Tip 1

Care About Your Craft

Why spend your life developing software unless you care about doing it well?

Tip #2

Think! About Your Work

Turn off the autopilot and take control. Constantly critique and appraise your work.

Tip #3

You Have Agency

It’s your life. Grab hold of it and make it what you want.

Tip #4

Provide Options, Don’t Make Lame Excuses

Instead of excuses, provide options. Don’t say it can’t be done; explain what can be done.

Tip #5

Don’t Live with Broken Windows

Fix bad designs, wrong decisions, and poor code when you see them.

Tip #6

Be a Catalyst for Change

You can’t force change on people. Instead, show them how the future might be and help them participate in creating it.

Tip #7

Remember the Big Picture

Don’t get so engrossed in the details that you forget to check what’s happening around you.

Tip #8

Make Quality a Requirements Issue

Involve your users in determining the project’s real quality requirements.

Tip #9

Invest Regularly in Your Knowledge Portfolio

Make learning a habit.

Tip #10

Critically Analyze What You Read and Hear

Don’t be swayed by vendors, media hype, or dogma. Analyze information in terms of you and your project.

Tip #11

English is Just Another Programming Language

Treat English as Just Another Programming Language. Write documents as you would write code: honor the DRY principle, ETC, automation, and so on.

Tip #12

It’s Both What You Say and the Way You Say It

There’s no point in having great ideas if you don’t communicate them effectively.

Tip #13

Build Documentation In, Don’t Bolt It On

Documentation created separately from code is less likely to be correct and up to date.

Tip #14

Good Design Is Easier to Change Than Bad Design

A thing is well designed if it adapts to the people who use it. For code, that means it must adapt by changing.

Tip #15

DRY—Don’t Repeat Yourself

Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.

Tip #16

Make It Easy to Reuse

If it’s easy to reuse, people will. Create an environment that supports reuse.

Tip #17

Eliminate Effects Between Unrelated Things

Design components that are self-contained, independent, and have a single, well-defined purpose.

Tip #18

There Are No Final Decisions

No decision is cast in stone. Instead, consider each as being written in the sand at the beach, and plan for change.

Tip #19

Forgo Following Fads

Neal Ford says, “Yesterday’s Best Practice Becomes Tomorrow’s Antipattern.” Choose architectures based on fundamentals, not fashion.

Tip #20

Use Tracer Bullets to Find the Target

Tracer bullets let you home in on your target by trying things and seeing how close they land.

Tip #21

Prototype to Learn

Prototyping is a learning experience. Its value lies not in the code you produce, but in the lessons you learn.

Tip #22

Program Close to the Problem Domain

Design and code in the language of the problem domain.

Tip #23

Estimate to Avoid Surprises

Estimate before you start. You’ll spot potential problems up front.

Tip #24

Iterate the Schedule with the Code

Use experience you gain as you implement to refine the project time scales.

Tip #25

Keep Knowledge in Plain Text

Plain text won’t become obsolete. It helps leverage your work and simplifies debugging and testing.

Tip #26

Use the Power of Command Shells

Use the shell when graphical user interfaces don’t cut it.

Tip #27

Achieve Editor Fluency

An editor is your most important tool. Know how to make it do what you need, quickly and accurately.

Tip #28

Always Use Version Control

Version control is a time machine for your work; you can go back.

Tip #29

Fix the Problem, Not the Blame

It doesn’t really matter whether the bug is your fault or someone else’s—it is still your problem, and it still needs to be fixed.

Tip #30

Don’t Panic

This is true for galactic hitchhikers and for developers.

Tip #31

Failing Test Before Fixing Code

Create a focused test that reveals the bug before you try fixing it.

Tip #32

Read the Damn Error Message

Most exceptions tell both what failed and where it failed. If you’re lucky you might even get parameter values.

Tip #33

“select” Isn’t Broken

It is rare to find a bug in the OS or the compiler, or even a third-party product or library. The bug is most likely in the application.

Tip #34

Don’t Assume It—Prove It

Prove your assumptions in the actual environment—with real data and boundary conditions.

