



Better Embedded System Software

Philip Koopman
Carnegie Mellon University

Drumnadrochit Education LLC, 2010
Hardcover, 397 pages, acid free paper
ISBN-13: 978-0-9844490-0-2
ISBN-10: 0-9844490-0-0

<http://koopman.us/book.html>

Topic Quick Reference	
1. Introduction	1
SOFTWARE DEVELOPMENT PROCESS	
2. Written Development Plan	9
3. How Much Paper Is Enough?	17
4. How Much Paper Is Too Much?	29
REQUIREMENTS & ARCHITECTURE	
5. Written Requirements	37
6. Measurable Requirements	47
7. Tracing Requirements To Test	57
8. Non-Functional Requirements	65
9. Requirement churn	77
10. Software Architecture	85
11. Modularity	93
DESIGN	
12. Software Design.	107
13. Statecharts and Modes	117
14. Real Time	125
15. User Interface Design	147
IMPLEMENTATION	
16. How Much Assembly Language Is Enough?	157
17. Coding Style	167
18. The Cost of Nearly Full Resources	175
19. Global Variables Are Evil.	185
20. Mutexes and Data Access Concurrency	199
VERIFICATION & VALIDATION	
21. Static Checking and Compiler Warnings	219
22. Peer Reviews	227
23. Testing and Test Plans	237
24. Issue Tracking & Analysis.	255
25. Run-Time Error Logs	265
CRITICAL SYSTEM PROPERTIES	
26. Dependability	275
27. Security	295
28. Safety	315
29. Watchdog Timers	333
30. System Reset	341

Contents

Topic Quick Reference	v
Preface	xxiii
1. Introduction	1
1.1. Overview	2
1.2. Using this book	2
1.3. Chapter organization	2
1.3.1. Chapter sections	2
1.3.2. Getting more information	3
1.4. What do you do first?	3
1.5. It's printed in a book, so it must be true, right?	4
1.6. For more information	4
1.6.1. General topic search keywords	4
1.6.2. Other resources	4
2. Written Development Plan	9
2.1. Overview	10
2.1.1. Importance of a written development plan	10
2.1.2. Possible symptoms	10
2.1.3. Risks of an inadequate development plan	10
2.2. Development plan overview	11
2.2.1. Benefits of a development plan	11
2.2.2. Creating a development plan	12
2.2.3. Risk management and success criteria	12
2.3. Starting point for a development plan	13
2.3.1. Process steps and paper generated	13
2.3.2. Risk management	14
2.3.2.1. Peer reviews	14
2.3.2.2. Gate reviews	14
2.3.3. Success criteria	15
2.4. Pitfalls	15
2.5. For more information	15
2.5.1. General topic search keywords	15
2.5.2. Recommended reading	16
2.5.2.1. General project management	16
2.5.2.2. Software development plans	16
2.5.2.3. Software development models	16

2.5.2.4. Software risk management	16
3. How Much Paper Is Enough?	17
3.1. Overview	18
3.1.1. Importance of creating enough paper	18
3.1.2. Possible symptoms	18
3.1.3. Risks of insufficient amounts of paper	19
3.2. Good ideas for creating paper	19
3.2.1. What paper should I create?	21
3.2.2. How much paper should I create?	21
3.3. A starting point for what paper to create	23
3.3.1. Customer requirements	23
3.3.2. Engineering requirements	23
3.3.3. Architecture	23
3.3.4. Design	24
3.3.5. Implementation	24
3.3.6. Test plan	24
3.3.7. Bug list	24
3.3.8. User Guide	25
3.3.9. Marketing materials	25
3.3.10. Schedule and staffing plan	25
3.3.11. A software development plan	25
3.4. Pitfalls	26
3.5. For more information	26
3.5.1. General topic search keywords	26
3.5.2. Suggested reading	26
3.5.3. Examples of documentation standards	27
3.5.4. Additional reading	28
4. How Much Paper Is Too Much?	29
4.1. Overview	30
4.1.1. Importance of keeping useful paper	30
4.1.2. Possible symptoms	30
4.1.3. Risks of creating useless paper	30
4.2. How to avoid useless paper	31
4.2.1. Make paper useful	31
4.2.2. Keep paper up to date	32
4.2.3. Eliminate paper that can't be made useful	33
4.3. Pitfalls	33
4.4. For more information	34
4.4.1. General topic search keywords	34
4.4.2. Recommended reading	34

