

---

# Table of Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Software Architecture . . . . .	1
1.2	Sustainability . . . . .	3
1.3	Technical Debt . . . . .	5
1.3.1	No Knowledge of Software Architecture . . . . .	8
1.3.2	Complexity and Size . . . . .	9
1.3.3	Architectural Erosion Takes Place Unnoticed . . . . .	11
1.3.4	We Don't Pay Extra For Quality! . . . . .	12
1.3.5	Types of Technical Debt . . . . .	13
1.4	The Systems I Have Seen . . . . .	14
1.5	Who Is This Book For? . . . . .	15
1.6	How To Use This Book? . . . . .	16
<b>2</b>	<b>Tracking Down Technical Debt</b>	<b>19</b>
2.1	Building Block Terminology . . . . .	19
2.2	Target and Actual Architecture . . . . .	21
2.3	Improvements to a Running System . . . . .	25
2.4	False Positives and Generated Code . . . . .	43
2.5	Cheat Sheet for Sotograph . . . . .	46
<b>3</b>	<b>Architecture in Programming Languages</b>	<b>47</b>
3.1	Java Systems. . . . .	47
3.2	C# Systems. . . . .	52
3.3	C++ Systems. . . . .	54
3.4	ABAP Systems . . . . .	56
3.5	PHP Systems . . . . .	57

<b>4</b>	<b>Architecture Analysis and Improvement</b>	<b>61</b>
4.1	Developers and Architects	61
4.2	Working on Architecture is a “Holschuld”	62
4.3	Live Architecture Improvement Workshop	63
4.4	Dealing with Mothers and Fathers	65
4.5	Modularity Maturity Index (MMI)	67
4.6	Technical Debt in the Lifecycle	69
<b>5</b>	<b>Cognitive Psychology and Architectural Principles</b>	<b>73</b>
5.1	Modularity	74
5.1.1	Chunking	74
5.1.2	Transfer to Design Principles	76
5.1.2.1	Units	78
5.1.2.2	Public Interface	79
5.1.2.3	Coupling	81
5.2	Pattern Consistency	82
5.2.1	Establishing Schemata	82
5.2.2	Transfer to Design Principles	84
5.3	Hierarchy	89
5.3.1	Formation of Hierarchies	89
5.3.2	Transfer to Design Principles	91
5.4	Cycles = Failed modularity + Pattern	93
5.5	Consequences for Architectural Analysis	93
<b>6</b>	<b>Architectural Styles that Reduce Technical Debt</b>	<b>97</b>
6.1	Rules of Architectural Styles	97
6.2	Separation of Business and Technical Building Blocks	98
6.3	Layered Architecture	101
6.3.1	Technical Layering	101
6.3.2	Domain Layering	103
6.3.3	The Infrastructure Layer	104
6.3.4	Integration of Domain-oriented Layers	106
6.4	Hexagonal, Onion, and Clean Architecture	107
6.5	Microservices and Domain-Driven Design	109
6.6	Pattern Languages	112
6.6.1	The Tool & Material Pattern Language	114
6.6.2	The DDD Pattern Language	117
6.6.3	Typical Framework Patterns	119
6.7	Sustainability and Architectural Styles	120

---

<b>7</b>	<b>Pattern in Software Architecture</b>	<b>121</b>
7.1	Mapping the Target to the Actual Architecture. . . . .	121
7.2	The Ideal Structure: Domain-Oriented or Technical? . . . . .	124
7.3	Public Interfaces for Building Block . . . . .	130
7.4	Interfaces: The Architectural Miracle Cure? . . . . .	134
7.4.1	Basic Therapy . . . . .	135
7.4.2	Side Effects . . . . .	137
7.4.3	Field Studies on “Living Patients” . . . . .	140
7.4.4	Fighting the Monolith . . . . .	143
7.5	The Need for Microservices . . . . .	145
<b>8</b>	<b>Pattern Languages: A True Architectural Treasure!</b>	<b>149</b>
8.1	The Treasure Hunt . . . . .	149
8.2	Software Archaeology . . . . .	151
8.3	From the Treasure Chest . . . . .	153
8.4	How Much Gold Is There? . . . . .	157
8.5	Annual Growth Rings . . . . .	158
8.6	Unclear Patterns Provoke Cycles . . . . .	159
<b>9</b>	<b>Chaos Within Layers: The Daily Pain</b>	<b>163</b>
9.1	Evaluating the Mess . . . . .	166
9.1.1	The Extent of the Chaos . . . . .	166
9.1.1.1	Architectural Styles and Cycles . . . . .	168
9.1.1.2	Lines of Code in Cycles . . . . .	169
9.1.1.3	Dependency Injection and Cycles . . . . .	171
9.1.2	Scope and Interconnectedness . . . . .	171
9.1.3	Cycle Range Within an Architecture . . . . .	174
9.2	The Big Ball of Mud . . . . .	179
9.2.1	The Black Hole Effect . . . . .	180
9.2.2	Breaking Free . . . . .	183
9.2.3	“Weaponizing” Technical Layering . . . . .	184
9.2.4	Pattern Language as a Lighthouse . . . . .	186
9.3	Uneven Modules . . . . .	190

<b>10 Refining Modularity</b>	<b>193</b>
10.1 Building Block Cohesion . . . . .	194
10.2 Building Block Sizes . . . . .	198
10.3 Class Sizes . . . . .	198
10.4 Method Size and Complexity . . . . .	204
10.5 Loose Coupling . . . . .	206
10.6 Coupling and Class Size . . . . .	213
10.7 How Modular Are You? . . . . .	215
<b>11 Real-World Case Studies</b>	<b>217</b>
11.1 The Java System <i>Alpha</i> . . . . .	218
11.2 The C# System <i>Gamma</i> . . . . .	225
11.3 The C++ System <i>Beta</i> . . . . .	234
11.4 The Java System <i>Delta</i> . . . . .	243
11.5 The Java System <i>Epsilon</i> (with C# Satellites) . . . . .	250
11.5.1 Java-Epsilon . . . . .	250
11.5.2 C#- <i>Epsilon</i> 1 . . . . .	258
11.5.3 C#- <i>Epsilon</i> 2 . . . . .	262
11.6 The ABAP system Lambda . . . . .	266
<b>12 Conclusion: The Path Toward Sustainable Architecture</b>	<b>273</b>
<b>Appendix</b>	<b>277</b>
<b>A Analysis Tools</b>	<b>279</b>
A.1 Lattix . . . . .	281
A.2 Sonargraph Architect . . . . .	282
A.3 Sotograph and SotoArc . . . . .	284
A.4 Structure101 . . . . .	285
<b>References</b>	<b>289</b>
<b>Index</b>	<b>295</b>