

*If you lead your life the right way, the karma will take care of itself. The dreams will come to you.*

—RANDY PAUSCH (The Last Lecture)

This chapter introduces a systematic scoping study performed with a set of five research questions, in which 33 studies, dated from 1999 to 2011, were evaluated. The study aims to investigate the state-of-the-art regarding to testing tools, synthesize available evidence, and identify gaps among the tools, available in the literature.

Even with several existing testing tools, they are mostly unable of supporting a SPL Testing process directly. Moreover, the majority of the proposed tools and prototypes regarding SPL were not implemented, and some times when the implementation is available, the literature just gives a brief overview. It indicates that additional investigation, empirical and practical, should be performed.

Furthermore, we believe that this chapter can also provides insights for new research in the software testing tools area, serving as a baseline, by analyzing the available tools and presenting the existing gaps.

The remainder of this chapter is organized as follows: Section 3.1 presents the related work. In Section 3.2, the method used in this study is described. Section 3.3 presents the planning phase and the research questions addressed by this study. Section 3.4 describes its execution, presenting the search strategy used and the selected studies and tools. Section 3.5 presents the classification scheme adopted in this study and reports the findings. In Section 3.6, the threats to validity are described.
3.1 Related Work

Several studies emphasize the need for testing tools to support the Testing Process, however, just a few ones focus on Software Product Lines. Amongst them, we have identified some studies in order to gather and evaluate the available evidence in the area. Thus, they were considered as having similar ideas to our scoping study and are next described.

A survey on SPL Testing was performed by Tevanlinna et al. (2004) in which the authors evaluate the state-of-the-art in SPL Testing highlighting the importance of testing tools for support SPL. In summary, the study investigates the need for product lines specific testing tools that should help to manage the reusable testing assets, automate the test execution activity, and the analysis of their results.

In (Tevanlinna et al., 2004), the authors explore specifically regression testing, testing of partial programs, frameworks and component testing. Our work analyzed all the test levels (unit, integration, system and acceptance) of the SPL Testing Process including regression testing.

In (Edwin, 2007), the author describes a systematic review, which aims to find out the primary studies relating to Software Product Lines Testing Tools. It investigates the tools that can be used to support testing in software product lines context to ensure high quality in software product line and its specific products.

Furthermore, according to Edwin (2007) more research is required in this area of specific and sophisticated tools for the software product lines testing process. In order to extend the research of Edwin (2007), we elaborated a scoping study that investigates single system testing tools and SPL testing tools. Finally, we also identified the possibility to adapt single systems testing tools to test SPL.

Lamancha et al. (2009) present a systematic review of the literature, which deals with testing in software product lines. The authors analyzed the existing approaches to testing in SPL, discussing the significant issues related to this area. The work also discussed which papers described software product lines testing tools.

Neto et al. (2011a) presented a systematic mapping study performed in order to map out the SPL Testing field, through synthesizing evidence to suggest important implications for practice, as well as identifying research trends, open issues, and areas for improvement.
3.2 Research Methodology

This study has been structured combining ideas from Petersen et al. (2008) with good practices defined in the guidelines proposed by Kitchenham and Charters (2007), such as the protocol definition. Therefore, we applied a process for a mapping study (MS), including best practices for conducting systematic reviews (SR), making the best use of both techniques.

According to Budgen et al. (2008), a MS provides a systematic and objective procedure for identifying the nature and extent of the empirical study data that is available to answer a particular research question. While a SR is a mean of identifying, evaluating and interpreting all available research relevant to a particular question (Kitchenham and Charters, 2007).

Figure 3.1 presents the phasing mapping study adapted from (Petersen et al., 2008). Some elements were changed and/or rearranged. The process was divided into three main phases: Research Directives which establishes the protocol and research questions, Data Collection which comprises the execution of the MS and the Results responsible for report the primary study outcomes and analysis. All phases are detailed in next sections.

Figure 3.1 The Systematic Mapping Process (adapted from Petersen et al. (2008))

3.3 Research Directives

This section presents the first phase of the study process, in which the protocol and the research questions are defined.
3.3. RESEARCH DIRECTIVES

3.3.1 Protocol Definition

This artifact was adopted from systematic review guidelines. The protocol defines the research plan and directives for an empirical study. Our initial activity was to develop a protocol in order to guide the research objectives and define how it should be executed.

Incremental reviews were performed according to the MS method. The protocol was also revisited in order to update the data based on new information collected during the study progress.

3.3.2 Question Structure

The research questions were framed by three criteria:

- **Population.** Published scientific literature reporting single systems testing tools and SPL testing tools, thus the population is composed by test analysts and/or test experts seeking a way to have a more automated process support, and by researchers in the test engineering/software product line areas.

- **Intervention.** Empirical studies involving single systems testing tools and SPL testing tools, software technology, and procedures. This review must search for indications that the SPL testing process can be fully supported.

- **Outcomes.** Type and quantity of evidence related to several single systems testing tools and SPL testing tools, in order to identify the main functionalities concerning to this area. The objective of this study is to map how the tools are supporting a SPL testing process. If the process is not fully supported, i.e., it has many gaps existing in the tools or if there is a necessity of using several tools in the whole process; the necessity for a new SPL testing tool increases.

3.3.3 Research Questions

The objective of this study is to investigate how the available tools support the Software Product Lines Testing process? Through this Research Question (RQ), it will be possible to identify if the existing tools support the same functionalities among them and how good enough is it.

- **RQ1 - Do the tools support the SPL Testing Process?** The idea of not being focused on software product lines process initially increases the chances of embrac-
RQ2 - Where the tools were developed and used? It aims at identifying where the tools were developed and how they have been validated. Through these descriptions, it is possible to map the current adoption of the tools.

There are also others three research questions that complete the research. Answering these questions led a detailed investigation of testing tools, which support both industrial and academic context.

- RQ3 - Is it possible to use the single system testing tools to test software product lines? The objective is to analyze the possibility to use the single system testing tools to test SPL instead of create tools from the scratch.

- RQ4 - Which test levels the existing testing tools support? It aims to classify the tools according to the test levels in order to identify what the tools can offer.

- RQ5 - How the testing tools are evolving? The last research question intended to identify the evolution of the Testing tools considering the rise of the functionalities complexity. We also analyzed the amount of activities the tools could handle.

3.4 Data Collection

Aiming at answering the research questions, data was collected from the research literature. These activities involve the search strategy, the studies selection and the data analysis, extraction and synthesis.

3.4.1 Search Strategy

The search strategy was developed by reviewing the data needed to answer each of the research questions. We divided the strategy in two phases. The first one focused on Single System Testing Tools (SSTT). The second part focused on Software Product Lines Testing Tools (SPLTT). The initial set of keywords was refined after a preliminary search that returned many results with few relevance. We used several combinations of search items until achieving a suitable set of keywords.

Furthermore, search strings could then be constructed using boolean AND’s and OR’s. At the second part of the search, all terms were combined with the term “Product Lines”,
3.4. DATA COLLECTION

“Product Family” and “SPL” by using Boolean “AND” operator. All of them were joined by using “OR” operator so that it could improve the completeness of the results. The complete list of search strings is available in Table 3.1.

<table>
<thead>
<tr>
<th>Research Strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (“Testing” OR “Test” OR “Tests”) AND Tool</td>
</tr>
<tr>
<td>2 “Tool for Testing” OR “Tool for Test”</td>
</tr>
<tr>
<td>3 (“Testing” OR “Test”) AND (Framework OR Application)</td>
</tr>
<tr>
<td>4 “Application for Test” OR “Application for Testing”</td>
</tr>
<tr>
<td>5 (“Automatic” OR “Automation”) AND (“Testing” OR “Test”) AND Tool</td>
</tr>
<tr>
<td>6 (“Testing” OR “Test”) AND Generation AND Tool</td>
</tr>
<tr>
<td>7 Automatic AND (“Testing” OR “Test”) AND Generation AND Tool</td>
</tr>
<tr>
<td>8 (“Testing” OR “Test” OR “Tests”) AND Tool AND (“product lines” OR “product family” OR “SPL”)</td>
</tr>
<tr>
<td>9 (“Tool for Testing” OR “Tool for Test”) AND (“product lines” OR “product family” OR “SPL”)</td>
</tr>
<tr>
<td>10 (“Testing” OR “Test”) AND Generation AND Tool AND (“product lines” OR “product family” OR “SPL”)</td>
</tr>
<tr>
<td>11 (“Application for Test” OR “Application for Testing”) AND (“product lines” OR “product family” OR “SPL”)</td>
</tr>
<tr>
<td>12 (“Automatic” OR “Automation”) AND (“Testing” OR “Test”) AND Tool AND (“product lines” OR “product family” OR “SPL”)</td>
</tr>
<tr>
<td>13 (“Testing” OR “Test”) AND Generation AND Tool AND (“product lines” OR “product family” OR “SPL”)</td>
</tr>
<tr>
<td>14 Automatic AND (“Testing” OR “Test”) AND Generation AND Tool AND (“product lines” OR “product family” OR “SPL”)</td>
</tr>
</tbody>
</table>

3.4.2 Data Sources

The search was executed using three steps: (i) the search strings in Table 3.1 were adjusted and applied in each digital database, all of the search strings were systematically checked by more than one author. The list of sources, in alphabetical order, is the following: ACM Digital Library⁴, Elsevier⁵, IEEE Computer Society Digital Library⁶, Science@Direct⁷, The DBLP Computer Science Bibliography⁸ and Springer Link⁹; (ii) a manual search was performed in the main Journals and Conferences, which are detailed in Appendix A. These libraries were chosen because they are some of the most relevant sources in software engineering (Kitchenham and Charters, 2007); (iii) and finally, the search was also performed using the ‘snow-balling’ process (Budgen et al., 2008), following up the references in papers and it was extended to include grey literature sources, seeking

---

¹http://portal.acm.org/
²http://www.elsevier.com/
³http://ieeexplore.ieee.org/
⁴http://www.sciencedirect.com/
⁵http://www.informatik.uni-trier.de/ley/db/
⁶http://springerlink.metapress.com/home/main.mpx
relevant white papers, industrial (and technical) reports, theses, work-in-progress, and books. Table 3.2 shows the number of tools selected in each category.

<table>
<thead>
<tr>
<th>Search Type</th>
<th>SSTT</th>
<th>SPLTT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic Search</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>Manual Search</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>Search on sites + snow-balling</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>24</strong></td>
<td><strong>9</strong></td>
</tr>
</tbody>
</table>

Furthermore, the described search engines are focused on academic results; given the fact that the goal was to find the largest number of tools as possible, and these engines would not find commercial tools (as long as they do not have any paper or journal published), these keywords were also used in search in web search engines, such as Google. In the web engines, the target was tools information and their grey literature (i.e. technical reports, white-papers, manuals and works in progress).

### 3.4.3 Study Selection

A set of 69 studies involving both single system and SPL testing tools was found. Figure 3.2 describes each filter applied during the study selection. Moreover, the Figure depicts the amount of studies remaining after applying each filter.

![Figure 3.2 Study Selection Summary](http://www.google.com)
The inclusion criteria were used to select all studies during the search step. After that, the same exclusion criteria was firstly applied in the studies title and next in the abstracts and conclusions. All excluded studies can be seen by differentiating the results among filters. Regarding the inclusion criteria, the studies were included if they described:

- **Tools that offer support to the testing phase.** The found tools must support some functionality in the testing process;

- **Tools that support single system testing and software product lines testing.** Tools developed for single system were considered because they allow to analyze if the SSTT could be used at the SPL testing process;

- **Tools with available executable.** Prototypes, demonstrations or finalized products allow to verify how the tools fulfilled the testing process;

- **Tools with documentation.** Documentations are useful because they describe the tools functionalities.

Studies were excluded if they described:

- **Testing Tools that were not implemented.** Tools with written documentation that did not implement the functionalities were discarded;

- **Conceptual Testing and conceptual testing frameworks.** In order to measure how existing testing tools supported SPL, conceptual testing tools and conceptual testing frameworks were eliminated;

- **Duplicated studies.** When a study has been published in more than one publication, the most complete version was considered.

After the first stage, 47 papers were selected (32 from single and 15 from SPL). In the second stage, we limited the publication venues to international journals (no magazines were included) and excluded studies on the basis of exclusion criteria applied to abstract and conclusion.

After this stage, there were 33 papers, which were considered to analysis (24 from single system and 9 from SPL). Figure 3.3, presents the single system and SPL testing tools regarding the publication years.

An important point to highlight is that in 2007 two international conferences, 29th International Conference on Software Engineering (ICSE’07) and 22nd International Conference on Automated Software Engineering (ASE’07) demonstrated the interest
3.4. DATA COLLECTION

of the research community on improving the Testing Tools field. Figure 3.4 shows the amount of publications considering these sources. In fact, it can be seen that peak in Figure 3.3 matches with the year when theses conferences occurred.

![Figure 3.3 Distribution of primary studies of single systems and SPL testing tools by their publication years](image)

**Figure 3.3** Distribution of primary studies of single systems and SPL testing tools by their publication years

![Figure 3.4 Amount of Studies vs. sources](image)

**Figure 3.4** Amount of Studies vs. sources

3.4.4 Data Extraction

All the 33 studies were fully read and submitted to a predefined form to accurately record the information obtained by the researchers from the primary studies. The form for data
3.4. DATA COLLECTION

extraction provides some standard information, such as:

- Tool’s name;
- Date of data extraction;
- Title, authors, journal, publication details (if available);
- Prototype information (if available);
- Website (if available); and,
- A list of each conclusion and statement encountered for each question

3.4.5 Testing Tools

Based on the research results, inclusion and exclusion criteria, a set of tools were selected, which is detailed next. A brief description of SSTT and SPLTT tools are following presented ordered by the publication year.

Selected Single System Testing Tools

- **IDATG** ([Beer et al., 1998](#)) - The IDATG (Integrating Design and Automated Test Case Generation) specification technique and tool design supports both the specification of the behavior of a user interface and the generation of two types of test case for GUI coverage and for checking the usability of the application.

- **SOSTEM** ([Liut et al., 2000](#)) - The testing tool, called SOSTEM (SOfl Specification TEsting Machine), is designed using SOFL (formal language and method for system specification and design) and implemented using Java. It supports five functionalities: generation of proof obligations, interactive testing, batch testing, test results analysis, and test information management. The tool can be used for both unit testing and integration testing.

- **AsmL** ([Barnett et al., 2003](#)) - The AsmL supports the generation of parameters, call sequences, and the conduction of conformance tests. The tool performs a semi-automatic approach, requiring a user to annotate models with information for generating tests.
3.4. DATA COLLECTION

- **DUTF (Wu and Gray, 2005)** - The DUTF (Domain-Specific Language Unit Test Framework) assists in the construction of test cases for DSL programs. Failed test cases reported within the DUTF reveal the presence of a program fault.

- **STSC (Shukla et al., 2005)** - The “STSC” prototype tool supports the statistical testing of software components. The tool supports a wide range of operational profiles and test oracles for test case generation and output evaluation. The tool also generates appropriate values for different types of input parameters of operations.

- **JTest (Xie and Notkin, 2006)** - Jtest can automatically generates test inputs to perform white-box testing. When specifications are provided with the class, Jtest can make use of them to perform black-box testing.

- **COMPTest (Gao et al., 2006)** - The COMPTest is a component test tool that can be used to automatically identify component-based API changes and impacts, as well as reusable test cases in a component test suite.

- **SimC (Xu and Zhang, 2006)** - The SimC is a prototype tool, which automatically generates test data for unit testing of C programs involving pointer and structure operations. The tool symbolically simulates the execution of the given program.

- **Taxi (Bertolino et al., 2007)** - The TAXI implements the XML-based Partition Testing approach for the automated generation of XML Instances conforming to a given XML Schema. In addition, it provides a set of weighted test strategies to guide the systematic derivation of instances.

- **WebSob (Martin et al., 2007)** - WebSob is a tool for automatically generating and executing web-service requests given a service provider’s Web Service Description Language (WSDL) specification.

- **Korat (Milicevic et al., 2007)** - Korat is a tool for constraint-based generation of structurally complex test inputs for Java programs. Korat takes an imperative predicate that specifies the desired structural integrity constraints and a finalization that bounds the desired test input size.

- **ATTEST (Ren and Chang, 2007)** - ATTEST is a toolkit to address problems in test automation and maintenance. It provides easy-to-use mechanisms for helping testers to write and maintain automated test scripts through describing system behaviors at a high abstract level.
3.4. DATA COLLECTION

- **TaRGeT** (*Nogueira et al., 2007*) - Test and requirements generation tool (TaRGet) is a tool for automatic test case generation from use case scenarios written in natural language (NL). It automates a systematic approach for dealing with requirements and test artifacts in an integrated way.

- **CodeGenie** (*Lemos et al., 2007*) - CodeGenie is a tool that implements a test-driven approach to search and reuse of code available on large-scale code repositories. With CodeGenie, developers design test cases for a desired feature first, similar to Test Driven Development (TDD).

- **Jwalk** (*Simons, 2007*) - Jwalk is a unit-testing tool developed to address the need for systematic unit testing within the context of agile methods. The tool operates on the compiled code for Java classes and uses a new lazy method for inducing the changing design of a class on the fly.

- **Smart** (*Xie et al., 2007*) - Smart is a SysteM for Application Reference Testing, novel approach for generating tests for web services from legacy GAPs (Graphical User Interface (GUI) APplications). This tool enables non-programmers to generate unit test cases for web services by performing drag-and-drop operations on GUI elements of legacy GAPs.

- **UnITeD** (*Pinte et al., 2008*) - The tool UnITeD enables the automatic generation of optimized component test sets from UML behavioral description by state machines or activity diagrams according to structural model-based testing criteria, such as the coverage of states, transitions and pairs of transitions. The tool is also capable of supporting integration testing, provided the behavior of interacting components is described by communicating state machines.

- **AgitarOne** (*Daniel and Boshernitsan, 2008*) - AgitarOne is a comprehensive unit testing product for Java. It can use run-time feedback and limited on-the-fly static analysis to generate input data or creates regression tests using constraint solving and mock objects.

- **REST** (*Xie et al., 2008*) - REST is a Reducing Effort in Script-based Testing for guiding test personnel through changes in test scripts so that they can use these modified scripts to test new versions of their respective GAPs. This tool enables test personnel to maintain and evolve test scripts with a high degree of automation and precision.
• **JUnitMX** (*Wloka et al.*, 2009) - JUnitMX is a unit testing tool that leverages a change model to assist developers in the creation of new unit tests. The tool provides quantitative feedback and detailed information about change effects, which facilitate the writing of more effective tests and motivate developers with an achievable coverage goal.

