The CTO Guide to AI, including:

10 AI Prompts Every CTO Should Be Using in 2025

Built by CTOs for CTOs serious about embedding AI into engineering workflows - not just playing with it.

What do you get in this guide?

- **10 AI Prompts** That Will Level Up Your Engineering Team
- **A Guide to Embedding AI** Into Daily Engineering Workflows
- **Step-by-Step Playbook** for Building an AI-Native Team
- **Quick Start Guide:** Where to Begin Today
- **Getting 10x More from AI:** How the Best Teams Prompt

Let’s get started →

CloudEmployee

Table of contents:

Read at your leisure, or jump to the best parts: (#6 😉)

What it’s all about.

1. Executive Summary

This guide isn't about AI hype. It's about giving your engineering team real advantages. Inside is:

- **How to choose** the right AI models for different tasks
- **10 proven prompts** for real engineering use cases (PR reviews, security audits, legacy onboarding)
- **How to build better prompts** through persona design and multi-stage workflows
- A phased **roadmap to embed AI** into your engineering workflows

Everything is practical, battle-tested, and ready for use.

Seb Hall
CEO & Founder @ Cloud Employee

★★★★★

“Our PR Review Synthesizer prompt cut review cycles by 32% and caught architectural issues we were missing. Worth it for that alone.”

Anto Cabraja
Chief Technology Officer @ QED

Reward for reading.

2. TLDR: What You'll Get

Strategic Framework: Guidelines for selecting appropriate AI models for different tasks, addressing security concerns, and measuring real business impact

Core Prompts: 10 battle-tested prompt templates for common engineering challenges, from code reviews to security audits

Advanced Techniques: Methods to significantly improve AI output quality through better context, persona engineering, and multi-stage approaches

Organizational Integration: Playbook for embedding AI into engineering workflows and fostering an AI-native culture

Implementation Roadmap: Phased approach to transform your engineering organization's AI capabilities

? **Busy?** Start with Prompts #1, #3, and #10. They'll show value immediately.

The problem.

3. Why This Guide Exists

Most engineering teams are still using AI as a productivity toy.

Writing isolated snippets. Copy/pasting code.

That's not enough anymore.

This guide is for CTOs ready to move *beyond the terminal* — and turn AI into a strategic advantage across their teams, processes, and knowledge base.

Practical applications of AI.

4. From Copilot to Strategy: Scaling AI Across Engineering

If you're already using Copilot, Cursor, or Cody - you're ahead of the curve. But this guide isn't about writing code faster. It's about transforming how AI is used across your entire engineering organization.

Too many teams are stuck in the "prompt toy" phase. This guide is for leaders ready to scale AI beyond the terminal.

This is not for:

- Writing isolated code snippets with ChatGPT
- Copy/pasting prompts back and forth
- Replacing smart developers with automated output

This is for:

- Standardizing pull request reviews with automated first-pass AI inputs
- Onboarding new devs into undocumented legacy systems using AI-assisted walkthroughs
- Building a prompt library tied to your architecture, security model, and workflows
- Embedding AI into your documentation process, ADRs, and code quality reviews
- Turning senior dev practices into AI-powered workflows any team member can use

Things to consider.

5. Picking the Right AI Model (and Avoiding Expensive Mistakes)

SUMMARY: Choose the right AI model based on task complexity and data sensitivity. Establish clear data privacy protocols for code sharing with AI. Measure impact using DORA metrics and developer satisfaction rather than vague time savings claims.

🤖 1. AI Selection Criteria: Choosing the Right Model for the Job

Different AI models have distinct strengths, limitations, and cost implications:

Task type	Recommended AI	Rationale
Simple completions, routine coding	GitHub Copilot, smaller open-source models	Lower cost, sufficient capability, faster response
Complex architectural design, security analysis	GPT-4, Claude Opus	Advanced reasoning, nuanced understanding of complex systems
Large codebase analysis	Claude Sonnet/Opus	Larger context windows handle more code at once
Sensitive IP/compliance requirements	Enterprise versions with data privacy guarantees, local models	Data doesn't leave your environment

Pro Move

Build a simple cheat sheet for your team on which AI to use when, balancing cost, capability, and security needs.

