From Coder to Software Engineer

Simple 6-Month Plan

Start Date: January 03, 2026

Just 1 hour per day + weekends

The Simple Truth

The difference between a coder and a software engineer isn't how much code you write. It's about **HOW you think** before, during, and after writing code.

Coder Thinks	Engineer Thinks
How do I make this work?	Why does this problem exist?
It works = Done	Will this work in 6 months?
Write code quickly	Design first, code second
My task only	Impact on the whole system

Your Daily Commitment

1 hour per day during the week + **2-3 hours on weekends**. That's it. Consistency beats intensity.

The 6-Month Plan

Month 1: Think Before You Code

Goal: Stop coding immediately. Start thinking first.

Every Day (30 minutes):

Before writing ANY code, answer these 3 questions in a document:

1. What problem am I solving?
2. What are 2-3 different ways to solve it?
3. Which way is best and WHY?

At end of day: Look at code you wrote. Ask: 'Is this clear? Could it be simpler?'

Weekend (2 hours):

- Read 'Clean Code' Chapter 1-2 (meaningful names, functions)
- Pick one old piece of code and make it cleaner

By End of Month:

✓ 30 documented 'Think Before Code' exercises

✓ Read Clean Code Chapters 1-4

✓ Refactored 4-5 old functions/classes

Month 2: Learn Design Principles

Goal: Understand SOLID principles and apply them.

Every Day (30 minutes):

- Week 1: Learn Single Responsibility Principle (one class = one job)
- Week 2: Learn Open/Closed Principle (open for extension, closed for modification)
- Week 3: Learn Dependency Inversion (depend on interfaces, not concrete classes)
- Week 4: Practice - find code that breaks these rules and fix it

Weekend (3 hours):

- Watch Laracasts 'SOLID Principles in PHP' (1 principle per weekend)
- Refactor one feature applying what you learned

By End of Month:

- ✓ Understand all 5 SOLID principles
- ✓ Refactored 5+ classes following SOLID
- ✓ Can explain each principle simply

Month 3: Design Patterns

Goal: Learn common patterns that solve common problems.

One Pattern Per Week:

- Week 1: Repository Pattern (separate data access from business logic)
- Week 2: Strategy Pattern (different algorithms, same interface)
- Week 3: Observer Pattern (events and listeners)
- Week 4: Factory Pattern (flexible object creation)

Daily (30 minutes):

- Read about the pattern of the week
- Look for where you could apply it in your code
- Implement it in one place

Weekend (3 hours):

- Read refactoring.guru for the pattern
- Implement the pattern in a real feature

By End of Month:

- ✓ Implemented 4 design patterns
- ✓ Know when to use each pattern
- ✓ Can explain patterns to others

Month 4: Architecture & Structure

Goal: Think about the big picture and system design.

Daily (40 minutes):

Before building any feature: Draw a simple diagram showing:

- What components are needed?

- How do they talk to each other?

- Where does data come from and go to?

Think: How would this work if we had 100x more users?

Weekend (3 hours):

• Read 'Clean Architecture' - key chapters

• Draw architecture diagrams for your project

• Identify what could be better structured

By End of Month:

✓ Can draw architecture diagrams

✓ Understand layered architecture

✓ Think about scalability naturally

Month 5: Quality & Testing

Goal: Write code that's reliable and maintainable.

Daily (40 minutes):

Write tests BEFORE writing code (TDD):

1. Write a failing test
2. Write minimum code to pass
3. Refactor to make it clean

Use PHPStan or similar tools to catch errors before running code

Weekend (3 hours):

- Add tests to old code that doesn't have any
- Set up code quality tools (PHPStan, PHP CS Fixer)
- Refactor code based on quality tool suggestions

By End of Month:

✓ Write tests for all new code

✓ Use static analysis tools regularly

✓ Code quality is measurably better

Month 6: Performance & Scale

Goal: Make code fast and scalable.

Daily (40 minutes):

- Week 1-2: Find slow database queries. Fix N+1 problems. Add indexes.
- Week 3: Learn and implement caching (Redis) for expensive operations
- Week 4: Implement queues for slow tasks (emails, reports, etc.)

Weekend (3 hours):

- Profile your application, find bottlenecks
- Learn about horizontal vs vertical scaling
- Write a plan: How would my app handle 10x traffic?

By End of Month:

- ✓ Optimized slow queries
- ✓ Implemented caching
- ✓ Understand scalability concepts

Your Simple Daily Routine

Morning (15 minutes)

Time	Activity
5 min	Review what you learned yesterday
10 min	Read/watch current month's learning material

During Work

Action	What to Do
Before coding	Think: Problem? Solutions? Best approach?
While coding	Apply current month's principles/patterns
Before commit	Review: Is it clear? Simple? Tested?

Evening (30 minutes)

Time	Activity
15 min	Practice current month's skill on real code
10 min	Write down what you learned today
5 min	Plan tomorrow's focus

What You Need to Learn

Books (Read in Order):

1. 'Clean Code' by Robert Martin - Months 1-2
2. 'Design Patterns' by Gang of Four - Month 3
3. 'Clean Architecture' by Robert Martin - Month 4
4. 'Refactoring' by Martin Fowler - Month 5

Free Online Resources:

- Laracasts - SOLID Principles in PHP (Month 2)
- Refactoring.guru - Design Patterns (Month 3)
- YouTube: CodeAesthetic channel (Throughout)
- Martin Fowler's blog - martinfowler.com (Throughout)

Tools to Use:

- PHPStan - Find bugs without running code
- PHP CS Fixer - Auto-format your code
- Draw.io - Create architecture diagrams (free)

Weekly Progress Check

Every Friday (20 minutes):

- ■ What did I learn this week?
- ■ Did I apply it to real code?
- ■ What will I focus on next week?
- ■ Rate my code quality this week (1-10)
- ■ What can I improve?

Monthly Review (30 minutes):

- ■ Did I complete this month's goals?
- ■ How is my code different from last month?
- ■ What do I need to practice more?
- ■ Am I ready for next month?

Remember

1. Consistency > Intensity
1 hour every day beats 7 hours on Sunday

2. Apply, Don't Just Learn
Reading about patterns is useless if you don't use them

3. Think First, Code Second
The best code is often the code you don't write

4. Be Patient
6 months will pass anyway. Make them count.

5. You Don't Need to Be Perfect
Progress is better than perfection

**After 6 months, you won't just write code.
You'll engineer solutions.**

Start today. Start simple. Stay consistent.

Good luck! ■