• **WebDiff** (*Choudhary et al.*, 2010) - WebDiff is a technique and tool for detecting cross-browser issues automatically and reporting the locations of such issues in the corresponding web page.

• **CoGenTe** (*Rajeev et al.*, 2010) - CoGenTe is a tool for meta-model based testing of code generators. It generates test cases to cover the syntactic aspects of a translation and complex semantic aspects.

• **TED** (*Mishra and Sonawane, 2010*) - TED enables integrated testing and debugging of software stack and underlying hardware while providing more flexibility and control to user. It can be used over any implementation of user direct access programming library and is available as open source.

• **AutoBlackTest** (*Mariani et al.*, 2011) - AutoBlackTest is a tool for automatic generation of test cases for interactive application. It interacts with the application though its GUI, and uses reinforcement learning techniques to understand the interaction modalities and to generate relevant testing scenarios.

**Selected Software Product Lines Testing Tools**

• **Tool** (*Stephenson et al.*, 2004) - The authors implemented a tool that can automatically generate test data to distinguish between models. Such automation is based on the ability of simulating or running models.

• **ScenTED DTCD** (*Reuys et al.*, 2005) - ScenTED-DTCD tool (Domain Test Case Scenario Derivation) was implemented in order to support the ScenTED technique (Scenario based TEst case Derivation), which aims at reducing effort in product family testing.

• **XUnit** (*Galli et al.*, 2005) - XUnit in its various forms is a widely used open-source unit testing framework. It has been ported to most object-oriented programming
languages and is integrated in many common Integrated Development Environment (IDE)s such as Eclipse\textsuperscript{8}.

- **ASADAL** (Kim et al., 2006) - ASADAL (A System Analysis and Design Aid tool) supports the entire life cycle of software development process based on a Product Line Software Engineering method called FORM (Feature-Oriented Reuse Method). It supports domain analysis, architecture and component design, code generation, and simulation-based verification and validation.

- **Objecteering** (Nebut et al., 2004, 2007) - Objecteering is a prototype tool to automatically generate both functional and robustness test cases specific to a product from the Product Families requirements.

- **GATE** (Feng et al., 2007) - GATE is a prototype tool that implements an efficient mechanism of testing component with high automation. The framework facilitates the reusability and automatic creation of aspect test cases, with software product lines techniques.

- **ParTeG** (Weissleder et al., 2008) - The tool Partition Test Generator (ParTeG), supports the automatic generation of boundary tests. ParTeG generates JUnit test suites for both configurations. Thus, the test suites are executable without any further effort. The tool also allows the user to choose a coverage criterion, which should be satisfied as good as possible by the generated test suite.

- **Kesit** (Uzuncaova et al., 2008) - The prototype Kesit shows that incremental test generation provides significant performance improvements over conventional means of test input generation. Kesit not only generates tests, but also scales to generation of larger inputs, which enables novel strategies for software testing.

- **MoSo-PoLiTe** (Oster et al., 2011) - The MoSo-PoLiTe framework provides a test framework for SPL. It is a tool chain that contains a pairwise configuration selection component on the basis of a feature model.

### 3.5 Outcomes

In this section, each research question is answered by analyzing different point of views, highlighting the findings gathered from the data extraction process.

\textsuperscript{8}http://www.eclipse.org/
3.5. OUTCOMES

How the available tools support the Software Product Lines Testing process?

A testing process should specify and separate the testing characteristics, such as the testing stages and also testing techniques (Machado et al., 2011). As showed in Figure 3.5, a small number of tools support the testing process, only 3 single system test tools and 4 SPL test tool support a testing process, corresponding to 21% of the analyzed tools. Only 12.5% of SSTT and 45% of SPLTT support the testing process. Table 3.3 organizes the tools according with the Testing process support.

![Figure 3.5 Tools that support the Testing process](image)

Tools Development and Usage

From the selected tools, most of them were developed in the academic environment (14 for single systems and 7 for SPL) while 7 were developed exclusively in industry (6 for single system and 1 for SPL). The remaining 5 tools were developed in both environments, academic and industrial (4 for single systems and 1 for SPL), as shown in Figure 3.6 and detailed in Table 3.4. There were some troubles in finding where the tools were used. Since there was the information in which environment the tool was developed, it was assumed that the tool was used at least once in this environment, even without a report detailing it.

Software Product Lines Adaptability

According to the results, 33% of the single system testing tools can be used to test SPL. The other tools were implemented to specific programming language, techniques and approaches that cannot be suitable to the SPL context.

Websob (Martin et al., 2007), Korat (Milicevic et al., 2007), CodeGenie (Lemos et al., 2007), JWalk (Simons, 2007), Smart (Xie et al., 2007), REST (Xie et al., 2008) and

38
### Table 3.3 Testing Process Support

<table>
<thead>
<tr>
<th>Testing Process support</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFTT</td>
<td>(Beer et al., 1998),</td>
</tr>
<tr>
<td></td>
<td>(Simons, 2007),</td>
</tr>
<tr>
<td></td>
<td>(Nogueira et al., 2007)</td>
</tr>
<tr>
<td></td>
<td>(Liut et al., 2000),</td>
</tr>
<tr>
<td></td>
<td>(Barnett et al., 2003),</td>
</tr>
<tr>
<td></td>
<td>(Wu and Gray, 2005),</td>
</tr>
<tr>
<td></td>
<td>(Shukla et al., 2005),</td>
</tr>
<tr>
<td></td>
<td>(Xie and Notkin, 2006),</td>
</tr>
<tr>
<td></td>
<td>(Gao et al., 2006),</td>
</tr>
<tr>
<td></td>
<td>(Xu and Zhang, 2006),</td>
</tr>
<tr>
<td></td>
<td>(Bertolino et al., 2007),</td>
</tr>
<tr>
<td></td>
<td>(Martin et al., 2007),</td>
</tr>
<tr>
<td></td>
<td>(Milicevic et al., 2007),</td>
</tr>
<tr>
<td></td>
<td>(Ren and Chang, 2007),</td>
</tr>
<tr>
<td></td>
<td>(Lemos et al., 2007),</td>
</tr>
<tr>
<td></td>
<td>(Xie et al., 2007),</td>
</tr>
<tr>
<td></td>
<td>(Pinte et al., 2008),</td>
</tr>
<tr>
<td></td>
<td>(Daniel and Boshernitsan, 2008),</td>
</tr>
<tr>
<td></td>
<td>(Xie et al., 2008),</td>
</tr>
<tr>
<td></td>
<td>(Wloka et al., 2009),</td>
</tr>
<tr>
<td></td>
<td>(Choudhary et al., 2010)</td>
</tr>
<tr>
<td></td>
<td>(Rajeev et al., 2010),</td>
</tr>
<tr>
<td></td>
<td>(Mishra and Sonawane, 2010),</td>
</tr>
<tr>
<td></td>
<td>(Mariani et al., 2011)</td>
</tr>
<tr>
<td>SPLTT</td>
<td>(Reuys et al., 2005),</td>
</tr>
<tr>
<td></td>
<td>(Nebut et al., 2007),</td>
</tr>
<tr>
<td></td>
<td>(Weissleder et al., 2008),</td>
</tr>
<tr>
<td></td>
<td>(Oster et al., 2011)</td>
</tr>
<tr>
<td></td>
<td>(Stephenson et al., 2004),</td>
</tr>
<tr>
<td></td>
<td>(Galli et al., 2005),</td>
</tr>
<tr>
<td></td>
<td>(Kim et al., 2006),</td>
</tr>
<tr>
<td></td>
<td>(Feng et al., 2007),</td>
</tr>
<tr>
<td></td>
<td>(Uzuncaova et al., 2008)</td>
</tr>
</tbody>
</table>

JUnitMX (Wloka et al., 2009) are tools able to be utilized at the SPL testing process.

Testing tools with specific purpose such as: test management, bug report, security test, can be used in Software Product Lines testing if the methodology of using is adapted to suit the SPL aspects such as commonalities and variabilities.

### Testing Levels

This review identified that many of the analyzed tools have similar functionalities. Moreover, the extracted functionalities have analogous goals, so it was possible to group them. This classification matched the description presented by McGregor (2001). The groups are:

- **Unit Testing** - Tools that test the smallest unit of software implementation. This unit can be basically a class, or even a module, a function, or a software component Neto et al. (2011a).
3.5. OUTCOMES

Figure 3.6 Where the tools were developed and used

- **Integration Testing** - Tools that tests the integration between modules or within the reference in domain-level when the architecture calls for specific domain components to be integrated in multiple systems *Neto et al. (2011a).*

- **System Testing** - Tools that ensure that the final product matches the required features *Nebut et al. (2007).*

- **Acceptance Testing** - Tools that will be used by customers during the validation of applications *Neto et al. (2011a).*

- **Regression Testing** - Even though regression testing is not a test level, some tools were developed to work with it. For this reason, we considered regression testing as part of the classification.

The classification can be applied not only for single system testing tools but also for **SPL** testing tools as showed at Figure 3.7. The difference is that **SPL** divide Testing according to two activities *Neto et al. (2011a):* core asset (grouping Unit and integration testing) and product development (grouping system and acceptance testing). Table 3.5 details the classification of the tools according to the test level plus regression testing.

**Testing Tools Evolution**

In order to identify the evolution of the tools, we constructed a timeline for single system showed in Figure 3.8 and the SPL testing tools is showed in Figure 3.9. Every year, since 2005 at least one single system testing tool for unit testing level was published. There is a clear evolution of tools at the unit testing level because the complexity of the tools increases over the years. There are no visible evolution in other testing levels of single system and **SPL** testing tools including regression testing.
3.5. OUTCOMES

<table>
<thead>
<tr>
<th>Table 3.4</th>
<th>Where the Tools were developed and used</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Academy</strong></td>
<td><strong>Industry</strong></td>
</tr>
<tr>
<td><strong>SSTT</strong></td>
<td>(Beer et al., 1998), (Wu and Gray, 2005), (Shukla et al., 2005), (Gao et al., 2006), (Xu and Zhang, 2006), (Xie and Notkin, 2006), (Bertolino et al., 2007), (Martin et al., 2007), (Milicevic et al., 2007), (Nogueira et al., 2007), (Lemos et al., 2007), (Simons, 2007), (Mishra and Sonawane, 2010), (Mariani et al., 2011)</td>
</tr>
<tr>
<td><strong>SPLTT</strong></td>
<td>(Galli et al., 2005), (Kim et al., 2006), (Feng et al., 2007), (Nebut et al., 2007), (Weissleder et al., 2008), (Uzuncaova et al., 2008), (Oster et al., 2011)</td>
</tr>
</tbody>
</table>

3.5.1 Main Findings

The analysis of the results enables us to present the amount of studies that match each category addressed in this study. It makes possible to identify what have been emphasized in past research and thus to identify gaps and possibilities for future research.

Based on the analyzed tools, it was possible to identify that the tools are usually developed to support a specific testing level, under the justification that there are no tools supporting all functionalities of a testing process. None of the selected tools supports the overall SPL life cycle. For example, the prototype tool developed by Reuys et al. (2005) was created to assist the ScenTED technique, and does not support a specific testing process. The majority of the testing tools were implemented to solve a specific problem.

The amount of SPL test tools in academy (78%) is higher than the number of single system test tools (58%), in industry this percentage is inverted, 25% of the single system test tools and 11% of SPL test tools. The percentage is equivalent when the tools are applied in both industry and academy (17% for single system and 11% for SPL). In accordance with Figure 3.6, the amount of projects applied in industrial context lacks investments.
<table>
<thead>
<tr>
<th>Test Level</th>
<th>Unit Testing</th>
<th>Integration Testing</th>
<th>System Testing</th>
<th>Acceptance Testing</th>
<th>Regression Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSTT</td>
<td>(Liut et al., 2000), (Wu and Gray, 2005), (Gao et al., 2006), (Xie and Notkin, 2006), (Xu and Zhang, 2006), (Lemos et al., 2007), (Simons, 2007), (Xie et al., 2007), (Daniel and Boshernitsan, 2008), (Pinte et al., 2008)</td>
<td>(Liut et al., 2000), (Shukla et al., 2005), (Lemos et al., 2007), (Pinte et al., 2008)</td>
<td>(Beer et al., 1998), (Bertolino et al., 2007), (Martin et al., 2007), (Milicevic et al., 2007), (Ren and Chang, 2007), (Nogueira et al., 2007)</td>
<td>(Beer et al., 1998), (Barnett et al., 2003)</td>
<td>(Gao et al., 2006), (Daniel and Boshernitsan, 2008), (Xie et al., 2008), (Mariani et al., 2011)</td>
</tr>
<tr>
<td>SPLTT</td>
<td>(Stephenson et al., 2004), (Galli et al., 2005), (Feng et al., 2007), (Weissleder et al., 2008)</td>
<td></td>
<td>(Reuys et al., 2005), (Nebut et al., 2007)</td>
<td>(Kim et al., 2006)</td>
<td>(Uzuncaova et al., 2008)</td>
</tr>
</tbody>
</table>
3.5. OUTCOMES

During the research, we identified the possibility to use single system test tools such as JUnitMX (Wloka et al., 2009), and CodeGenie (Lemos et al., 2007) to support the SPL testing process, however, it will be necessary change the methodology of using to suit the tool for the SPL context. Another possibility should be implement specific functionalities from SPL such as variability management of the assets, but this will be possible only for open-source applications. Tools such as AsmL (Barnett et al., 2003), and SimC (Xu and Zhang, 2006) were developed to solve specific problems of a specific environment, due to this, the effort to adapt these tools will be impracticable. For this reason it will be more viable construct a new tool from the beginning.

Despite the number of single system test tools be twice higher than the number of SPL test tools, when the tools are organized by testing levels, the percentage of single system and SPL test tools are equivalent. Unit Testing, (SSTT 42%, SPLTT 45%), Integration Testing (SSTT 13%, SPLTT 0%), System Testing (SSTT 19%, SPLTT 33%), Acceptance testing (SSTT 13%, SPLTT 11%) and Regression Testing (SSTT 13%, SPLTT 11%).

As discussed below, the amount of single system test tool used in the industry added with tools used in both academy and industry correspond to 41% of the total. When we focus on SPL testing tools, this number decreases to 22%. Thus, we believe that SPL testing tools do not achieve maturity enough to be applied in industrial projects. Moreover, the effort required to implement a SPL testing process into organizations hampers the development of SPL testing tools.

Figure 3.8 illustrates that 2007 was the year with a higher number of publications describing single system testing tools (7 papers). 3 in the International Conference on Software Engineering (ICSE’07) and 2 in the International Conference on Automated Software Engineering (ASE’07). These two conferences together published the largest number of papers describing single system test tools, 5 papers each one.
3.5. OUTCOMES

Figure 3.8 Selected Single System Testing Tools time line

Figure 3.9 Selected SPL Testing Tools time line
There was a peak in the development in 2007 of tools both in single system and SPL testing tools. Coincidentally, it was the year with more tools focused on System Testing. From all the tools of system testing analyzed, only 3 were applied in the industry: 2 single system testing tools and 1 SPL testing tool.

Finally, we identified a tendency directed to Automatic Test Case Generation Tools and according to Bertolino et al. (2007) “The dream would be a powerful integrated test environment which by itself, as a piece of software is completed and deployed, can automatically take care of possibly instrumenting it and generating or recovering the needed scaffolding code (drivers, stubs, simulators), generating the most suitable test cases, executing them and finally issuing a test report”.

3.6 Threats to validity

The main threats to validity identified in the review are described next:

- **Tools selection.** A possible threat in such review is to exclude some relevant tool. In order to reduce this possibility, the selection of tools was based on the identification of the key research portals in computer science and wide range of web search engines, besides the main journals and conferences (Appendix A). The defined criteria intended to select tools supporting some functionalities of the SPL Test Process and not just supporting specifics requirements.

- **Data Extraction.** In order to ensure the validity, multiple sources of data were analyzed, i.e. papers, prototypes, technical reports, white papers and manuals, in addition to the tools executable.

- **Research Questions.** The questions defined could not have covered the whole Testing Tools, which implies that some one cannot find answers to the questions that concern them. To mitigate this feasible threat, we had several discussions with project members and experts in the area, allowing the calibration of the question. Thus, even if we had not selected the most adequate set of questions, we attempted to address the most asked and considered open issues in the field.

- **Publication Bias.** We cannot guarantee that all relevant primary studies were selected. It is possible that relevant studies were not chosen during the search process. We mitigate this threat by following references in the primary studies.
• **Research Strings.** The terms used in the research strings can have many synonyms, it is possible that some work were overlooked. To mitigate this threat, we discussed with project members and experts in the area.

### 3.7 Chapter Summary

This chapter presented a scoping study on software product lines testing tools, whose goal was to identify how the available SPLT tools are supporting the SPL Testing process. Through the review, it was possible to identify, which current functionalities are being supported by the tools, and which ones should be present in every SPLT tool based on their priorities.

The research was conducted using techniques from mapping study, a helpful approach for identifying the areas where there is sufficient information for a systematic review to be effective, as well as those areas where is necessary more research (Budgen *et al.*, 2008).

We noticed that publications describing industrial experiences are rare in literature. The existing case studies and experiments report only small projects, containing results obtained from tools that solve specific problems related with testing. Consequently, more experiments involving SPL testing tools are needed.

Next chapter presents the proposed tool based on the information obtained through the results analysis of this study.
A Software Product Lines System Test Case Tool

*Experience is what you get when you didn't get what you wanted.*

—RANDY PAUCH (The Last Lecture)

Nowadays, the decision whether to work with Software Product Lines (*SPL*) or not becomes a requirement for the strategic planning of companies working in specific domain. Thus, learning how to choose the ideal tools to test a *SPL* is beneficial for companies in this planning process. Even though the ascending number of available software engineering testing tools, *SPL* testing lacks specific tools able of supporting the *SPL* Testing Process and managing the variability of test assets. This chapter presents a software product lines testing tool to build system tests from use cases that addresses challenges for *SPL* Testing identified in the literature review.

### 4.1 Introduction

Software testing tools are available for testing in every stage of the software development life cycle, although it is not commercially viable for the tools vendors to produce a tool that suits every organization. However, the customization of such tools is desirable and inevitable (Fewster and Graham, 1999).

Software product lines testing tools are not an exception, and choose testing tools suitable for test applications and support process can be one of the most critical tasks of a project. In the *SPL* context, the amount of available tools decrease drastically, and the need of tools to reduce the effort during the *SPL* testing process is a gap that need to be
4.2. REQUIREMENTS

In order to manage the variability, avoid the explosion of test cases due to the great number of variation points combinations and reduce the effort to test a software product lines, we need testing tools that would allow for improvements in costs, time-to-market and quality (Almeida et al., 2007).

