Working Effectively with AI in Software Development

Mental Models, Patterns, and Practical Workflows

Petr Čala

Sympulse s.r.o.

December 8, 2025

Duration: 1.5 hours (40 min presentation + 50 min demo)

You're Already Using AI

You're likely already using AI for:

- Autocomplete suggestions while typing
- Answering questions about APIs and syntax
- Generating boilerplate code
- Explaining errors

Today's Goal

Being more **systematic and intentional** about AI use

How Language Models Work: Tokens

What they are:

- Text broken into chunks
- Processed sequentially
- One token at a time

Why it matters:

- Context window = working memory
- Relevance over volume
- Code $\neq$ prose (denser tokens)

Code uses more tokens per line than English text—50 lines of code $\approx$ 2 pages of prose

Key Insight

Showing code is more efficient than describing it

Probabilistic Generation

"Models predict the most likely next token, then the next, then the next"

Three consequences:

1. **Non-deterministic:** Same input → different outputs
2. **No "knowing":** Predicting plausible continuations
3. **Variable confidence:** Some outputs are guesses

Why this matters: Verification is essential

Mental Model: Knowledgeable but Literal

AI is like a very literal assistant:

- Broad knowledge, lacks *YOUR* context
- Does exactly what you ask (not what you meant)
- Won't push back unless prompted
- Takes instructions literally

When it helps:

- Getting started
- Initial drafts

When it fails:

- Implicit requirements
- Domain decisions

Mental Models: Pattern Matcher & Generator

The Pattern Matcher:

- Excels with familiar patterns
- Works well: examples provided, conventions exist
- Struggles: novel problems, ambiguous requirements

The Confident Generator:

- Generates plausible text regardless of accuracy
- Won't say "I don't know"
- Can't distinguish confidence levels

Always Verify

You need to verify, not just review

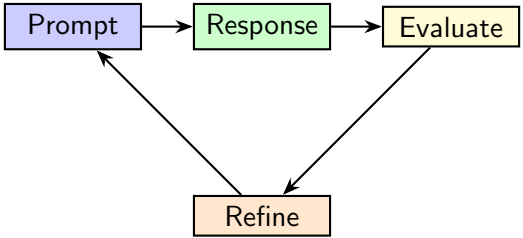
Effective AI Use

Effective AI Use

Understanding + Communication + Verification

- **Understand:** How they actually work
- **Communicate:** What you really need
- **Verify:** Trust but verify always

The Feedback Loop



- Working with AI is iterative
- 2-3 iterations typically gets you to quality

Principle 1: Be Specific

Vague:

```
Write a function
to process data
```

Specific:

```
Write a function that
takes user objects,
returns verified
email addresses
```

Key: Be precise about outcome, not verbose about method

Principles 2-4

2. **Provide Context**
   - Relevant code, constraints, purpose, environment
3. **Show, Don't Tell**
   - "Make it follow REST conventions"
   - [Show example] "Create similar for /posts/:id"
4. **Break Down Complex Tasks**
   - "Build an authentication system"
   - Data model → Password functions → Login endpoint

Pattern 1: Structured Request

```
## Task
[Clear problem statement]

## Current Code
[Paste relevant code here]

## Requirements
- Specific constraint 1
- Specific constraint 2

## Constraints
- Technology limits
- Style requirements
```

Structure creates clarity

Pattern 2: Iterative Refinement

Process: Broad → Specific → Implementation → Refinement

Example sequence:

1. "Three approaches to implement caching"
2. "Expand approach #2"
3. "Write the invalidation logic"

Why it works:

- Each step builds on verified results
- Reduces wasted iterations

Pattern 3: Rubber Duck

Use case: Decision making

```
Approach A vs B for permissions:
- A: JSON array on user record
- B: Separate permissions table
- Context: ~10k users, rare changes,
  query on every request
What tradeoffs should I consider?
```

Result: Structured analysis of options

Use AI to think through decisions, not just implement them

Handling Poor Results

Four strategies:

1. **Add More Context** – AI might be missing information
2. **Provide Negative Examples** – Show what you DON'T want
3. **Ask for Alternatives** – "Current approach doesn't work because..."
4. **Correct and Continue** – Point out specific issues

Iteration is built-in. 2-3 rounds gets you to quality

Anti-Pattern: Prompt Dumping

DON'T:

```
[10,000 line codebase]

"Fix the bug"
```

DO:

```
[Relevant 50 lines]

"Throws null when
array is empty.
Add guard clause."
```

Context pollution degrades output quality

AI Strengths vs Limitations

STRENGTHS

- Pattern recognition
- Error interpretation
- Security patterns
- Best practices