🔒 2. Data Privacy & Security Framework

Critical considerations:

- **IP Protection:** Sanitize proprietary algorithms before inputting to public models
- **PII Handling:** Establish protocols for removing customer data from code examples
- **Sensitive Domain Knowledge:** For regulated industries, use enterprise AI versions with appropriate data handling agreements

Pro Move

Draft a one-page AI usage policy that clearly outlines what code/data your team can share with different AI systems.

📊 3. Impact Measurement Strategy

Move beyond anecdotal evidence to quantifiable metrics:

Engineering Velocity Metrics:

- DORA metrics correlation with AI adoption (Deployment Frequency, Lead Time for Changes, etc.)
- PR review time reduction
- Time-to-resolution for bugs/incidents

Quality Metrics:

- Change failure rate pre/post AI adoption
- Regression defects post-release
- Code quality scores via static analysis

Developer Experience Metrics:

- AI-specific sections in developer satisfaction surveys
- Onboarding time to productivity
- Knowledge access equality across experience levels

🎓 Pro Move

Benchmark your team's baseline now, before you roll AI out at scale, then track meaningful improvements as usage matures.

What you’re really here for.

6. The 10 AI Prompts Your Engineering Team Should Start With

EXECUTIVE SUMMARY: These 10 prompts address the most high-leverage engineering tasks. Start with PR reviews and documentation generation for quick wins, then advance to architecture design and security analysis as your team gains expertise.

📝 1. PR Review Synthesizer

Use When: You're reviewing a pull request and want a fast, high-level summary with critical insights.

Prompt

Act as a senior software architect reviewing this pull request with particular attention to architectural impact, security implications, and scalability concerns.

Provide:

1. A concise summary of the primary purpose and scope
2. Key architectural changes and their implications
3. Potential security or performance concerns
4. Any deviation from established patterns or style guides
5. Specific questions that should be addressed before merging

Format as a GitHub PR comment, prioritizing issues by severity.

Team Integration

- Add to your GitHub PR template as guidance for reviewers
- Configure as automated first-pass review via GitHub Actions
- Track common issues to identify needs for architectural guidance

🏛️ 2. Architectural Decision Record Generator

Use When: Capturing important architectural decisions and their rationale for future reference.

Prompt

Act as a principal solutions architect drafting an Architectural Decision Record (ADR). Based on the following context, create a comprehensive ADR that includes:

1. Title: A descriptive title for the architectural decision
2. Status: Proposed
3. Context: The forces at play, including technological, business, and team constraints
4. Decision: The response to these forces, clearly stating the direction taken
5. Consequences: The resulting context after applying the decision, including positive, negative, and neutral consequences
6. Compliance: How this decision aligns with architectural principles and regulatory requirements
7. Alternatives Considered: Other options evaluated with pros/cons

Context for this decision:

[Insert problem statement, requirements, constraints, and any discussion summary]

Team Integration

- Automatically generate draft ADRs from architecture meeting notes
- Store in central knowledge repository with cross-linking to affected systems
- Review in architecture governance meetings

📜 3. Legacy System Knowledge Extraction

Use When: Onboarding new developers to complex legacy systems or preparing for modernization.

Prompt

Act as a system archaeology expert analyzing this legacy codebase. Create a comprehensive understanding guide that includes:

1. Core system architecture and dataflow diagram (described in text)
2. Key abstractions and their relationships
3. Critical implementation details that wouldn't be obvious to newcomers
4. Dependency map highlighting internal and external dependencies
5. Identified technical debt categories and their impact
6. Potential modernization paths with risk assessment
7. Documentation gaps requiring further investigation

Provide concrete examples from the code to support your analysis.
[Insert legacy code or system description]

Team Integration

- Run this on critical legacy systems during onboarding preparation
- Create a legacy knowledge map in your wiki
- Use as input for modernization planning

🧠 4. System Design Critique

Use When: Validating proposed architecture designs before implementation begins.