The availability of tools makes testing a more systematic activity and can minimize the cost and time consumed, as well as the errors caused by human intervention (Nakagawa et al., 2007). As discussed in the previous chapter, a wide range of tools, with both commercial and academic purposes can be found. However, these tools have almost always been implemented for specific purpose and they are isolated from others, presenting its own architecture and internal structures (Nakagawa et al., 2007). As a consequence, difficulties in integration, evolution, maintenance, and reuse of these tools are very common. Moreover, these tools often focus on automating specific testing techniques and criteria, without considering the whole testing process (Nakagawa et al., 2007).

Thus, this work presents a SPL testing tool for creating system test cases based on use cases that supports the SPL testing process. Furthermore, in order to figure out the needs of the research field, we split our investigation into two steps (presented in the previous chapter). Firstly, we analyzed testing tools that have been developed for testing single systems; and secondly, we focused on tools that have been developed specifically to product lines. The question that motivates our work is: How to handle variation points and their combination within a test case?

The remainder of this chapter is organized as follows. Section 2.3.3 is an introduction to software product lines testing tools. Section 4.2 lists the requirements proposed for the tool and Section 4.3 presents the tool architecture. Section 4.4 describes the technologies utilized during the implementation (Section 4.5). Section 4.6 discusses the proposal. Section 4.7 shows the operation of the tool; and finally, Section 4.8 summarizes and concludes this paper.

4.2 Requirements

Based on the scoping study performed, each one of the tools from system testing was analyzed in order to map its functionalities that were considered as requirements for the SPLMT-TE development. As we identified common functionalities between Single System and SPL tools, we considered every tool from both contexts. Table 4.1 details
which functionalities each tool offers support. The numbers refer to the functionalities described. This table facilitates the identification of the gaps in the selected tools, and it can help discovering which tool best satisfies specific needs.

Nakagawa et al. (2007) identified functional requirements through investigation of software testing processes, testing tools and testing ontology. It is noticed that these requirements refer to the core activities of testing. Since our work focuses on testing tools we considered only the requirements related with our context. Others requirements were detected during the analysis of the tools. A set of functional and non-functional requirements are presented next.

4.2.1 Functional Requirements

The identified functional requirements and their rationale are described as follows:

- **i) Import Test Cases** (Nakagawa et al., 2007) - Allows the user to import the test cases from templates and other documents such as Extensible Markup Language (XML) files into the tool.

- **ii) Include Test Cases Manually** (Nakagawa et al., 2007) - The tool must provide manually test cases creation, including information such as, name, pre condition, pos condition, steps, type (positive, negative or undefined) and others.

- **iii) Store Test Cases** (Nakagawa et al., 2007) - Stores the test cases into a repository. This functionality gathers the manipulated information and save it in the database.

- **iv) Generate Test Cases Automatically** (Nakagawa et al., 2007) - This functionality creates test cases automatically helping at the writing process, in order to reduce the effort during this activity.

- **v) Test Suite Reduction** (Nakagawa et al., 2007) - Assists at the reduction of the number of test cases from a test suite, eliminating the similar and duplicated tests, trying to maintain the coverage criteria.

- **vi) Enable Test Cases** (Nakagawa et al., 2007) - Activates test cases inside of the test suite, allowing the inclusion of new test cases at the suite.

- **vii) Disable Test Cases** (Nakagawa et al., 2007) - The opposite from the previous ones. Deactivate test cases from the test suite, allowing the exclusion of obsolete tests.
4.2. REQUIREMENTS

- **viii) Export Test Cases** *(Nakagawa et al., 2007)* - The opposite from the first functionality (Import Test Case). The tool must be capable of export test cases into other formats and templates such as XML files. Nowadays, the tool exports test cases into Portable Document Format (PDF) files. Future implementations pretend to support others format of files.

- **ix) Test Cases Report** *(Nakagawa et al., 2007)* - Generates reports about the selected test cases.

- **x) Remove Test Cases** *(Nakagawa et al., 2007)* - Allows the exclusion of duplicated and obsolete test cases.

- **xi) Test Cases Visualization** *(Nakagawa et al., 2007)* - Provides the visualization of test cases created manually and automatically. All information described at the test cases can be seen individually.

- **xii) Test Suite Management** *(Nakagawa et al., 2007)* - Groups test cases according to project demands. The tool also provides the creation of new test suites, deletion of unnecessary test suites and maintenance of obsolete test suites.

- **xiii) Calculate Test Coverage** *(Nakagawa et al., 2007)* - Defines the coverage criteria of the test cases, and identify if the amount of tests is sufficient to evaluate all the features.

- **xiv) Execute Test Requirements Using Test Cases** *(Nakagawa et al., 2007)* - Uses a combination of test cases to execute test requirements.

- **xv) Generate Test Input/Data** - Generates test input/data automatically, creating possible combinations of input/data. Common used to generate specific information according to the necessities of tools strongly related with source code and programming language such as Java and C.

- **xvi) Test Plan Management** - Groups the test suites and test cases according to project demands. Moreover, the tool allows the creation of new test plans (master, unit, integration, system and acceptance test plan individually), deletion of unnecessary test plans and maintenance of obsolete test plans. Figure 4.1 shows how the tool works grouping the test plans. Future work will implement the exchange of information between the master test plan and unit, integration, system and acceptance test plans.
4.2. REQUIREMENTS

Figure 4.1 Test Plan Management

• xvii) Variability Management - Manages the variation points and the variants of the software product lines assets.

The last line of the Table 4.1 describes the features of the proposal. The functionalities i, v, xiii, xiv, and xv, were not implemented because they were out of scope.

<table>
<thead>
<tr>
<th>Functionalities/Tools</th>
<th>i</th>
<th>ii</th>
<th>iii</th>
<th>iv</th>
<th>v</th>
<th>vi</th>
<th>vii</th>
<th>viii</th>
<th>ix</th>
<th>x</th>
<th>xi</th>
<th>xii</th>
<th>xiii</th>
<th>xiv</th>
<th>xv</th>
<th>xvi</th>
<th>xvii</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSTT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDATAG</td>
<td>•</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TAXI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WebSob</td>
<td>•</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Korat</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATTEST</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TaRGcT</td>
<td>•</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ScenTED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Objecteering</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MoSo-PoLiTe</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proposal</td>
<td>•</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.2.2 Non-Functional Requirements

The non-functional requirements are following described:

• xviii) Reusability - the proposed tool must allow the reusability of test assets such as test plans, test suites, and test cases. They can be reused to create new test assets.
4.2. REQUIREMENTS

• **xix) Extensibility** - In general, any software must take into account future growth. The architecture must presume the addition of new functionalities and the level of effort required to implement them without impacting to existing system functions. For this reason, the proposed tool must be well-structured, with low coupling modules to accommodate maintenance and extension demanded.

• **xx) Usability** - The graphic interface of the tool must be built-in with intuitive components to perform the functionalities.

• **xxi) Performance** - Performance is measured in terms of the response time. In centralized systems, the involved variable is the hardware processing power. In distributed scenarios, however, other variables must be considered, such as network traffic, geographical distance, and number of components.

• **xxii) Platform Independence** - Organizations used to have heterogeneous development plataforms and, for this reason, an integrated reuse environment must be used in order to maximize its user base and provide more effective results. The implementation of the environment functionalities must be based on technologies portable across plataforms.

4.2.3 Functionalities Priority

Although the mentioned requirements are considered important for the development of the proposed tool, some of them were not entirely or partially accomplished due to time constraints and scope of the proposal. In this sense, some requirements had more priority than others. Table 4.2 shows the expected requirements against its situation of development according to its priority, thus, in order to formalize the situation of each one, some priority criteria were adopted:

- **Essential.** It represents the indispensable and high-priority requirements that must be carried out. The lack of them turns the application useless;

- **Important.** It represents the medium-priority requirements that are strongly advisable for better usage of the tool; and

- **Aimed.** It represents the low-priority requirements that are required for particular situations or enhancements for current development.

For the situation of realization, three criteria were adopted:
Table 4.2 Summary of requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirements</th>
<th>Priority</th>
<th>Situation</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>Import Test Cases</td>
<td>Important</td>
<td>Not Achieved</td>
</tr>
<tr>
<td>ii</td>
<td>Included Test Cases Manually</td>
<td>Essential</td>
<td>Achieved</td>
</tr>
<tr>
<td>iii</td>
<td>Store Test Cases</td>
<td>Essential</td>
<td>Achieved</td>
</tr>
<tr>
<td>iv</td>
<td>Generate Test Cases Automatically</td>
<td>Essential</td>
<td>Important</td>
</tr>
<tr>
<td>v</td>
<td>Test Suite Reduction</td>
<td>Important</td>
<td>Not Achieved</td>
</tr>
<tr>
<td>vi</td>
<td>Enable Test Cases</td>
<td>Important</td>
<td>Achieved</td>
</tr>
<tr>
<td>vii</td>
<td>Disable Test Cases</td>
<td>Important</td>
<td>Achieved</td>
</tr>
<tr>
<td>viii</td>
<td>Export Test Cases</td>
<td>Important</td>
<td>Achieved</td>
</tr>
<tr>
<td>ix</td>
<td>Test Cases Report</td>
<td>Essential</td>
<td>Achieved</td>
</tr>
<tr>
<td>x</td>
<td>Remove Test Cases</td>
<td>Essential</td>
<td>Achieved</td>
</tr>
<tr>
<td>xi</td>
<td>Test Cases Visualization</td>
<td>Important</td>
<td>Achieved</td>
</tr>
<tr>
<td>xii</td>
<td>Test Suite Management</td>
<td>Essential</td>
<td>Achieved</td>
</tr>
<tr>
<td>xiii</td>
<td>Calculate Test Coverage</td>
<td>Important</td>
<td>Not Achieved</td>
</tr>
<tr>
<td>xiv</td>
<td>Execute Test Requirements Using Test Cases</td>
<td>Important</td>
<td>Not Achieved</td>
</tr>
<tr>
<td>xv</td>
<td>Generate Test Input/Data</td>
<td>Important</td>
<td>Not Achieved</td>
</tr>
<tr>
<td>xvi</td>
<td>Test Plan Management</td>
<td>Essential</td>
<td>Achieved</td>
</tr>
<tr>
<td>xvii</td>
<td>Variability Management</td>
<td>Essential</td>
<td>Achieved</td>
</tr>
<tr>
<td>xviii</td>
<td>Reusability</td>
<td>Important</td>
<td>Achieved</td>
</tr>
<tr>
<td>xix</td>
<td>Extensibility</td>
<td>Important</td>
<td>Partially Achieved</td>
</tr>
<tr>
<td>xx</td>
<td>Usability</td>
<td>Important</td>
<td>Achieved</td>
</tr>
<tr>
<td>xxi</td>
<td>Performance</td>
<td>Essential</td>
<td>Partially Achieved</td>
</tr>
<tr>
<td>xxii</td>
<td>Platform Independence</td>
<td>Important</td>
<td>Achieved</td>
</tr>
</tbody>
</table>

- **Achieved.** It means that the requirement was completely carried out and tested;
- **Partially Achieved.** It means that the requirement was implemented but there was not opportunity for testing or validation; and
- **Not Achieved.** It represents the requirements that were not definitely implemented.

Table 4.2 summarizes the requirements showing the priority and the situation of the proposed requirements. The requirements were not definitely implemented because they were out of scope. The requirements Extensibility and Performance were partially implemented because the proper tests were not fully executed.

### 4.3 Tool Architecture Overview

The *SPLMT* architecture was designed to be extensible providing the capacity to add new features by including new components, which allows the creation of the SPLMT-TE.

Figure 4.2 describes the architecture of the proposed tool combined with the *SPL* Metamodel (*Cavalcanti et al.*, 2011). Details of the layers are described below:
4.3. TOOL ARCHITECTURE OVERVIEW

- **Browser**: Users access the application through web browsers.

- **Template**: A Django template separates the application from the data. A template defines the presentation logic that regulates how the information should be displayed.

- **URL Dispatcher**: It defines which view is called for a given URL pattern. Basically, it is a mapping between URL patterns and the view functions that should be called for those URL patterns.

- **View**: It contains the business logic. View layer is itself a stack. Python view methods call a restricted template language, which can itself be extended with custom tags. Exceptions can be raised anywhere within a Python program, and if not handled they rise up as far as the environment.

- **Model**: It describes the data structure/database schema. It contains a description of the database table, represented by a Python class. This class is called a model. Using it, it is possible to create, retrieve, update and delete records in the database using simple Python code. SPLMT is composed by the Product Management (PM), Risk Management (RM), Scope (SC), Requirements (RQ), and Test Modules proposed in this work.

- **Persistence Layer**: All the data information is recorded at the repository. Django attempts to support as many features as possible on all database backends. However, not all database backends are alike, and we have to make design decisions on which features to support and which assumptions we can make safely.

### 4.3.1 Metamodel for managing variability

In this work, variability is introduced by the SPL Metamodel (Cavalcanti et al., 2011) that represents how variability in test assets could be managed in a SPL project. This model is the basis of designing, tracking and evolution of the test assets in the RiPLE-TE(Machado et al., 2011).

Cavalcanti et al. (2011) presents the RiPLE metamodel for assets variability management, divided into *SPL project management (PM)*, which includes the *risk management (RM)* sub-metamodel, and the *SPL core development*, which are the *scoping (SC)*, *requirements (RQ)* and *testing* sub-metamodels. The processes are sequentially arranged from the top to the bottom, in each layer depicted.
Every test case created can represent variability in a way that reuse is encouraged. Figure 4.3 shows an extract of a metamodel (Cavalcanti et al., 2011) developed in order to capture the relationship among the artifacts created in a SPL project. This figure basically depicts the Test model fragment, in which test cases are considered the main artifact to be handled.

Since each use case in a SPL project is supported by one or more test cases, the variability herein is handled as the same way than in use cases elaboration. This latter expands on the abstract use case definition, in which variability is represented in the use cases. A use case is herein composed by the entity AbstractFlow that comprises the subentity Flow, which actually represent the use case steps. Every step can be associated with a subflow that can represent a variation point. In both, fragments compose every (use or test) case, and are thus considered when deriving some product. This way, dependencies among test cases fragments and use case fragments make sense.

In addition the model also include other involved artifacts, such as planning and management assets.

### 4.4 Technologies

The architecture of the tool was structured based on a set of technologies, as follows:
4.4. TECHNOLOGIES

The tests model excerpt - from Cavalcanti et al. (2011)

- **Django**\(^1\) - is a high-level Python Web framework that encourages rapid development and pragmatic design that follows the Model-View-Controller (MVC) pattern. Through Django, the metamodel mapped entities and their relationship into Python\(^2\) classes, and then it is automatically created a relational database for these entities. Finally, Django generates a Web application where it is possible to test the mapping by inserting some test data, the documentation regarding features, requirements and so on.

- **Python Software Foundation (2011)** - It is the programming language used to implement the logic business of the tool. We preferred to work with python because there is plenty of material available. It is also free to use because of its OSI-approved open source license.

- **MySQL** - It is the database for the application. Such database was chosen because it features indexing of content, stability, good performance, good documentation, strong open source community, and it is a free project.

- **re - Regular expression operations**\(^3\) - It is a Python module that provides regular expression matching operations. re was used to construct the test cases steps through the manipulation of strings.

Currently, it is possible to use the tool to document all the assets regarding the metamodel, however the test cases derivation from use cases is not supported yet (Cavalcanti

---

1. [http://www.djangoproject.com](http://www.djangoproject.com)
2. [http://www.python.org](http://www.python.org)
3. [http://docs.python.org/library/re.html](http://docs.python.org/library/re.html)
4.5. IMPLEMENTATION

et al., 2011). This is the focus of our proposal and to bridge this gap we propose the architecture showed in Figure 4.2.

4.5 Implementation

The proposed tool was implemented using the Django Framework and the Python programming language. The motivation for choosing this combination was its flexibility with text processing, which is essential task for the creation of test cases, and rapid prototype construction. Moreover, Prechelt (2000) has presented that programming in dynamic typed languages is more productive than programming in static typed languages. Other language used to develop the tool was the Structured Query Language (SQL).

The proposed tool was developed during 4 months of development (20 hours/week). It contains about 3000 lines of code, with approximately 85% of Django/Python code, 10% of HTML code and 5% of SQL. All the modules presented in the tool can run on multiple operating systems and supports several browsers.

Figure 4.4 presents the low level architecture that translates the process explained at the previous sections and shows how the tool work precisely. After selecting the use cases at the proposed tool and selecting the option to create the test cases, the commands will be converted into ORM - Object Relational Mapping and the information will be directly extracted from the database.

![Figure 4.4 Proposed Low Level Architecture](image)

The server will manipulate the data using Python methods implemented using regular expressions in order to create new test cases. Subsequently, these new test cases will be
saved directly at the repository. Finally, the tool can visualize the test cases and the test assets can be organized into test suites and test plans, reducing the effort spent to manage the test assets of the product line.

4.6 Software Product Lines System Test Case Tool

The literature still lacks works describing tool support for testing software product lines (Neto et al., 2011a; Edwin, 2007; Tevanlinna et al., 2004). In this effect, we propose a tool focused on the elaboration of system test cases for the SPL projects, thus encouraging reusable artifacts to be produced. We will use the test model previously mentioned (and fully described in Cavalcanti et al. (2011)) to manage the test assets and dependencies.

Additionally, Figure 4.5 illustrates how the system test cases are built from use cases. According to Dias et al. (2008), a use case goal can be decomposed recursively into sub-goals, as a result is necessary to create test cases for each fragments ensuring the use cases coverage. Thus, the following steps were defined:

- **1a.** The use case document is composed by the use cases of the SPL project.
- **1b.** The tool allows users to select all the use cases or part of these. The selected use cases will be extracted from the document described in 1a.
- **2a.** Each use case can generate a system test case. When the use case is more complex the tool will generate a test cases group described below.
- **2b.** Test Cases groups will be composed by two or more system test cases.
- **2c.** The combination of system test cases generated by the tool will compose the Test Case Document. The tool also allows users to select specific test cases in order to generate customized test cases documents.

Next, the system test cases are formed by the following fields: an ID (unique identification number created automatically by the tool), the name of the test case, its summary and classification (positive and negative test cases), steps of the test case, pre and post conditions, variability type, use case references and screen path (describes the path of the system that should be tested).

