

Table of Contents

Introduction	xi
Jean-Louis BOULANGER	
Chapter 1. SPARK – A Language and Tool-Set for High-Integrity Software Development	1
Ian O’NEILL	
1.1. Introduction	1
1.2. An overview of SPARK	2
1.2.1. What is SPARK?	2
1.3. The rationale behind SPARK	3
1.3.1. Flow analysis	6
1.3.2. Code proof	11
1.3.3. Correctness by construction	14
1.4. Industrial applications of SPARK	15
1.4.1. SHOLIS	16
1.4.2. Lockheed-Martin C-130J mission computer	17
1.4.3. MULTOS CA	19
1.4.4. Tokeneer	20
1.4.5. Aircraft monitoring software	23
1.4.6. iFACTS.	24
1.4.7. SPARK Skein	24
1.5. Conclusion	25
1.6. Bibliography	26
Chapter 2. <i>Model-Based Testing</i> Automatic Generation of Test Cases Using the Markov Chain Model	29
H�el�ene LE GUEN, Frederique VALL�EE and Anthony FAUCOGNEY	
2.1. Preliminaries on the test process	29
2.1.1. Findings	29

2.1.2. Test optimization	30
2.1.3. The statistical usage test	30
2.1.4. Generating test cases	32
2.2. Modeling using Markov chains	32
2.2.1. Origin	32
2.2.2. Mathematical formalization	34
2.2.3. Principles of generation	42
2.2.4. Some indicators	44
2.2.5. Calculating reliability	45
2.3. The MaTeLo tool	52
2.3.1. Engineering tests directed by models, model-based testing	52
2.3.2. A chain of tools	54
2.3.3. The usage model	55
2.3.4. Configuration of test strategies	64
2.3.5. Generating test campaigns	65
2.3.6. Analysis of the results and indicators	69
2.4. Examples of industrial applications	75
2.4.1. AUDI	75
2.4.2. Magneti marelli	77
2.4.3. Other industrial applications	78
2.4.4. Industrialization of the tests	78
2.5. Conclusion	79
2.6. Bibliography	80
Chapter 3. Safety Analysis of the Embedded Systems with the AltaRica Approach	83
Pierre BIEBER and Christel SEGUIN	
3.1. Introduction	83
3.2. Safety analysis of embedded systems	83
3.3. AltaRica language and tools	85
3.3.1. The AltaRica language	85
3.3.2. Modeling the propagation of failures with AltaRica	92
3.3.3. Tools associated with AltaRica	95
3.4. Examples of modeling and safety analysis	99
3.4.1. Integrated modular avionics architecture	99
3.4.2. System of electric power generation and distribution	104
3.5. Comparison with other approaches	110
3.5.1. Some precursors	111
3.5.2. Tools for preexisting formal languages	111
3.5.3. Languages for physical systems	112
3.5.4. Injecting faults in nominal models	112

3.6. Conclusion	113
3.6.1. An approach to assess the safety of systems tested in aeronautics	113
3.6.2. Clarification of the system architecture and horizontal exploration of the failure propagation: impacts on the scope of analyses	113
3.6.3. Clarification of the nominal system characteristics: impacts on the generic definitions of the failure modes	115
3.6.4. Compositional models of failure propagation: impacts on the overall safety process	116
3.7. Special thanks	117
3.8. Bibliography	118
Chapter 4. Polyspace®	123
Patrick MUNIER	
4.1. Overview	123
4.2. Introduction to software quality and verification procedures	124
4.3. Static analysis	126
4.4. Dynamic tests	126
4.5. Abstract interpretation	127
4.6. Code verification	127
4.7. Robustness verification or contextual verification	131
4.7.1. Robustness verification	132
4.7.2. Contextual verification	132
4.8. Examples of Polyspace® results	133
4.8.1. Example of safe code	133
4.8.2. Example: de-referencing of a pointer outside its bounds	135
4.8.3. Example: inter-procedural calls	136
4.9. Carrying out a code verification with Polyspace®	138
4.10. Use of Polyspace® can improve the quality of embedded software	140
4.10.1. Begin by establishing models and objectives for software quality	141
4.10.2. Example of a software quality model with objectives	141
4.10.3. Use of a subset of languages to satisfy coding rules	142
4.10.4. Use of Polyspace® to reach software quality objectives	143
4.11. Carrying out certification with Polyspace®	145
4.12. The creation of critical onboard software	146
4.13. Concrete uses of Polyspace®	146
4.13.1. Automobile: Cummins engines improve the reliability of their motors' controllers	147

