Practices of an Agile Developer

This card summarizes the guidelines from
Practices of an Agile Developer: Working in the Real World
(ISBN 0-9745140-8-X)
by Venkat Subramaniam and Andy Hunt.

For more information about THE PRAGMATIC BOOKSHELF please visit
`www.pragmaticprogrammer.com`.

TIPS 1 TO 11

1. Blame doesn't fix bugs.
Instead of pointing fingers, point to possible solutions. It's the positive outcome that counts. *(pg. 14)*

2. Don't fall for the quick hack.
Invest the energy to keep code clean and out in the open. *(pg. 18)*

3. Criticize ideas, not people.
Take pride in arriving at a solution rather than proving whose idea is better. *(pg. 22)*

4. Do what's right.
Be honest, and have the courage to communicate the truth. It may be difficult at times; that's why it takes courage. *(pg. 25)*

5. Keep up with changing technology.
You don't have to become an expert at everything, but stay aware of where the industry is headed, and plan your career and projects accordingly. *(pg. 31)*

6. Raise the bar for you and your team.
Use brown-bag sessions to increase everyone's knowledge and skills and help bring people together. Get the team excited about technologies or techniques that will benefit your project. *(pg. 33)*

7. Learn the new; unlearn the old.
When learning a new technology, unlearn any old habits that might hold you back. After all, there's much more to a car than just a horseless carriage. *(pg. 37)*

8. Keep asking Why.
Don't just accept what you're told at face value. Keep questioning until you understand the root of the issue. *(pg. 39)*

9. Tackle tasks before they bunch up.
It's easier to tackle common recurring tasks when you maintain steady, repeatable intervals between events. *(pg. 42)*

10. Let your customers decide.
Developers, managers, or business analysts shouldn't make business-critical decisions. Present details to business owners in a language they can understand, and let them make the decision. *(pg. 48)*

11. A good design is a map; let it evolve.
Design points you in the right direction. It's not the territory itself; it shouldn't dictate the specific route. Don't let the design (or the designer) hold you hostage. *(pg. 52)*

Copyright © 2006 Venkat Subramaniam and Andy Hunt, excerpted from *Practices of an Agile Developer*, ISBN 0-9745140-8-X

12. Choose technology based on need.
Determine your needs *first*, and then evaluate the use of technologies for those specific problems. Ask critical questions about the use of any technology, and answer them genuinely. *(pg. 55)*

13. Keep your project releasable at all times.
Ensure that the project is always compilable, runnable, tested, and ready to deploy at a moment's notice. *(pg. 59)*

14. Integrate early, integrate often.
Code integration is a major source of risk. To mitigate that risk, start integration early and continue to do it regularly. *(pg. 61)*

15. Deploy your application automatically from the start.
Use that deployment to install the application on arbitrary machines with different configurations to test dependencies. QA should test the deployment as well as your application. *(pg. 64)*

16. Develop in plain sight.
Keep your application in sight (and in the customers' mind) during development. Bring customers together and proactively seek their feedback using demos every week or two. *(pg. 69)*

17. Develop in increments.
Release your product with minimal, yet usable, chunks of functionality. Within the development of each increment, use an iterative cycle of one to four weeks or so. *(pg. 74)*

18. Estimate based on real work.
Let the team actually work on the current project, with the current client, to get realistic estimates. Give the client control over their features and budget. *(pg. 76)*

19. Use automated unit tests.
Good unit tests warn you about problems immediately. Don't make any design or code changes without solid unit tests in place. *(pg. 84)*

20. Use it before you build it.
Use Test Driven Development as a design tool. It will lead you to a more pragmatic and simpler design. *(pg. 88)*

21. Different makes a difference.
Run unit tests on each supported platform and environment combination, using continuous integration tools. Actively find problems before they find you. *(pg. 92)*

22. Create tests for core business logic.
Have your customers verify these tests in isolation, and exercise them automatically as part of your general test runs. *(pg. 94)*

23. Measure how much work is left.
Don't kid yourself—or your team—with irrelevant metrics. Measure the backlog of work to do. *(pg. 97)*

24. Every complaint holds a truth.
Find the truth, and fix the real problem. *(pg. 100)*

25. Write code to be clear, not clever.
Express your intentions clearly to the reader of the code. Unreadable code isn't clever. *(pg. 106)*

