



Dick Hamlet

Composing Software Components

A Software-testing
Perspective

 Springer

Contents

The Griffin	vii
Acknowledgements	ix
1 Introduction	1
1.1 A ‘Clear Drop’	2
1.1.1 Testing Components and Recording Approximations	2
1.1.2 Synthesizing a System	5
1.1.3 Discussion of the Example	6
1.2 Roadmap of this Monograph	8
1.2.1 Theory vs. Practice	9
1.2.2 Formal Theory of Software Testing	10
1.2.3 Exploratory Tools	11
1.2.4 Insights into Component Composition	11
1.2.5 Implications for Testing in General	12
 Part I Components and Component-based Development	
2 Engineering, Components, and Software	17
2.1 Standardized Components Make Engineering Possible	17
2.2 Mechanical Engineering of a Vacuum System	19
2.3 Electrical/Computer Engineering of a Laptop	21
2.4 Can It Be Done with Software?	23
3 Software Components and Component-based Development	29
3.1 The Parts: Components	30
3.1.1 Common Environment for Software	30
3.1.2 Reuse	31
3.1.3 Information Hiding	32
3.1.4 Object-oriented Design	32
3.1.5 Szyperski’s Definition	33
3.2 The Systems: Component-based Software Development (CBSD) ...	34

3.2.1	Product Families	36
3.2.2	Component Development and Cataloging	36
3.2.3	System Design using Components	38
3.3	The Viewpoint: Testing Simple Components and Systems	38
3.3.1	Simple Components	38
3.3.2	Simple Systems	39
3.3.3	<i>Critique of the Model</i>	39
4	CBSD in Practice and Theory	41
4.1	Components and Connectors	41
4.2	System Architecture	43
4.3	Component Models	44
4.3.1	Middleware and Container Services	44
4.4	Immutable Components	45
4.5	Broader Theory of CBSD	46
4.5.1	General Component-based System Design Theory	46
4.5.2	Component-based Verification	48
4.5.3	Testing vs. Proving	49
4.6	Summary of CBSD Issues	50
 Part II Software Testing: Practice and Theory		
5	Software Testing's Place in Development	55
5.1	'Lifecycle' Models of Development	56
5.1.1	Development Phases	56
5.1.2	Waterfall Models	57
5.1.3	Agile Models	58
5.1.4	Which Model is Best?	59
5.2	Functional/Requirements Testing	60
5.2.1	Unit Testing vs. System Testing	60
5.3	Preventing Bugs	61
5.3.1	Software Inspection	61
5.3.2	Formal Methods	62
5.3.3	<i>Creating Perfection vs. Finding Failure</i>	63
5.4	Testing in CBSD	63
6	Software Testing Theory	65
6.1	Floyd-Hoare-Mills Semantics	65
6.2	Functional Testing Theory	67
6.2.1	Functional Testing Theory without State	67
6.2.2	Extending Functional Theory to Include State	69
6.2.3	Testing Concurrent Software	72
6.3	<i>Summary of Testing Theory</i>	75

7	Subdomain Testing	77
7.1	Divide and Conquer (or Multiply and Founder?)	78
7.2	History of ‘Coverage’ Testing	79
7.2.1	Functional Coverage	80
7.2.2	Structural Coverage	81
7.2.3	Combining Functional and Structural Coverage	85
7.3	Usage Profiles	86
7.4	Subdomain Testing in the Presence of State	88
7.5	Concurrency	91
7.6	Comparing Subdomain Techniques	92
7.6.1	The ‘Subsumes’ Partial Ordering	92
7.6.2	Random Testing	93
7.6.3	Comparing Random- and Subdomain-testing	94

