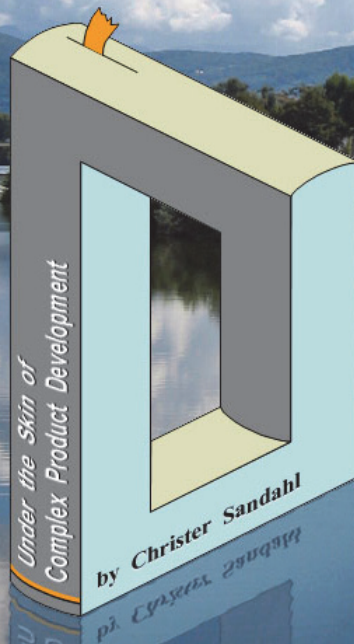

Complex Product Development and its Management



by Christer Sandahl

This Book

Dedicated to my dear family,
for taking proper care of me,
when I fade into book coma.

To be printed and marketed by AuthorHouse

Revision of 23 May 2011

All rights reserved.
Christer Sandahl

Overview Table of Content

Overall CHAPTER 1	<i>Warning Bells</i>	23
Overall CHAPTER 2	<i>Product Development Overview</i>	37
Overall CHAPTER 3	<i>Value Chain</i>	63
Overall CHAPTER 4	<i>Optimise Concurrency</i>	77
Overall CHAPTER 5	<i>Portfolio & platform</i>	105
Detail CHAPTER 6	<i>Requirements</i>	109
Detail CHAPTER 7	<i>Architectures</i>	151
Detail CHAPTER 8	<i>Finalize design</i>	195
Detail CHAPTER 9	<i>Integration</i>	199
Detail CHAPTER 10	<i>Verification and Validation</i>	203
Unite CHAPTER 11	<i>Line Management</i>	207
Unite CHAPTER 12	<i>Interfering Operations</i>	225
Unite CHAPTER 13	<i>Balancing Players</i>	227
Unite CHAPTER 14	<i>Configuration control</i>	231
Meta CHAPTER 15	<i>Information Management</i>	243
Meta CHAPTER 16	<i>Quality System</i>	245
Meta CHAPTER 17	<i>Process & methodology</i>	247
Meta CHAPTER 18	<i>Improvement & Assessment</i>	251

Detailed Table of Content

Overall CHAPTER 1 *Warning Bells* **23**

- 1.1 What has gone wrong ? **24**
 - 1.1.1 *Success from the low complexity world* **24**
 - 1.1.2 *Success from the high complexity world* **25**
 - 1.1.3 *Transition from low complexity to high* **25**
 - 1.1.4 *Is there any trick with scaling up ?* **25**
 - 1.1.5 *Short about complexity* **26**
- 1.2 When should you hear the warning bell ? **27**
 - 1.2.1 *EXAMPLE: Unrealistic campaigns continually restarted* **28**
 - 1.2.2 *EXAMPLE: Management not being accountable* **28**
 - 1.2.3 *EXAMPLE: Root cause analysis being suppressed* **29**
 - 1.2.4 *EXAMPLE: New wrapping with same content* **30**
 - 1.2.5 *EXAMPLE: Ego people being change agents* **30**
 - 1.2.6 *EXAMPLE: Sub optimization within local organizations* **30**
 - 1.2.7 *EXAMPLE: Scapegoats of a blame game* **31**
 - 1.2.8 *EXAMPLE: Products driven by engineers* **31**
 - 1.2.9 *EXAMPLE: Product structure being degenerated* **32**
 - 1.2.10 *EXAMPLE: Impossible principles applied* **32**
 - 1.2.11 *EXAMPLE: Artificial complexity being pushed* **33**
 - 1.2.12 *EXAMPLE: Intrinsic complexity being ignored* **33**
 - 1.2.13 *EXAMPLE: Devils in the details being ignored* **34**
- 1.3 Your way out of this **34**

Overall CHAPTER 2 *Product Development Overview* **37**

- 2.1 The product **38**
 - 2.1.1 *About products* **38**
 - 2.1.2 *EXAMPLE: chairs development* **38**
- 2.2 Product life cycle (PLC) **38**
 - 2.2.1 *About product life cycle* **38**
 - 2.2.2 *EXAMPLE: chairs life cycle* **39**
- 2.3 Product development (PD) **40**
 - 2.3.1 *About product development* **40**
 - 2.3.2 *EXAMPLE: chairs product development* **40**

2.4 Strategic study	41
2.4.1 <i>About road maps</i>	41
2.4.2 <i>EXAMPLE: chairs strategic study</i>	41
2.5 Concept study	42
2.5.1 <i>About concept studies</i>	42
2.5.2 <i>EXAMPLE: chairs concept brief</i>	42
2.6 Portfolio management	43
2.6.1 <i>About product portfolio management</i>	43
2.6.2 <i>EXAMPLE: chairs portfolio management</i>	43
2.6.3 <i>About reuse planning</i>	44
2.6.4 <i>EXAMPLE: chairs reuse planning</i>	44
2.7 Business study	44
2.7.1 <i>About business cases</i>	44
2.7.2 <i>EXAMPLE: chairs business case</i>	45
2.7.3 <i>About business gates (BG)</i>	45
2.7.4 <i>EXAMPLE: business gate number 1 (BG1)</i>	45
2.8 Prototyping and customer feedback	46
2.8.1 <i>About prototyping</i>	46
2.8.2 <i>EXAMPLE: chairs prototyping</i>	46
2.8.3 <i>About customer requirements</i>	46
2.8.4 <i>EXAMPLE: customer feedback</i>	46
2.9 Requirement management	47
2.9.1 <i>About requirement management</i>	47
2.9.2 <i>EXAMPLE: chairs requirement management</i>	47
2.9.3 <i>Test cases</i>	48
2.9.4 <i>EXAMPLE: chairs test cases</i>	48
2.10 Designing a system	49
2.10.1 <i>About architectures</i>	49
2.10.2 <i>EXAMPLE: chairs architecture</i>	50
2.10.3 <i>System decomposition</i>	50
2.10.4 <i>EXAMPLE: chairs component list</i>	50
2.10.5 <i>Interface descriptions</i>	51
2.10.6 <i>EXAMPLE: chairs component interface</i>	51
2.11 Detailing	52
2.11.1 <i>EXAMPLE: chairs detailing</i>	53
2.12 Product propositions and promotion	53
2.12.1 <i>EXAMPLE: chairs product planning</i>	53
2.12.2 <i>About promotion campaign</i>	54
2.12.3 <i>EXAMPLE: chairs promotion campaign</i>	54
2.12.4 <i>EXAMPLE: business gate number 2 (BG2)</i>	54
2.13 Material supply	55

2.13.1	<i>About material and supply</i>	55
2.13.2	<i>About bill of material</i>	55
2.13.3	<i>Cost down activities</i>	55
2.13.4	<i>EXAMPLE: chair material supply</i>	56
2.14	Integration, verification and validation	57
2.14.1	<i>About integration</i>	57
2.14.2	<i>EXAMPLE: chairs integration</i>	57
2.14.3	<i>EXAMPLE: chairs verification</i>	57
2.14.4	<i>About validation and β-test</i>	57
2.14.5	<i>EXAMPLE: chairs validation</i>	57
2.14.6	<i>EXAMPLE: Business gate 3 (BG3)</i>	58
2.15	Ramp up and Launch	58
2.15.1	<i>About ramp up and launch</i>	58
2.15.2	<i>EXAMPLE: chairs ramp up and market launch</i>	58
2.16	The motor chairs	58
2.16.1	<i>Organising the new development</i>	59
2.16.2	<i>Modules with different lead times</i>	60
2.16.3	<i>Extended architecture</i>	61

Overall CHAPTER 3 *Value Chain* **63**

3.1	General about value chain	64
3.1.1	<i>EXAMPLE: value chains</i>	64
3.1.2	<i>Value chain appearance</i>	64
3.2	Value chain basics	65
3.2.1	<i>Short about flows</i>	65
3.2.2	<i>The value chain is a kind of executable flow</i>	65
3.2.3	<i>Total value chain</i>	66
3.2.4	<i>Value added</i>	67
3.3	Chair development value chain	67
3.3.1	<i>EXAMPLE: chair development basic value chain</i>	67
3.3.2	<i>EXAMPLE: chair development activity flow</i>	68
3.4	Results	69
3.4.1	<i>Short about results</i>	69
3.4.2	<i>EXAMPLE: chair development activities including results</i>	69
3.5	Result orientation	72
3.5.1	<i>EXAMPLE: chair development result dependences</i>	72
3.5.2	<i>Life cycle status of results</i>	73
3.6	Building blocks in value chains	73