Copyright © 2006 Venkat Subramaniam and Andy Hunt, excerpted from *Practices of an Agile Developer*, ISBN 0-9745140-8-X

26. Comment to communicate.
Document code using well-chosen, meaningful names. Use comments to describe its purpose and constraints. Don't use commenting as a substitute for good code. *(pg. 112)*

27. Actively evaluate trade-offs.
Consider performance, convenience, productivity, cost, and time to market. If performance is adequate, then focus on improving the other factors. Don't complicate the design for the sake of perceived performance or elegance. *(pg. 115)*

28. Write code in short edit/build/test cycles.
It's better than coding for an extended period of time. You'll create code that's clearer, simpler, and easier to maintain. *(pg. 116)*

29. Develop the simplest solution that works.
Incorporate patterns, principles, and technology only if you have a compelling reason to use them. *(pg. 119)*

30. Keep classes focused and components small.
Avoid the temptation to build large classes or components or miscellaneous catchall classes. *(pg. 122)*

31. Tell, don't ask.
Don't take on another object's or component's job. Tell it what to do, and stick to your own job. *(pg. 125)*

32. Extend systems by substituting code.
Add and enhance features by substituting classes that honor the interface contract. Delegation is almost always preferable to inheritance. *(pg. 130)*

33. Maintain a log of problems and their solutions.
Part of fixing a problem is retaining details of the solution so you can find and apply it later. *(pg. 133)*

34. Treat warnings as errors.
Checking in code with warnings is just as bad as checking in code with errors or code that fails its tests. No checked-in code should produce any warnings from the build tools. *(pg. 137)*

35. Attack problems in isolation.
Separate a problem area from its surroundings when working on it, especially in a large application. *(pg. 140)*

36. Handle or propagate all exceptions.
Don't suppress them, even temporarily. Write your code with the expectation that things will fail. *(pg. 143)*

37. Present useful error messages.
Provide an easy way to find the details of errors. Present as much supporting detail as you can about a problem when it occurs, but don't bury the user with it. *(pg. 147)*

38. Use stand-up meetings.
Stand-up meetings keep the team on the same page. Keep the meeting short, focused, and intense. *(pg. 153)*

Copyright © 2006 Venkat Subramaniam and Andy Hunt, excerpted from *Practices of an Agile Developer*, ISBN 0-9745140-8-X

TIPS 39 TO 45

39. Good design evolves from active programmers.
Real insight comes from active coding. Don't use architects who don't code—they can't design without knowing the realities of your system. *(pg. 156)*

40. Emphasize collective ownership of code.
Rotate developers across different modules and tasks in different areas of the system. *(pg. 158)*

41. Be a mentor.
There's fun in sharing what you know—you gain as you give. You motivate others to achieve better results. You improve the overall competence of your team. *(pg. 161)*

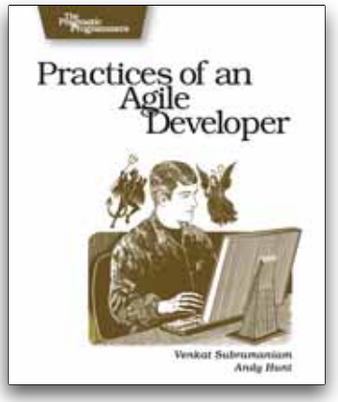
42. Give others a chance to solve problems.
Point them in the right direction instead of handing them solutions. Everyone can learn something in the process. *(pg. 164)*

43. Share code only when ready.
Never check in code that's not ready for others. Deliberately checking in code that doesn't compile or pass its unit tests should be considered an act of criminal project negligence. *(pg. 166)*

44. Review all code.
Code reviews are invaluable in improving the quality of the code and keeping the error rate low. If done correctly, reviews can be practical and effective. Review code after each task, using different developers. *(pg. 170)*

45. Keep others informed.
Publish your status, your ideas and the neat things you're looking at. Don't wait for others to ask you the status of your work. *(pg. 172)*

Venkat & Andy



www.PragmaticProgrammer.com/titles/pad

Copyright © 2006 Venkat Subramaniam and Andy Hunt, excerpted from *Practices of an Agile Developer*, ISBN 0-9745140-8-X