Tip #35

Learn a Text Manipulation Language

You spend a large part of each day working with text. Why not have the computer do some of it for you?

Tip #36

You Can’t Write Perfect Software

Software can’t be perfect. Protect your code and users from the inevitable errors.

Tip #37

Design with Contracts

Use contracts to document and verify that code does no more and no less than it claims to do.

Tip #38, pg. 113:

Crash Early

A dead program normally does a lot less damage than a crippled one.

Tip #39

Use Assertions to Prevent the Impossible

If it can’t happen, use assertions to ensure that it won’t. Assertions validate your assumptions. Use them to protect your code from an uncertain world.

Tip #40

Finish What You Start

Where possible, the function or object that allocates a resource should be responsible for reallocating it.

Tip #41

Act Locally

Keep the scope of mutable variables and open resources short and easily visible.

Tip #42

Take Small Steps—Always

Small steps always; check the feedback; and adjust before proceeding.

Tip #43

Avoid Fortune-Telling

Only look ahead as far as you can see.

Tip #44

Decoupled Code Is Easier to Change

Coupling ties things together, so that it’s harder to change just one thing.

Tip #45

Tell, Don’t Ask

Don’t get values from an object, transform them, and then stick them back. Make the object do the work.

Tip #46

Don’t Chain Method Calls

Try not to have more than one dot when you access something.

Tip #47

Avoid Global Data

It’s like adding an extra parameter to every method.

Tip #48

If it’s Important Enough To Be Global, Wrap It in an API

But only if you really, really want it to be global.

Tip #49

Programming Is About Code, But Programs Are About Data

All programs transform data, converting an input into an output. Start designing using transformations.

Tip #50

Don’t Hoard State; Pass It Around

Don’t hang on to data within a function or module. Take one down and pass it around.

Tip #51

Don’t Pay Inheritance Tax

Consider alternatives that better fit your needs, such as interfaces, delegation, or mixins

Tip #52

Prefer Interfaces to Express Polymorphism

Interfaces make polymorphism explicit without the coupling introduced by inheritance.

Tip #53

Delegate to Services: Has-A Trumps Is-A

Don’t inherit from services: contain them.

Tip #54

Use Mixins to Share Functionality

Mixins add functionality to classes without the inheritance tax. Combine with interfaces for painless polymorphism.

Tip #55

Parameterize Your App Using External Configuration

When code relies on values that may change after the application has gone live, keep those values external to the app. When you application will run in different environments, and potentially for different customers, keep the environment and customer specific values outside the app.

Tip #56

Analyze Workflow to Improve Concurrency

Exploit concurrency in your user’s workflow.

Tip #57

Shared State Is Incorrect State

Shared state opens a large can of worms that can often only be fixed by rebooting.

Tip #58

Random Failures Are Often Concurrency Issues

Variations in timing and context can expose concurrency bugs, but in inconsistent and irreproducible ways.

Tip #59

Use Actors For Concurrency Without Shared State

Use Actors to manage concurrent state without explicit synchronization.

Tip #60

Use Blackboards to Coordinate Workflow

Use blackboards to coordinate disparate facts and agents, while maintaining independence and isolation among participants.

Tip #61

Listen to Your Inner Lizard

When it feels like your code is pushing back, it’s really your subconscious trying to tell you something’s wrong.

Tip #62

Don’t Program by Coincidence

Rely only on reliable things. Beware of accidental complexity, and don’t confuse a happy coincidence with a purposeful plan.

Tip #63

Estimate the Order of Your Algorithms

Get a feel for how long things are likely to take before you write code.

Tip #64

Test Your Estimates

Mathematical analysis of algorithms doesn’t tell you everything. Try timing your code in its target environment.

Tip #65

Refractor Early, Refractor Often

Just as you might weed and rearrange a garden, rewrite, rework, and re-architect code when it needs it. Fix the root of the problem.

Tip #66

Testing Is Not About Finding Bugs

A test is a perspective into your code, and gives you feedback about its design, api, and coupling.

Tip #67

A Test Is the First User of Your Code

Use its feedback to guide what you do.

