No, vibe coding does not create tech debt.

# Bad coders do.

Vibe coding just lets bad coders create tech debt **100x faster**.

**Khur Boon Kgim**

AI-Powered Dad and Technopreneur

# Vibe-Coding With Discipline

7 principles to stop building slopware with AI

If you're vibe-coding without discipline, you're not building software — you're building **slopware**.

# What Is Slopware?

Slopware looks fine on the surface. It "runs". It "demos well".

But the moment it faces real customers, real data, real traffic...

**It breaks.**

And with AI coding assistants, you can now generate **more slopware faster than ever before**.

# The Solution

To prevent slopware, you need these **8 principles of good coding**, rewritten for the AI era.

- ✓ Don't Repeat Yourself (DRY)
- ✓ Single Responsibility
- ✓ Pyramid Principle
- ✓ Minimise Coupling
- ✓ Set Up Guardrails
- ✓ Colocate Related Code
- ✓ Favour Pure Functions
- ✓ Refactor Continuously

Let's break down each one...

1

# Don't Repeat Yourself

Avoid duplication of logic, knowledge, or intent

# DRY: What It Means

Each behaviour should live in **one authoritative place** — never duplicated across files.

### SYMPTOMS WHEN VIOLATED

- ✗ AI edits similar-looking code in many places
- ✗ You fix one place, but other similar places stay broken
- ✗ AI generates the same logic in every new file
- ✗ "Didn't I already ask you to fix this?"

# DRY: Why It Matters With AI

AI often rewrites similar code in multiple places unless you explicitly enforce DRY.

Duplication leads to **inconsistent behaviour** and bugs that hide across "near-identical" copies.

## EXTRACT COMMON PATTERNS INTO

- ✓ Utility functions
- ✓ Domain services
- ✓ Reusable UI components
- ✓ Shared schemas & validation rules
- ✓ Middleware
- ✓ Config files

# DRY: How To Apply

**DO**

Centralise logic

**DON'T**

Let AI silently copy-paste logic across files

**AI AGENT RULE**

"When similar logic appears, extract it into a reusable shared component or function."

**FOLLOW-UP PROMPT**

"Identify duplicate logic and refactor it using DRY principles."

# 2

# Single Responsibility

Function → Class → Module → File →
Schema → Table

# SRP: What It Means

Every unit of code — and every table — should do **exactly one thing**.

### SYMPTOMS WHEN VIOLATED

- X AI keeps adding to the same file until it's 1000+ lines
- X You paste a whole file as context just to change one thing
- X AI hallucinates because the file is too long to fit in context

# SRP: Why It Matters With AI

### AI LOVES GENERATING "GOD FILES" THAT MIX

- ✗ API calls
- ✗ Validation
- ✗ Business rules
- ✗ Transformations
- ✗ I/O
- ✗ Error handling

### SRP MAKES YOUR CODE

- ✓ Predictable
- ✓ Testable
- ✓ Debuggable

# SRP: The AI Era Bonus

**Small, focused files reduce token cost.**

You only need to provide the relevant file as context — not a thousand-line blob.

### THIS REDUCES

- ✓ Token usage
- ✓ Hallucination risk
- ✓ Context overflow
- ✓ Repair cost

# SRP: How To Apply

**DO**

Keep files tight and scoped

**DON'T**

Mix unrelated responsibilities

**AI AGENT RULE**

"Split code into small, single-responsibility functions and modules."

**FOLLOW-UP PROMPT**

"Refactor this into smaller, single-responsibility modules."

3

# Pyramid Principle

Intent → Supporting Logic →
Implementation Details

# Pyramid: What It Means

Both files and folders should be structured **top-down**:

> **Top** = high-level orchestration

> **Middle** = supporting logic

> **Bottom** = helpers and details

### SYMPTOMS WHEN VIOLATED

X AI puts helper functions before the main logic

X You can't find where the flow starts

X AI reads the file wrong because structure is confusing

X Every prompt needs "look at the bottom of the file"

# Pyramid: Why It Matters With AI

AI writes code in whatever order it "thinks".

**Humans understand code top-down.**

### APPLIES TO FILE STRUCTURE

Top of the file = high-level orchestration.
The first section should answer:

- ✓ What does this file do?
- ✓ What is the main flow?

Put helper functions at the bottom.