5. Written Requirements.	37
5.1. Overview	38
5.1.1. Importance of written requirements	38
5.1.2. Possible symptoms	38
5.1.3. Risks of inadequate requirements	38
5.2. What to put in written requirements	39
5.2.1. The big picture on requirements	39
5.2.2. Types of requirements	40
5.3. Good requirement practices	41
5.3.1. Measurement	41
5.3.2. Tolerance	42
5.3.3. Consistency	42
5.3.4. Use of shall and should	43
5.3.5. Trackability	43
5.3.6. Readability	44
5.4. Pitfalls	45
5.5. For more information	45
5.5.1. General topic search keywords	45
5.5.2. Recommended reading	45
5.5.3. Suggested standards.	45
6. Measurable Requirements	47
6.1. Overview	48
6.1.1. Importance of having measurable requirements	48
6.1.2. Possible symptoms	48
6.1.3. Risks of unmeasurable requirements	49
6.2. Successful quantification of requirements	49
6.2.1. Avoid a requirement for perfection	49
6.2.2. Being precise	51
6.3. Measuring the unmeasurable.	52
6.3.1. Don't ignore unmeasurable topics when writing requirements	52
6.3.2. Create a proxy measurement	52
6.3.3. Get testers involved up front	53
6.3.4. Create an engineering feedback system	54
6.3.5. Rely upon process.	54
6.3.6. Formal methods	55
6.4. Pitfalls	55
6.5. For more information	55
6.5.1. General topic search keywords	55
6.5.2. Recommended reading	56

7. Tracing Requirements To Test.	57
7.1. Overview	58
7.1.1. Importance of tracing requirements	58
7.1.2. Possible symptoms	58
7.1.3. Risks of inadequate traceability	58
7.2. Traceability.	59
7.2.1. Requirement to test traceability	59
7.2.2. Exploratory testing and traceability	61
7.2.3. Other uses of traceability	62
7.3. Pitfalls	62
7.4. For more information	62
7.4.1. General topic search keywords	62
7.4.2. Recommended reading	62
8. Non-Functional Requirements	65
8.1. Overview	66
8.1.1. Importance of meeting non-functional requirements.	66
8.1.2. Possible symptoms	67
8.1.3. Risks of inadequate non-functional requirements.	67
8.2. Common non-functional requirements	67
8.2.1. Performance	68
8.2.2. Resource usage	68
8.2.3. Dependability	69
8.2.4. Fault handling	70
8.2.5. Security and intellectual property protection	71
8.2.6. Safety and critical functions	72
8.2.7. Usability.	74
8.2.8. Conformance to standards and certification requirements	74
8.3. Pitfalls	75
8.4. For more information	75
8.4.1. General topic search keywords	75
9. Requirement Churn.	77
9.1. Overview	78
9.1.1. Importance of minimizing requirement churn	78
9.1.2. Possible symptoms	78
9.1.3. Risks of frequent requirement changes	79
9.2. Tracking requirement churn	79
9.2.1. Tracking the requirement change rate.	79
9.3. Controlling requirement churn	81
9.3.1. The Change Control Board	81
9.3.2. Requirement freezes	82

9.3.3. Assessing the cost of changes	82
9.4. Pitfalls	83
9.5. For more information	84
9.5.1. General topic search keywords	84
9.5.2. Recommended reading	84
10. Software Architecture	85
10.1. Overview	86
10.1.1. Importance of a well defined software architecture	86
10.1.2. Possible symptoms	86
10.1.3. Risks of an inadequately defined software architecture	87
10.2. Representing software architecture	87
10.2.1. Common architectural representations	88
10.2.2. Multiple architectural diagrams	89
10.3. What makes an architecture good?	90
10.3.1. Architectural patterns	91
10.4. Pitfalls	91
10.5. For more information	92
10.5.1. General topic search keywords	92
10.5.2. Recommended reading	92
11. Modularity	93
11.1. Overview	94
11.1.1. Importance of modular systems	94
11.1.2. Possible symptoms	94
11.1.3. Risks of non-modular systems	94
11.2. Evaluating modularity	94
11.2.1. Small size	95
11.2.2. High cohesion	96
11.2.3. Low coupling	96
11.2.4. Information hiding	97
11.2.5. Composability	98
11.2.6. Low complexity	99
11.2.6.1. LOC (lines of code)	99
11.2.6.2. McCabe cyclomatic complexity	99
11.2.6.3. Short term memory chunks	100
11.2.6.4. Function points	100
11.2.6.5. Using complexity metrics	100
11.3. Improving modularity	101
11.3.1. Factoring	101
11.3.2. Aggregation	102
11.4. Pitfalls	103