Part III Composition of Components

8	Subdomain Theory of Stateless Component Composition	97
8.1	Software Testing is ‘Non-compositional’	97
8.2	Approximating and Measuring Component Properties	99
8.3	Calculating Properties of Stateless Systems	101
8.3.1	Series System	102
8.3.2	Conditional System Control Structure	105
8.3.3	Iterative System Control Structure	106
8.4	Combining Different Component Approximations	107
8.5	Synthesizing a Component-based System	108
8.5.1	Combining Testing and Proving	108
8.6	Summary of the Subdomain Testing Theory	110
9	Tutorial Example – SYN Tools for Stateless Components	111
9.1	Getting Started	112
9.2	A Simple Complete Example	113
9.3	Approximation and Prediction Errors	124
9.4	Better Component Approximations	128
9.4.1	Splitting and Adjusting Subdomains	130
9.4.2	Piecewise-linear Component Approximation	130
9.4.3	How Well Can a Component Developer Do?	133
9.5	Internal Profiles	134
9.6	Incremental Processing	137
9.7	Tutorial Summary	138
10	Persistent State	139
10.1	Extended Subdomain Theory of Composition	139
10.1.1	Algorithms for Synthesizing Systems from Components with State	140
10.1.2	Verifying the Algorithms	144
10.2	Testing Measurements	145

10.2.1	3-D Graphs and Approximation Errors	145
10.2.2	Equi-spaced vs. Sequence Sampling	146
10.3	System Predictions	147
10.3.1	Synthesis of Systems with State	148
10.4	A Tutorial Example with State	150
10.4.1	Tutorial: Modes (Preferences)	151
11	Concurrent Execution	167
11.1	Adding Concurrency to Composition Theory	167
11.1.1	Algorithm for Synthesizing Components in Parallel	168
11.2	Testing Measurements, Behavior Graphs, and System Predictions	169
11.3	A Tutorial Example with Concurrency	170
11.3.1	Tutorial: Multiversion Software	171
12	The Other Non-functional Property: Reliability	179
12.1	Reliability in Other Engineering Disciplines	179
12.2	Software Reliability Theory	181
12.2.1	Software ‘Time’ Parameter	182
12.2.2	The Minefield Analogy	182
12.2.3	A Speculative Software Failure Rate	184
12.2.4	Measuring Software Failure Rate	188
12.2.5	Failure Rate in Subdomains	191
12.3	Component Independence	192
12.4	Reliability Synthesis	193
12.4.1	Difficulties in Component Measurements	194
12.4.2	Synthesis Rules	194
Part IV Supporting Tools		
13	CBSD Support Tools	201
13.1	Component Developers and System Designers	201
13.2	Ideal Tools for I-CBSD	202
13.2.1	Ideal Component-level Testing/Measurement Tools	202
13.2.2	Ideal System-level Synthesis (CAD) Tools	203
13.2.3	SYN Tools: An Existence Proof	203
14	Tool Implementation	205
14.1	Component Conventions	207
14.1.1	Artificial Components	208
14.2	Underlying Algorithms	209
14.3	Execution by Table-lookup	210
14.3.1	Validating Tools	211
14.3.2	A Nasty Mistake	213
14.4	Tool Performance	216

