CORE PROMPT ENGINEERING PRINCIPLES

Module 2 — Harwell Prompt Engineering

LEARNING OBJECTIVES

By the end of this module you will be able to:

- Write developer-focused prompts that are clear, contextual, and constrained
- Apply zero-shot, few-shot, and chain-of-thought prompting appropriately
- Iteratively refine prompts when output is wrong or incomplete
- Use these techniques for technical problem-solving in chat tools

BRIDGE FROM MODULE 1

What we learned yesterday:

- **Where** to use AI (public vs. enterprise)
- **When** to trust vs. verify

The problem:

- ❌ You've tried asking AI for help, but the answer was too vague
- ❌ It didn't understand your context
- ❌ It gave you something that doesn't fit your codebase
- ❌ You don't know how to fix it

Today: Learn **how** to prompt effectively.

THE DEVELOPER'S PROMPT: THE PROBLEM

Bad prompt example:

> *"Write some code for Spring Boot"*

Problems:

- ❌ Too vague — what kind of code?
- ❌ No context — which version? What's the task?
- ❌ No constraints — what style? What patterns?

Result: Generic, unusable output

THE DEVELOPER'S PROMPT: THREE CS

Clarity — What do you want?

- Task: generate, explain, refactor, test
- Format: code block, inline, with comments

Context — What does the model need to know?

- Stack: Spring Boot 3, Java 17, JPA
- File: "In the UserController class"
- Current state: "This method already exists"

Constraints — What are the rules?

- Language: Java 17 style
- Style: Follow Spring Boot best practices
- Don’ts: “Don’t use @Autowired”

CLARITY: TASK AND FORMAT

Example — vague:

> *“Write a controller”*

Example — clear:

> *“Generate a Spring Boot REST controller with one GET endpoint that returns a list of users”*

Better:

- ✅ Task: Generate
- ✅ What: REST controller
- ✅ Format: GET endpoint, returns list

CONTEXT: STACK, FILE, CONSTRAINTS

Example — no context:

> *“Generate a controller”*

Example — with context:

“Generate a Spring Boot 3 REST controller using Java 17. The endpoint should return ResponseEntity. I’m working in the UserController class.”

Better:

- ✅ Stack: Spring Boot 3, Java 17
- ✅ File: UserController
- ✅ Return type: ResponseEntity

CONSTRAINTS: LANGUAGE, STYLE, DON’TS

Example — no constraints:

> *“Generate a Spring Boot controller”*

Example — with constraints:

“Generate a Spring Boot 3 REST controller using Java 17. Use constructor injection (not @Autowired), return ResponseEntity, and include proper error handling. Don’t use Optional for return types.”

Better:

- ✅ Style: Constructor injection
- ✅ Patterns: ResponseEntity, error handling
- ✅ Don’ts: No Optional returns

ZERO-SHOT PROMPTING

What is zero-shot?

Just ask, no examples.

Example:

> *“Explain what @RestController does in Spring Boot”*

When to use:

- ✅ Well-known concepts
- ✅ Quick answers
- ✅ Standard patterns

Limitation:

- ❌ May not match your style or needs

FEW-SHOT PROMPTING

What is few-shot?

Add 1–2 examples to guide style.

Example:

“Generate a controller like this:

```
@RestController
public class ProductController {
    private final ProductService service;
    public ProductController(ProductService
        this.service = service;
    }
}
```

Now generate another one for User.”

When to use:

- ✅ When you want specific style
- ✅ When format matters
- ✅ One example can change everything

CHAIN-OF-THOUGHT PROMPTING

What is chain-of-thought?

Ask for step-by-step reasoning.

Example:

> *“Refactor this method. First explain what it does, then suggest improvements step by step, then show the refactored code.”*

When to use:

- ✅ Complex tasks
- ✅ When accuracy matters
- ✅ Debugging or design decisions

WHEN TO USE WHICH TECHNIQUE

Technique	Use when
Zero-shot	Well-known patterns, quick answers
Few-shot	You want specific style or format
Chain-of-thought	Complex refactoring, debugging, design

ITERATIVE REFINEMENT: THE PROBLEM

Demo: Run a vague prompt, show poor output

Problems:

- ❌ Output doesn’t match your needs
- ❌ Missing context or constraints
- ❌ Wrong style or patterns

Question: How do you fix it?

ITERATIVE REFINEMENT: READ THE OUTPUT

Diagnosis:

1. **What's missing?** — Identify gaps
2. **What's wrong?** — Spot issues
3. **What would make this better?** — Think about improvements

Example:

- Output uses @Autowired → Missing constraint
- Output uses Spring Boot 2 → Missing context (we use Spring Boot 3)
- Output doesn’t match our style → Need few-shot example

ITERATIVE REFINEMENT: REFINE THE PROMPT

Add missing context:

> *“I’m using Spring Boot 3, not 2”*

Tighten instructions:

> *“Use constructor injection, not @Autowired”*

Add examples:

“Like this: [example]”

Add constraints:

“Don’t use Optional for return types”

ITERATIVE REFINEMENT: RE-RUN AND COMPARE

Before: Vague prompt → Generic output

After: Clear prompt with context and constraints → Targeted output

Key point:

Prompting is iterative, not one-shot.

Practice: Refine based on output until it works.

SUMMARY

1. **Clarity**: Task + format
2. **Context**: Stack + file + constraints
3. **Constraints**: Language + style + don’ts
4. **Techniques**: Zero-shot, few-shot, chain-of-thought
5. **Iterative**: Refine based on output

BRIDGE TO MODULE 3

What we've learned:

- **How** to prompt effectively (3Cs, techniques, iteration)

What's next:

Module 3: AI-Assisted Java Development — **what** to prompt for (code generation, explanation, refactoring, testing).

Apply the 3Cs and iterative refinement throughout.

QUESTIONS?

Module 2 — Core Prompt Engineering Principles