LIMITATIONS

- Business logic
- Architecture
- Your context
- Race conditions

AI excels at mechanical analysis.
Human judgment essential for context

Code Review Workflow

PRE-COMMIT

- You ask AI to review your changes
- Address obvious issues before human review

DURING REVIEW

- Verify complex logic
- Explain unfamiliar patterns

POST-MERGE

- Generate release notes
- Update documentation

AI as first pass, humans for context-dependent decisions

Review Prompts

Template:

```
Review this [type] code for:
- [Concerns: security, performance]
- [Language/framework] best practices

[paste code]
```

Variations:

- **Comparative:** "Compare these and find differences"
- **Focused:** "Security review only"

Debugging: Error Interpretation

Template:

```
I'm getting this error:
[error message and stack trace]

Happens when: [trigger]

Relevant code: [paste code]

Question: What could cause this?
```

Structures information and reduces hallucination

Augment, Don't Replace

AI makes humans more effective—
it doesn't make humans unnecessary

Key Insight

The goal is **amplification**, not substitution

Why Augment, Not Replace?

AI is probabilistic, not deterministic

- Same input can produce different outputs
- Confidence is not the same as correctness
- "Plausible" doesn't mean "right"

AI lacks your context

- Business requirements, team conventions
- Historical decisions and their rationale
- Organizational constraints and politics

Accountability remains with humans

- You sign off on the code, not the AI
- Debugging AI mistakes is still your job

The Augmentation Spectrum

Principle: The higher the stakes, the more human judgment required

Rule of thumb: Would you blindly accept this from a junior developer?

Practical Examples

Augmentation done right:

- AI drafts PR description → Human reviews and edits
- AI suggests code improvements → Human accepts/rejects each
- AI summarizes long thread → Human verifies accuracy
- AI generates test cases → Human validates coverage

Replacement gone wrong:

- AI auto-merges PRs without review
- AI decides deployment timing
- AI chooses database schema changes
- AI handles customer escalations alone

Three Essential Patterns

1. **Generate and Review**
   - AI generates → Human reviews → System applies
2. **Triage and Route**
   - AI classifies → System routes → Humans handle
3. **Augment and Assist**
   - Human works → AI suggests → Human decides

CI/CD Integration Points

Each stage has different risk/benefit tradeoffs

Pipeline Considerations

FAIL-SAFE DESIGN

- Don't break pipelines on AI failures
- Make AI steps optional

COST CONTROL

- Limit to key branches
- Cache results

OBSERVABILITY

- Log decisions
- Track false positives

Security & Data

DATA EXPOSURE

- What goes to AI services?
- Filter sensitive files, redact secrets

SUPPLY CHAIN RISK

- Review AI suggestions
- Don't auto-apply changes

FEEDBACK LOOPS

- Record dismissals
- Monitor false positive rate

Threat Categories

APPLICATION

- Authorization
- Access control

AI LAYER

(our focus)

- Prompt injection
- Data exposure

INFRASTRUCTURE

- API security
- Data storage

Key Threat: Prompt Injection

Warning

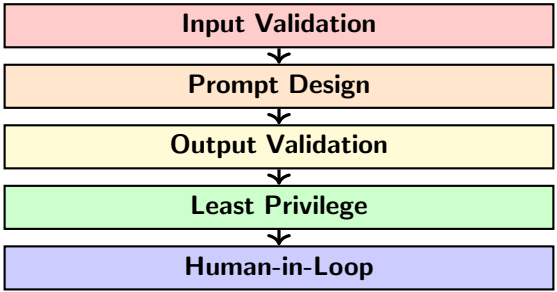
No complete solution exists. Goal: reduce risk, limit impact

The problem: AI can't reliably distinguish between user input and instructions

Attack types:

1. Direct injection (in user input)
2. Indirect injection (in data sources)
3. Jailbreaking (roleplay attacks)

Defense in Depth



Multiple layers, not single solution

Practical Guidance

DO:

- ✓ Review code carefully
- ✓ Watch for anti-patterns
- ✓ Redact sensitive data
- ✓ Consult security team

DON'T:

- Blindly accept
- Paste credentials
- Assume AI knows threats
- Use AI alone

Watch for: SQL concatenation, missing validation, hardcoded credentials

What We Covered

- **Foundation** – How models work, mental models
- **Prompting** – 4 principles, 3 patterns
- **Code Inspection** – Review & debug workflows
- **Automation** – Patterns, CI/CD, reliability
- **Security** – Threats, defense, guidelines

Now let's see it in practice...

Live Demo

We'll walkthrough:

1. Effective prompting in action
2. Code review and debugging workflow
3. Using these tools day-to-day

Questions are always welcome

Thank You!

Questions?