- 3.6.1 Activity-activity linkage 73
- 3.6.2 Activity-result alteration 74
- 3.6.3 Result-result dependencies 75
- 3.6.4 State machine 75
- 3.6.5 Executor and producer roles 76

Overall CHAPTER 4 *Optimise Concurrency* 77

- 4.1 Sequential Waterfall 78
- 4.2 Flow Parallelism 78
 - 4.2.1 Development example series of parallel patterns 79
 - 4.2.2 Layout of the following series of examples 79
- 4.3 Patterns of basic development 80
 - 4.3.1 EXAMPLE: basic waterfall 81
 - 4.3.2 EXAMPLE: trying parallel mania 81
 - 4.3.3 EXAMPLE: trying moderate parallelism 83
 - 4.3.4 Patterns of Parallelism 84
 - 4.3.5 EXAMPLE: trying late requirements 84
 - 4.3.6 EXAMPLE: prototyping 86
 - 4.3.7 EXAMPLE: double staffing 87
 - 4.3.8 EXAMPLE: iterative development 87
 - 4.3.9 Patterns of iterative development 89
 - 4.3.10 EXAMPLE: system decomposition 89
 - 4.3.11 Generic layered V-model 90
 - 4.3.12 EXAMPLE: iterative decomposition 91
 - 4.3.13 EXAMPLE: big bang integration of system modules 91
- 4.4 Patterns of feature development 92
 - 4.4.1 EXAMPLE: one single feature extension 93
 - 4.4.2 EXAMPLE: feature parallel mania 93
 - 4.4.3 EXAMPLE: two feature extension 94
 - 4.4.4 EXAMPLE: two feature extension on isolated branches 95
 - 4.4.5 EXAMPLE: four features extension 96
 - 4.4.6 EXAMPLE: four features extending with one single system verification 97
 - 4.4.7 EXAMPLE: four features extending with automated system verification 97
- 4.5 Conclusion from series of examples 98
- 4.6 Critical path 99
 - 4.6.1 About lead time 99
 - 4.6.2 EXAMPLE: New Years rave. 100

4.6.3	<i>Counting backwards to identify flow ?</i>	101
4.6.4	<i>EXAMPLE: restaurant management</i>	101
4.6.5	<i>EXAMPLE: prefabricates in restaurants</i>	102
4.6.6	<i>EXAMPLE: late requirements revisited</i>	103

Overall CHAPTER 5 *Portfolio & platform* 105

5.1	Forerunner or follower	106
5.1.1	<i>Fashion analogy</i>	106
5.1.2	<i>Time to market</i>	106
5.1.3	<i>Hygiene factor versus differentiator</i>	106
5.1.4	<i>Make/buy analysis</i>	106
5.2	Product paradigm shift	106
5.2.1	<i>EXAMPLE: mechanical versus electronic calculator</i>	106
5.2.2	<i>EXAMPLE: graphical interface in computer and telephone</i>	106
5.3	Portfolio	106
5.3.1	<i>Diversity of products</i>	107
5.3.2	<i>Cannibalism</i>	107
5.4	Platforms	107
5.4.1	<i>Platform basics</i>	107
5.4.2	<i>Building all products in parallel</i>	107
5.4.3	<i>Implementing platform products gradually</i>	107
5.4.4	<i>Most advanced product drive the platform</i>	107
5.5	Reuse	107
5.5.1	<i>Ad hoc</i>	107
5.5.2	<i>On opportunity</i>	107
5.5.3	<i>Planned in advance</i>	107
5.5.4	<i>Granularity of reuse</i>	107
5.6	Product line (Platform)	108
5.6.1	<i>Product portfolio</i>	108
5.6.2	<i>Variability versus flexibility</i>	108

Detail CHAPTER 6 *Requirements* 109

6.1	Requirements	110
6.1.1	<i>Purpose</i>	110
6.1.2	<i>Restrictions</i>	111
6.1.3	<i>System hierarchies</i>	111
6.1.4	<i>Requirements connection to architecture and vice versa</i>	111

6.1.5	<i>Requirements traceable to each other</i>	111
6.2	Constitute restrictions	112
6.2.1	<i>Value chain</i>	112
6.2.2	<i>Capture environment restrictions</i>	112
6.2.3	<i>EXAMPLE Capture environment and house regulations</i>	112
6.2.4	<i>Environment design make/buy analysis</i>	113
6.2.5	<i>EXAMPLE: Environment design make/buy analysis</i>	113
6.2.6	<i>Hand over to "Constitute environment"</i>	114
6.3	Refine restrictions and specify top-system	114
6.3.1	<i>Top-system value chain</i>	114
6.3.2	<i>Reconcile restrictions against environment architecture</i>	115
6.3.3	<i>EXAMPLE: Reconcile house regulations against environment</i>	115
6.3.4	<i>Capture top-system requirements</i>	116
6.3.5	<i>EXAMPLE: Capture house requirements</i>	116
6.3.6	<i>Prioritize and consolidate top-system requirements</i>	118
6.3.7	<i>EXAMPLE: Prioritize and consolidate house requirements</i>	118
6.3.8	<i>Top-system design make/buy analysis</i>	118
6.3.9	<i>EXAMPLE: House design make/buy analysis</i>	120
6.3.10	<i>Hand over to "Decompose top-system and design its elements"</i>	120
6.4	Refine top-system and specify mid-systems	121
6.4.1	<i>Mid-system value chain</i>	121
6.4.2	<i>Reconcile top-system requirements against architecture</i>	121
6.4.3	<i>EXAMPLE: Reconcile house requirements against architecture</i>	122
6.4.4	<i>Capture mid-systems requirements</i>	122
6.4.5	<i>EXAMPLE: Capture kitchen requirements</i>	122
6.4.6	<i>Prioritize mid-system requirements</i>	124
6.4.7	<i>EXAMPLE: Prioritize room requirements</i>	124
6.4.8	<i>Mid-system design make/buy analysis</i>	124
6.4.9	<i>EXAMPLE: Room design make/buy analysis</i>	124
6.4.10	<i>Hand over to "Decompose mid-systems and design their elements"</i>	125
6.5	Refine mid-systems and specify bottom-systems	125
6.5.1	<i>Bottom-system value chain</i>	125
6.5.2	<i>Reconcile mid-system requirements against architecture</i>	125
6.5.3	<i>EXAMPLE: Reconcile rooms requirements against architecture</i>	125

6.5.4	<i>Capture bottom-systems requirements</i>	126
6.5.5	<i>EXAMPLE: Capture fixtures requirements</i>	126
6.5.6	<i>Prioritize bottom-system requirements</i>	128
6.5.7	<i>EXAMPLE: Prioritize machinery fixtures requirements</i>	128
6.5.8	<i>Bottom-system design make/buy analysis</i>	128
6.5.9	<i>EXAMPLE: Machinery fixtures design make/buy analysis</i>	128
6.5.10	<i>Hand over to "Decompose bottom-systems and design their elements"</i>	128
6.6	Refine bottom-systems	129
6.6.1	<i>Finalising value chain</i>	129
6.6.2	<i>Reconcile bottom-system requirements against architecture</i>	129
6.6.3	<i>EXAMPLE: Reconcile fixtures requirements against architecture</i>	129
6.6.4	<i>End of refine and decompose zig-zag</i>	129
6.7	Static and dynamic requirements	130
6.7.1	<i>Static requirements</i>	130
6.7.2	<i>Dynamic stimulus-response requirements</i>	131
6.7.3	<i>EXAMPLE: Multiplication table toy</i>	131
6.8	Use cases	133
6.8.1	<i>Dynamic textual use case requirements</i>	133
6.8.2	<i>EXAMPLE: Specify the Windows calculator</i>	133
6.8.3	<i>EXAMPLE: Calculator dual operand use case</i>	134
6.9	Function requirements	134
6.9.1	<i>Dynamic function requirements</i>	134
6.9.2	<i>EXAMPLE: Egg function</i>	136
6.10	Scenarios requirements	136
6.10.1	<i>EXAMPLE: Breakfast scenario</i>	137
6.10.2	<i>State machines</i>	138
6.10.3	<i>EXAMPLE Scenario state machine requirements</i>	139
6.10.4	<i>Functions penetrating architecture</i>	139
6.10.5	<i>EXAMPLE: Egg function penetrating architecture</i>	140
6.10.6	<i>Scenarios penetrating architectures</i>	142
6.10.7	<i>EXAMPLE: Calculator normal case scenario requirements</i>	142
6.10.8	<i>EXAMPLE: How many states ?</i>	144
6.10.9	<i>EXAMPLE Calculator full case requirements</i>	144
6.11.1	<i>EXAMPLE: Calculator requirements test case</i>	147
6.13	Road maps	148