11.4.1. Using performance as an excuse for poor modularity	103
11.4.2. Over-aggressive simplification	103
11.5. For more information	103
11.5.1. General topic search keywords	103
11.5.2. References	104
12. Software Design	107
12.1. Overview	108
12.1.1. Importance of having a good design	108
12.1.2. Possible symptoms	108
12.1.3. Risks of not having a concrete design	109
12.2. The role of design	109
12.2.1. The right level of design abstraction	109
12.2.2. Going straight to implementation	110
12.3. Different design representations	110
12.3.1. Pseudocode	111
12.3.2. Flowcharts	112
12.3.3. Statecharts	113
12.3.4. Other design representations	113
12.3.5. Code synthesis and model-based design	113
12.4. Pitfall – using comments as the design	114
12.5. For more information	116
12.5.1. General topic search keywords	116
12.5.2. Recommended reading	116
13. Statecharts and Modes	117
13.1. Overview	118
13.1.1. Importance of using statecharts	118
13.1.2. Possible symptoms	118
13.1.3. Risks of not using statecharts	118
13.2. Statecharts	119
13.2.1. Statechart construction	119
13.2.2. Statechart implementation	121
13.3. Pitfalls	123
13.4. For more information	123
13.4.1. General topic search keywords	123
13.4.2. Recommended reading	124
14. Real Time	125
14.1. Overview	126
14.1.1. Importance of getting real time operation right	126
14.1.2. Possible symptoms	127
14.1.3. Risks of inadequate real time design	127

14.2. Real time analysis overview	127
14.2.1. Assumptions and terminology for analysis and scheduling	128
14.2.1.1. All tasks T_i are perfectly periodic	128
14.2.1.2. All tasks T_i are completely independent	128
14.2.1.3. Worst case execution time C_i for each task is known	128
14.2.1.4. The deadline D_i is greater than or equal to the period P_i	129
14.2.1.5. There is zero overhead to switch between tasks	129
14.2.1.6. Meeting the assumptions	129
14.2.2. Selecting an approach	130
14.3. Pure static scheduling (non-preemptive)	130
14.3.1. Pure static scheduling strengths	132
14.3.2. Pure static scheduling weaknesses	132
14.3.3. Scheduling analysis	132
14.3.4. Embellishments	133
14.3.5. Pure static scheduling pitfalls	133
14.4. Static scheduling with small helper interrupts	134
14.4.1. Strengths	135
14.4.2. Weaknesses.	135
14.4.3. Scheduling analysis – the easy case	136
14.4.4. Scheduling analysis – the difficult case	136
14.4.5. Pitfalls	139
14.5. Rate monotonic scheduling with harmonic periods	139
14.5.1. Strengths	141
14.5.2. Weaknesses.	141
14.5.3. Scheduling analysis	142
14.5.4. Embellishments	143
14.5.5. RMS pitfalls	143
14.6. Overarching pitfalls	144
14.6.1. Don't write your own RTOS	144
14.6.2. Don't use EDF scheduling	145
14.6.3. Don't use big ISRs	145
14.6.4. Don't just rely on testing to ensure deadlines are met.	145
14.7. Knowing when to get help.	145
14.8. For more information	146
14.8.1. General topic search keywords	146
14.8.2. Recommended reading.	146
14.8.3. Additional reading	146

15. User Interface Design.	147
15.1. Overview	148
15.1.1. Importance of good usability.	148
15.1.2. Possible symptoms	148

15.1.3. Risks of a bad user interface	149
15.2. User interface design process	149
15.2.1. Identifying the user population.	149
15.2.2. Identifying the task	150
15.3. Good user interface design	151
15.3.1. User interface design principles.	151
15.3.2. Testing the design for usability	152
15.3.3. Reviewing the design	153
15.3.4. Iterated design efforts	153
15.4. Pitfalls	154
15.5. For more information	154
15.5.1. General topic search keywords	154
15.5.2. Recommended reading.	154
15.5.3. References	154