15 Debugging Components, Component-based Systems, and Support Tools 219

15.1 Debugging Components 219

 15.1.1 Checking Tests Against Requirements 220

 15.1.2 Executing Code Outside the SYN Tools 221

 15.1.3 Finding Good Subdomains 221

 15.1.4 Graphical Aids 222

15.2 Debugging Component-based Systems 222

 15.2.1 Component Mismatch 225

 15.2.2 Interface Profiles 227

15.3 Debugging of Support Tools 228

 15.3.1 Problem Decomposition 228

 15.3.2 Iterative Enhancement 229

 15.3.3 Components and Debugging of SYN Tools 230

16 Unfinished Business: Volunteer Tool Makers 233

16.1 Unstable Algorithms and Code 233

16.2 Improving the SYN Tools 234

16.3 Who’s Next? 236

Part V Case Studies

17 Accuracy of Component Measurements and System Predictions 239

17.1 Better Component Approximation, Better System Prediction 239

 17.1.1 Tuning Subdomains with Tool Support 244

17.2 Predicting Prediction Accuracy 247

 17.2.1 Prediction Error is Linear in Measurement Error 247

 17.2.2 Theory of Error Propagation 248

 17.2.3 Prediction Error is an Emergent System Property 251

 17.2.4 Approximating System Prediction Errors 252

17.3 Approximation Accuracy as a Component Test-quality Metric 253

17.4 The Right Subdomains for Component Testing 254

18 Case Studies of I-CBSD 257

18.1 Fundamental Questions about Subdomain Testing 257

 18.1.1 How to Sample Subdomains? 258

 18.1.2 Is Series Synthesis Associative? 261

18.2 Moving Control Structures between Components and Systems 262

 18.2.1 Series Composition within a Component 263

 18.2.2 Conditionals in Code and Conditional Components 264

 18.2.3 Raising the Level of Programming 266

18.3 Persistent State 266

 18.3.1 Infeasible States 266

 18.3.2 Modes and Storage in State 270

 18.3.3 A Controlled ‘Editor’ System 277

18.4 Iteration at System Level 284

18.5	Component and System Reliability	288
18.6	Substituting one Component for Another	292
18.6.1	Meeting a Non-functional System Requirement Bound	293

Part VI Implications for Software Testing

19	Unit vs. System Testing	299
19.1	Components Make Ideal Software ‘Units’	300
19.1.1	Solving Unit-testing Problems	301
19.1.2	Choosing Unit-test Subdomains	303
19.2	Unit Testing Is More Than it Seems	305
19.2.1	Saving and Using Unit-test Results	305
19.2.2	Unit Tests the Only Tests	305
19.3	Trusting Unit Tests	307
19.3.1	Trustworthy Component Testing	307
19.3.2	Matching Interface Profiles	307
19.4	Comparing System Predictions to Requirements	308
20	Functional vs. Non-functional Properties	311
20.1	Non-functional Depends on Functional	311
20.2	Non-functional ‘Compositional’ Properties	312
20.2.1	Run Time	313
20.2.2	Reliability	313
20.2.3	Safety Factors and Prediction Accuracy	314
20.3	Predicting Emergent Properties	315
20.3.1	Memory Leaks	315
20.3.2	Security	316
20.3.3	‘Emergent’ Prediction Error	317
21	Conclusion: Lessons Learned from I-CBSD	319
21.1	Software Components are Unlike Mechanical Components	319
21.2	Software Functions Are Inherently Discontinuous	320
21.2.1	Simple Component Behaviors Lead to Complicated System Behaviors	322
21.3	Testing Theory is Unlike Other Formal Methods	323
21.3.1	Conservative Reductions	323
21.3.2	Special Role of Persistent State	324
21.4	The Several Meanings of ‘Compositional’	325
21.4.1	Compositional Properties	326
21.4.2	Testing Can Be Made Compositional	326
21.5	Simple Tools are Remarkably Powerful	327

22	Open Problems	331
22.1	Subdomain Testing in Non-numeric Domains	331
22.2	Completing a Testing Theory including State	332
22.2.1	Reliability in the Presence of State	332
22.2.2	Better SYN Tools for State	333
22.3	Limited Input Domain	333
	References	335
	Appendix	
A	Tool Specifications	343
A.1	Documentation	344
A.2	SYN Documentation Tricks	345
A.2.1	Stand-alone Script Execution	346
A.2.2	Error Messages	346
A.2.3	'Message-discovery' Documentation	348
A.2.4	'#debug' Statements	349
A.2.5	The Script Header Comments	349
A.3	Details of the Tool Scripts	349
A.3.1	File Formats	350
A.3.2	Testing and Approximating Components: COMP and friends	353
A.3.3	Synthesizing and Predicting Systems: SYN and Calc	354
A.3.4	Auxiliary Scripts	361
	Index	363