6.15 Frequently asked questions about requirements	148
6.15.1 <i>Why black-boxes ?</i>	148
6.15.2 <i>EXAMPLE: Why not early open the house black-box ?</i>	149
6.15.3 <i>Isn't it too expensive to refine and specify requirements ?</i>	149
6.15.4 <i>EXAMPLE: Life cycle cost of an house</i>	149
6.15.5 <i>Isn't it delaying the development to refine and specify requirements ?</i>	149
6.15.6 <i>Who are requirement stakeholders ?</i>	149
6.15.7 <i>How is a durable prioritization made</i>	150
6.15.8 <i>Why not ask the developers to specify during developing ?</i>	150
6.15.9 <i>Traceability of requirements</i>	150
6.15.10 <i>Why top-down, and not bottom-up refinement ?</i>	150
6.15.11 <i>Is it smart to specify what a system should not do ?</i>	150
6.15.12 <i>Requirements on textual requirements</i>	150

Detail CHAPTER 7 *Architectures* 151

7.1.1 <i>Purpose</i>	152
7.1.2 <i>Black-boxes</i>	152
7.1.3 <i>White-boxes</i>	153
7.1.4 <i>Systems and architecture hierarchies</i>	153
7.1.5 <i>Views of architectures</i>	153
7.2 <i>Constitute environment</i>	154
7.2.1 <i>Value chain</i>	154
7.2.2 <i>Restrictions impact on environment design</i>	154
7.2.3 <i>EXAMPLE: House regulations impact on its environment design</i>	155
7.2.4 <i>Consolidate environment architecture</i>	156
7.2.5 <i>EXAMPLE: Environment architecture résumé</i>	156
7.2.6 <i>Identifying black-box in environment architecture résumé</i>	157
7.2.7 <i>EXAMPLE: Environment logical architecture drawing</i>	158
7.2.8 <i>Layout environment white-box</i>	158
7.2.9 <i>EXAMPLE: Environment physical architecture layout</i>	159
7.2.10 <i>Hand over to "Refine restrictions and specify top-system"</i>	159

7.3 Decompose top-system and design its elements 160

- 7.3.1 Value chain 160*
- 7.3.2 Top-system requirements impact on top-system design 160*
- 7.3.3 EXAMPLE: House requirements impact on top-system design 161*
- 7.3.4 Consolidate top-system design 162*
- 7.3.5 EXAMPLE: House architecture résumé 163*
- 7.3.6 Identify black-boxes in top-system architecture résumé 165*
- 7.3.7 EXAMPLE: House logical architecture drawing 165*
- 7.3.8 Layout top-system white-box architecture 166*
- 7.3.9 EXAMPLE: House physical architecture layout 167*
- 7.3.10 Hand over to "Refine top-system and specify mid-systems" 167*

7.4 Decompose mid-systems and design their elements 168

- 7.4.1 Value chain 168*
- 7.4.2 Mid-system requirements impact on mid-system design 168*
- 7.4.3 EXAMPLE: Room requirements impact on room design 169*
- 7.4.4 Consolidate mid-system architecture 170*
- 7.4.5 EXAMPLE: Kitchen architecture résumé 171*
- 7.4.6 Identifying black-boxes in mid-system architecture résumé 171*
- 7.4.7 EXAMPLE: Kitchen logical architecture drawing 171*
- 7.4.8 Layout mid-system white-box architecture 172*
- 7.4.9 EXAMPLE: Kitchen physical architecture layout 172*
- 7.4.10 Hand over to "Refine mid-systems and specify bottom-systems" 173*

7.5 Decompose bottom-systems and design their elements 174

- 7.5.1 Value chain 174*
- 7.5.2 Bottom-system requirements impact on bottom-system design 175*
- 7.5.3 EXAMPLE: Machinery fixtures requirements impact on fixtures design 175*
- 7.5.4 Consolidate bottom-system architecture 177*
- 7.5.5 EXAMPLE: Machinery fixtures architecture résumé 177*
- 7.5.6 Identifying black-boxes in bottom-system architecture résumé 178*
- 7.5.7 EXAMPLE: Logic machinery fixtures architecture 178*
- 7.5.8 Layout bottom-systems white-box architecture 179*
- 7.5.9 EXAMPLE: Fixtures physical architecture layout 179*

7.5.10	<i>Hand over to "Refine bottom-systems"</i>	180
7.6	The two-sided zig-zag coin	181
7.6.1	<i>EXAMPLE: House refining and decomposition zig-zag</i>	181
7.6.2	<i>General refining and decomposition zig-zag</i>	182
7.7	Generalizing the value chain	183
7.7.1	<i>EXAMPLE: What the house example didn't tell</i>	183
7.7.2	<i>Generic requirement and architecture value chain</i>	183
7.8	More architectures	184
7.8.1	<i>EXAMPLE: Multiplication table toy</i>	184
7.9	Requirements and architectures relationship	186
7.9.1	<i>Awkward relationship types</i>	186
7.9.2	<i>Module coupling</i>	186
7.9.3	<i>Module cohesion (or strength)</i>	186
7.9.4	<i>EXAMPLE: professions</i>	186
7.10	Architecture hierarchies	186
7.10.1	<i>Top-down versus bottom-up</i>	186
7.10.2	<i>What is great with black-boxes and interfaces</i>	187
7.10.3	<i>How to find the black-boxes</i>	187
7.10.4	<i>What decides the size of a black box</i>	187
7.10.5	<i>When to specify properties</i>	187
7.10.6	<i>Different decomposition depth</i>	187
7.10.7	<i>EXAMPLE: House logical box hierarchy</i>	187
7.11	The use of black-boxes	189
7.11.1	<i>Finding black-boxes</i>	189
7.11.2	<i>Aggregated properties and their summarization</i>	189
7.11.3	<i>Bill of material (aggregated price)</i>	189
7.11.4	<i>Decompose for reuse</i>	189
7.11.5	<i>Partition and aggregation</i>	189
7.12	Large and complex architectures	189
7.12.1	<i>Very different boxes inside each other</i>	189
7.12.2	<i>EXAMPLE: Complete motor chair architecture</i>	190
7.13	Architecture types	191
7.13.1	<i>Principles</i>	191
7.13.2	<i>EXAMPLE: OSI model</i>	191
7.13.3	<i>EXAMPLE: mobile phone</i>	191
7.14	Open versus embedded	191
7.14.1	<i>Extend before compile time</i>	191
7.14.2	<i>Extend before link time</i>	191
7.14.3	<i>Extend at run time</i>	191
7.14.4	<i>Prepare for specialization</i>	191

7.14.5	<i>EXAMPLE: Microsoft windows</i>	191
7.15	Transforming requirements to architecture	192
7.15.1	<i>Heuristic rather than deterministic</i>	192
7.15.2	<i>Iteration process</i>	192
7.16	Decomposition to Sub-systems	192
7.16.1	<i>Principle</i>	192
7.16.2	<i>Practical aspects</i>	192
7.17	Modularization	192
7.17.1	<i>Levels according to Myers 1975</i>	192
7.18	Components	192
7.19	Maintenance cost reduction	192
7.20	Property performance	192
7.21	Atom architecture level	193
7.22	Degeneration	193
7.22.1	<i>Extend a shanty town in width</i>	193
7.22.2	<i>Extend a shanty town in height</i>	193
7.23	Cost, quality, projecting estimations	193

Detail CHAPTER 8 *Finalize design* **195**

8.1	About finalizing design	196
8.2	Finalize the house	196
8.3	Finalizing the multiplication table tool	196
8.3.1	<i>By gate network</i>	196
8.4	Difference between sw and hw	196
8.4.1	<i>Visibility</i>	196
8.4.2	<i>Margin cost for an extension</i>	196
8.4.3	<i>Development versus production cost</i>	196
8.5	Finalizing the multiplication table tool	196
8.5.1	<i>Development environment</i>	196
8.5.2	<i>By single chip computer</i>	196
8.6	Finalizing the calculator	196
8.6.1	<i>By single chip computer</i>	196
8.6.2	<i>Interrupt</i>	196
8.6.3	<i>Real time processes</i>	196
8.6.4	<i>Operating system</i>	197
8.6.5	<i>By an application</i>	197
8.7	Experimental Prototypes	197