# Pyramid: How To Apply

### APPLIES TO FOLDER STRUCTURE TOO

Top-level folders should represent high-level intent:

`api/` `domain/` `services/` `infrastructure/` `utils/`

Low-level details belong deeper.

**DO**

Make files and folders self-explanatory

**DON'T**

Bury orchestration among helpers

### AI AGENT RULE

"Place orchestration at the top. Move helper functions to the bottom. Organise folders around high-level intent."

### FOLLOW-UP PROMPT

"Restructure this file so orchestration comes first and helpers last."

# 4

# Minimise Coupling

Loose connections → flexible, safer systems

# Coupling: What It Means

Modules should interact through **clear boundaries**, not tangled dependencies.

### SYMPTOMS WHEN VIOLATED

- X AI needs to read 10 files just to make one change
- X One small fix causes errors in unrelated modules
- X AI suggests changes that break things you didn't mention
- X Your context window fills up before AI understands the code

# Coupling: Why It Matters With AI

### WHEN CODE IS STRONGLY COUPLED

- X Changing one part breaks unobvious other parts
- X Debugging becomes unpredictable
- X Refactoring becomes risky
- X Need to understand many adjacent code to make a small change

### AND CRITICALLY

AI needs to read **more files** to fix tightly coupled code.

# Coupling: The Cost Equation

**MORE COUPLING = MORE CONTEXT =**

- X Higher token cost
- X Higher hallucination risk
- X Slower iteration
- X More brittle fixes

**Loose coupling = smaller context = cheaper, safer AI-assisted development.**

# Coupling: How To Apply

**DO**

Design clear boundaries

**DON'T**

Let logic leak across layers or share global state

**AI AGENT RULE**

"Keep modules loosely coupled with explicit interfaces. Avoid hidden dependencies."

**FOLLOW-UP PROMPT**

"Decouple these modules and propose cleaner interfaces."

5

# Set Up Guardrails

Typecheck, Tests, Linter, Formatter

# Guardrails: What It Means

Your codebase **must** have:

- ✓ Type checking
- ✓ Unit tests
- ✓ Lint rules
- ✓ Formatting rules

## SYMPTOMS WHEN VIOLATED

- X AI generates code that "looks right" but crashes at runtime
- X You only discover bugs when users report them
- X AI keeps making the same type of mistake
- X No way to verify if AI output actually works

# Guardrails: Why They Matter With AI

## THESE TOOLS LET AI CHECK ITSELF

AI generates code → tools produce errors → AI fixes errors.

> **Without these, the AI is coding blind.**

## TESTABLE CODE IS USUALLY LOOSELY COUPLED CODE

If you can't test it easily, it usually means:

- X Hidden state
- X Tight coupling
- X Unclear boundaries
- X Mixed responsibilities

**Tests reveal architecture quality.**

# Guardrails: How To Apply

These tools are your **non-negotiable guardrails**.

**DO**

Enforce checks before merge

**DON'T**

Deploy raw AI output

**AI AGENT RULE**

"Ensure code passes strict type checks, lint rules, formatting, and meaningful tests."

**FOLLOW-UP PROMPT**

"Fix all type errors, add tests, and ensure the code passes linting."

6

# Colocate Related Code

Files that change together should live together

# Colocation: What It Means

Group code by **feature**, not by type. Keep related files close — components, tests, styles, types, and utilities in the same folder.

### SYMPTOMS WHEN VIOLATED

- X AI updates one file but misses related files in other folders
- X You have to tell AI "also check the tests folder" and "also check the types folder"
- X Feature changes require jumping across 5 different directories
- X AI forgets to update related code because it's too far away

# Colocation: Why It Matters With AI

### WHEN RELATED CODE IS SCATTERED

- ✗ AI misses files that should change together
- ✗ You need to specify every related path manually
- ✗ Context becomes fragmented and incomplete

### WHEN RELATED CODE IS COLOCATED

- ✓ AI sees everything it needs in one place
- ✓ Changes are complete and consistent
- ✓ Less "I forgot to update X"

# Colocation: How To Apply

**DO**

Put component, test, style, and types in the same folder

**DON'T**

Scatter related files across src/components/, src/tests/, src/styles/, src/types/

**AI AGENT RULE**

"Keep related code together. Group by feature, not by file type."

**FOLLOW-UP PROMPT**

"Reorganise this codebase to colocate related files by feature."