When all the use cases are selected, the application focus at the construction of mandatory test cases. Thus the optional test cases can be built in accordance with the needs of specific products of the SPL product development phase. However, every
mandatory use case has to be validated, which consequently demands the creation of mandatory test cases. Besides, test cases priority are also classified as High, Medium and Low (Neiva et al., 2010).

In addition to the proposal, test cases are composed by extracting information from use cases. Figure 4.6 illustrates the data extraction from the use case document (1x) for the construction of test case document (2x). The main flow (1y) of the use cases leads to the building of positive test cases (2y - when determined action should succeed (Condron, 2004)) that analyze if the main functionalities are working properly. Secondary flows (1z) of the use cases are divided in Alternative and Exception flows. Alternative secondary flows result in positive test cases (2z) that validate instructions that should succeed (Condron, 2004). Finally, exception secondary flows result in negative test cases (2w - when determined action should not succeed Condron (2004)) that verifies errors messages and unsuccessful procedures.

In order to manage the variability, all the variation points are associated with requirements specified in the SPL requirement document detailed by Cavalcanti et al. (2011). As requirements include variability, test cases must also contain explicit variability information. Test cases that include variability are called variant test cases. Test cases without variability are called common test cases and can be applied to all applications (Reuys et al., 2005).

Moreover, each variation point is related with a use case, thus, for every single use
case is possible to create test cases. Hence, variation points leads to the creation of test cases preserving the variability within the tests cases.

The prioritization of the variation points is established in line with the variability type and priority of the requirements. Mandatory requirements with high priority have to be examined first. For this reason, we propose that tests cases of mandatory variation points with high priority should be created first.

Figure 4.7 explains how the test case document can support the test architect. Using the document, the test architect can delegate which tests should be created by developers (Unit and Integration tests because they need source code knowledge to support White-box techniques. White-box test cases consider the entire source code of the program while grey-box test cases only consider a portion of it (Segura et al., 2008)) and which tests must be done by test analysts (System and Acceptance tests because they need part or the entire system working, will be useful to support Black-box techniques. Black-box test cases are those in which no details about the implementation, in terms of code, are provided. These test cases are based exclusively on the inputs and outputs of the system under test (Segura et al., 2008)).

### 4.6.1 SPLMT-TE Steps and activities

Based on the requirements identified before, it was possible to define the functionalities supported by the tool. Its flowchart is depicted in Figure 4.8 detailing the execution flow.

Initially, the user can select use cases in order do create test cases automatically or create test cases manually. When the tests already exist is possible to organize them into
4.6. SOFTWARE PRODUCT LINES SYSTEM TEST CASE TOOL

Figure 4.7 Using of the Test Case Document

test suites. Finally, test plans can be created in order to organize the test suites. All these functionalities allow handling the variability of a software product line.

Figure 4.8 Tool’s Execution Flow

This process was defined to be executed by the test analyst and test architect, who are the principal users of the tool, however, test manager and test designer may also fulfill these steps as shows Figure 4.9. The RiPLE-TE process has 5 activities in Core Asset Development and Product Development (Machado, 2010) but the proposed tool only support 2 phases, Test Planning (assets: Test plans and Test cases) and Test Asset Design (assets: Test suites and Test cases).
4.7 SPLMT-TE in Action

In order to demonstrate how the tool works, this section shows the operation of it. The numbers in Figure 4.10 present the common sequence to use the SPLMT-TE. The test analyst or testers typically use the test case module according to the following flow: (1) first it is inserted the search keywords and the required filters, after the search, the users examine the results to identify the test cases (2), if necessary, the testers could create more test cases (3) or invoke others functionalities (4).

The functionalities are described as following:

1. **Search Bar** - This is the part of the tool, which the tester can search for specific test cases. It is also in that field where the search are specified by name of the test case, specific words and responsible for the tests.

2. **Test Case Visualization** - This field shows all the test cases of the repository, it also display the results of the searches. Each line represents a test case, with information about: unique identification number of the test, Name of the test case, brief summary about the test case, variability type and priority. Such information can be organized according to the columns.

3. **Adding Button** - The adding button can be selected when the creation of manually test cases is needed. During the creation, testers can “Save and add another
test cases”, “Save and continue editing” or “Save” just one test case as shows Figure 4.11. The test case will be saved only if the mandatory fields (Test case name, variability type, use case related, steps and priority) were filled properly. Warning messages will be displayed until the test case obeys the demands. Figure 4.12 presents one of these alert messages, requiring the filling of the Name field.

4. **Combo Box** - The combo box displays all the extra features of the tool such as the creation of reports, the deletion of one or more selected items, the test case creation, etc.

   Figure 4.13 presents the creation of test suites. When the test suite creation option is selected, the test architects can “Save and add another test suites”, “Save and continue editing” or “Save” just one suite. The test suite will be saved only if the mandatory fields (Name of the Test suite, priority and test cases related) were filled properly. Warning messages will be displayed until the test suite obeys the demands. The responsible for the test suite can be associated with the suite and a summary can explain and add information referred to the suite.
4.8 Chapter Summary

In this chapter, it was presented a web-based tool to create test cases from use case and manage test assets such as test cases, test suites and test plans. The tool was built based on the needs to facilitate these tasks reducing the effort and the costs of the Software Product Lines testing process. Thus, the requirements were presented as well as the architecture, the components of the architecture, features for create and manage the test assets, details of implementation (programming language and frameworks used), and some examples of use of it.

Next chapter presents an experimental study to validate the effectiveness of the tool.
Figure 4.11 Tool in Action: Test Case Creation
4.8. CHAPTER SUMMARY

Figure 4.12 Alert Message

Figure 4.13 Tool in Action: Test Suite Creation
Figure 4.14 Tool in Action: Test Plan Creation
5

Experimental Study

Have something to bring to the table, because that will make you more welcome.
—RANDY PAUCH (The Last Lecture)

Wohlin et al. (2000) state that the only real evaluation of a process is to have people using it, since the process is just a description until people use it. In fact, the path from subjectivity to objectivity is paved by testing or empirical comparison with reality (Juzgado et al., 2004).

In this sense, we performed a controlled experimental study (in vitro) in order to evaluate the proposed tool. The experimental study consisted of two phases. Firstly, a pilot study was performed in order to analyze the experiment feasibility, as well as to identify possible deficiencies in the design of the proposed experiment. Then, the actual experiment was performed.

This chapter describes and motivates the design of the experiments and discusses the threats to validity. The experimental study was conceived and structured based on the concepts of experimental software engineering and the evaluation of SE methods and tools provided by (Wohlin et al., 2000)

Following the experiment process defined in (Wohlin et al., 2000), we discuss its definition in Section 5.1, planning in Section 5.2, operation in Section 5.4, analysis and interpretation in Section 5.5; and finally, lessons learned at Section 5.7.
5.1 Definition

The definition phase consists of the moment in which the objectives and goals are defined. This study applied the Goal Question Metric (GQM) method (Basili, 1992) in order to collect and analyze metrics that are intended to define those goals and was structured as follows:

**Goal.** The objective of this experimental study is to analyze the SPLMT-TE tool for the purpose of evaluation with respect to its efficiency and effectiveness from the point of view of the potential users (testers) in the context of a SPL testing project in an academic environment.

**Questions.** To achieve this goal, the following questions were defined:

- **Q1.** Is the time required to design system test cases reduced when the tool is used? It includes the time required to search and analyze the use cases, which are the base to build test cases. Thus, it is important to know if time will be saved in such tasks.

- **Q2.** Is the amount of test cases increased when the tool is used?

- **Q3.** Is the time required to execute the designed test cases reduced when the tool is used? The execution includes the time required to execute and report errors.

- **Q4.** Is the amount of errors detected increased when the tool is used?

- **Q5.** Is the effectiveness of test cases improved when the tool is used?

**Metrics.** Once the questions were defined, they need to be mapped to a measurement value, in order to characterize and manipulate the attributes in a formal way (Basili et al., 1986). Hence, the metrics used in this analysis are next described:

- **M1. Designed Test Cases (DTC)(Juzgado et al., 2004):** It refers to the number of designed test cases. This metric refers to Q2.
5.2. PLANNING

- **M2. Efficiency in Test Case Design (ETCD)** (Juzgado et al., 2004): This metric is related to the number of designed test cases \((DTC)\) over the amount of time spent to create test cases \((TSC)\) (Juzgado et al., 2004). It is related to Q1.

\[
ETCD = \frac{DTC}{TSC} \times 100
\]  

5.2.1 Context Selection

In order to avoid the risks of conducting experiments in real projects, the experiment was conducted in an academic environment (not industrial software development). We

- **M3. Efficiency of Test Cases Execution (ETCE)** (Juzgado et al., 2004): Comprehends to the amount of executed test cases \((ETC)\) over the amount of time necessary to execute the designed test cases \((TSE)\) (Juzgado et al., 2004), including errors report. This metric is connected to Q3.

\[
ETCE = \frac{ETC}{TSE} \times 100
\]

- **M4. Number of Errors Found (NEF)** (Juzgado et al., 2004): It represents the number of errors found by the subjects during the execution. This metric is linked to Q4.

- **M5. Test Cases Effectiveness (TCE)**: Chernak (2001) proposed the metric to measure test case effectiveness. He defines this metric as the ratio of defects found by test cases to the total number of defects reported during a test cycle. Test cases that find more defects are considered more effective. We adapted this metric to our context, so TCE consists of the amount of errors found \((M4)\) reported to the total number of test cases created \((M1)\). This metric is associated with the Q5.

\[
TCE = \frac{NEF}{DTC} \times 100
\]

5.2 Planning

After the definition of the experiment, the planning prepares for how the experiment should be conducted (Wohlin et al., 2000). Sometimes referred to as experimental design or protocol, this phase describes the plan or protocol that is used to perform the experiment and then to analyze the results.

5.2.1 Context Selection

In order to avoid the risks of conducting experiments in real projects, the experiment was conducted in an academic environment (not industrial software development). We
also performed a pilot study to analyze the experiment feasibility, as well as to identify possible deficiencies in the design of the proposed experiment. The subjects of the pilot study were 14 undergraduate students from the Software Engineering course (MATA62)\(^1\) at Federal University of Bahia, Brazil.

MATA62 classes hosted the ‘experimental lab’, which included the selection of subjects, training, and execution of the experiment. This course was designed to explain fundamental concepts and principles of Software Engineering.

The pilot was executed from June to July in 2011 and addressed a problem that was analyzed based on two specific SPL projects scenarios. All activities of the experiment were performed at the pilot study.

After performing the pilot study, the actual experiment was implemented. The subjects of the experiment were composed by graduate students (7 M. Sc. Students and 5 Ph. D. Students) from Federal University of Bahia and Federal University of Pernambuco, Brazil.

The experiment was performed in July 2011, and used the same projects as pilot study.

### 5.2.2 Hypothesis Formulation

In an experiment, it is necessary to formally and clearly state what is intended to evaluate (Juzgado and Moreno, 2001). In this experimental study, we focused on six hypotheses. They are formally stated with the measures needed to evaluate them.

**Null Hypothesis \((H_{0n})\)**

There is no benefit of using the proposed tool (described below as Tool) to support the design and execution of system test cases, if compared to the manual process (described below as manual), in terms of effectiveness. The Null hypotheses are:

\[
H_{01} : \mu_{DTC_{manual}} \geq \mu_{DTC_{Tool}}
\]

\[
H_{02} : \mu_{ETCD_{manual}} \geq \mu_{ETCD_{Tool}}
\]

\[
H_{03} : \mu_{ETCE_{manual}} \geq \mu_{ETCE_{Tool}}
\]

\(^1\)http://disciplinas.dcc.ufba.br/MATA62
5.2. PLANNING

\[ H_{04} : \mu_{\text{NEF\_manual}} \geq \mu_{\text{NEF\_Tool}} \]
\[ H_{05} : \mu_{\text{TCE\_manual}} \geq \mu_{\text{TCE\_Tool}} \]

**Alternative Hypothesis \((H_{1n})\)**

Conversely to what was defined as Null Hypotheses, the alternative hypothesis determines that the proposed tool produces benefits that justify its use. We herein define the set of alternative hypotheses, as follows:

\[ H_{11} : \mu_{\text{DTC\_manual}} < \mu_{\text{DTC\_Tool}} \]
\[ H_{12} : \mu_{\text{ETCD\_manual}} < \mu_{\text{ETCD\_Tool}} \]
\[ H_{13} : \mu_{\text{ETCE\_manual}} < \mu_{\text{ETCE\_Tool}} \]
\[ H_{14} : \mu_{\text{NEF\_manual}} < \mu_{\text{NEF\_without\_Tool}} \]
\[ H_{15} : \mu_{\text{TCE\_manual}} < \mu_{\text{TCE\_Tool}} \]

5.2.3 Selection of subjects

The subjects of this experiment were selected by *convenience sampling* (Wohlin et al., 2000). The nearest and most convenient persons are selected as subjects. *Fourteen* subjects were selected for the pilot study and, *twelve* for the experiment.

The students acted as test analysts and testers, following the roles defined in the RiPLE-TE process (Machado et al., 2011). The subjects were responsible for the tasks of designing assets and executing test cases. One group was responsible for performing test case creation in an ad-hoc fashion, whereas the other performed the test case creation using the proposed tool. The division of groups was based on the subjects’ expertise, gathered from a background form, as will be explained in details in instrumentation section.

The motivation for the participants to attend the experimental study is based upon the assumption that they will have an opportunity to use the proposed tool in a project so that
they could gain experience to execute future studies.

5.2.4 Experiment design

Once the problem has been stated, and we have chosen the experiment variables, we are able to design the experiment. An experiment consists of a series of tests of the treatments, or even a set of tests.

**Design Type**

The design type used in the experiment is *one factor with two treatments*, in which we would like to compare the two treatments against each other (Wohlin *et al.*, 2000). In this sense, the factor is the use of the proposed tool and the treatments are following described:

1. **Creating test cases with the tool support.** In the first treatment, the group of subjects was trained in the tool, and they had access to the requirements document where the use cases were described during the performing session.

2. **Create test cases manually.** In the second treatment, the group of subjects did not receive any training regarding the tool, they also had access to the requirements document during the performing session.

As the subjects that participated from the pilot did not have experience in Software Product Lines, they were introduced to the principles of SPL. All the subjects were trained in Software Testing too. It included how to write good system test cases, how to execute test cases and how to report errors. Figure 5.1 shows how the experiment execution was performed according to the treatments defined.

5.2.5 Instrumentation

The background information about the subject’s experience was gathered from a questionnaire handled out at the first experiment session. They had to fill in a questionnaire named *background questionnaire*, in which they had to provide information about their experience in testing software, participation in projects, experience with software product lines, software development, and the tools which the experiment requires (see Appendix B.1).

Subjects had access to the artifacts necessary to perform the experiment, in this case, the requirements document with the use cases. In addition, the subjects had access to
5.2. PLANNING

Figure 5.1 Experiment Design. The subjects received a requirements document with the use cases and performed the creation of test cases with and without the tool support. After that, the subjects received another list to report the errors found during the test execution.

the tutorials used in the training as well as a guideline on how to write test cases, how to report the errors, and how to create test cases using the proposed tool.

Before being informed about the goals and general information on the experiment, all subjects signed a consent form presented in Appendix B.2, as a means of agreeing to join the study. Their signature means that they understand the information given about the study. They were informed that they could withdraw from the experiment at any time, without penalty. During the pilot study, the consent form was applied after the explanation of the experiment; we changed the order of this activities because it helped at the concentration of the subjects.

An important way of collecting information is the test case creation form, where the subjects were supposed to write the test cases manually. Appendix B.3 depicts a copy of the test case creation form. It includes details about every test case that must be related with the use case. The subjects also used a form to report the execution of the test cases. Appendix B.4 shows a copy of the test case execution form, including information about the results of the test cases and time spent to execute a test case.

Another important form is the error collection form, where the subjects are supposed to report every defect found. Appendix B.5 presents the errors reporting form. It includes details about every error found by the subjects. We used such a form instead of using a bug tracking system due to timing constraints. Although the fields were extracted from such systems, we believe that a spreadsheet would be suitable for this experimental
In order to analyze how much time was spent at the creation of test cases with the proposed tool, the subjects used the automatic test case creation form described in Appendix B.6.

In addition, four types of feedback questionnaires were designed: one intended to the group which conducted the experiment using the proposed tool (Appendix B.7, and another that created the tests manually (Appendix B.8). The feedback questionnaires were designed to provide useful information on the use of the tool, through gathering information about the participants satisfaction with the tool.

The third one refers to questionnaire presented in Appendix B.9, designed with the intention to gather feedback on the subjects which performed the experiment using the proposed tool, regarding their opinion about the possibility of finding more defects using the tool. The fourth one refers to questionnaire from Appendix B.10, which asked the same question and added the questions about the use of the tool.

Subjects were monitored during the filling out of the questionnaires, since these provide useful data for future analysis and improvements, regarding both tool and the experiment design.

### 5.2.6 Training

It was explained to the subjects the context of the experiment and how it had to be conducted. Both groups worked 4 days each, attending to training sessions and the test execution session.

Related to the training sessions, initially, the subjects became aware of the experiment purpose and associated tasks. In order to balance the knowledge, a brief training on how to create and execute test cases were performed, which included practical exercises.

As mentioned before, the group 1 performed the activities using the proposed tool at the first phase with previous training based on the tool. The group worked without tool support in the second phase. On the other hand, group 2 started in the first phase writing the test cases manually, at that moment the training in the tool was not necessary. In the second phase, the group received the training before using the tool.

Feedback A represents the moment in which subjects from Group 1 reported their feedbacks on the experiment through filling in a questionnaire. As previously mentioned, the subjects of this group created test cases using the proposed tool. The questionnaire presented in Appendix B.7 focus on general information about learning in training and about the tool usage. Feedback B asked the subjects from Group 2 about the learning
During the training. The group 2, which performed tests without the tool, used the feedback questionnaire presented in Appendix B.8.

Feedbacks C and D represent the moment in which additional feedback questionnaires were applied to the subjects from Group 1 and Group 2 respectively, right after they finished the second phase of the experiment. They gave feedback on aspects that would be improved upon, by comparing the experience with possible opportunities to improve the results they reported by using the tool.

5.3 The Experimental Study Project

Two SPL projects were chosen for the experiment. The first consists of the cell phone or mobile devices games domain called Arcade Game Maker (AGM) Pedagogical Product Line produced by the Software Engineering Institute (SEI) in order to support learning about and experimenting with software product lines.

The product line consists of nine products: three games on three platforms (Win32, Linux on wireless devices, and platform-independent version). The games are graphical and depend heavily on the graphical nature of the environment. The source code of the AGM was developed through the Java Platform, Micro Edition (Java ME), the environment emulation was handled by the Eclipse framework.