8.8 Software languages	197
8.8.1 Machine	197
8.8.2 Assembler	197
8.8.3 Structured	197
8.8.4 Object oriented	197
8.8.5 Code generators	197
8.9 Software design elements	197
8.9.1 Interrupt	197
8.9.2 Control	197
8.9.3 Drivers	198
8.9.4 Protocols	198
8.9.5 Applications	198
8.9.6 User Interface	198
8.10 Managing software size	198
8.10.1 Head full	198

Detail CHAPTER 9 *Integration* **199**

9.1 Planning	200
9.1.1 Integration from hardware and upwards	200
9.1.2 Counting backwards	200
9.1.3 Integration driven development	200
9.1.4 Efficient integration	200
9.2 Integration Prototypes	200
9.2.1 To integrate step wise	200
9.2.2 Anatomy	200
9.2.3 Feature principle	200
9.2.4 Incremental development	200
9.3 Revision control and branching	200
9.3.1 Principles	200
9.3.2 Sequential flow with no overlap	200
9.3.3 Implementation overlap bug corrections	200
9.3.4 Implementation partly overlap system test	201
9.3.5 Implementation overlap whole system test	201
9.3.6 Continuous implementation overlap	201
9.3.7 Many parts for revision control	201
9.3.8 Layered architecture revision control	201
9.4 Compatibility	201
9.4.1 Principle	201
9.4.2 Backward	201
9.5 Working against a community	201

- 9.5.1 *Time to market* 201
- 9.5.2 *Proprietary implementations* 201

Detail CHAPTER 10 *Verification and Validation* **203**

- 10.1 Attitudes to defects 204
- 10.2 Roles during test 204
- 10.3 Test Cases 204
 - 10.3.1 *Coverage* 204
 - 10.3.2 *Traceability to requirements* 204
 - 10.3.3 *Reference included in delivery* 204
 - 10.3.4 *Defects leakage during test* 204
- 10.4 Component and system test 204
 - 10.4.1 *Value chain* 204
- 10.5 Regression test 204
- 10.6 Acceptance criteria 205
- 10.7 Automatization 205
 - 10.7.1 *Design is vital* 205
- 10.8 Involve customers 205

Unite CHAPTER 11 *Line Management* **207**

- 11.1 Line Organization Basics 208
 - 11.1.1 *Short about organisation structure* 208
 - 11.1.2 *The line organisation is a kind of architecture* 208
 - 11.1.3 *EXAMPLE: similarities between architectures* 208
 - 11.1.4 *Maintaining the architecture* 209
- 11.2 Scaling up line organisations 210
 - 11.2.1 *Small size team* 210
 - 11.2.2 *Middle size firm* 210
 - 11.2.3 *Large size company* 211
- 11.3 Overlaying value chain on line organisation 212
 - 11.3.1 *Conservation of the value chain* 212
 - 11.3.2 *Different overlay possibilities* 213
 - 11.3.3 *“Work along” oriented work force distribution* 213
 - 11.3.4 *Chair example of “work along” line organisation* 214
 - 11.3.5 *“Work across” oriented work force distribution* 216
 - 11.3.6 *EXAMPLE: “work across” oriented work force distribution* 216

	11.3.7 <i>Mix between work along and work across</i>	217
11.4	Large size enterprise	219
	11.4.1 <i>Engineering line organisation versus system architecture</i>	219
	11.4.2 <i>EXAMPLE: engineering line organisation related to system architecture</i>	220
	11.4.3 <i>EXAMPLE: motor chair development enterprise</i>	221
11.5	Line accountability by ownership	221
	11.5.1 <i>Owner concept</i>	221
	11.5.2 <i>Control a company</i>	222
	11.5.3 <i>Management accountability</i>	222
	11.5.4 <i>Value chain ownership</i>	223
	11.5.5 <i>Work environment</i>	224
	11.5.6 <i>Equipment and tools ownership</i>	224
Unite CHAPTER 12	<i>Interfering Operations</i>	225
	12.1 Concept	226
	12.2 Definition of a project	226
	12.3 Requests and Allocation	226
	12.4 Follow-up and reporting	226
	12.5 Project office	226
	12.6 Multiproject setup	226
	12.7 Checkpoints in development flow	226
	12.7.1 <i>Definition</i>	226
	12.7.2 <i>EXAMPLE: milestone</i>	226
Unite CHAPTER 13	<i>Balancing Players</i>	227
	13.1 Conflicting Parameters	228
	13.1.1 <i>Lead Time</i>	228
	13.1.2 <i>Quality</i>	228
	13.1.3 <i>Cost of development</i>	228
	13.1.4 <i>Cost of product</i>	228
	13.1.5 <i>Delivery time</i>	228
	13.1.6 <i>Other</i>	228
	13.2 Balanced Scorecards	228
	13.3 <i>EXAMPLE: Project expectations overdetermined</i>	228

13.4 EXAMPLE: Overloading creates underachievement	228
13.5 Interaction between Project and Line	229
13.5.1 <i>Matrix organization</i>	229
13.5.2 EXAMPLE: <i>described line and project</i>	229
13.5.3 <i>Learning from projects, improving the line</i>	229
13.6 Gates in development flow	229
13.6.1 <i>Definition</i>	229
13.6.2 EXAMPLE: <i>tollgate</i>	229

Unite CHAPTER 14 *Configuration control* **231**

14.1 Different lead time development flow	232
14.1.1 EXAMPLE: <i>cooking the main course</i>	232
14.1.2 <i>Abbreviations</i>	232
14.1.3 EXAMPLE: <i>motor chair development with three different lead times.</i>	233
14.1.4 <i>Generic V-model for up-front system management</i>	234
14.1.5 EXAMPLE: <i>recurrent development at different lead times</i>	235
14.1.6 <i>V-model for recurrent development</i>	236
14.1.7 EXAMPLE: <i>ignoring system engineering</i>	236
14.2 Backward and Forward compatibility	237
14.2.1 EXAMPLE: <i>backward compatibility</i>	238
14.2.2 EXAMPLE: <i>extending both software and hardware</i>	238
14.2.3 EXAMPLE: <i>extending all three modules</i>	240
14.2.4 EXAMPLE: <i>continuation of extending modules</i>	240
14.3 Feature driven structures of patterns	241
14.3.1 EXAMPLE: <i>different lead time feature development</i>	241
14.3.2 EXAMPLE: <i>combination of module and feature development</i>	241

Meta CHAPTER 15 *Information Management* **243**

15.1 Abstraction Levels	244
15.1.1 <i>Meta Level</i>	244
15.1.2 <i>Description Level</i>	244
15.1.3 <i>Reality</i>	244
15.2 Availability versus security	244
15.3 Information Classification	244

15.3.1 Requirements	244
15.3.2 Attributes	244
15.3.3 Architecture	244
15.3.4 Interaction	244
15.3.5 Governance	244
15.4 Life Cycle Status	244
15.5 Revision Control	244
15.6 Tools	244
15.6.1 Web	244

Meta CHAPTER 16 *Quality System* **245**

16.1 Nature of Quality	246
16.2 Separation of Quality	246
16.3 Quality Shortage Sources	246
16.4 Chain of quality links	246
16.5 EXAMPLE: Wine Quality	246
16.6 EXAMPLE: Vandal Links	246
16.7 Measurements	246
16.8 Lessons Learned and Follow-up	246

Meta CHAPTER 17 *Process & methodology* **247**

17.1 Process elements	248
17.1.1 Strategies	248
17.1.2 Governance	248
17.1.3 Workflow	248
17.1.4 Metrics	248
17.2 Benefit of an explicit process description	249
17.3 As is or to be	249
17.4 Views of a process	249
17.5 Partitioning of a process	249
17.5.1 Process architecture	249
17.6 Symbols of process descriptions	249
17.7 Results or activities	249
17.8 Process roles	249
17.8.1 Process owner	249

17.8.2 Process manager	249
17.8.3 Process support	249
17.9 Often too detailed	250
17.10 Nothing new under the sun	250
17.11 Ad Hoc	250
17.12 Prototyping	250
17.13 Waterfall	250
17.13.1 Clean Room	250
17.14 Iterative	250
17.14.1 Rational Universal Process	250
17.14.2 Agile concepts	250
17.15 Languages	250
17.15.1 Object oriented	250
17.15.2 UML	250
17.15.3 Chrilles “all in a picture”	250
17.16 Make your own combination	250

Meta CHAPTER 18 *Improvement & Assessment* **251**

18.1 Improvement Planning	252
18.1.1 Step wise progress	252
18.1.2 Settle principles	252
18.2 Improvement Execution and Follow-up	252
18.2.1 Piloting	252
18.3 Change Project	252
18.3.1 Structure	252
18.4 Improvement anti-patterns	252
18.4.1 Ignore the process, then if problems arise, blame it	252
18.4.2 Throw the baby out with the bath water	252
18.4.3 Most excellence view is from Everest	252
18.4.4 Hunt scapegoats in place of analyse root causes	252
18.4.5 Preferring people’s facade to their outcome	252
18.4.6 Faking improvement by renaming and buzz wording	252
18.4.7 Management decisions are good recommendation	252
18.4.8 Processes inhibit creativity	252
18.4.9 Invalidate the law of gravitation	252
18.4.10 Let gurns save us by fixing their failures	253
18.5 Improvement Roles	253
18.5.1 Business Engineering	253

18.5.2 <i>Process Management</i>	253
18.5.3 <i>Process architecture</i>	253
18.6 Organisational Cultures	253
18.7 ISO 9000	254
18.8 Baldrige	254
18.9 CMM	254

“There is a lot of noise in the jungle,
you must only be aware of the dangerous”



Why nailing with a cudgel ?

