40 Essential Rules of Software Engineering

Presented by Anton Martyniuk

antondevtips.com

0. The best code is the one you don't write

Avoid unnecessary complexity by not introducing unneeded features or code. Less code means fewer bugs and easier maintenance.

1. You're not paid to write code - you're paid to solve problems

Focus on delivering valuable solutions that address actual business or user needs.
Code is just a tool to solve business problems

2. Everything is a trade-off. There's no "best" tool

Every tool or technology has strengths and weaknesses. Choose the most appropriate one based on your project's specific context and constraints.

4. Write meaningful commit messages

Good commit messages clearly communicate the purpose of changes, helping your team quickly understand the project's history.

5. First make it work, then make it pretty

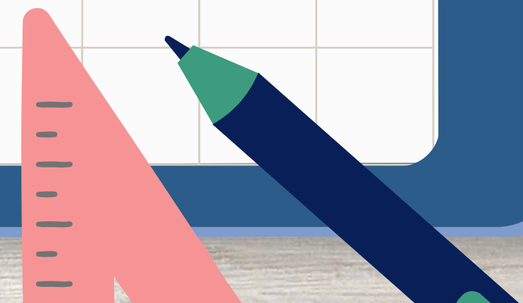
Prioritize functionality first, then refactor your code to improve readability, efficiency, and maintainability.

6. Ship early, iterate often

Delivering smaller features frequently allows rapid feedback, enabling adjustments and improvements to meet user expectations.

7. Estimations are never true

Estimates are inherently uncertain. Plan accordingly and continuously refine your estimates based on real-world experience.



8. Refactor continuously and incrementally

Regular, small improvements prevent code decay and maintain quality without disrupting project momentum.
Boy Scout Rule is amazing for continious code improvement

9. Code reviews improve more than just quality — they improve teams

Reviewing code collaboratively fosters knowledge sharing, promotes best practices, and strengthens team relationships.

10. Never trust user input

Always validate, sanitize, and securely handle user input to protect your application from vulnerabilities and errors.

11. Log precisely, not excessively

Focused logging helps debug issues effectively without overwhelming teams with irrelevant information.

12. Automate everything that can be automated

Automation reduces manual errors, saves time, and enhances reliability, especially for repetitive tasks.
Use AI to 10x your productivity

13. Complexity kills projects, don't over-engineer

Keep solutions as simple as possible to avoid unnecessary complexity that can kill project success.

14. Fix root causes, not symptoms

Addressing underlying issues prevents repeated problems and ensures long-term stability.

15. Measure first, optimize second

Identify actual performance bottlenecks through measurement before attempting optimization to ensure impactful improvements. In 90% of cases you will find out - you don't need any optimization

16. Minimize coupling, maximize cohesion

Ensure modules have clearly defined responsibilities and minimal interdependencies (coupling), improving maintainability and flexibility.
Code that changes together needs to be put together (cohesion)

17. Keep third-party dependencies minimal and well-managed

Reducing external dependencies minimizes risk, simplifies updates, and helps manage security vulnerabilities effectively.

18. Never hard-code sensitive information

Sensitive data should always be stored securely, separate from code, to protect against exposure or compromise.

19. Errors should fail loudly and immediately

Immediate and clear error handling facilitates quicker debugging and more robust systems.

20. Choose clarity over cleverness

Clear, straightforward code is easier to understand, maintain, and debug, benefiting the whole team.

21. Choose descriptive naming over explanatory comments

Well-chosen names eliminate confusion and reduce need for comments, making the code self-explanatory.

22. Favor composition over inheritance

Composition offers greater flexibility and clearer relationships between components than deep inheritance hierarchies.

23. Complexity doesn't scale; simplicity does

Simple solutions scale more easily, maintaining clarity and manageability as projects grow.

24. Respect the principle of least surprise

Components should behave predictably, minimizing confusion and improving usability.

25. Remove unused code without hesitation

Unused code adds unnecessary complexity.
Keep your codebase clean and focused.

26. Think and code in small, testable units

Smaller units facilitate easier testing, debugging, and modifications, promoting higher quality code.

27. Be consistent with your coding standards

Consistent coding standards enhance readability, simplify collaboration, and maintain project quality.

28. Abstraction should hide complexity, not create it

Effective abstraction simplifies understanding and usage, rather than introducing additional complexity.

29. Favor explicit over implicit

Explicit code clearly communicates intent, reducing misunderstandings and enhancing maintainability.

30. Every line of code should justify its existence

Ensure each line of code contributes meaningfully to functionality or clarity.
Remove redundant code ruthlessly

31. Always strive to understand the business behind the code

Knowing the broader context and objectives ensures your solutions effectively meet business needs.

32. Keep Pull Requests small and manageable

Smaller pull requests simplify reviews, accelerate integration, and reduce potential issues.

33. Invest in CI/CD right from the start

Continuous Integration and Continuous Deployment improve delivery speed, quality, and consistency.
Invest early so you don’t have surpises later

34. Design APIs that are easy to use correctly and hard to misuse

Intuitive and robust API design reduces errors, simplifies integration, and increases developer satisfaction.

You can use HATEOAS as GPS for your APIs

35. Document why, not just what

Clearly documenting reasoning behind decisions provides invaluable context for future development.

36. Forget about "this works on my machine"

Ensure consistent environments and thorough testing to avoid platform-specific issues.

37. Be aware of technical debt; repay it incrementally

Regularly addressing technical debt maintains long-term productivity and codebase health.

38. Balance YAGNI ("You Aren’t Gonna Need It") with thoughtful design

Avoid speculative code but thoughtfully anticipate foreseeable future needs.

39. Ask for help when you're stuck

Seeking assistance promptly accelerates problem resolution and promotes a healthy, collaborative culture.

40. Never stop learning and questioning your assumptions

Continuous improvement and curiosity drive innovation and help you adapt to evolving technologies and practices.

Thank You!

Repost to help others and follow me

Presented by  Anton Martyniuk

antondevtips.com