In its initial planning, three products were derived from the core asset base:

- **Brickles.** A player gets three pucks to break all of the bricks in the brick pile, if the player breaks all the bricks, he wins, if the player runs out of pucks, he loses. Figure 5.2 presents the game in execution.

- **Pong.** The version of Pong is for one player. It has a few simple rules, the objective is to keep the puck in play as long as possible. Figure 5.2 shows the game in execution.

- **Bowling.** The player attempts to knock down as many pins as possible in 10 frames. The player can throw two balls in each frame, knocking down all pins in a frame with the first ball is termed a strike, knocking all pins in a frame with both balls is termed a spare. Figure 5.2 introduces the game during execution.

---


3 [http://www.sei.cmu.edu](http://www.sei.cmu.edu)
Each game has a different set of rules. Some of the games have a simple won/lost scoring system, while others award points for certain actions. Each game has a different set of moving and stationary items.

The amount of features were described in the requirement document of the Arcade Game Maker, as can been see in Figure 5.3 which depicts the feature model.

The second SPL project used in the experiment consisted of a word processor tool, named NotepadSPL. The product line was extracted from the FeatureVisu⁴ project that provides data and supplementary material of an empirical study on forty software product lines of different sizes and domains. The products of the NotepadSPL family implement specific functionalities that satisfy necessities of the users.

The product line consists of seven products. The word processors are graphical and were developed based on word processors of the market. The source code of the NotepadSPL was developed through the Java Platform, Standard Edition (Java SE), the environment emulation was handled by the Eclipse framework.

In its initial planning, two products were derived from the core asset base:

⁴http://fosd.de/FeatureVisu
5.4 Operation

In the operational phase of an experiment, the treatments are applied to the subjects. It means that this part of the experiment is the part where the experimenter actually meets the subjects (Wohlin et al., 2000). The operation consists of three steps: Preparation, where the subjects are selected and informed by the experiment and the material such as material and tools are prepared. Execution, where the subjects realize their tasks in accordance with the treatments and data collected. Finally, Data Validation, where we validate the collected data (Wohlin et al., 2000). Following, each step of the pilot study and the experiment will be detailed.

Figure 5.3 Feature Model from Arcade Game

- **Simple.** This product is a common text-only editor. This version has restrictions such as, that is not possible to find next words, print, save as, etc.

- **Advanced.** This product is a complete word processor without restrictions, the advanced notepad aggregates all functionalities of a advanced word processor and support to different type of files. Figure 5.4 presents the NotepadSPL in execution.

The amount of features were described in the requirement document of the NotepadSPL, as can been see in Figure 5.5 which depicts the feature model.
5.4. OPERATION

Figure 5.4 NotepadSPL in execution

5.4.1 Pilot Study

Preparation

As described earlier, before the experiment, we performed a pilot study to acquire experience. The subjects of the pilot study were students that had previously enrolled in courses on object-oriented programming based on the Java programming language, but without experience on software testing, verification and validation concerns.

The subjects had to agree with the research goals. They obeyed the experiment rules by signing up the consent form. The results of the personal subjects performance in the experiment were kept confidential. Furthermore, the subjects were informed that we would like to investigate the effectiveness of the proposed tool. However, they were not conscious of what aspects we intended to study, thus including the hypotheses stated.

The requirement documents, all the forms and reports described in Appendix B were printed and provided before the execution of the experiment.

Execution

The activities involved in the pilot study were executed by the subjects. The characterization was made when the subjects filled in the background form, allowing us to organize the subjects into two groups. Both groups performed the pilot using the tool in one phase and without using the tool in the other phase.
Thus, when the group 1 worked with tool support, group 2 worked manually. When the group 2 worked with tool support, group 1 worked manually. Moreover, the balancing strategy was then applied in order to have two groups as balanced as possible in terms of expertise. Subjects’ expertise, age, gender, current undergraduate semester and grade point average was considered for balancing purposes.

The first training was composed by an explanation of SPL concepts, in which subjects from both groups were introduced to this new software engineering and testing approach. Moreover, explanations about the objective of the experiment, their rules in this study were addressed.

The second training was focused on software testing concepts, in which subjects from both groups learned how to write good test cases, how to execute test cases, and how to report errors. They also participated from practice lessons, where they learned how to use the instruments from the experiment.

The tasks of the subjects were described at the RiPLE-TE process. They analyzed the application, created test assets (Appendix B.3), executed test cases (Appendix B.4), and reported the errors (Appendix B.5). They had to perform system tests only in one product of the product line, although they should access the documentation from all the product line, in order to known how the products worked. In addition, subjects were encouraged to reuse test cases they create in terms of reduce the effort, regarding to the variability of the products. The test cases they produced with the proposed tool containing the test cases were recorded in the tool repository, the test cases write manually were archived.

**Figure 5.5** Feature Model from NotepadSPL
with the other forms of the experiment for further analysis.

At the end of the third and fourth day of the pilot study, each of the 14 participants completed a feedback questionnaire, feedback A (Appendix B.7) or B (Appendix B.8) in the third day and feedback C (Appendix B.9) or D (Appendix B.10) in the last day.

Data Validation

To verify if the data are reasonable and collected properly, all data were checked. This deals with aspects such as the participants’ dedication and understanding required to filling in the forms. If the subject did not work seriously, some data therefore should be removed before the analysis (Wohlin et al., 2000).

Data were collected from 14 subjects. However, data from 2 subjects (IDs 12 and 14) were removed because they did not participate in all activities or they did not complete the forms as requested since the beginning of the study. Appendix C detailed the analysis of the pilot study information.

Lessons Learned

After concluding the pilot study, we gathered information that can be used as a guide to the experiment. Some important aspects should be considered, specially the ones seen as limitations in the pilot study. The general impressions gathered from the pilot study are described next. In general, subjects who reported in the characterization activity (see Table C.3) a lower level of experience in the software testing, were the same who reported difficulty for performing the software testing activities.

Motivation. As the projects documents were not short and most of the subjects complained about boredom, it was difficult to keep the subject’s motivation during all the execution. Thus, this aspect should be analyzed in order to mitigate it. A possible solution can be defining only one activity by subject. The subjects with experience will be responsible for the creation; the inexperienced will only execute and report errors.

Questionnaires. After concluding the pilot study, we noticed that some useful information was not collected, such as the subject’s impression of using the tool, if they had experience in industry or only in academia, and so on.

Design Type. The design type used at the pilot study was defined with the subjects analyzing the same domain with tool and without tool. We believed that the design type interfered at the final result of the experiment.

Measurement. During the pilot study, the subjects executed the activities as a testing group. For this reason, the data was compromised for the analysis, what could lead
to uncertain and not worth results. Hence, we did not perform any kind of hypotheses testing. Instead of verify the data of each subject, for example, the number of test cases created per subject, we had to check the number of test cases written by the group. The same happened with the test cases execution.

Beside this problem, the result was important to check the reliability and validity of the results allowing the definition and calibration of the metrics for the experiment.

Another problem happened during the test cases creation using the tool. The process of creation with the tool was too fast that the subjects from the pilot study forgot to register how many time was spent in the activity.

Finally, at the day 3 of the pilot study execution, the subjects tried to finish the activities as fast as possible because they were worried about an exam of another discipline.

### 5.4.2 Experiment

#### Preparation

The subjects of the experiment were M. Sc. and Ph. D. students in software engineering, some of them with experience in software testing. They also represented a non-random subset from the universe of subjects.

As in the pilot study, the subjects had to follow the same steps and rules of the experiment. They signed up the consent form and were informed with the same information used at the pilot.

The experiment was held at the Software Engineering Laboratory (LES) from the Federal University of Bahia. The subjects used the Eclipse IDE, the software of the Arcade Game Maker, the software of the NotepadSPL, and the proposed tool. The requirements documents, all the forms and reports were provided in digital copies.

#### Execution

The experiment was conducted in the end of July 2011. Table 5.1 shows all activities executed by the subjects. It describes the training and execution sessions. After the execution, data was collected.

Two groups were organized after the subjects filled out the background form (see Table 5.2). The balancing strategy was applied to harmonize the groups in terms of expertise.

The execution was performed in two phases. One group used the tool while the other worked without tool support. The pilot study execution was limited to 2 hours of
Table 5.1 Experiment Training and Execution Agenda

<table>
<thead>
<tr>
<th>Activity Length</th>
<th>Groups</th>
<th>Activity Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Day 1 - July 25th, 2011</td>
<td>Characterization - B.1 / Consent Term - B.2</td>
<td>0:30</td>
</tr>
<tr>
<td></td>
<td>Explanation on the Experiment</td>
<td>0:30</td>
</tr>
<tr>
<td>Day 2 - July 26th, 2011</td>
<td>Training on Testing</td>
<td>1:30</td>
</tr>
<tr>
<td></td>
<td>Exercises</td>
<td>0:30</td>
</tr>
<tr>
<td>Day 3 - July 27th, 2011</td>
<td>Creation / Execution</td>
<td>N/D*</td>
</tr>
<tr>
<td></td>
<td>Adhoc Creation / Execution</td>
<td>N/D*</td>
</tr>
<tr>
<td></td>
<td>Feedback A - B.7</td>
<td>0:15</td>
</tr>
<tr>
<td></td>
<td>Feedback B - B.8</td>
<td>0:30</td>
</tr>
<tr>
<td>Day 4 - July 28th, 2011</td>
<td>Adhoc Creation / Execution</td>
<td>N/D*</td>
</tr>
<tr>
<td></td>
<td>Creation / Execution</td>
<td>N/D*</td>
</tr>
<tr>
<td></td>
<td>Feedback C - B.9</td>
<td>0:30</td>
</tr>
<tr>
<td></td>
<td>Feedback D - B.10</td>
<td>0:30</td>
</tr>
</tbody>
</table>

(*) N/D stands for Not Defined

test cases creation and execution because of the discipline schedule. However, at the experiment we did not restrict the time of creation and execution.

Table 5.3 presents a concentrated view of subjects’ profile, group 1 is detached with gray lines and group 2 with white lines. In this table, the experience of the subjects with software development, Java, testing, SPL and with testing tools is organized in terms of months dealing with such aspects.

Furthermore, their participation in industrial development and test projects is showed as well. The experience of each subject regarding english reading was classified as: advanced, intermediate and basic. This information could impact on the results because all the artifacts that compose the set of instruments are written in english, except for the requirement document from the NotepadSPL that was written in portuguese.

Another information that can be visualized in this table is related to the subject’s knowledge on software testing tools. The form (Appendix B.1) included a field, which they could write down tools that they had some expertise.

After the characterization, the subjects were trained at the concepts of software testing. Both groups learned how to create a good test case, how to execute test cases, and how to report errors. The training in SPL concepts was not necessary since the subjects had experience in the area.

Before using the proposed tool, the subjects learned how to use the tool. At the end, they had to fill out the feedback questionnaire, this process was also detailed at the pilot study execution.
Data Validation

Data was collected from 12 subjects. However, data from 2 subjects (IDs 10 and 11) were removed because they did not participate in all activities. Table 5.4 shows the final grouping of subjects. Once again, we believe that the absence of these subjects did not invalidated the results, in terms of statistical analysis and interpretation.

5.5 Analysis and Interpretation

After collecting experimental data in the operation phase, we can draw conclusions based on this data. Quantitative interpretation may be carried out in three steps. The data is characterized using descriptive statistics in the first step. In the second step, abnormal or false data points are excluded, reducing the data set of valid data points. Finally, in step three, the data is analyzed by hypothesis testing (Wohlin et al., 2000).

We analyzed all the artifacts produced by the subjects, including the error report forms and the feedback questionnaires. Table 5.5 shows all information gathered from the experiment. ID means the subject identification. The information regarding time constraints are expressed in minutes.

5.5.1 Descriptive Statistic

Designed Test Cases

In terms of test cases created, Table 5.6 shows the distribution of designed test cases (considering data from Table 5.5). Min. stands for minimum, 1st Qu. and 3rd Qu. means 1st and 3rd Quartile respectively, Max. stands for maximum, and finally Std.Dev. means Standard Deviation.

Boxplot shown in Figure 5.6 contains data from the distribution of test cases created by tool usage. In groups (1 and 2) using the tool, the mean was 26.10 with Std.Dev. of 6.06; and 15.70 of mean with Std.Dev. of 3.83 in groups (1 and 2) without using the tool. Median of groups using the tool is higher than groups that did not use tool support.

According to Figure 5.6 the amount of designed test cases created with the tool is
### Table 5.3  Subjects’ Profile from Group 1 (gray lines) and Group 2 (white lines) - Experiment

<table>
<thead>
<tr>
<th>Group</th>
<th>Subject ID</th>
<th>English Reading</th>
<th>Part. in Industry</th>
<th>Experience in Programming*</th>
<th>Java*</th>
<th>Testing*</th>
<th>SPL*</th>
<th>Testing Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Dev. Project</td>
<td>Test. Project</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>advanced</td>
<td>No</td>
<td>No</td>
<td>48</td>
<td>12</td>
<td>6</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>intermediate</td>
<td>No</td>
<td>No</td>
<td>204</td>
<td>100</td>
<td>40</td>
<td>24</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>advanced</td>
<td>Yes</td>
<td>Yes</td>
<td>78</td>
<td>24</td>
<td>42</td>
<td>24</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>advanced</td>
<td>No</td>
<td>No</td>
<td>30</td>
<td>12</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>advanced</td>
<td>Yes</td>
<td>No</td>
<td>12</td>
<td>6</td>
<td>4</td>
<td>24</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>advanced</td>
<td>No</td>
<td>Yes</td>
<td>30</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>advanced</td>
<td>Yes</td>
<td>No</td>
<td>48</td>
<td>28</td>
<td>6</td>
<td>36</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>intermediate</td>
<td>Yes</td>
<td>No</td>
<td>31</td>
<td>15</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>intermediate</td>
<td>Yes</td>
<td>No</td>
<td>180</td>
<td>60</td>
<td>24</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>advanced</td>
<td>Yes</td>
<td>Yes</td>
<td>24</td>
<td>12</td>
<td>24</td>
<td>36</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>intermediate</td>
<td>Yes</td>
<td>Yes</td>
<td>36</td>
<td>10</td>
<td>36</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>intermediate</td>
<td>No</td>
<td>No</td>
<td>24</td>
<td>6</td>
<td>6</td>
<td>18</td>
</tr>
</tbody>
</table>

(* ) the experience is expressed in months
Table 5.4 Final grouping of Subjects from the Experiment

<table>
<thead>
<tr>
<th>Group</th>
<th>Subjects ID</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 4 5 6 12</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>2 3 7 8 9</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 5.5 Experiment information

<table>
<thead>
<tr>
<th>Tool</th>
<th>Group</th>
<th>ID</th>
<th>DTC</th>
<th>ETCD*</th>
<th>ETC</th>
<th>ETCE*</th>
<th>NEF</th>
</tr>
</thead>
<tbody>
<tr>
<td>With</td>
<td>1</td>
<td>1</td>
<td>18</td>
<td>43</td>
<td>17</td>
<td>29</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>22</td>
<td>31</td>
<td>22</td>
<td>32</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>21</td>
<td>12</td>
<td>20</td>
<td>44</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>20</td>
<td>25</td>
<td>19</td>
<td>30</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>22</td>
<td>19</td>
<td>20</td>
<td>33</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>33</td>
<td>29</td>
<td>33</td>
<td>11</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>34</td>
<td>20</td>
<td>34</td>
<td>38</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>29</td>
<td>42</td>
<td>29</td>
<td>39</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>32</td>
<td>19</td>
<td>32</td>
<td>30</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>30</td>
<td>17</td>
<td>30</td>
<td>42</td>
<td>4</td>
</tr>
<tr>
<td>Without</td>
<td>1</td>
<td>1</td>
<td>19</td>
<td>59</td>
<td>19</td>
<td>26</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>18</td>
<td>65</td>
<td>18</td>
<td>13</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>17</td>
<td>106</td>
<td>17</td>
<td>25</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>18</td>
<td>60</td>
<td>17</td>
<td>40</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>18</td>
<td>63</td>
<td>18</td>
<td>23</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>12</td>
<td>50</td>
<td>12</td>
<td>35</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>10</td>
<td>83</td>
<td>10</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>21</td>
<td>198</td>
<td>14</td>
<td>58</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>13</td>
<td>74</td>
<td>13</td>
<td>30</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>11</td>
<td>62</td>
<td>11</td>
<td>30</td>
<td>0</td>
</tr>
</tbody>
</table>

(*) - amount of time expressed in minutes

Higher than without tool support. The proposed tool enabled the creation of more test cases, 261 with tool support and 157 without it.

Table 5.6 Designed Test Cases

<table>
<thead>
<tr>
<th>Tool</th>
<th>Group</th>
<th>Min.</th>
<th>1st Qu.</th>
<th>Median</th>
<th>Mean</th>
<th>3rd Qu.</th>
<th>Max.</th>
<th>Std.Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>With</td>
<td>1</td>
<td>18.00</td>
<td>20.00</td>
<td>21.00</td>
<td>20.60</td>
<td>22.00</td>
<td>22.00</td>
<td>1.67</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>29.00</td>
<td>30.00</td>
<td>32.00</td>
<td>31.60</td>
<td>33.00</td>
<td>34.00</td>
<td>2.07</td>
</tr>
<tr>
<td></td>
<td>1 and 2</td>
<td>18.00</td>
<td>21.25</td>
<td>25.50</td>
<td>26.10</td>
<td>31.50</td>
<td>34.00</td>
<td>6.06</td>
</tr>
<tr>
<td>Without</td>
<td>1</td>
<td>17.00</td>
<td>18.00</td>
<td>18.00</td>
<td>18.00</td>
<td>18.00</td>
<td>19.00</td>
<td>0.71</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>10.00</td>
<td>11.00</td>
<td>12.00</td>
<td>13.40</td>
<td>13.00</td>
<td>21.00</td>
<td>4.40</td>
</tr>
<tr>
<td></td>
<td>1 and 2</td>
<td>10.00</td>
<td>12.25</td>
<td>17.50</td>
<td>15.70</td>
<td>18.00</td>
<td>21.00</td>
<td>3.83</td>
</tr>
</tbody>
</table>

Efficiency in Test Case Design

Table 5.7 presents the time spent in test cases design of each group, considering (1) groups using and without using the tool and (2) the value for the whole groups using and without using the tool.
5.5. ANALYSIS AND INTERPRETATION

Figure 5.6 Boxplot with Designed Test Cases per Group

Figure 5.7 presents the distribution of time to design the test cases per subject. In groups (1 and 2) using the tool, the *mean value* was 25.70 with *Std.Dev.* of 10.48, while in groups (1 and 2) without using tool support, the *mean value* was 82.00, with *Std.Dev.* of 43.69. The *median value* of groups using the tool is greater than groups without using tool support.