1.1 What has gone wrong ?

In development organisations, it is not seldom seen that people work backwards like nailing with a cudgel (primitive club). Why ? You never get a carpenter to your house with a cudgel to nail with, do you ?

Why are skilled people behaving like nailing with a cudgel ?

Product development is nowadays in many respects an established and ordinary business. For example, house and bridge development are several thousands years of age. Other fields of product development are much younger, for example dealing with software begun in the decade of 1960.

However, in all development fields, there are still products which fail to satisfy end users. In some newer fields like software, trouble looks to be the standard, but in other fields it is a bit better. And note, there is no conspiracy behind this, no supplier like to disappoint an end user. So what is the problem ?

Uncontrolled complexity risk to emerge, when a business are getting scaled up.

If digging deeper in the development business in order to find the root cause to why products fail to satisfy end users, the most common reason seems to be that up-scaled development introduce multiplied levels of complexity, which in turn cause capability and competence problems when organising development of these products.

Two thousand years ago, the Pantheon building in Rome was an ultimate complex house construction on the very front of development knowledge at that time, but should nowadays be a rather modest target for a mid sized construction company. A complex building of today is a several hundred meter high sky scrape, which need high tech solutions and sophisticated calculations for strength of construction materials, and several hundred of workers within many disciplines must be well organized to make the building being raised. Complexity seem to have increased over time, which implies that it is certainly possible to coop with larger and larger complexity.

To analyse complexity a bit further, imagine the two diametrically opposed ends of complexity, the “ordinary low end”, and the “utmost high end”.

1.1.1 Success from the low complexity world

Houses have been constructed with success for very long. A normally handy person (with some drive) can, for example, extend his private house with some new space. He may have to contact experts to sort out problems beyond his competency and hire specialists to help him build, but in the whole, this is not more complex than he can lead the construction work and also take part in the craftsmanship. The extended space will be of desired cost and quality, and will function as planned. What possibly fails can afterwards most often be repaired at modest costs.

This scene is true for many ordinary scaled business in our modern society. (Let be that the house business recently face new complexity, when being target for a mas-

sive energy saving requirements or when competition press prices for house building below what is reasonable for persistent quality.)

1.1.2 Success from the high complexity world

There might be air plane crashes and medicines with severe side effects, but to travel by air or follow a doctor's subscription is generally very safe. In these cases, the high complexity of developing aircraft or medicine are undoubtedly handled with success. Obviously, in these fields, the scaling up of complexity has worked out very well, even if not totally clean from disasters.

1.1.3 Transition from low complexity to high

However, not all business and companies has manage to make this transition in a proper way. Computers often hang up and spoils large amount of work, consumer electronics fall into pieces and must be expensively repaired, kitchen equipment barely keep together until warranties are expired, etc.

Some companies retain success and others fail when getting into complexity.

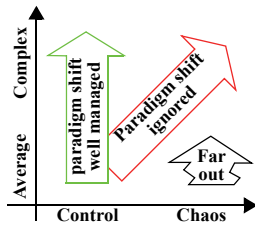
In these unsuccessful cases there are of cause a lot of extenuating circumstances, like everything must be developed in a rush because the market change quickly, testing is not given enough time and is forwarded to the end users, money is spent on commercials rather than development etc. And in the software discipline, typically one after another line of code is added, and the scale-up come very creeping and invisible, and all of a sudden has ruined the structure of the system.

Many unsuccessful companies might argue, that it is not really their problem if they fails to deliver satisfying complex products. Who hasn't heard "the customer simply gets what they pay for". But most often an analysis would have showed, that poor products costs more than they save for both producer and customer. These products are often in the poor end of the quality scale, and in fact it would have been more profitable to develop them better from the beginning (at least when consider the full life cycle of the product).

1.1.4 Is there any trick with scaling up ?

Scaling up is like $1 + 1 = 3$. At some point more than the size has changed, a paradigm shift has occurred.

It is recognised for long that when things dramatically change in scale, the thing is not just anymore the same but with another size. It get more like $1 + 1 = 3$. Sometimes this is referred as "at some point when increasing quantity, there is a change in quality" or in our case "at some point along the scaling-up there must be a change in approach, sometimes referred as a "paradigm shift" is approaching.



If the paradigm has shifted, the old methodology and approach must be replaced, and a radical new way of thinking must be applied. For example, house stairways must be replaced by elevators when houses get higher, growing software must be partitioned into smaller pieces separated with clear interfaces, slide rules get replaced with digital calculators, key hole surgery being far more efficient than open big wounds etc. The world is full of (smaller and bigger) paradigm shifts.

And back to the initial question, why nailing with a cudgel. Developers might nail with a cudgel because their approach hasn't been scaled up to the actual complexity facing them. A paradigm shift has occurred but was ignored. The cudgel was successful against primitive enemy tribes, but has got very inappropriate for nailing.

1.1.5 Short about complexity

With complexity means, when a set of system parts have relations to each other, in a way that forms a total system that is hard to understand and predict.

Let's look at the solar system. When copernicus placed the sun in the middle, Isaac Newton was able to describe the motion of all planets with his "laws of motion". This is rather ordinary mathematics, referred as the "n-body problem", which can be analytically handled.

But complexity reappear at this stage. It was rather simple to solve the equation for $n = 2$, e.g. two planets like the sun and the earth being alone in the solar system. It took several hundreds of years to solve it for $n = 3$, e.g. three planets like the sun, the earth and the moon being the only planets. For n greater than 3 it is still not completely analytically solved, but the challenge has lead to a lot of chaos research.

This is in short what happens when scaling-up. Very soon the system parts form a total system that possibly might be described, but gets hard to understand and predict. Often such systems are referred as systems in chaos. The system itself doesn't know that it is in chaos, of course it is our understanding that is not good enough.

The general way to treat complexity is to make research, in order to enough understand the complexity. If still too complex to be handled, some mitigation can be tried. One way might be to limit the degrees of freedom and accept a lesser accuracy of understanding, for example by approximations (the moon has no influence on the sun), or to freeze some relations (the sun is assumed fixed in the centre).



Have you ever reflected over why houses preferably have right angles between most building elements. Do you get the point? Simply because this lowers the complexity and makes a house easier to understand, predict and build. The Beijing Bird's Nest, not having two similar angles anywhere, has such large complexity that it had been impossible to handle in the slide rulers era, but could be mastered by designing with powerful computers.

1.2 When should you hear the warning bell ?

A company having problem with mastering its complexity, shows many symptoms from this if anybody care to watch. And most often these symptoms has continued for a long time. Scaling up is a slowly creeping effect, and problems may be small and silent in the beginning, but hitting hard after a while if not cured.

And there are reasons to watch out. The first company to identify an approaching paradigm shift and succeeding to overcome the challenge, is coming out very strong and competitive. Instead of being gradually slowed down by growing legacy complexity, they can capitalize on their new way of mastering it, and in short time get ahead of their competitors.

The most obvious examples are paradigm shifts in warfare, which could even change the balance of power between whole countries. A lot of effort is spent on intelligence and reconnaissance in order to watch the enemies effectiveness.

Disorder during development, result in that this disorder is also built into developed products.

One may perhaps argue that it is not a big problem if a development organisation is internally messy and in disorder, because this will not trouble their customers. But this is often wrong, because this disorder will as well be built into their products, which will get the same lack of structure and quality, and will thereby finally hit their customers.