16. How Much Assembly Language Is Enough?	157
16.1. Overview	158
16.1.1. Importance of avoiding assembly language.	158
16.1.2. Possible symptoms	158
16.1.3. Risks of too much assembly language	159
16.2. Why avoid using assembly language?	159
16.2.1. Higher cost and longer development time	159
16.2.2. Poor portability.	160
16.3. Techniques for avoiding assembly language.	160
16.3.1. Buying a bigger, faster CPU	160
16.3.2. Profiling and optimizing	161
16.3.3. Tuning source code for the compiler	161
16.3.4. Giving more information to the compiler	162
16.3.5. Rewriting code to make optimization easier	163
16.4. When is it OK to use assembly language?	164
16.4.1. Really tiny processors.	164
16.4.2. Small, frequently used routine	165
16.4.3. Extended precision math.	165
16.4.4. Specialized hardware resources.	165
16.5. Pitfalls	165
16.6. For more information	166
16.6.1. General topic search keywords	166
16.6.2. Recommended reading.	166
16.6.3. Additional reading	166

17. Coding Style	167
17.1. Overview	168

17.1.1. Importance of a consistent coding style	168
17.1.2. Possible symptoms	168
17.1.3. Risks of not having a written coding style guideline	168
17.2. Good coding style	169
17.2.1. Style issues	169
17.2.2. Source file templates	169
17.2.3. Commenting guidelines	170
17.2.3.1. Traceability comments	170
17.2.3.2. Assertions	170
17.3. Language usage rules	171
17.3.1. Formatting guidelines	171
17.3.2. Language usage rules	171
17.3.3. MISRA C	172
17.4. Naming conventions	172
17.5. Pitfalls	173
17.5.1. So-called self-documenting code	174
17.6. For more information	174
17.6.1. General topic search keywords	174
17.6.2. Recommended reading	174
18. The Cost of Nearly Full Resources	175
18.1. Overview	176
18.1.1. Importance of having slack in resources	176
18.1.2. Possible symptoms	176
18.1.3. Risks of too-full resources	177
18.2. How full is too full?	177
18.3. An example of hardware/software cost tradeoffs	179
18.4. Fixing overly full resources	183
18.5. Pitfalls	183
18.5.1. Optimizing to increase slack capacity	183
18.5.2. Over-zealous avoidance of optimization	184
18.6. For more information	184
18.6.1. General topic search keywords	184
18.6.2. References	184
19. Global Variables Are Evil	185
19.1. Overview	186
19.1.1. Importance of avoiding globals	186
19.1.2. Possible symptoms	186
19.1.3. Risks of using globals	187
19.2. Why are global variables evil?	187
19.2.1. Definition of global variables	187

19.2.2. Quasi-global variables	189
19.2.3. Global variable problem: implicit coupling	189
19.2.4. Global variable problem: bugs due to interference	190
19.2.5. Global variable problem: lack of mutex	191
19.2.6. Pointers to globals	191
19.3. Ways to avoid or reduce the risk of globals	191
19.3.1. Get a more capable processor	192
19.3.2. Move the global inside the procedure that needs it	192
19.3.3. Make a shared variable quasi-global instead of global	193
19.3.4. Make the global an object, and provide access methods	194
19.3.5. If you must use a global, document it well	196
19.4. Pitfalls	196
19.5. For more information	196
19.5.1. General topic search keywords	196
19.5.2. Recommended reading	196
19.5.3. Additional reading	197
20. Mutexes and Data Access Concurrency	199
20.1. Overview	200
20.1.1. Importance of using data access concurrency techniques	200
20.1.2. Possible symptoms	200
20.1.3. Risks of incorrect concurrency management	201
20.2. Data sharing hazards	201
20.2.1. Data updated in mid-read	201
20.2.2. Updates of data held in registers	203
20.2.3. Multiple writers	204
20.3. Data protection strategies	205
20.3.1. Volatile keyword	205
20.3.2. Atomic modifications and disabling interrupts	207
20.3.2.1. Atomic operations for multiple threads	207
20.3.2.2. Atomic operations for hardware access	208
20.3.2.3. Atomic operations within ISRs	208
20.3.3. Queues	208
20.3.3.1. Queues to avoid concurrency problems	209
20.3.3.2. Double buffering	210
20.3.4. Mutexes	210
20.4. Reentrant code	213
20.5. Pitfall	214
20.6. For more information	214
20.6.1. General topic search keywords	214
20.6.2. Recommended reading	214