The efficiency in test case design was higher with groups that used the tool, which allowed the creation of more test cases faster than without tool support as presented in Figure 5.7. The time needed to create system test cases decreases using the tool, reducing from 820 minutes to 257 minutes, saving 563 minutes.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Group</th>
<th>Min.</th>
<th>1st Qu.</th>
<th>Median</th>
<th>Mean</th>
<th>3rd Qu.</th>
<th>Max.</th>
<th>Std.Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>With</td>
<td>1 and 2</td>
<td>0.42</td>
<td>0.73</td>
<td>1.15</td>
<td>1.18</td>
<td>1.70</td>
<td>1.76</td>
<td>0.51</td>
</tr>
<tr>
<td>Without</td>
<td>1 and 2</td>
<td>0.11</td>
<td>0.16</td>
<td>0.21</td>
<td>0.22</td>
<td>0.28</td>
<td>0.32</td>
<td>0.07</td>
</tr>
</tbody>
</table>

**Efficiency in Test Cases Execution**

Table 5.8 summarizes the data from the distribution of time to execute the test cases per subject. The *mean* in groups (1 and 2) using the tool was 0.94 with *Std.Dev.* of 0.74. In groups without the tool, the *mean* was 0.59 with *Std.Dev.* of 0.33. *Median* of groups using the tool is higher than in groups without using the tool (see Figure 5.8).
5.5. ANALYSIS AND INTERPRETATION

There was no significant difference during the execution of the created test tools. Both groups, with and without tool support, executed a similar number of test cases during the same period of time. The effort of test case execution were almost the same too, 328 minutes with tool support and 300 minutes without, we need to consider that the number of test cases executed were higher with the tool (256 test cases) than without tool support (149 test cases).

![Boxplot with Efficiency in Test Case Design](image)

**Figure 5.7** Boxplot with Efficiency in Test Case Design

By analyzing the errors found during the test case execution, the following dataset (Table 5.9) was structured. Figure 5.9 shows the Boxplot with distribution of the number of errors found per subject.

Mean of the groups (1 and 2) using the tools was 5.20 with Std.Dev. of 2.97. The mean of the groups (1 and 2) without using the tool was 2.90 with Std.Dev. of 2.23. Median of the groups that used the tool is higher than in the groups that did not use it.

According to Figure 5.9 the amount of errors found with the test cases created by the tool was slightly higher than with the test cases created manually. Moreover, the subjects were capable to find more errors during the use of the tool, 38 errors found with tool support and 26 without.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Group</th>
<th>Min.</th>
<th>1st Qu.</th>
<th>Median</th>
<th>Mean</th>
<th>3rd Qu.</th>
<th>Max.</th>
<th>Std.Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>With</td>
<td>1 and 2</td>
<td>0.46</td>
<td>0.61</td>
<td>0.70</td>
<td>0.94</td>
<td>0.86</td>
<td>3.00</td>
<td>0.74</td>
</tr>
<tr>
<td>Without</td>
<td>1 and 2</td>
<td>0.24</td>
<td>0.38</td>
<td>0.47</td>
<td>0.59</td>
<td>0.72</td>
<td>1.38</td>
<td>0.33</td>
</tr>
</tbody>
</table>

**Table 5.8** Efficiency in Test Cases Execution

**Number of Errors Found**
5.5. ANALYSIS AND INTERPRETATION

**Figure 5.8** Boxplot with the Efficiency in Test Cases Execution

**Table 5.9** Number of Errors Found

<table>
<thead>
<tr>
<th>Tool</th>
<th>Group</th>
<th>Min.</th>
<th>1st Qu.</th>
<th>Median</th>
<th>Mean</th>
<th>3rd Qu.</th>
<th>Max.</th>
<th>Std.Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>With</td>
<td>1</td>
<td>2.00</td>
<td>2.00</td>
<td>2.00</td>
<td>3.80</td>
<td>3.00</td>
<td>10.00</td>
<td>3.49</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4.00</td>
<td>6.00</td>
<td>7.00</td>
<td>6.60</td>
<td>8.00</td>
<td>8.00</td>
<td>1.67</td>
</tr>
<tr>
<td></td>
<td>1 and 2</td>
<td>2.00</td>
<td>2.25</td>
<td>5.00</td>
<td>5.20</td>
<td>7.75</td>
<td>10.00</td>
<td>2.97</td>
</tr>
<tr>
<td>Without</td>
<td>1</td>
<td>3.00</td>
<td>3.00</td>
<td>4.00</td>
<td>4.40</td>
<td>5.00</td>
<td>7.00</td>
<td>1.67</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
<td>1.00</td>
<td>1.40</td>
<td>2.00</td>
<td>4.00</td>
<td>1.67</td>
</tr>
<tr>
<td></td>
<td>1 and 2</td>
<td>0.00</td>
<td>1.25</td>
<td>3.00</td>
<td>2.90</td>
<td>4.00</td>
<td>7.00</td>
<td>2.23</td>
</tr>
</tbody>
</table>

**Figure 5.9** Boxplot with the Errors Found
Test Cases Effectiveness

Concerning the use and applicability of the TCE metric, to determine whether a test case created by the SPLMT-TE was effective, we compared the TCE from both groups, using and without using tool support. We measured the test case effectiveness as a ratio of the total amount of designed test cases by the total of errors found.

By applying the TCE formula, we obtained the distribution presented in Figure 5.10 and detailed in Table 5.10. It shows the data similarity. Mean of the groups (1 and 2) using the tool was 0.19 with Std.Dev. of 0.11. The mean of the groups (1 and 2) without using the tool was the same, 0.16 with Std.Dev. of 0.12. Median of groups using and not using the tool is the same, 0.17.

There was no significant difference between the test case effectiveness. The number of errors found per number of test cases was almost the same using and without using the tool. Test case effectiveness with and without the tool support was almost the same, 14.55 using the tool and 16.56 without using it.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Group</th>
<th>Min.</th>
<th>1st Qu.</th>
<th>Median</th>
<th>Mean</th>
<th>3rd Qu.</th>
<th>Max.</th>
<th>Std.Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>With</td>
<td>1</td>
<td>0.09</td>
<td>0.09</td>
<td>0.10</td>
<td>0.18</td>
<td>0.16</td>
<td>0.47</td>
<td>0.16</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
<td>0.07</td>
<td>0.09</td>
<td>0.19</td>
<td>0.20</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td>1 and 2</td>
<td>0.09</td>
<td>0.10</td>
<td>0.17</td>
<td>0.19</td>
<td>0.23</td>
<td>0.47</td>
<td>0.11</td>
</tr>
<tr>
<td>Without</td>
<td>1</td>
<td>0.16</td>
<td>0.16</td>
<td>0.22</td>
<td>0.24</td>
<td>0.26</td>
<td>0.41</td>
<td>0.10</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.13</td>
<td>0.18</td>
<td>0.23</td>
<td>0.20</td>
<td>0.24</td>
<td>0.24</td>
<td>0.09</td>
</tr>
<tr>
<td></td>
<td>1 and 2</td>
<td>0.00</td>
<td>0.09</td>
<td>0.17</td>
<td>0.16</td>
<td>0.21</td>
<td>0.41</td>
<td>0.12</td>
</tr>
</tbody>
</table>

5.5.2 Hypotheses Testing

Since the experiment has one factor with two treatments, completely randomized design, the data collected during the experiment were submitted to parametric tests, t-test to analyze the hypothesis. The tests are primarily presented for a significance level of 5%.

Regarding the Q2, the amount of test cases created using the tool is higher than without using it (see Figure 5.6), t-value = -4.59, degrees of freedom (df) = 15.2, and p-value = 0.0001729. The number of the p-value is lower than the significance level, rejecting the null hypothesis. Null Hypothesis $H_{01}$ is rejected, since $H_{01}: \mu_{DTC_{\text{manual}}} \geq \mu_{DTC_{\text{Tool}}}$.

Time spent creating test cases without the tool is higher than using it (see Figure 5.7). For this reason, in order to answer the Q1, the Null Hypothesis $H_{02}$ is rejected, since
5.5. ANALYSIS AND INTERPRETATION

Figure 5.10 Boxplot with the Test Case Effectiveness

$H_{02} : \mu_{ETCD_{manual}} \leq \mu_{ETCD_{Tool}}$, \( t\)-value $= 5.9$, $df = 9.4$, and $p$-value $= 0.0001999$. This $p$-value allowed the rejection with high significance level.

Thus, the efficiency in test case execution supports the Null Hypothesis $H_{03}$. It was not found any significant difference between the efficiency of create test cases with the tool and without it, as shows Figure 5.8. Finally, the question Q3 is answered, $t$-value $= 1.4$, $df = 12$ and $p$-value $= 0.09917$. As the $p$-value is higher than 0.05 the null hypothesis were not rejected.

Considering data from Table 5.9, the Null Hypothesis $H_{04}$ is rejected, since $H_{04} : \mu_{NEF_{manual}} \geq \mu_{NEF_{Tool}}$, and the number of errors found were higher using the tool support, $t$-value $= -1.96$, $df = 16.7$, and $p$-value $= 0.0373$, answering the Q4. The $p$-value is lower than 5% rejecting the null hypothesis.

As a result, the Null Hypothesis $H_{05}$ cannot be rejected, since $H_{05} : \mu_{TCE_{manual}} \geq \mu_{TCE_{Tool}}$, $t$-value $= -0.498$, $p$-value $= 0.3122$, and $df = 17.9$. Since the $p$-value is higher than the significance level the hypothesis cannot be rejected and no conclusion can be drawn. Regarding Q5, there is no significant differences between the effectiveness test cases values during the tool usage and without use it.
5.6. VALIDITY EVALUATION

5.5.3 Qualitative analysis

Data from subjects who used the tool were qualitatively analyzed. According to the subject’s opinion, factors next listed were considered so that subjects must attribute them with a Yes/No value. In addition, a text field was available for subjects to freely comment his/her choice. The opinion was gathered from the feedback questionnaire, applied right after they had executed the experiment.

Only 20% of the subjects needed additional information other than the available artifacts. The majority of subjects approved the use of the tool, except the factor on how effective the tool was. In summary, the overall comments regarding difficulties referred to lack of expertise in software testing, which directly impacted on their activities.

Ninety percent of the subjects agreed that the SPLMT-TE helped them at the creation of test cases and find more errors. Sixty percent of the subjects believed that the tool created sufficient test cases. Moreover, 33% changed the test cases created by the tool and 35% created more test cases.

5.6 Validity Evaluation

A fundamental question concerning results from an experiment is how valid the results are. According to Wohlin et al. (2000), it is important to consider the question of validity already in the planning phase in order to anticipate possible threats involved in the context of an experiment. Wohlin et al. (2000) adopt the four-type-categorization of the threats to the validity of the experimental results. Each classification scheme is described below, including threats that fit in the context of this experiment:

5.6.1 Conclusion Validity

Threats for the conclusion validity are concerned with issues that affect the ability to draw the correct conclusion about relationship between the treatment and outcome.

- **Experience of subjects.** Subjects without experience in test case creation, test cases execution and finding errors also can affect this validity, since it is harder for them to understand the context of the problem and how to handle with the requirements document, because of lack of professional inexperience;

- **Experience of SPL.** Subjects without experience, in special, Software Product Lines, it is harder for them understand the concepts such as commonalities and
variabilities, and how to handle combination of variation points of the test assets;

- **Reliability of Measures.** The validity of an experiment is highly dependent on the reliability of the measures. Once the measurements are not adequately, it can bring unreliable data. Aiming to mitigate this threat, it was validated with RiSE members.

### 5.6.2 Internal Validity

Threats to internal validity are influences that can affect the independent variable with respect to causality, without the researcher knowledge (Wohlin et al., 2000).

- **Maturation.** This is the effect that subjects react differently as time passes. Some subjects can be affected negatively (feel bored or tired) during the experiment, and their performance may be below normal. In order to mitigate this boredom, two familiar domains were provided. Other subjects can be affected positively (learning) during the course of the experiment. Subjects were free to stop for some moments, but they could not share information to other subjects;

- **Gained Experience.** It is the effect caused by the experiment execution order, in our case, there were two scenarios at the first phase group 1 execution using tool support and at the second phase without the tool. At the second scenario, group 2 executed the first phase manually and at the second phase with tool support. To reduce this risk, two distinct problems were analyzed, one for evaluating the tool support, and the other one to evaluate the manually test case creation and compare the two treatments.

### 5.6.3 Construct Validity

Threats to external validity are conditions that limit our ability to generalize the results of the experiment to industrial practice.

- **Mono-operation bias.** Since the experiment includes only one independent variable, the experiment may under-represent the construct and thus not give the full picture of the theory;

- **Experimenter Expectancies.** Surely the experimenter expectancies may bias the results, and for that reason, all formal definition and planning of the experiment
was carefully designed beforehand, and reviewed by other RiSE members that performed other experiments and advisors. We also consulted two experts at the area, Dr. David M. Weiss\textsuperscript{5} and Dr. John D. McGregor\textsuperscript{6};

- **Hypothesis guessing.** The subjects involved in the experiment might try to figure out what the purpose and intended result of the experiment is. They are likely to base their behavior on their guesses about the hypotheses. In order to minimize this risk, all formal definition and planning of the experiment were carefully designed and we searched for valid measures in the literature to aid in hypotheses and metrics definition;

- **Evaluation apprehension.** Some people are afraid of being evaluated (Wohlin \textit{et al.}, 2000). The subjects of the pilot were taking the discipline and they were afraid that the experiment could affect their grades. In order to avoid this risk, we explained that the execution of the pilot was not influence on their grades. During the experiment, the subjects executed their activities without pressure and without time restriction.

### 5.6.4 External Validity

Refers to the extent to which the experiment setting actually reflects the construct under study.

- **Generalization of subjects.** This is an effect of having a subject population not representative of the population we want to generalize to, i.e. the wrong people participating in the experiment. The pilot study was conducted with undergraduate students without knowledge about software testing. In this case, if these subjects succeed using the tool support, we cannot conclude that a experienced testing analyst would use it successfully too. Analyzing from another point of view, negative conclusions have external validity, i.e., if the subjects fail using the tool, then this is an evidence that an experienced testing analyst would fail too. For this reason, we executed the experiment with experienced subjects. Another complicating factor could be an experiment involving two SPL projects. Thus, some training sessions on the topic were held, involving the subjects in practical sessions, in which they could become familiar with the context as well as the purpose of product lines;

\textsuperscript{5}Professor from the Department of Computer Science at Iowa State University
\textsuperscript{6}Associate Professor of Computer Science at Clemson University Clemson University
5.7 Lessons Learned

After concluding the experimental studies, we gathered information that can be used as a guide to future replications. The structure presented in this Chapter can be reused in other general experiments in the SPL Testing practice.

Some important aspects should be considered, specially the ones seen as limitations in the experiment. We tried to mitigate some of these problems that we raised at the pilot study. The general impressions gathered from the experiment are described below.

**Training.** To eliminate the problem with lack of experience, we selected subjects with experience in the SPL and software testing area - see Table 5.3.

**Motivation.** To mitigate the complaints with boredom, we removed the 2 hours limitation for creation and execution. A possible solution for future replications could be to define only one activity by subject. The subjects with experience will be responsible for test creation activities and the inexperienced subjects will only execute and report errors.

**Questionnaires.** To suppress some problems related to the wrong data collection, we changed some questions of the questionnaires. The questionnaires should be constantly updated and calibrated before new replications.

**Project.** We would select projects with more specification available in advance. Subjects, mainly from the experiment, with a large experience in industry, complained about the lack of documentation to aid them to create the test cases. Moreover, as we are dealing with SPL projects, we need projects containing many variation points and variants in order to analyze the impact of the tool usage.

**Design Type.** We changed the design type at the experiment. At the first round, the subject from group 1 used the tool with the domain 1, and the group 2 also used the domain 1 but without tool support. At the second round, group 1 analyzed the domain 2
without tool support and group 2 analyzed the domain 2 with tool support.

**Measurement.** In order to eliminate the data loss that happened in the pilot study, we changed the instructions for the experiment, where each subject did the activities individually allowing a better interpretation of the data. We also enforced the importance of collect information during the creation of test cases with the tool.

### 5.8 Chapter Summary

This Chapter presented the experimental studies conducted to evaluate the proposed tool. It included the definition, planning, operation, analysis and interpretation of a pilot study and a experiment. The guidelines defined by Wohlin et al. (2000) were used to perform the experiment. The pilot study was conducted at the Federal University of Bahia, involving undergraduate students, and the experiment was conducted at the Software Engineering Laboratory (LES), involving M. Sc. and Phd. students.

The studies analyzed the SPLMT-TE effectiveness, in order to gather empirical evidences. Thus, we can conclude that there was no gain using the tool to save time during the execution of test cases. In the analysis of test case effectiveness, the t-test also did not reject the null hypothesis, concluding that there is no advantage in using the tool.

On the other hand, the number of test cases created and the number of errors found were higher when the subjects used the proposed tool. Furthermore, the effort spent during the test case design decrease when the tool was used.

Even with the reduced number of subjects (12 at the pilot study and 10 at the experiment), we could identify some directions for improvements, specially regarding the subjects difficulties. We believe that more experiments should be performed considering the lessons learned. A replication in industrial context will be interesting.

Unfortunately, only part of the functionalities of the SPLMT-TE could be evaluated at the experiment due to the users’ lack of experience in software testing. Activities that should be executed by test architects, test managers, and test designers, such as test suite and test plan creation could not be performed to avoid the infeasibility of the experiment.

Next chapter presents the conclusions of this dissertation, some related work, and directions for future work.
Testing software seems like one of the easiest things imaginable, but is not. Some of the difficulties in testing are incorrect requirements and tight time schedules (Craig and Jaskiel, 2002). To suppress some of these problems, testing tools are indispensable.

Currently, despite of the increasing interest by the research community regarding SPL testing (Neto et al., 2011a), it is very difficult to find suitable tools to support SPL testing processes. It has a direct impact in the high costs this activity poses, since testing becomes a cumbersome and laborious work (Edwin, 2007).

This dissertation presented SPLMT-TE - Test module of the Software Product Lines Management Tool, which is based on an extensive scoping study of current available tools, and on the RiSE group expertise.