To identify problems with mastering complexity in development organisations, watch out for the following 13 warning-bells:

1. "EXAMPLE: Unrealistic campaigns continually restarted" (page 28).
2. "EXAMPLE: Management not being accountable" (page 28)
3. "EXAMPLE: Root cause analysis being suppressed" (page 29)
4. "EXAMPLE: New wrapping with same content" (page 30)
5. "EXAMPLE: Ego people being change agents" (page 30)
6. "EXAMPLE: Sub optimization within local organizations" (page 30)
7. "EXAMPLE: Scapegoats of a blame game" (page 31)
8. "EXAMPLE: Products driven by engineers" (page 31)
9. "EXAMPLE: Product structure being degenerated" (page 32)
10. "EXAMPLE: Impossible principles applied" (page 32)

- 11. "EXAMPLE: Artificial complexity being pushed" (page 33)
- 12. "EXAMPLE: Intrinsic complexity being ignored" (page 33)
- 13. "EXAMPLE: Devils in the details being ignored" (page 34)
- 14. /* EXAMPLE: Continually decision reset */

1.2.1 EXAMPLE: Unrealistic campaigns continually restarted

In this case, improvement campaigns are being broadcasted from management and the message from them might look like Figure 1-1 below.

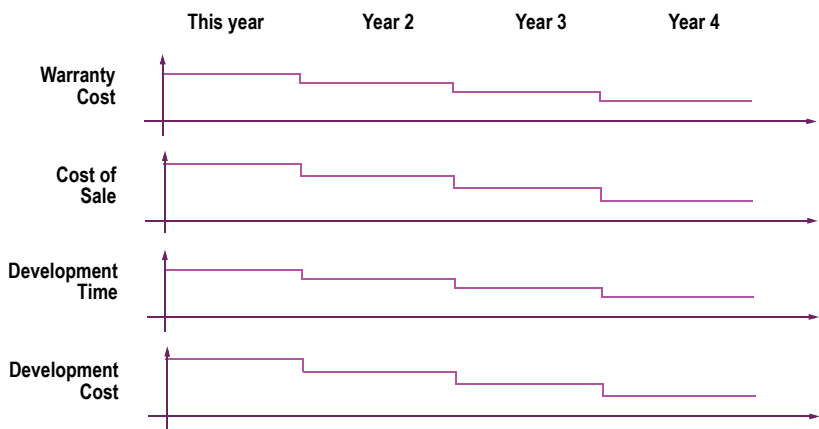


FIGURE 1-1 Improvement campaign goals

Probably you have seen such programs sometimes passing by. Most of the energy is put on cheering and making noise, and less on analysing, understanding and implementation of changes. Very often these campaigns are very intense in the beginning but are fading out as time pass. You may also have ended up with the feeling, that not much have been achieved at the end of the campaign (even if graphs are circulated proving the contrary). And for that so little is delivered, seldom anybody are found responsible.

Why are many organizations running improvement projects, one after another, without sustainable results ?

When a new manager enter the organisation, the campaign is restarted again to show drive and energy. But of course with different names, concepts and symbols, but with the same type of unrealistic campaign.

1.2.2 EXAMPLE: Management not being accountable

Managers are the most important group of employees, when to establishing good and efficient working models. Over and over again, it has been found that making improvements and establishing working models is more or less impossible without

active support from managers. Keep a watch on the following criteria, which often prevent from setting a sound company culture:

- Managers not specially interested in how his inferiors are working, and claiming that they are expected to sort out that themselves.
- Managers thinking it is more important keeping peace in his organisation and reporting to superior management that everything is working fine, instead of risking noise from solving severe problems.
- Managers not ever acting pro actively to problems. Their habit is always to wait until the failure is a matter of fact, before they take action.
- Managers pretending to be interested in work models, but just desire to silence their conscience. They might engage persons to document and improve, with the hidden agenda to archive the results in binders in order to be forgotten.
- Managers acting grandiose, and claim that they already have everything under control. For example, imagine that this book were shown to them. They would answer that here is nothing new they didn't already knew. It's a great book, but we already work according to it.

It is seldom possible to influence on which managers there are in a company, but yet it sets the level for improvement success. It is waste of energy to try to improve companies with inappropriate management.

“The management culture” is the most influential on how complexity is handled and improved.

1.2.3 EXAMPLE: Root cause analysis being suppressed

Sometimes a frenetic “improvement wave” can be spread over an organisation. One after another wants to be the best on improving the way of work. It may even happen that upper managers tries to beat each other with efficiency programs and rewarding improvement proposals.

Their eagerness admit no time to structure the improvement work, and improvements get started on every imaginable spot of the organisation. Everything are object for reparation and improvement. Current work models are declared insufficient and are discarded (and forgotten), like throwing the baby out with the bath water, in favour for new bright improvements to come.

An objective and honest root cause analysis might be embarrassing and inconvenient, but is most essential for serious improvement management

Unfortunately, sufficient analysis are not being made, pointing out the poor parts being most urgent targets for improvements, and what parts in fact is “good enough”, at least for a while. Neither it is planned in what order the poor parts need to be fixed. Often organisation are very well aware of existing real problems and bottle necks, but courage are lacking to present these facts and get these problems visible. It is much more convenient to sweep the most ugly problems under the carpet and report about more harmless shortages. They are not so embarrassing and they are much easier to fix.

1.2.4 EXAMPLE: New wrapping with same content

Even if development is complex, there is certainly a limited amount of fundamental ways how to organize product development. This fact is troublesome for methodology consultants, salvation authors (myself being an exception:-) and other confidence trick makers. But like in the fashion business, this is solved by change the wrapping and reintroduce old things as being the latest inventions, that are urgently needed by everybody.

Trousers can not be designed in so many different ways. But the fashion business succeeds, over and over again, to surprise us with yet new trousers.

People too young in the development business was maybe not there the last time these things were in fashion. And managers may not have time enough to penetrate and disclose all “package” tricks. Like fashion consumers, people in general fear the risk to be regarded uninformed and old fashioned.

When these arguments arise from consultant sales persons or by improvement proposals from inferiors, it easily happens that it is decided to acquire similar things (but differently named and described) that might already be acquired and even might be in place.

1.2.5 EXAMPLE: Ego people being change agents

Many persons love to start up new things, to be inventive persons, to get a lot of attention and look busy, and to be hang-arounds to influential managers. They use their charisma to sell in solutions to anyone in need for anything.

But after a while, when it gets harder to deliver and show promised results, these persons pop up somewhere else in the organisation with other newly started improvements. If follow-up from management is poor, these persons are never made responsible for what they promised but not delivered, and can proceed to jump around.

Charisma people are very useful, but never appoint them in expert positions.

Needless to say, this example is a disaster for improvement activities and the organisation moral. Much more of this will be discussed in the chapter "Meta CHAPTER 18 Improvement & Assessment", at page 251.

1.2.6 EXAMPLE: Sub optimization within local organizations

When organizations get that large, that everybody don't meet each other face to face any more, there will appear more and more individuals not being in contact with each other. This is nothing wrong in itself, in a big company everybody can not work together with everybody else.

To still bring employees to share the company culture and value chain, there is now an emerging need to work with formal documentation and improvements on many abstraction levels. As a consequence of this, it can often be seen that groups internally works very efficient and structured. But if looking on how these groups contribute together for the company result, it might be very inefficient, or the groups might even destroy each others work results.

Even if a group performs very well, it is not thereby given that it brings any substantial value to the organisation

The higher up in an organisation the bigger are the effects of problems and rewards from efficient solutions. But in many companies this fact is not recognized, and even the revers may occur. On low level there might be a dedicated improvement work ongoing, but the higher up in the organisation, the more uninteresting managers get for improvements and an efficient way of working.

1.2.7 EXAMPLE: Scapegoats of a blame game

If crisis of any kind hits a development company, it is quite natural that everybody tries to protect themselves, and managers tries to protect their organisations. One way to protect yourself, if you can make believe the cause of the crisis are outside of your own domains, is to declare that you and your organisation was only an innocent victims. Often a company executive group get pressure from the boards and owners, and must present drive and improvement programs to overcome the crisis.

Altogether, there is a high desire to point out scapegoats in order to try to hide own responsibility. And in the same pattern as common mobbing mechanisms, the weakest parts of the company organisations is less dangerous to attack.

The value chain and people working with the value chain, are often targets when searching for scapegoats of a crisis.