21. Static Checking and Compiler Warnings	219
21.1. Overview	220
21.1.1. Importance of performing static checks	220
21.1.2. Possible symptoms	220
21.1.3. Risks of omitting static checks	220
21.2. The role of compiler warnings	221
21.2.1. Example warnings	221
21.2.1.1. Uninitialized variable	221
21.2.1.2. Suspicious language use	221
21.2.1.3. Risky type conversions	222
21.3. Lint and static analysis tools	223
21.4. Pitfalls – working around warnings	224
21.4.1. Warning pragmas	224
21.4.2. Switch statement default cases	224
21.5. For more information	225
21.5.1. General topic search keywords	225
21.5.2. Recommended reading	225
22. Peer Reviews	227
22.1. Overview	228
22.1.1. Importance of doing peer reviews	228
22.1.2. Possible symptoms	228
22.1.3. Risks of not doing peer reviews	228
22.2. Peer review effectiveness	228
22.3. Conducting peer reviews	230
22.3.1. What can be reviewed?	230
22.3.2. How do you perform peer reviews?	231
22.3.3. Review productivity	233
22.4. Pitfalls	234
22.5. For more information	234
22.5.1. General topic search keywords	234
22.5.2. Recommended reading	234
22.5.3. References	235
23. Testing and Test Plans	237
23.1. Overview	238
23.1.1. Importance of testing and test plans	238
23.1.2. Possible symptoms	238
23.1.3. Risks of inadequate test efforts	239
23.2. Overview of testing	239
23.2.1. When can we test?	239
23.2.2. What styles of testing can be used?	240

23.2.3. How much testing is enough?	242
23.3. Why do we test?	243
23.3.1. Testing to fix bugs (debug-oriented testing)	243
23.3.2. Testing to verify development process quality	244
23.3.3. The spectrum of testing goals	248
23.4. Test plans	249
23.4.1. What to test	249
23.4.2. How to test	250
23.4.3. How thoroughly do you test?	251
23.4.4. Test automation	251
23.4.5. Test result reporting	252
23.5. Pitfalls	252
23.6. For more information	252
23.6.1. General topic search keywords	252
23.6.2. Recommended reading	253
24. Issue Tracking & Analysis	255
24.1. Overview	256
24.1.1. Importance of tracking issues	256
24.1.2. Possible symptoms	256
24.1.3. Risks of not tracking issues	256
24.2. Issue tracking	257
24.2.1. Why track issues?	257
24.2.2. What information should be tracked?	257
24.3. What's a bug, really?	259
24.3.1. Is it a bug or an issue?	259
24.3.2. When does an issue count for reporting?	259
24.4. Defect and issue analysis	260
24.4.1. Issue reporting	260
24.4.2. Identification of bug farms	261
24.4.3. Identification of process problems	262
24.5. Pitfalls	262
24.6. For more information	262
24.6.1. General topic search keywords	262
24.6.2. Recommended reading	263
25. Run-Time Error Logs	265
25.1. Overview	266
25.1.1. Importance of run-time error logs	266
25.1.2. Possible symptoms	267
25.1.3. Risks of not keeping error logs	267
25.2. Error logging	267