7

# Favour Pure Functions

Same input → same output, no side effects

# Pure Functions: What It Means

A pure function always returns the **same output for the same input** and has **no side effects** — no database writes, no API calls, no global state changes.

## SYMPTOMS WHEN VIOLATED

- X AI generates functions that secretly modify global state
- X Tests pass sometimes, fail other times
- X AI can't predict what a function does without reading everything it touches
- X Debugging requires tracing through 10 files of side effects

# Pure Functions: Why It Matters With AI

### PURE FUNCTIONS ARE EASY FOR AI TO

- ✓ Understand — input/output is explicit
- ✓ Test — no mocking required
- ✓ Refactor — no hidden dependencies
- ✓ Compose — plug them together safely

### IMPURE FUNCTIONS FORCE AI TO

- X Read surrounding context to understand behaviour
- X Track hidden state changes
- X Guess at side effects

# Pure Functions: Examples

Access Control Logic

**IMPURE**

```
function canAccess(userId) {
  const user = db.find(userId);
  logAccess(userId);
  // ... access control logic
}
```

**PURE**

```
function canAccess(user, role) {
  // ... access control logic
}

const user = db.find(userId);
if (canAccess(user, 'admin')) { logAccess(userId); }
```

**Impure functions** → slow integration tests

**Pure functions** → fast unit tests

# Pure Functions: How To Apply

**DO**

Isolate side effects at the edges

**DON'T**

Mix I/O, state, and logic in the same function

**AI AGENT RULE**

"Keep business logic in pure functions. Push side effects (database, API, file I/O) to the outer layers."

**FOLLOW-UP PROMPT**

"Refactor this to separate pure logic from side effects."

8

# Refactor Continuously

Not once a quarter — continuously

# Refactor: What It Means

Refactoring improves structure **without changing behaviour**.

### SYMPTOMS WHEN VIOLATED

- X AI-generated code piles up without cleanup
- X Each new feature makes the codebase harder to prompt
- X You spend more time explaining context than coding
- X Eventually AI can't help because the mess is too big

# Refactor: Why It Matters With AI

AI produces code quickly — and **messy code quickly**.

Without regular cleanup, small shortcuts turn into monstrous tangles.

### IF YOU DELAY REFACTORING

- X Logic becomes tangled
- X Responsibilities blur
- X Files swell
- X Everything intertwines
- X You end up with spaghetti code

> **Spaghetti code is expensive to fix and painful to untangle.**

# Refactor: Use Principles 1-7

## YOUR REFACTORING CHECKLIST

- ✓ **DRY:** Extract duplicates into shared components
- ✓ **SRP:** Split large files into focused modules
- ✓ **Pyramid:** Move orchestration to the top
- ✓ **Coupling:** Decouple tangled modules
- ✓ **Guardrails:** Add missing tests and types
- ✓ **Colocation:** Group related files together
- ✓ **Pure Functions:** Separate logic from side effects

# Refactor: How To Apply

## Fix the mess while it's still small.

**DO**

Tidy constantly

**DON'T**

Allow entropy to accumulate

**AI AGENT RULE**

"After generating code, propose refactoring: break functions apart, improve naming, and clarify structure."

**FOLLOW-UP PROMPT**

"Refactor this code using DRY, SRP, Pyramid, Coupling, Guardrails, Colocation, and Pure Functions principles."

# The 8 Principles

> 1. **DRY:** One authoritative place for each behaviour
> 2. **SRP:** Every unit does exactly one thing
> 3. **Pyramid:** Intent first, details last
> 4. **Coupling:** Loose connections, clear boundaries
> 5. **Guardrails:** Types, tests, lint, format
> 6. **Colocation:** Related code lives together
> 7. **Pure Functions:** Same input, same output, no side effects
> 8. **Refactor:** Clean up continuously using 1-7

# The Bottom Line

AI helps you code faster.

These principles ensure you don't produce **slopware** faster.

**Vibe-coding with discipline is the future.**

Vibe-coding without it burns time, money, tokens — and eventually, your entire system.

# Let's Connect

Which principle do you find hardest to enforce with AI?

> Follow for more AI coding insights
> Repost to help others avoid slopware
> Save this for your next AI coding session

#VibeCoding #AICode #SoftwareEngineering #CleanCode #DeveloperProductivity

**Khur Boon Kgim**

AI-Powered Dad and Technopreneur