Not seldom seen, is that executives point out the engineering value chain to be the cause for the crisis. There are examples, that even if the crisis obviously was caused by outdated product portfolios delivered by incompetent market organisations, it ends up with the management declares that developers are working in the wrong way.

A good way to eliminate such mismanagement, is to demand root cause analysis. But the opposite is more common, it is easier and more controllable to point out a weak scapegoat, than to launch a root cause analysis which may find skeleton in anybody's cupboard.

1.2.8 EXAMPLE: Products driven by engineers

Many product markets (even if being very technical products) are not different from the fashion market. It is the price tag and the appearance of the product that is most important, and the rest of the product characteristics must only reach above a normal "hygiene standard".

Engineer driven development is often carefully watched. But to ensure it will not happen is hard.

If a company is governed by mostly technical people, the reverse might occur. Then the technical systems within the cabinet get most important and heavily improved but the outlook is kept as boring as forever. User perceived quality may slip because feedback from the market is ignored. A high return rate and warranty cost is often the result.

A company may have a lot of market driven people and managers, but despite this might still be engineering driven, because a strong channel might be missing to convey market driven requirements into the centre of the engineering departments, for more details about this see the chapter "Detail CHAPTER 6 Requirements", at page 109.

1.2.9 EXAMPLE: Product structure being degenerated

Product structure and architecture are often the most misunderstood of all development concepts. It is very strange, because for the house construction business the architect is both important and well understood. In hardware development it is fairly well understood, because the components are tangible and can be likened to rooms in a house.

But when coming to software, it might be totally impossible to make analogies to rooms, flats, floors etc. A software construction that has been uncontrolled extended for long time, may have an architecture very similar to a allotment-garden cottage. Small rooms has been added to the house body every summer but the body itself has never been reconstructed. This ends up in a cottage with a large amount of rooms, nooks and corners, but nowhere any continuous space for living. More about understanding architecture will be discussed in the chapter "Detail CHAPTER 7 Architectures", at page 151.

A software structure might be as degenerated as an allotment-garden cottage. A lot of rooms, nooks and corners, but nowhere to live.

1.2.10 EXAMPLE: Impossible principles applied

To build a company on strong principles are generally a good sign, for example like the successful companies Toyota and Ikea. But be aware of, that if wrong or impossible principles are selected, then the damaged will be equally strong as the success would have been.

A striking example on implying impossible principles is the construction of the Swedish warship Wasa. Sweden was in war with Poland, and needed better fire power in their navy. Thus, there was a heavy force from the Swedish king to equip the ships with as many cannons as possible. Consequently the ship Wasa had too many cannons compared to its size and ballast, but despite this the king ordered the launch of the ship. After ten minutes in fine weather, the ship capsized and sunk in 1628.

What a competitive edge it would be for a company which succeed to "suspend gravitation". But what are the chance ?

Another example is a company having big problems to handle their system, which grew bigger and bigger for each day, resulting in uncontrollable organisations. The system engineering department was not in control and was even seen to be unneeded. However, a big improvement project was started and the upper management announced “the principle of de-coupling”, meaning that different parts of the system should be developed separately from each other and handed over from smaller organisations for integration and assembly.

The “only” annoying problem was that the architecture of the system was far from being decoupled, and no sub-system interfaces were described, visualized or managed. Few of the persons in the management understood, that if the system wasn’t partitioned in parts, it was useless to partition the organisation. A lot of time, energy and money was thrown away when to divide the company into decoupled organisations, but without decouple the technical system parts from each other. More of this will be addressed in the chapter “Detail CHAPTER 7 Architectures”, at page 151.

1.2.11 EXAMPLE: Artificial complexity being pushed

One of the best examples on this, even if far from the product development world, is the old solar system model with the earth in the centre. The solar system model showed the movement of all planets assuming that they were connected on spheres that were rotating at different speed. But some observations were really difficult to explain with this model. For example, it was not so seldom seen that a planet moving over the sky, suddenly reversed its direction and retrograded for some time, before reversing once again and proceeding in the original direction. The more irregularities of planet movements that was observed, the more epicycle spheres was put into the solar system model.

Complexity in this model was actually added by humans, and consequently most of it disappeared when Copernicus released his model with the sun in centre (by the way, a substantial paradigm shift).

Things should be made as simple as possible - but not simpler.

- Albert Einstein

In product development it is easy to end up with too many “spheres” and everything gets more and more impossible to understand. Every model builder is not as brilliant as Copernicus, but the warning bell should be ringing, when working models gradually gets too complex to understand. It might be the consequence of a model that has evolved beyond its initial simple context, and must be replaced by a more well-reasoned one.

1.2.12 EXAMPLE: Intrinsic complexity being ignored

This is the opposite to the preceding example and might ring the warning bell as well. Product development can be really complex, due to very complex products that are being developed in shorter time that is possible (so to say).

One good example of this is when a system to be developed has varying lead times in different architectural parts. For example, turn around time for making a new

software release might take a couple of days, while making a new ASIC may take some years to prepare.

Again, things should be made as simple as possible - but not simpler.
- Albert Einstein

If you neglect the different lead times for the software and the ASIC, you never succeed to get requirement management started in the right time for different architectural parts. That implies that different parts will not be ready for integration at the same time, which implies that some parts must wait for other parts to get ready. Or that some parts must be used despite they are not yet

finalized.

The point is, that if the complexity of different lead time is not taken care of in the work model, the requirement management will never succeed.

1.2.13 EXAMPLE: Devils in the details being ignored

Of course it is important that top management are supporting improvement work, but it might be dangerous to drive and organize everything strictly top-down. Some nasty details can be as hard to overcome as a law of nature. An improvement without experts checking in advance for devil's details may come out to nothing.

A good example is when totally development lead time must be shorten. That means to do the same things like before, but on shorter time, which in turn often results in doing more and more things in parallel. To understand that the synchronisation between parallel work will still be possible in practice, a critical path analysis must be performed on the new parallelism to apply.

The saying "little strokes fell great oaks" might be terrible decisive in complex product development.

At some point in trying shortening the lead time, it is not possible to further increase the degree of parallelism, because some details of the devil sets the limit. For example, it take some minimal time to get an ASIC (silicon chip) produced after you have submitted all ASIC drawings and this time isn't realistic to shorten beyond a certain duration. For analyses of critical path it is essential to involve experts of details, knowing the critical path and where unconditional limitations sets in for each activity in the path.

1.3 *Your way out of this*

Above chapters describe symptoms when complex development are deviating in wrong ways. It is very important to diagnose such symptoms in an early stage, when it is still easy and cheap to cure. But how to carry out this treatment? This book give one clear answer to it - the development organisation must better master their complexity they face.

This book do help you out. (In this book, important statements are shown like this)

To learn about complexity is hard and there is certainly no short cuts. This book intends to give you a palate of aspects on complexity, that can help you to discuss, understand and come up with improvement proposals that really help.

(If you find this book being impolite when claiming that you need help, but in fact you don't, please instead review this book and send back your comments to the author).

This is not just an academic book that only enumerate all plausible facts of complex development, but it is instead a book covering real experience and long time learning. It contain a almost endless amount of examples from various relevant development worlds. But this book doesn't stop even there, it also put all these examples into context of the whole lot, in order to capture and illustrate the complexity behind. And that is in fact what you are in big need of, but seldom get from other books. This book will take you underneath the skin of complex development, and prevent you to fall into any of the pit-falls presented above. Good luck!