Tip #68

Build End-To-End, Not Top-Down or Bottom Up

Build small pieces of end-to-end functionality, learning about the problem as you go

Tip #69

Design to Test

Start thinking about testing before you write a line of code.

Tip #70

Test Your Software, or Your Users Will

Test ruthlessly. Don’t make your users find bugs for you.

Tip #71

Use Property-Based Tests to Validate Your Assumptions

Property-based tests will try things you never thought to try, and exercise your code in ways is wasn’t meant to be used.

Tip #72

Keep It Simple and Minimize Attack Surfaces

Complex code creates a breeding ground for bugs and opportunities for attackers to exploit.

Tip #73

Apply Security Patches Quickly

Attackers deploy exploits as quick as they can, you have to be quicker.

Tip #74

Name Well; Rename When Needed

Name to express your intent to readers, and rename as soon as that intent shifts.

Tip #75

No One Knows Exactly What They Want

They might know a general direction, but they won’t know the twists and turns.

Tip #76

Programmers Help People Understand What They Want

Software development is an act of co-creation between users and programmers.

Tip #77

Requirements Are Learned in a Feedback Loop

Understanding requirements requires exploration and feedback, so the consequences of decisions can be used to refine the initial ideas.

Tip #78

Work with a User to Think Like a User

It’s the best way to gain insight into how the system will really be used.

Tip #79

Policy Is Metadata

Don’t hardcode policy into a system; instead express it as metadata used by the system.

Tip #80

Use a Project Glossary

Create and maintain a single source of all the specific terms and vocabulary for a project.

Tip #81

Don’t Think Outside the Box—Find the Box

When faced with an impossible problem, identify the real constraints. Ask yourself: “Does it have to be done this way? Does it have to be done at all?”

Tip #82

Don’t Go into the Code Alone

Programming can be difficult and demanding. Take a friend with you.

Tip #83

Agile Is Not a Noun; Agile Is How You Do Things

Agile is an adjective: it’s how you do something.

Tip #84

Maintain Small Stable Teams

Teams should be small and stable, where everyone trusts each other and depends on each other.

Tip #85

Schedule It to Make It Happen

If you don’t schedule it, it’s not going to happen. Schedule reflection, experimentation, learning and skills improvement.

Tip #86

Organize Fully Functional Teams

Organize Around Functionality, Not Job Functions. Don’t separate UI/UX designers from coders, frontend from backend, testers from data modelers, design from deployment. Build teams so you can build code end-to-end, incrementally and iteratively.

Tip #87

Do What Works, Not What’s Fashionable

Don’t adopt a development method or technique just because other companies are doing it. Adopt what works for your team, in your context.

Tip #88

Deliver When Users Need It

Don’t wait weeks or months to deliver just because your process demands it.

Tip#89

Use Version Control to Drive Builds, Tests, and Releases

Use commits or pushes to trigger builds, tests, releases. Use a version control tag to deploy to production.

Tip #90

Test Early, Test Often, Test Automatically

Tests that run with every build are much more effective than test plans that sit on a shelf.

Tip #91

Coding Ain’t Done ’Til All the Tests Run

’Nuff said.

Tip #92

Use Saboteurs to Test Your Testing

Introduce bugs on purpose in a separate copy of the source to verify that testing will catch them.

Tip #93

Test State Coverage, Not Code Coverage

Identify and test significant program states. Testing just lines of code isn’t enough.

Tip #94

Find Bugs Once

Once a human tester finds a bug, it should be the last time a human tester finds that bug. Automatic tests should check for it from then on.

Tip #95

Don’t Use Manual Procedures

A computer will execute the same instructions, in the same order, time after time.

Tip #96

Delight Users, Don’t Just Deliver Code

Develop solutions that produce business value for your users and delight them every day.

Tip #97

Sign Your Work

Artisans of an earlier age were proud to sign their work. You should be, too.

Tip #98

First, Do No Harm

Failure is inevitable. Make sure no one will suffer because of it.

Tip #99

Don’t Enable Scumbags

Because you risk becoming one, too.

Tip #100

It’s Your Life. Share it. Celebrate it. Build it. AND HAVE FUN!

Enjoy this amazing life we have, and do great things.