Better Embedded System Software

Philip Koopman
Empirical Approach To Content

- Based on 90+ industry design reviews
  - Real companies, products, problems
  - Some reviews were to save failing projects
  - Other reviews were to check up on otherwise good projects

- Professional book for practicing embedded system designers
  - Dug out the “red flag” issues from the review reports
  - Sorted, aggregated, sifted
  - 6 areas; 29 topics within those areas
  - Each chapter is 8-15 pages about a red flag topic
  - This is the stuff designers get wrong in real projects

- Also see my blog at:
  http://betterembsw.blogspot.com/
2. No Written Development Plan
   - And, often, no defined methodical development process

3. Insufficient paper trail
   - Things other than the code itself not written down

4. Creation of useless paper rather than useful paper
   - Creation of paper for paper’s sake (although this is unusual)
   - Belief that paper trail is a waste of time
Requirements & Architecture

5. **No written software requirements**
   - But often, thorough non-software requirements (digital HW, mechanical)

6. **Poor requirement quantification**
   - “Runs fast” or “user friendly”

7. **No traceability from requirements to acceptance test**
   - So you don’t know if the acceptance test actually tests everything that matters

8. **No non-functional requirements**
   - No stated targets for dependability, safety, security

9. **High requirements churn**
   - No change control process or formal change approvals; no freeze date

10. **No defined architecture**
    - Only a hardware-only block diagram

11. **Poor modularity**
    - Often just a big pile of code; multi-page Interrupt Service Routines
Design

12. No software design
   • Just implementation. Few flowcharts; usually no statecharts

13. No statecharts for state-intensive systems
   • Fuzzy understanding of behavior results in deeply nested, buggy “if” statements

14. No real time scheduling
   • Often ad hoc tasking approach

15. No methodical approach to user interface
   • Engineers take a shot without considering usability
Implementation

16. Heavy use of assembly language
   • Instead of writing code that is easy to compile or investing in good tools

17. Inconsistent coding style
   • Don’t use a style sheet or common style approach

18. Optimizing for hardware instead of total system cost
   • “Engineers are free” – spend time squeezing into the last 1% of memory

19. Use of many global variables
   • Some learned to program with unscoped languages (e.g., BASIC)

20. No use of concurrency management
   • E.g., no use of a mutex when warranted. In general no notion of time triggered
Verification & Validation

21. Poor static checking or compiler warnings
   • Warnings not generated or ignored

22. Ineffective peer reviews
   • Sometimes informal hall checks, but often nobody else even looks at code

23. No test plan
   • No methodical approach to testing. Often hardware-centric testing

24. No formal issue tracking
   • May not be a central bug log

25. No run time error logs
   • Or, sometimes, logs without enough useful information (e.g., no time stamps)
Critical System Properties

26. Dependability
   • Usually no dependability plan beyond “software shall never crash”

27. Security
   • Usually little or no security plan even for network-connected systems

28. Safety
   • Often no recognition that a system is somewhat safety critical (SIL 2 or SIL 3)

29. No or improper use of watchdog timers
   • Timers turned off or kicked from a hardware timer

30. Insufficient attention to system reset
   • May disrupt running system; may not anticipate multiple proximate resets