Glossary of terminology

Term	In this book	Out there
Architecture	How the structure of an artefact is assembled by its parts, including immaterial artefacts like software.	About the same
Artefact	Something developed or produced by humans, including immaterial artefacts	About the same
Black-box (BB)	Demarcation which not disclose anything of its internal content, but only showing properties of its surface, and behaviour via its interface. Properties and behaviour are described by requirements.	About the same
Component	A sub-system being explicitly developed or extracted from a finished system, in the way that it can be easily understood and reused without significant rework. Electronic components are typical, even if they often have small granularity.	About the same
Decompose	To split up an architecture black-box into its white-box, containing design elements (some of which can be underlying black-boxes).	About the same
Description	Information about an artefact. Always be careful when referred to an artefact, is it the artefact itself or is it its description being referred.	About the same
Design (noun)	See design element.	
Design (verb)	Transform black-box requirements, to design elements in resulting white-box.	Rather similar
Design element	Anything showing up when decomposing a black-box, e.g. underlying black-boxes, electronics, software commands etc. Design elements can be developed and finished without a new layer of formal requirements.	Many definitions
Development	Translate an product idea (satisfying a market demand), into descriptions and prototypes, making a production possible. Be aware, to not mix production and development together.	About the same
Formal	Comply to concepts that are shared by colleagues.	More vague
Granularity	Size of components (compared to size of assembled system).	About the same
Interface	Interactions with a system (or sub-system) are through its interfaces. If the system (or sub-system) is judged a black-box, the interface requirements must be described and managed.	About the same
Life cycle status	Regarding a named artefact to only change maturity states, during its life from birth to death. The life cycle status reflecting the maturity might be stated after the name.	About the same
Line (management)	The static and most often hierarchal structure of managers to lead a company.	About the same
Managed information	Information is said to be managed, if it is uniquely identifiable, documented, obeyed and updated to always reflect the reality.	About the same
Module	A sub-system without clear interfaces to its system. It can have a explicit name, but it is not easy to understand how it interacts with the system or how it is demarcated from the system.	Unclear distinction between module and component.
Object	A self-contained and autonomy artefact. Close in its meaning to a small black-box.	About the same
Product	A product is an artefact, created by somebody, from raw materials, to finished goods, for a market, to satisfy a need.	About the same

Overview Table of Contents

Term	In this book	Out there
Product life cycle (PLC)	Time period, from a product idea is invoking a development, until the product is out-phased on the market, including eventual warranties.	About the same
Production	To realise and multiply construction in volumes, based on developed construction documentation and prototypes. Be aware, to not mix production and development together.	About the same
Project	A temporary structure in a company, to accomplish an assigned objective, and restricted by limited conditions (time, cost etc.).	About the same
Project leader	The manager of a project, reporting to the project sponsor.	About the same
Project office	A line manager with a pool of project leaders	About the same
Project sponsor	A line manager ordering and controlling a project.	About the same
Property	Observable characteristics on the surface of a system (or sub-system). If the system (or sub-system) is judged a black-box, the property requirements must be described and managed.	About the same
Refine	Narrow in black-box requirements to match contained underlying black-boxes.	Can mean whatever
Requirements	The behaviour and properties of an artefact.	About the same
Specify	Users capturing wishes from a system.	Much more vague
Stakeholder	Anybody that have interest in what are specified by requirements. A stakeholder might be an end user of a system to be developed, an orderer paying for the development, a verifier testing a developed system against its requirements, etc.	About the same
Sub-system	Partition or part of the system to be developed. If a sub-system is judged complex, it should be considered as a black-box with requirements to describe it.	Much more vague
System	The precise target demarcation for what is to be developed, no less, no more.	Much more vague
Value chain	The factual way that work are performed in a company (to add value to it), regardless if it is understood or described.	More vague
White-box (WB)	Demarcation of an amount of visible design elements. When opening a black-box it gets a white-box containing embedded black-boxes and other design elements. Be aware that a white-box always has requirements, that has been updated to still satisfy the chosen white-box design.	About the same

Overview Table of Contents

Term	In this book	Out there
------	--------------	-----------

Index

A

Architecture 32, 33
Architecture maintenance 209
ASIC 34

B

Big bang integration (example) 91
Bird's Nest, Beijing 27
Board of directors 222

C

Chaos 26
Compatibility, backward 237, 238
Compatibility, forward 237
Complexity, artificial 33
Complexity, development 30
Complexity, growing 264
Complexity, high 25
Complexity, intrinsic 33
Complexity, low 24
Complexity, mitigation 26
Complexity, multiplied levels 24
Complexity, short about 26
Complexity, transition from low to high 25
Computer program 65, 66
Cooking the main course (example) 232
Copernicus 33
Critical path 68, 99
Critical path analysis 34
Customer 66

D

De-coupling 33
Details, devil in 34
Development, bridges 24
Development, conclusion of series of examples 98
Development, different lead time 232
Development, different lead times (example) 233, 235
Development, double staffing 87
Development, extending all modules 240
Development, extending continuously 240

Development, extending features (example) 241
Development, extending modules and features (example) 241
Development, extending sw & hw (example) 238
Development, feature parallel mania (example) 93
Development, four features (example) 96, 97
Development, houses 24
Development, ignoring system (example) 236
Development, in disorder 27
Development, iterative (example) 87
Development, iterative decomposition (example) 91
Development, late requirements (example) 84
Development, moderate parallelism (example) 83
Development, parallel mania (example) 81
Development, products 24
Development, prototyping 86
Development, single feature (example) 93
Development, sky scrapes 24
Development, system decomposition (example) 89
Development, two feature (example) 94
Development, two feature on isolated branches (example) 95

E

Ego people 30
Entrepreneurship 264

F

Finished goods 38
Flow of activities 65, 69
Flow, executable 65
Flow, support 67
Forum 221

G

Gnosjö area 264

H

Hygiene standard 31

I

IKEA 38

Immaterial asset 64

Improvement campaigns 28

Interfaces 33

L

Lead time 34, 99

Leading group 222

Life cycle status 73

Line accountability 221

Line organisation 64

M

Management, acting grandiose 29

Management, bad conscience 29

Management, keeping peace 29

Management, line 207

Management, not accountable 28

Management, not interesting 29

Management, not proactively 29

Managing director 222

Man-time (actual) 80

Man-time (effective) 80

N

Nail with a cudgel 24, 26

N-body problem 26

Newton Isaac 26

O

Organisation structure 208

Organisation versus architecture
(example) 220

Organisation, a kind of architecture 208

Organisation, large size 211, 219

Organisation, middle size 210

Organisation, small size 210

Organisation, uncontrollable 33

Organisation, very large size 221

Organization, line 208

Owner 221

P

Pantheon 24

Paradigm shift 25, 26, 27, 33

Parallel work 34

Prefabricate 102

Principles, impossible 32

Producer 76

Product 66

Product development 33, 37, 38, 40

Product life cycle 38

Product, aspects of 38

Product, driven by engineers 31

Product, in disorder 27

Product, structure 32

Production description 64

Products, degenerated 32

Products, profitable quality 25

Prototype 64

R

Raw material 38

Requirement management 34

Requirements, late (example
revisited) 103

Result orientation 72

Results 69

Roles 76

Root cause analysis 24, 29

S

Sandahl Christer, the author 264

Scaling up 25, 26, 27

Scaling up line organisations 210

Software 24, 32, 34

Solar system 33

State machine 75

Structure of parts 76, 110, 130, 131, 135,
140, 152, 153, 208

Supplier 66

System engineering 33

T

Time to market 40

Top-down 34

V

Value added 67

Value added tax (VAT) 67

Value chain 64, 66

Value chain block activity-activity 73

Value chain block activity-result 74

Value chain block executor and
producer 76

Value chain block result-result 75
Value chain block, state machine 75
Value chain follow-up 69
Value chain life cycle status 73
Value chain links 68
Value chain overlaying line
 organisation 212
Value chain ownership 223
Value chain results 69
Value chain state machine 75
Value chain, activities and results
 (example) 69
Value chain, activity flow (example) 68
Value chain, basics (example) 67
Value chain, building blocks 73
Value chain, concept 65
Value chain, conservation 212
Value chain, development 64
Value chain, product 64
Value chain, result orientation
 (example) 72
V-model 90

W

Warning-bells 27
Wasa, Swedish warship 32
Waste 80
Waterfall 78
Work across value chain (example) 216
Work along and work across mix 217
Work along value chain (example) 214
Wrapping, new with old content 30



Christer Sandahl has for over 40 years been living in the engineering world. Beginning with photo and chemistry in his teenage, over to electrical engineering in university, and into computer design as professional.

In his early working life Christer has all by himself several times constructed large computer systems, both hardware and software.

When computers got large and complex, he has for long periods managed software groups in successful local companies as well as in large world wide combines, such as Sony Ericsson, and Axis Communications.

Christer has grown up in the “Gnosjö area” of Sweden, the origin of entrepreneurship. His family has a large transportation company, he and his brother has quality wine production in Hungary, and Christer has of cause taken on this way of life in his engineering profession.

- Why is malfunctions common in computer products?
- Why is it frustrating to operate our everyday products?
- Why do most development projects get out of hand?

It become more and more evident that behind those problems, is mainly the ever growing complexity, which is not enough understood and mastered.

If deciding to master complexity, there are no single easy cures. You need generalists, you need specialists, but most essential, you need people that are both, to establish efficient bridging between all different areas of development.

This book describes all those concrete elements of complex product development, and ties them together to a uniform conception. With you in mind, all descriptions comes with plenty of real examples and illustrations, from prestigious board rooms down to the plain floor.

Don't ever tell me you didn't get a chance.
