The Software Commandments V2.0

GENERAL

1. Commit to deliver, to help, to learn, and to improve
2. Be honest, be reliable, and work hard
3. When you mess up, own up!
4. Don't stay stuck, get help! If you're in a hole, stop digging. If in doubt, ask - and keep asking.
5. Communicate! Keep people informed - avoid surprises.
6. Write effectively, speak effectively - keep it short and make it matter to the recipients
7. Finish your project. Know your deadlines, hit them if you can, shout if you realize you can't
8. Solve problems, and help others to solve problems too – everyone learns faster that way
9. Clean and document as you go: don't leave a mess in the bathroom, the kitchen or your code
10. Think for yourself: challenge bad decisions, escalate if necessary - stay calm and polite
11. Make sure you *really* understand the objectives/target of the project
12. Keep a log book and use it every day
13. Always keep your own backups, and make sure you can restore

CODING

14. Code LESS: every line creates a work-chain
15. Read the documentation, use existing code if possible. Don't re-create what already exists.
16. If there isn’t a spec/definition, write a short one (with constraints, goals, non-goals) and agree it
17. If it isn’t specified as a requirement, don't code it. If the agreed spec is wrong, challenge it
18. Understand why code style is important, and comply
19. Get someone to review your code. Learn from the reviews, don't take it personally
20. Use version control - ideally GIT. Commit early, commit often, *with* descriptive comments
21. Quality code WORKS! All the time
22. The bug stops here! Be responsible for testing and fixing your own code
23. Remove uncertainty as fast as possible. Prototype, learn and iterate.
24. Set yourself doable targets every day and every week. If you're not making daily progress, shout

TESTING

25. Test behaviour, not structure. Don’t be gentle, break it!
26. Write down what you need to test before or at the same time as you write the code
27. Look for difficult, unexpected cases too
28. If it can’t be tested, it shouldn’t be there
29. If there's a test framework, use it. Aggressively replace/remove tests when things change
30. If someone else finds a bug after you’ve said it works, kick yourself!

DESIGN

31. Establish constraints and write them down
32. Plan to remove uncertainty fast - prototype and test against realistic scenarios ASAP
33. Design for testability, and design defensively: garbage in does not mean garbage out!
34. Design must be communicated clearly in writing
35. The first idea isn’t necessarily best: consider alternatives, discuss
36. Great design (and leadership) is methodology-independent
37. K I S S. Avoid any complex approach which will not be easily understood by later coders

ESTIMATING

38. When estimating, discuss the requirement with others – get perspective
39. If you’re unsure, say so. If you need more information, ask for it
40. Break the problem down into component tasks – get someone to challenge the breakdown
41. Compare like-for-like: try to check against previous productivity on similar work
42. Estimate the whole job, not just the coding.
43. State what your estimate includes, and what it excludes. And state that an estimate is a GUESS
44. Don’t be over-optimistic. It’s ALWAYS worse than you think.

MANAGING

45. Get the best people you can find/afford, and clear the way so they can be effective
46. Provide strong leadership – make sure your people know what they need to do, and when
47. Allow and foster open and honest communication. Listen and learn. React effectively and fairly
48. Encourage debate where necessary, but if things get bogged down, be a benevolent dictator
49. Set short timescales for targets and reviews – weekly is probably best.
50. If you're squeezed on time and cost, reduce scope – or you'll be late/low quality/over budget

In case things still don't work out after all of the above, keep making friends...