Prompt

You are a principal architect specializing in distributed systems with 20+ years of experience. Review this system design with a critical eye for:

1. Scalability bottlenecks under the specified load conditions
2. Single points of failure or resilience gaps
3. Data consistency and integrity risks
4. Operational complexity and maintainability concerns
5. Security vulnerabilities at the architectural level
6. Cost optimization opportunities
7. Compliance with architectural principles

For each issue identified, suggest alternative approaches from your experience, considering the constraints.

System design:
[Insert system design description/diagram]

Load requirements:
[Insert expected load, growth projections, SLAs]

Architectural principles:
[Insert relevant architectural principles]

Team Integration

- Incorporate into architecture review process
- Use for training junior architects
- Create design pattern library from recurring recommendations

🧩 5. Multi-Stage Refactoring Planner

Use When: Planning complex refactoring efforts that span multiple components or sprints.

Prompt

As a refactoring expert specializing in complex system transformations, create a detailed refactoring plan for this codebase. Your plan should include:

1. System analysis: Identify core components and their interdependencies
2. Technical debt assessment: Categorize and prioritize debt by risk and impact
3. Refactoring strategy: Recommend appropriate refactoring patterns for each component
4. Dependency mapping: Identify the optimal sequence of changes to minimize risk
5. Testing strategy: Outline approach to ensure behavioral preservation throughout
6. Rollout plan: Break down the refactoring into logical phases with clear completion criteria
7. Risk assessment: Identify potential risks and mitigation strategies for each phase

Provide specific examples of changes you'd make to illustrate the approach.

[Insert codebase or system description]

Constraints:

[Insert team size, time constraints, business priorities]

Team Integration

- Use during quarterly planning for technical debt reduction
- Incorporate into strategic roadmap for system modernization
- Create refactoring playbooks for common patterns in your codebase

🔌 6. API Integration Scaffolding Generator

Use When: Integrating with new APIs or building client libraries for internal services.

Prompt

As an integration specialist, generate scaffolding code for integrating with this API. Create:

1. Client library structure following [LANGUAGE] best practices
2. Interface definitions for all API operations
3. Request/response models with appropriate validation
4. Error handling approach covering all documented error states
5. Retry strategy for transient failures
6. Authentication implementation
7. Logging and telemetry points
8. Unit test stubs covering happy and error paths

Make the implementation idiomatic to the stack and follow coding standards.

API specification:
[Insert OpenAPI spec, documentation link, or description]

Stack:
[Insert language, frameworks, and coding standards]

Team Integration

- Generate during project kickoff for new integrations
- Add to internal developer portal for service-to-service communication
- Create reusable templates for common integration patterns

⚙️ 7. Cross-Cutting Concern Analysis

Use When: Ensuring consistent implementation of non-functional requirements across a codebase.

Prompt

As a technical lead responsible for engineering excellence, analyze this codebase for consistent implementation of [CROSS-CUTTING CONCERN: error handling, logging, authentication, etc.].

Create a comprehensive report including:

1. Current implementation patterns identified across the codebase
2. Inconsistencies or gaps in implementation
3. Security or reliability risks from the identified inconsistencies
4. Recommended standardized approach with code examples
5. Implementation priority for addressing identified issues
6. Suggested automated checks to prevent future inconsistencies

[Insert codebase or relevant modules]

Defined standard for [CONCERN]:
[Insert existing standard if available]

Team Integration

- Schedule regular automated analyses for critical concerns
- Feed findings into engineering standards documentation
- Create linting rules based on identified patterns

🛡️ 8. Proactive Security Threat Modeling

Use When: Identifying potential security threats during the design phase of new features.

Prompt

Act as a security architect with expertise in threat modeling. Using the STRIDE methodology (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege), analyze this proposed feature for potential security threats.