25.2.1. How to log	267
25.2.2. What to log	268
25.2.3. Recording system reset	269
25.3. Error log analysis	270
25.3.1. Identifying failure causes	270
25.3.2. Identifying latent defects	271
25.4. For more information	271
25.4.1. General topic search keywords	271
25.4.2. Recommended reading	271
26. Dependability	275
26.1. Overview	276
26.1.1. Importance of achieving dependability	276
26.1.2. Possible symptoms	276
26.1.3. Risks of dependability gaps	276
26.2. Hardware dependability	276
26.2.1. Faults and failures	277
26.2.2. Reliability	277
26.2.2.1. Computing reliability	278
26.2.3. Reliability vs. MTBF	278
26.2.3.1. Serial component reliability	279
26.2.3.2. Redundancy and parallel component reliability	280
26.2.4. Availability	281
26.2.5. Dispatchability	283
26.2.6. Support for computing hardware reliability	284
26.3. Software dependability	284
26.3.1. High quality software	285
26.3.2. Software state scrubbing	286
26.3.3. Diverse software	287
26.3.3.1. Use different versions of third party software	288
26.3.3.2. Have a simple version as a fallback plan	288
26.3.3.3. Other approaches	289
26.4. Dependability plan	289
26.4.1. Dependability goal	290
26.4.2. Anticipated faults	291
26.4.3. Planned dependability approaches	292
26.5. Pitfalls	292
26.6. For more information	293
26.6.1. General topic search keywords	293
26.6.2. Recommended reading	293
26.6.3. Additional resources	293
26.6.4. References	293

27. Security	295
27.1. Overview	296
27.1.1. Importance of adequate security	296
27.1.2. Possible symptoms	296
27.1.3. Risks of inadequate security	297
27.2. Embedded security overview	297
27.2.1. Aspects of security	297
27.2.2. Embedded-specific attacks	298
27.2.3. Attack probability and motivation	299
27.2.4. Elements of typical countermeasures	300
27.3. Attack vectors (network and otherwise)	302
27.3.1. Internet connection	303
27.3.2. Factory-installed malware	303
27.3.3. Off-line spreading of malware	304
27.3.4. Indirectly Internet connected	305
27.3.5. Modem connected	306
27.3.6. Unauthorized users	306
27.4. Intellectual property protection	307
27.5. Security plan	308
27.5.1. Security plan elements	309
27.5.2. The <i>no security</i> plan	310
27.6. Pitfalls	310
27.7. For more information	312
27.7.1. General topic search keywords	312
27.7.2. Recommended reading	312
27.7.3. References	312
28. Safety	315
28.1. Overview	316
28.1.1. Importance of ensuring an appropriate level of safety	316
28.1.2. Possible symptoms	316
28.1.3. Risks of insufficient attention to safety	317
28.2. Embedded safety	317
28.2.1. Terminology	317
28.2.2. What, exactly, is safe software?	318
28.2.3. How big an issue is safety in your system?	319
28.2.4. Safety Integrity Levels	320
28.3. Safety analysis techniques	321
28.3.1. HAZOP (HAzard and OPerability)	321
28.3.2. PHA (Preliminary Hazard Analysis)	322
28.3.3. FTA (Fault Tree Analysis)	323
28.4. Avoiding safety critical software	325

28.4.1. Hardware implementation of critical functions	325
28.4.2. Hardware gating of critical functions	326
28.4.3. Hardware safety shutdown	327
28.4.4. Human oversight	328
28.5. Creating safety critical software	328
28.5.1. Safety plan	329
28.6. Pitfalls	329
28.7. For more information	330
28.7.1. General topic search keywords	330
28.7.2. Suggested reading	330
28.7.3. References	330
29. Watchdog Timers	333
29.1. Overview	334
29.1.1. Importance of using a watchdog	334
29.1.2. Possible symptoms	334
29.1.3. Risks of not using a watchdog	335
29.2. Watchdog timer overview	335
29.2.1. Watchdog timer operation	335
29.2.2. Watchdog effectiveness	336
29.2.3. Variations on watchdog timers	337
29.2.3.1. Software-settable watchdog parameters	337
29.2.3.2. Unlock sequence	337
29.2.3.3. Watchdog activated during startup	337
29.2.3.4. Windowed watchdog	337
29.2.4. Software watchdog	337
29.3. Good watchdog practices	338
29.4. Pitfalls	339
29.5. For more information	339
29.5.1. General topic search keywords	339
29.5.2. Recommended reading	340
30. System Reset	341
30.1. Overview	342
30.1.1. Importance of a good system reset strategy.	342
30.1.2. Possible symptoms	342
30.1.3. Risks of poor reset behavior	342
30.2. Ways to reset a system	342
30.2.1. External system resets	343
30.2.2. Internal system resets	343
30.3. Well behaved resets	344
30.3.1. Behavior during the reset	344

30.3.2. Behavior after the reset	344
30.3.3. Repeated resets	345
30.4. For more information	346
30.4.1. Recommended reading	346
31. Conclusion	347
About The Author	369