4.13.2. Aerospace: EADS guarantees the reliability of satellite launches	148
4.13.3. Medical devices: a code analysis leads to a recall of the device	149
4.13.4. Other examples of the use of Polyspace®	150
4.14. Conclusion	152
4.15. Bibliography	152
Chapter 5. Escher Verification Studio Perfect Developer and Escher C Verifier	155
Judith CARLTON and David CROCKER	
5.1. Introduction	155
5.1.1. Escher Technologies Ltd	155
5.1.2. Needs	156
5.2. Perfect Developer – its inspiration and foundations	156
5.2.1. Design-by-Contract	156
5.2.2. Verified Design-by-Contract	157
5.2.3. Perfect Developer	158
5.3. Theoretical foundations	159
5.4. The <i>Perfect</i> language	160
5.5. A Perfect Developer example	161
5.5.1. The specification	161
5.5.2. Verification conditions generated when the unrefined ring buffer is verified	164
5.5.3. Refining to a ring buffer	165
5.5.4. Verification conditions generated when the refined ring buffer is verified	168
5.6. Escher C verifier	168
5.7. The C subset supported by eCv	169
5.8. The annotation language of eCv	169
5.8.1. Applying verified design-by-contract to C	170
5.8.2. Arrays and pointers	171
5.8.3. Unions	172
5.8.4. Side effects	172
5.8.5. Quantifiers	173
5.9. Escher C verifier examples	173
5.9.1. First escher C verifier example	173
5.9.2. Second example	175
5.10. The theorem prover used by Perfect Developer and Escher C verifier	180
5.10.1. Logic	180
5.10.2. Term rewriter	180

5.11. Real-world applications of Perfect Developer and Escher C verifier	181
5.11.1. Safety-critical systems	181
5.11.2. Teaching university classes	182
5.11.3. IT system development: Precision Design Technology Ltd	182
5.12. Future work	188
5.12.1. Composite projects	188
5.12.2. Code generation	188
5.12.3. Concurrency	189
5.12.4. Extension	189
5.12.5. Conclusion	189
5.13. Glossary	190
5.14. Bibliography	191
Chapter 6. Partial Applications of Formal Methods	195
Aryldo G. RUSSO Jr.	
6.1. History	195
6.1.1. Context	196
6.2. Case studies	196
6.2.1. Early 2007 – the B method and railway domain – breaking the wall	196
6.2.2. Early 2008 – requirements verification	197
6.2.3. 2009 – tool comparison	201
6.2.4. Late 2009 – writing a formal specification – user point of view	202
6.2.5. Early 2010 – a methodological approach to a B formalization	205
6.2.6. Early 2011 changing the way to vital verification	209
6.2.7. The VeRaSiS plug-in	211
6.2.8. Results	211
6.3. Conclusion	212
6.4. Bibliography	213
Chapter 7. Event-B and Rodin	215
Michael BUTLER, Asieh SALEHI FATHABADI and Renato SILVA	
7.1. Event-B	215
7.1.1. The Event-B definition	215
7.1.2. Event-B structure and notation	216
7.1.3. Refinement in Event-B	218
7.1.4. Proof obligations	220
7.1.5. A comparison between Event-B and other formal methods	222

7.2. Rodin as an Event-B tool	222
7.3. Event-B model decomposition	223
7.3.1. Overview	223
7.3.2. Decomposition styles	223
7.4. Case study: metro system	225
7.4.1. Overview of the safety-critical metro system	226
7.4.2. Abstract model: MetroSystem_M0	228
7.4.3. First refinement: MetroSystem_M1	233
7.4.4. Second refinement: MetroSystem_M2	234
7.4.5. Third refinement and first decomposition: MetroSystem_M3	236
7.4.6. Machine Track	237
7.4.7. Machine Train	239
7.4.8. Machine Middleware	239
7.4.9. Refinement of Train: Train_M1	239
7.4.10. Further development	243
7.4.11. Conclusion	243
7.5. Acknowledgments	244
7.6. Bibliography	244
Chapter 8. Conclusion	247
Jean-Louis BOULANGER	
8.1. Introduction	247
8.2. Structured, semi-formal and/or formal methods	248
8.2.1. E/E/PE system	248
8.2.2. Rail sector	249
8.2.3. Taking into account techniques and formal methods	252
8.2.4. Software requirement specification	254
8.2.5. Coding	266
8.2.6. Verification and validation (V&V)	278
8.3. Conclusion	280
8.4. Bibliography	282
Glossary	287
List of Authors	293
Index	295