The remainder of this chapter is organized as follows: Section 6.1 summarizes the achieved goals of the work and Section 6.2 presents a comparison with some related work. Section 6.3 points out some directions for future work unexplored by this work. Academic contributions are listed in Section 6.4, and finally, the concluding remarks are described in Section 6.5

### 6.1 Research Contributions

This work has three main contributions: (i) a **scoping study**, in which a formal process for analyzing the state-of-the-art in the SSTT and SPLTT fields was performed; (ii) the definition of requirements, architecture, and implementation of the SPLMT-TE,
6.1. RESEARCH CONTRIBUTIONS

the proposed tool for support the RiPLE-TE process; and finally (iii) **the experimental study**, which enabled us to define a model for conducting experimental studies to evaluate **SPL** testing tools and evaluated the tool effectiveness. Each one following detailed.

6.1.1 Scoping Study

We applied the scoping study, to collect evidences in the literature that allowed us to sketch and comprehend the state-of-the-art of single system and **SPL** testing tools research and practice field. The motivation was to identify the evidence available on the topic and point out gaps in the existing tools.

Through this study, twenty-seven tools were identified and analyzed according to aspects related to the research question - *how the available tools support the Software Product Lines Testing process*. This analysis offered the base to define a set of requirements that should be presented in **SPL** testing tools, and to map new tools development and usage.

6.1.2 SPLMT-TE

Based on the **SPLMT** tool for managing the **SPL** phases, we built the SPLMT-TE, the test module for support the RiPLE-TE process. In summary, the proposed tool is intended to reduce the effort of testing in **SPL** projects, by performing activities in a systematic and manageable way.

Moreover, SPLMT-TE allows the automatic creation of test cases based on use cases, management of test assets, organization of test cases into test suites, and test plans.

6.1.3 Experimental study

A pilot study was performed before the experiment in order to identify some problems that could have compromised the experimental study. The studies analyzed the tool effectiveness to reduce the effort spent during the system testing level of the **SPL** testing process.

Due to lack of information in the literature, we defined aspects from scratch, even without having baselines to compare gained results. We need to investigate complementary metrics to be used in next experiments.
6.2 Related Work

There are few studies describing and detailing tools for testing SPL projects. If we decide to narrow the scope, encompassing the search for only tools for the system testing level, within the SPL context, the findings are worse.

According to Reuys et al. (2005) the ScenTED-DTCD (Domain Test Case Scenario Derivation) is a prototype tool focused on generating test cases scenarios that describe the test engineer’s activities and the system response if modeled within the activity diagram. ScenTED has been applied within a case study at Siemens AG Medical Solutions HS IM. The goal of the case study was to reduce the effort for test creation by reusing test case scenarios.

Another prototype tool can be found (Nebut et al., 2007). It generates system test scenarios from use cases. The approach is based on the automation of the generation of application system tests, for any chosen product from the system requirements of a product line. The author also performed an experimental validation at the academy, using 3 products to study the efficiency of the tests generated.

Nogueira et al. (2007) proposed the Test and Requirements Generation Tool (TaRGeT), a tool for automatic test case generation from use case scenarios written in Natural Language. TaRGeT automates a systematic approach for dealing with requirements and test artifacts in an integrated way. The authors applied two case studies at the industry to evaluate the tool.

Finally, Oster et al. (2011) presented a tool chain based on commercial tools since it was developed within industrial cooperations. It contained a pairwise configuration selection component on the basis of a feature model. The authors are applying MoSoPoLiTe in two industrial cooperation.

In order to develop our work, we considered every mentioned study, since they bring relevant information that comprises the knowledge to develop SPL system testing tools.

6.3 Future Work

Due to time constraints imposed on the master degree, this work can be seen as an initial climbing towards a tool for support the system testing level of the SPL testing process, and interesting directions remain to improve what was started here and new routes can be explored in the future. Thus, the following issues should be investigated as future work:

- **Metrics.** This dissertation analyzed only few metrics found in the literature that
suite for evaluating the proposed tool effectiveness. For this reason, more metrics are needed to allow a complete analysis.

- **Testing Tools for the other Testing Levels.** Since the dissertation focused on system testing level, the other testing levels, unit, integration, and acceptance testing need further attention.

- **Experiment Replications.** As said before, more experiments should be performed considering the lessons learned described in Chapter 5. A next step is to perform an experimental study in the academia and industry with more experienced subjects. This will give more evidences and consequently more confidence on the effectiveness of the tool.

- **Test Automation.** During the definition of our proposal, we identified the necessity of extend the architecture in order to allow support the management of source code and subsequently automatically generate unit and integration test cases. As a consequence, the next challenge will be understand how to define *which tests should be automated (first)?* In order to answer this question, it will be necessary to choose the fundamental tests that are run the most often in order to validate the essential functions. If we have a set of breadth tests (that will be provided by our proposal) for a particular product of the SPL, some tests will be automatable, and some will not. Of those, which are automatable, we need to decide what tests we want to automate. We will not be able to automate all at once, even of the ones that could be automated (Fewster and Graham, 1999).

- **New Functionalities.** New functionalities could be implemented to allow the report execution, testers will execute the test plan and report if the test cases passed or failed. At the end of the execution the tool will generate more detailed reports.

### 6.4 Academic Contributions

Besides the final contributions listed so far, some intermediate result of this work has been reported in the literature, as shown in the following:

6.5 Concluding Remarks

The contributions of this dissertation are result of a consistent research relied on single systems and SPL testing tools. Even with several existing testing tools, they are mostly unable of supporting a SPL Testing process directly. Moreover, the majority of the proposed tools and prototypes regarding SPL were not implemented.

In this context, this work presented the SPLMT-TE, a Software Product Lines System Test Case Tool, which aims to support the system testing level of the SPL testing process by creating system test cases based on use cases. This tool can also creates, manages, and organizes test plans and test suites.

Additionally, the tool was evaluated in an academic context, through an experimental study. According to the data collected and analyzed in the experiment, the tool presents indications of its effectiveness. We believe this dissertation is one more step to the maturation of the testing tools in software product lines testing.

The SPLMT has currently being applied in a SPL industrial project, in the medical information management domain. It is a large project, including about 800 features. Based on this project we will apply, along the 2011/2 semester, the SPLMT-TE in a part of the project, and collect data to improve the proposal.

Then, we will have the opportunity to gather data in a set of aspects, such as: understand the impact of the variability points and variants in a project. We have in mind that real SPL may have hundred of variation points and several hundreds of variants, hence it is necessary to understand, with evidence, how the proposed tool fits in different scenarios.

As constraints grow in complexity and number, the difficulty of modeling and generating test cases increases; we will try to verify the effectiveness and feasibility of the SPLMT-TE. For this reason, we propose to additionally investigate how much effort will be safe during the use of the tool in a industrial context.


Appendices
This appendix lists the journals A.1 and the conferences A.2 used in locating primary studies in the Mapping Study explained in Chapter 3.

A.1 List of Journals

<table>
<thead>
<tr>
<th>Table A.1 List of Journals</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Journals</strong></td>
</tr>
<tr>
<td>ACM Computing Survey</td>
</tr>
<tr>
<td>ACM Transactions on Software Engineering and Methodology (TOSEM)</td>
</tr>
<tr>
<td>Annals of Software Engineering</td>
</tr>
<tr>
<td>Automated Software Engineering</td>
</tr>
<tr>
<td>ELSEVIER Information and Software Technology (IST)</td>
</tr>
<tr>
<td>ELSEVIER Journal of Systems and Software (JSS)</td>
</tr>
<tr>
<td>IEEE Software</td>
</tr>
<tr>
<td>IEEE Transactions on Software Engineering</td>
</tr>
<tr>
<td>Journal of Systems and Software</td>
</tr>
<tr>
<td>Software Process: Improvement and Practice</td>
</tr>
<tr>
<td>Software Practice and Experience</td>
</tr>
<tr>
<td>Journal of Software Maintenance Research and Practice</td>
</tr>
<tr>
<td>Journal of Systems and Software (JSS)</td>
</tr>
<tr>
<td>Software Practice and Experience (SPE) Journal</td>
</tr>
<tr>
<td>Software Quality Journal</td>
</tr>
<tr>
<td>Software Testing, Verification and Reliability</td>
</tr>
</tbody>
</table>
### A.2 List of Conferences

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Conference Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>APSEC</td>
<td>Asia Pacific Software Engineering Conference</td>
</tr>
<tr>
<td>ASE</td>
<td>International Conference on Automated Software Engineering</td>
</tr>
<tr>
<td>CAiSE</td>
<td>International Conference on Advanced Information Systems Engineering</td>
</tr>
<tr>
<td>CBSE</td>
<td>International Symposium on Component-based Software Engineering</td>
</tr>
<tr>
<td>COMPASS</td>
<td>International Computer Software and Applications Conference</td>
</tr>
<tr>
<td>ECBS</td>
<td>International Conference and Workshop on the Engineering of Computer Based Systems</td>
</tr>
<tr>
<td>ECOWS</td>
<td>European Conference on Web Services</td>
</tr>
<tr>
<td>ECSA</td>
<td>European Conference on Software Architecture</td>
</tr>
<tr>
<td>ESEC</td>
<td>European Software Engineering Conference</td>
</tr>
<tr>
<td>ESEM</td>
<td>Empirical Software Engineering and Measurement</td>
</tr>
<tr>
<td>FASE</td>
<td>Fundamental Approaches to Software Engineering</td>
</tr>
<tr>
<td>ICCBSS</td>
<td>International Conference on Composition-Based Software Systems</td>
</tr>
<tr>
<td>ICSE</td>
<td>International Conference on Software Engineering</td>
</tr>
<tr>
<td>ICSM</td>
<td>International Conference on Software Maintenance</td>
</tr>
<tr>
<td>ICSR</td>
<td>International Conference on Software Reuse</td>
</tr>
<tr>
<td>ICST</td>
<td>International Conference on Software Testing, Verification and Validation</td>
</tr>
<tr>
<td>ICWS</td>
<td>International Conference on Web Services</td>
</tr>
<tr>
<td>ISSRE</td>
<td>International Symposium on Software Reliability Engineering</td>
</tr>
<tr>
<td>GPCE</td>
<td>International Conference on Generative Programming and Component Engineering</td>
</tr>
<tr>
<td>MODEL</td>
<td>International Conference on Model Driven Engineering Languages and Systems</td>
</tr>
<tr>
<td>MoTiP</td>
<td>Workshop on Model-based Testing in Practice</td>
</tr>
<tr>
<td>OOPSLA</td>
<td>ACM SIGPLAN conference on Object-Oriented Programming, Systems, Languages, and Applications</td>
</tr>
<tr>
<td>PROFES</td>
<td>International Conference on Product Focused Software Development and Process Improvement</td>
</tr>
<tr>
<td>QoSA</td>
<td>International Conference on the Quality of Software Architectures</td>
</tr>
<tr>
<td>QSIC</td>
<td>International Conference on Quality Software</td>
</tr>
<tr>
<td>ROSATEA</td>
<td>International Workshop on The Role of Software Architecture in Testing and Analysis</td>
</tr>
<tr>
<td>SAC</td>
<td>Annual ACM Symposium on Applied Computing</td>
</tr>
<tr>
<td>SEAA</td>
<td>Euromicro Conference on Software Engineering and Advanced Applications</td>
</tr>
<tr>
<td>SEKE</td>
<td>International Conference on Software Engineering and Knowledge Engineering</td>
</tr>
<tr>
<td>SPLC</td>
<td>Software Product Line Conference</td>
</tr>
<tr>
<td>SPLiT</td>
<td>Software Product Line Testing Workshop</td>
</tr>
<tr>
<td>TAIC PART</td>
<td>Testing: Academic and Industrial Conference</td>
</tr>
<tr>
<td>TEST</td>
<td>International Workshop on Testing Emerging Software Technology</td>
</tr>
<tr>
<td>WICSA</td>
<td>Working IEEE/IFIP Conference on Software Architecture</td>
</tr>
<tr>
<td>WS-Testing</td>
<td>International Workshop on Web Services Testing</td>
</tr>
</tbody>
</table>
Experimental Study Instruments

This appendix presents the instruments given to the subjects involved in the Experimental Study, prior presented in Chapter 5. The set of instruments included in this Appendix comprise the following forms: B.2 presents the consent form subjects must be given and signed before joining the Experimental Study, confirming permission to participate in the research; then B.1 details the background questionnaire, intended to collect data about the subjects background; and the feedback questionnaire, answered by the subjects after performing the experiment: B.7 to subjects who used the proposed tool first and B.8 to the ones who created tests manually. An addition questionnaire - B.9 - should be answered by some selected subjects which did not use the SPLMT-TE, after they have performed the experiment using it.
B.1 Background Questionnaire

A. GENERAL INFORMATION

1. Age: _____
2. Sex: [ ] Male [ ] Female
3. Current Undergraduate Semester*: _____
4. Grade Point Average (GPA)*: _____

(*) information used during the pilot study only

B. TECHNICAL KNOWLEDGE - PROFESSIONAL EXPERIENCE

1. English reading:
   [ ] Bad
   [ ] Medium
   [ ] Good

2. Previous experience with software development:
   [ ] I’ve never developed software
   [ ] I’ve already developed software, alone
   [ ] I’ve already developed software, in group classes
   [ ] I’ve already developed software, in companies

3. What is your experience with programming (in months/years)?

[ ] Industry [ ] School/Academia

4. What are the programming languages you have used / are using now?
5. What is your experience with software testing (in months/years)? Which Levels (Unit, Integration, Acceptance or System Testing)?

6. Previous experience with software testing:
   [ ] None
   [ ] I’ve already studied testing, either in class or in books
   [ ] I’ve already developed projects, in academic context, applying testing concepts
   [ ] I’ve already been involved in one industrial testing project
   [ ] I’ve already been involved in several industrial testing projects

7. Have you ever used any software testing tool? Which one(s)?

8. What is your experience with Software Product Lines (in months/years)?

9. Previous experience with Software Product Lines:
   [ ] None
   [ ] I’ve already tried it, either in class or in books
   [ ] I’ve already used it in academic project(s)
   [ ] I’ve already used it in one industrial testing project
   [ ] I’ve already used it in several industrial testing projects

10. Please, write down any suggestion you think might would be useful
# B.2 Consent form

<table>
<thead>
<tr>
<th>Subject Name:</th>
</tr>
</thead>
</table>

The information contained in this form is intended to establish a written agreement, whereby the student authorizes his/her participation in the experiment Software Product Lines System Test Tool, with full knowledge of the nature of the procedures he/she will submit as a participant, with free will and without any duress. This participation is voluntary and the subject is free to withdraw from the experiment at any time and no longer participate in the study without prejudice to any service that is being or will be submitted.

## I. STUDY TITLE:
On the behavior of the Software Product Lines System Test Tool.

## II. STUDY GOAL:
Evaluate the effectiveness of the proposed tool.

## III. RESPONSIBLE INSTITUTIONS:
Federal University of Pernambuco (UFPE) and Federal University of Bahia (UFBA).

## IV. RESPONSIBLE RESEARCHERS:
Eduardo Almeida, Dr. (UFBA) - Crescencio R. Lima Neto, MSc. Candidate (UFPE).

## V. CONSENT:
By signing this consent form, I certify that have read the information above and I am sufficiently informed of all statements, I fully agree to participate on the experiment. So, I authorize the execution of the research discussed above.

Salvador, BA, Brazil, / / /  

---  

**Signature**
## B.3 Test Case Creation Form

Table B.2: Test Case Creation Form

<table>
<thead>
<tr>
<th>TEST CASE CREATION FORM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Case ID: __________</td>
</tr>
</tbody>
</table>

Name of the Test Case: ____________________________________________________________

Summary:

_____________________________________________________________________________

_____________________________________________________________________________

Test Type: [ ] Positive [ ] Negative Related Use Case: ____________________________

Test Priority: [ ] High [ ] Medium [ ] Low

Responsible: ________________________________________________________________

Pre-condition:

_____________________________________________________________________________

_____________________________________________________________________________

Pos-condition:

_____________________________________________________________________________

Steps:

_____________________________________________________________________________

_____________________________________________________________________________

_____________________________________________________________________________
## B.4 Test Case Execution Form

Table B.3: Test Case Execution Form

<table>
<thead>
<tr>
<th>Subject ID #</th>
<th>Start Time: <em><strong>:</strong></em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Date: <em><strong>/</strong></em>/____</td>
<td>End Time: <em><strong>:</strong></em></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Id #</th>
<th>Time to Execute</th>
<th>Responsible</th>
<th>Result</th>
<th>Error ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Case ID</td>
<td>Time Spent at the execution of a test case</td>
<td>Responsible for execute the test case</td>
<td>Result of the execution of the test case</td>
<td>If the test fail, report the ID of the error</td>
</tr>
<tr>
<td><em><strong>:</strong></em></td>
<td>[ ] Passed [ ] Failed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em><strong>:</strong></em></td>
<td>[ ] Passed [ ] Failed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em><strong>:</strong></em></td>
<td>[ ] Passed [ ] Failed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em><strong>:</strong></em></td>
<td>[ ] Passed [ ] Failed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em><strong>:</strong></em></td>
<td>[ ] Passed [ ] Failed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em><strong>:</strong></em></td>
<td>[ ] Passed [ ] Failed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em><strong>:</strong></em></td>
<td>[ ] Passed [ ] Failed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em><strong>:</strong></em></td>
<td>[ ] Passed [ ] Failed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em><strong>:</strong></em></td>
<td>[ ] Passed [ ] Failed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em><strong>:</strong></em></td>
<td>[ ] Passed [ ] Failed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em><strong>:</strong></em></td>
<td>[ ] Passed [ ] Failed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em><strong>:</strong></em></td>
<td>[ ] Passed [ ] Failed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em><strong>:</strong></em></td>
<td>[ ] Passed [ ] Failed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em><strong>:</strong></em></td>
<td>[ ] Passed [ ] Failed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em><strong>:</strong></em></td>
<td>[ ] Passed [ ] Failed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em><strong>:</strong></em></td>
<td>[ ] Passed [ ] Failed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em><strong>:</strong></em></td>
<td>[ ] Passed [ ] Failed</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

120
## B.5 Error Reporting Form

Table B.4: Error Reporting Form

<table>
<thead>
<tr>
<th>Subject ID #:</th>
<th>Date:<strong>/</strong>/____</th>
<th>Total of Errors:</th>
<th>Start Time: <strong>:</strong></th>
<th>End Time: <strong>:</strong></th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>ID #</th>
<th>Req.</th>
<th>Use Case</th>
<th>Error Description</th>
<th>Time when error was found</th>
<th>Severity</th>
<th>Test Case ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td><strong>:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td><strong>:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td><strong>:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td><strong>:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td><strong>:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td><strong>:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td><strong>:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td><strong>:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td><strong>:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td><strong>:</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### B.6 Automatic Test Case Creation Report Form

Table B.5: Test Case Creation Report Form

<table>
<thead>
<tr>
<th>Subject ID #:</th>
<th>Start Time: <em><strong>:</strong></em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Date: <em><strong>/</strong></em>/____</td>
<td>End Time: <em><strong>:</strong></em></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test Case ID</th>
<th>Time to Create</th>
<th>Responsible</th>
</tr>
</thead>
<tbody>
<tr>
<td><em><strong>:</strong></em></td>
<td>Time spent at the creation of a test case</td>
<td>Responsible for creating the test case</td>
</tr>
<tr>
<td><em><strong>:</strong></em></td>
<td><em><strong>:</strong></em></td>
<td><em><strong>:</strong></em></td>
</tr>
<tr>
<td><em><strong>:</strong></em></td>
<td><em><strong>:</strong></em></td>
<td><em><strong>:</strong></em></td>
</tr>
<tr>
<td><em><strong>:</strong></em></td>
<td><em><strong>:</strong></em></td>
<td><em><strong>:</strong></em></td>
</tr>
<tr>
<td><em><strong>:</strong></em></td>
<td><em><strong>:</strong></em></td>
<td><em><strong>:</strong></em></td>
</tr>
<tr>
<td><em><strong>:</strong></em></td>
<td><em><strong>:</strong></em></td>
<td><em><strong>:</strong></em></td>
</tr>
<tr>
<td><em><strong>:</strong></em></td>
<td><em><strong>:</strong></em></td>
<td><em><strong>:</strong></em></td>
</tr>
<tr>
<td><em><strong>:</strong></em></td>
<td><em><strong>:</strong></em></td>
<td><em><strong>:</strong></em></td>
</tr>
<tr>
<td><em><strong>:</strong></em></td>
<td><em><strong>:</strong></em></td>
<td><em><strong>:</strong></em></td>
</tr>
<tr>
<td><em><strong>:</strong></em></td>
<td><em><strong>:</strong></em></td>
<td><em><strong>:</strong></em></td>
</tr>
</tbody>
</table>
B.7 Feedback Questionnaire A

A. GENERAL INFORMATION

1. Subject Name: ____________________________
2. ID: ____________________________

B. REGARDING THE EXPERIMENT

1. How effective was the training, in your opinion? Was it helpful to make you understand the procedures of the creation and execution of test cases? There is something missing, or something that you think could be done better?
   - [ ] Training was effective, it helped me to understand the tasks of creation and execution of test cases.
   - [ ] Training was effective, it helped me to understand the tasks of creation and execution of test cases, but training timing was too short.
   - [ ] It would have been more effective if it had more practical examples.
   - [ ] Activity fairly intuitive, but you need good experience to apply it according to the rules and estimated time.
   - [ ] It should have been shown an example, following step by step all the possible details that might arise during the testing activity.

2. Did you have doubts on any notion of the presented activities of creation and execution of test cases? If yes, how you handled it?
   - [ ] Yes. I asked the instructors for explanations.
   - [ ] Yes. I just revised training material.
   - [ ] No.

   Comments:

3. Besides the knowledge acquired in training on Software Product Lines and Testing, you needed other information to perform the experiment?
   - [ ] Yes  [ ] No.

   If yes, which additional information?

4. Which black box Testing method did you use?
   - [ ] Random Testing.
B.7. FEEDBACK QUESTIONNAIRE A

[ ] Equivalence Class Partitioning.
[ ] Boundary-value analysis.
[ ] Cause-effect graph.
[ ] Error Guessing.
[ ] None.

If not, why?

________________________________________

5. Was the method used above effective in help you finding defects?
[ ] Yes  [ ] No.

Comments:

________________________________________

C. REGARDING THE PROPOSED TOOL

1. Besides the knowledge acquired in training of the proposed tool, you needed other information to perform the experiment?
[ ] Yes  [ ] No.

If yes, which additional information?

________________________________________

2. Was the proposed Tool efficient in help you create test cases and finding defects?
[ ] Yes  [ ] No.

Comments:

________________________________________

3. Was the automatic test case creation sufficient? You need to change the test cases or create more test cases?
[ ] Yes  [ ] No.

Comments:

________________________________________
4. Do you think that the proposed tool, presented in details in training, contributed to the test case creation?
   [ ] Yes  [ ] No.
   
   *Comment your answer:*

5. Do you think there is any other important information that must be presented by the tool?
   [ ] Yes  [ ] No.
   
   *Comment your answer:*

6. Did you have any problem with the creation of test cases?
   [ ] Yes  [ ] No.
   
   *Comment your answer:*

7. What were the major difficulties you faced while performing the experiment?

8. Please, write down any suggestion you think might be useful
B.8 Feedback Questionnaire B

A. GENERAL INFORMATION

1. Subject Name: 

2. ID: 

B. REGARDING THE EXPERIMENT

1. How effective was the training, in your opinion? Was it helpful to make you understand the procedures of the creation and execution of test cases? There is something missing, or something that you think could be done better?
   - [ ] Training was effective, it helped me to understand the tasks of creation and execution of test cases.
   - [ ] Training was effective, it helped me to understand the tasks of creation and execution of test cases, but training timing was too short.
   - [ ] It would have been more effective if it had more practical examples.
   - [ ] Activity fairly intuitive, but you need good experience to apply it according to the rules and estimated time.
   - [ ] It should have been shown an example, following step by step all the possible details that might arise during the testing activity.

2. Did you have doubts on any notion of the presented activities of creation and execution of test cases? If yes, how you handled it?
   - [ ] Yes. I asked the instructors for explanations.
   - [ ] Yes. I just revised training material.
   - [ ] No.

Comments:

3. Besides the knowledge acquired in training on Software Product Lines and Testing, you needed other information to perform the experiment?
   - [ ] Yes  [ ] No.

If yes, which additional information?

4. Which black box Testing method did you use?
   - [ ] Random Testing.
5. Was the method used above effective in help you finding defects?
   [ ] Yes
   [ ] No.

Comments:

________________________________________________________________________________________

6. What were the major difficulties you faced while performing the experiment?

________________________________________________________________________________________

________________________________________________________________________________________

7. Please, write down any suggestion you think might would be useful

________________________________________________________________________________________

________________________________________________________________________________________

________________________________________________________________________________________
B.9 Feedback Questionnaire C

A. GENERAL INFORMATION

1. Subject Name: ________________________________

2. ID: ________________________________

B. REGARDING THE PROPOSED TOOL

1. Do you think that the proposed tool, just presented in the training session, would aid you in the creation of test cases and finding more defects than using an ad-hoc approach, like you did?
   [ ] Yes
   [ ] No.

*Comment your answer.*
B.10 Feedback Questionnaire D

A. GENERAL INFORMATION

1. Subject Name: 

2. ID: 

B. REGARDING THE PROPOSED TOOL

1. Do you think that the proposed tool, just presented in the training session, would aid you in the creation of test cases and finding more defects than using an ad-hoc approach, like you did?
   [ ] Yes
   [ ] No.

Comment your answer.

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

2. Besides the knowledge acquired in training of the proposed tool, you needed other information to perform the experiment?
   [ ] Yes  [ ] No.

If yes, which additional information?

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

3. Was the proposed Tool efficient in help you create test cases and finding defects?
   [ ] Yes  [ ] No.

Comments:
4. Was the automatic test case creation sufficient? You need to change the test cases or create more test cases?
   [ ] Yes  [ ] No.

   Comments:

5. Do you think that the proposed tool, presented in details in training, contributed to the test case creation?
   [ ] Yes  [ ] No.

   Comment your answer:

6. Do you think there is any other important information that must be presented by the tool?
   [ ] Yes  [ ] No.

   Comment your answer:

7. Did you have any problem with the creation of test cases?
   [ ] Yes  [ ] No.

   Comment your answer:

8. What were the major difficulties you faced while performing the experiment?

9. Please, write down any suggestion you think might be useful
This appendix presents the information gathered at the pilot study, prior presented in Chapter 5.

C.1 Execution

Table C.1 sketches the schedule of the pilot study describing the training and execution sessions. At the end, data was collected.

<table>
<thead>
<tr>
<th>Activity Length</th>
<th>Day 1 - June 28\textsuperscript{th}, 2011</th>
<th>Day 2 - June 30\textsuperscript{th}, 2011</th>
<th>Day 3 - July 05\textsuperscript{th}, 2011</th>
<th>Day 4 - July 07\textsuperscript{th}, 2011</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Explanation on the Experiment 0:15</td>
<td>Training on Testing 1:30</td>
<td>Training on the Tool 0:15</td>
<td>Adhoc Creation / Execution 0:15</td>
</tr>
<tr>
<td>2</td>
<td>Characterization - B.1 / Consent Term - B.2 0:15</td>
<td>Exercises 0:30</td>
<td>Feedback A - B.7 1:15</td>
<td>Feedback B - B.8 0:30</td>
</tr>
<tr>
<td></td>
<td>Introduction to SPL 1:30</td>
<td></td>
<td>Feedback B - B.8 0:30</td>
<td>Feedback B - B.8 0:30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Training on the Tool 0:15</td>
<td>Feedback B - B.8 0:30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Feedback B - B.8 0:30</td>
<td>Feedback B - B.8 0:30</td>
</tr>
</tbody>
</table>

Table C.2 introduces the initial division of subjects to group 1 and group 2. The division was equal and the rounds were held in different days, it caused that some subject could miss the first round or the second round. In order to eliminate this influence, we balanced the groups to normalize the results.

Table C.3 shows a concentrated view of the subjects’ profile, group 1 is detached with gray lines and group 2 with white lines. In this table, the experience of the subjects with software development, Java,


C.2.  ANALYSIS AND INTERPRETATION

Table C.2  Subjects of the pilot divided into groups

<table>
<thead>
<tr>
<th>Group</th>
<th>Subjects ID</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3 4 6 7 9 11 12</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>1 2 5 8 10 13 14</td>
<td>7</td>
</tr>
</tbody>
</table>

Testing, SPL and with testing tools is organized in terms of months dealing with such aspects. The lack of experience in Testing, SPL and using test tools could be a threat to the validity of the experiment, training sessions focused on these aspects in order to mitigate them.

Only two subjects had experience with testing tools at the industry, one of them worked with NUnit\(^1\) and the other worked with Junit. For this reason, each subject was placed in different groups. The grade point average was analyzed only at the pilot study since the subjects were undergraduate students.

Table C.4 presents the final grouping of subjects. Although we analyzed all the data generated by these two subjects, we believe that the absence of these two subjects did not invalidate our work, in terms of statistical analysis and interpretation of the results.

C.2  Analysis and Interpretation

Table C.5 shows the information gathered in the pilot study organizing the data generated by subjects in groups. The data were rearranged in groups to facilitate the identification. Since the subjects worked like a testing group the analysis was done considering the value generated by the group.

Table C.6 details the Designed Test Cases data with the test cases created per group. In groups (1 and 2) using the tool, the mean was 3.50 with Std.Dev. of 1.20, and 2.80 of mean with Std.Dev. of 1.10 in groups (1 and 2) without using the tool. Median of groups using the tool is slightly higher than groups that not use tool support.

There was no significant difference between the number of designed test cases with and without the tool. The number of test cases created with the tool support was similar to the number of test cases created without it.

Efficiency in Test Cases Design. In terms of efficiency of designed test cases per time spent with design, Table C.7 shows data gathered from the both groups.

In groups (1 and 2) using the tool, the mean value was 0.69 with Std.Dev. of 0.36, while in groups (1 and 2) without using tool support, the mean value was 0.12, with Std.Dev. of 0.05. The median value of groups using the tool is greater than groups without using tool support.

Table C.7 presents that the efficiency in test case design using the tool was higher than the efficiency without using it. The groups that worked with the tool created more test cases faster than the groups without tool support.

Efficiency in Test Cases Execution. Boxplot shown in Table C.8 contains data from the efficiency in test cases execution. The mean in groups (1 and 2) using the tool was 0.44 with Std.Dev. of 0.30. In groups without using the tool the mean was 0.41 with Std.Dev. of 0.26. Median of groups using the tool is slightly higher than in groups without using the tool.

---

\(^1\)Unit-testing framework for all .Net Languages - http://www.nunit.org/
<table>
<thead>
<tr>
<th>Group</th>
<th>Subject ID</th>
<th>GPA</th>
<th>English Reading</th>
<th>Part. in Industry</th>
<th>Experience in</th>
<th>Testing Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Dev. Project</td>
<td>Programming*</td>
<td>Java*</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Test. Project</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>7.0</td>
<td>advanced</td>
<td>No</td>
<td>24</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>7.6</td>
<td>intermediate</td>
<td>Yes</td>
<td>72</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>6.8</td>
<td>advanced</td>
<td>No</td>
<td>24</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>7.0</td>
<td>intermediate</td>
<td>No</td>
<td>24</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>6.3</td>
<td>advanced</td>
<td>Yes</td>
<td>24</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>6.3</td>
<td>intermediate</td>
<td>No</td>
<td>24</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>6.1</td>
<td>intermediate</td>
<td>No</td>
<td>18</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>7.5</td>
<td>intermediate</td>
<td>No</td>
<td>18</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>6.1</td>
<td>intermediate</td>
<td>No</td>
<td>24</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>6.0</td>
<td>advanced</td>
<td>Yes</td>
<td>72</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>6.5</td>
<td>intermediate</td>
<td>No</td>
<td>24</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>6.4</td>
<td>advanced</td>
<td>No</td>
<td>24</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>7.3</td>
<td>intermediate</td>
<td>No</td>
<td>24</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>7.4</td>
<td>intermediate</td>
<td>No</td>
<td>20</td>
<td>10</td>
</tr>
</tbody>
</table>

(*) the experience is expressed in months
### Table C.4 Final grouping of Subjects from the Pilot

<table>
<thead>
<tr>
<th>Group</th>
<th>Subjects ID</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3 4 6 7 9 11</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>1 2 5 8 10 13</td>
<td>6</td>
</tr>
</tbody>
</table>

### Table C.5 Pilot Study information

<table>
<thead>
<tr>
<th>Tool</th>
<th>Group</th>
<th>ID</th>
<th>DTC</th>
<th>ETCD*</th>
<th>ETC</th>
<th>ETCE*</th>
<th>NEF</th>
</tr>
</thead>
<tbody>
<tr>
<td>With</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3</td>
<td>8</td>
<td>3</td>
<td>10</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>13</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>18</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>15</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>3</td>
<td>10</td>
<td>3</td>
<td>13</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>15</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>2</td>
<td>12</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>13</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Without</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>25</td>
<td>4</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3</td>
<td>28</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>5</td>
<td>47</td>
<td>4</td>
<td>19</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>4</td>
<td>25</td>
<td>4</td>
<td>40</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>2</td>
<td>30</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>3</td>
<td>30</td>
<td>4</td>
<td>18</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>4</td>
<td>33</td>
<td>4</td>
<td>9</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>24</td>
<td>2</td>
<td>15</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>3</td>
<td>29</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>4</td>
<td>32</td>
<td>4</td>
<td>11</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>4</td>
<td>39</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>2</td>
<td>35</td>
<td>4</td>
<td>20</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

(*) - amount of time expressed in minutes

### Table C.6 Designed Test Cases - Pilot

<table>
<thead>
<tr>
<th>Tool</th>
<th>Min.</th>
<th>1st Qu.</th>
<th>Median</th>
<th>Mean</th>
<th>3rd Qu.</th>
<th>Max.</th>
<th>Std.Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>With</td>
<td>2.00</td>
<td>2.80</td>
<td>3.50</td>
<td>3.50</td>
<td>4.00</td>
<td>6.00</td>
<td>1.20</td>
</tr>
<tr>
<td>Without</td>
<td>1.00</td>
<td>2.00</td>
<td>3.00</td>
<td>2.80</td>
<td>3.00</td>
<td>5.00</td>
<td>1.10</td>
</tr>
</tbody>
</table>

### Table C.7 Efficiency in Test Case Design - Pilot

<table>
<thead>
<tr>
<th>Tool</th>
<th>Min.</th>
<th>1st Qu.</th>
<th>Median</th>
<th>Mean</th>
<th>3rd Qu.</th>
<th>Max.</th>
<th>Std.Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>With</td>
<td>0.13</td>
<td>0.36</td>
<td>0.83</td>
<td>0.69</td>
<td>1.00</td>
<td>1.00</td>
<td>0.36</td>
</tr>
<tr>
<td>Without</td>
<td>0.05</td>
<td>0.09</td>
<td>0.10</td>
<td>0.12</td>
<td>0.13</td>
<td>0.24</td>
<td>0.05</td>
</tr>
</tbody>
</table>
There was no significant difference between the efficiency in test case execution with and without the tool. The time spent to execute the test cases created by the tool was almost the same to execute the tests created manually.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Min.</th>
<th>1st Qu.</th>
<th>Median</th>
<th>Mean</th>
<th>3rd Qu.</th>
<th>Max.</th>
<th>Std.Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>With</td>
<td>0.10</td>
<td>0.21</td>
<td>0.36</td>
<td>0.44</td>
<td>0.53</td>
<td>1.00</td>
<td>0.30</td>
</tr>
<tr>
<td>Without</td>
<td>0.15</td>
<td>0.23</td>
<td>0.32</td>
<td>0.41</td>
<td>0.54</td>
<td>1.00</td>
<td>0.26</td>
</tr>
</tbody>
</table>

**Test Cases Effectiveness**, calculates the number of errors found per designed test case as describes Table C.9. It shows the data similarity. Mean of the groups (1 and 2) using the tool was 0.17 with Std.Dev. of 0.24. The mean of the groups (1 and 2) without using the tool was the same, 0.20 with Std.Dev. of 0.25. Median of groups using the tool was 0.00 and not using the tool was, 0.08.

The effectiveness with and without tool support is similar. There was no significant difference between the test case effectiveness with and without the tool.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Min.</th>
<th>1st Qu.</th>
<th>Median</th>
<th>Mean</th>
<th>3rd Qu.</th>
<th>Max.</th>
<th>Std.Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>With</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.17</td>
<td>0.33</td>
<td>0.67</td>
<td>0.24</td>
</tr>
<tr>
<td>Without</td>
<td>0.00</td>
<td>0.00</td>
<td>0.08</td>
<td>0.20</td>
<td>0.42</td>
<td>0.66</td>
<td>0.25</td>
</tr>
</tbody>
</table>

Finally, the **Number of errors found** was not analyzed in descriptive statistic because of the small amount of data, only 6 errors were found by the subjects using the tool and 8 without using it.