For each identified threat:

1. Describe the threat scenario in detail
2. Assess impact and likelihood
3. Recommend specific mitigations
4. Suggest verification approaches to confirm the threat is addressed

Feature description:
[Insert feature description, data flows, and technical approach]

Current security controls:
[Insert relevant existing security measures]

Compliance requirements:
[Insert any regulatory/compliance considerations]

Team Integration

- Incorporate into security review process for all new features
- Build a threat library specific to your domain
- Create security requirement templates from recurring threats

🚀 9. Performance Optimization Strategy

Use When: Addressing performance issues or proactively optimizing critical paths.

Prompt

As a performance engineering specialist, analyze this code for optimization opportunities with particular attention to:

1. Algorithmic complexity issues ($O(n^2)$ or worse behaviors)
2. Database query inefficiencies and N+1 problems
3. Memory usage patterns and potential leaks
4. Concurrency bottlenecks and thread safety
5. Network call optimization opportunities
6. Caching strategy improvements
7. Resource utilization (CPU, memory, I/O)

For each identified issue:

- Provide specific code locations
- Explain the performance impact with quantitative estimates
- Suggest concrete optimization approaches with example implementations
- Prioritize by expected impact vs. implementation effort
- Recommend performance tests to validate improvements

[Insert code or performance profile data]

Performance requirements:
[Insert performance SLAs or targets]

Team Integration

- Run proactively on critical path code during code review
- Create performance testing scenarios based on identified hotspots
- Build optimization patterns library from recurring recommendations

📚 10. Comprehensive Documentation Generator

Use When: Creating detailed technical documentation for modules, APIs, or systems.

Prompt

As a technical documentation specialist, create comprehensive documentation for this codebase following a documentation-as-code approach. Generate:

1. Executive summary for project stakeholders (1-2 paragraphs)
2. System context and architectural overview
3. Installation and environment setup guide
4. API reference with examples and edge cases
5. Data models and schema descriptions
6. Error handling approach and troubleshooting guide
7. Performance characteristics and scaling considerations
8. Security considerations and best practices
9. Deployment and operational guidance
10. Contributing guidelines and development workflow

Format in clean, structured Markdown with:

- Consistent heading hierarchy
- Syntax-highlighted code examples
- Navigable table of contents
- Cross-reference links between sections
- Callouts for important warnings and notes

[Insert codebase or module]

Target audience: [Insert audience skill level and role]

Team Integration

- Generate baseline documentation for all new services
- Schedule regular refresh with code changes
- Incorporate into definition of done for features

Implementing AI well.

7. Embedding AI Into Daily Engineering Workflows

SUMMARY: The difference between mediocre and exceptional AI results often comes down to prompt quality. Improve your results by providing richer context, assigning specific expertise personas, breaking complex tasks into steps, and iteratively refining outputs.

1. Context Enhancement Strategies

The effectiveness of AI assistance depends heavily on the quality of context provided:

Full-System Context Example:

Before diving into this specific module, let me provide system context:

- This is part of a microservices architecture with 12 services
- This particular service handles user authentication and session management
- It interfaces with PostgreSQL database and Redis cache
- The company has a regulatory requirement for SOC 2 compliance
- Engineering principles prioritize security over feature velocity

Now, regarding the specific module I need help with:
[Insert specific code/question]

Historical Context Example:

This codebase has evolved over 5 years:

- Initially built as a monolith in 2018
- Partially refactored to microservices in 2020
- Still contains some legacy patterns we're working to modernize
- Has significant technical debt in the data access layer
- Recent focus has been on performance optimization

Specific question:
[Insert specific code/question]

Pro Move

Benchmark your team's baseline now, before you roll AI out at scale, then track meaningful improvements as usage matures.

🎭 2. Persona Engineering

Explicitly assigning a persona to the AI can dramatically improve results:

Example: Security Review Persona

Act as a security architect with 15+ years of experience in application security, OWASP Top 10 mitigation, and secure coding patterns in [LANGUAGE]. You've performed hundreds of security reviews and have a particular expertise in identifying subtle vulnerabilities that automated tools miss.

Review this code for security vulnerabilities:
[Insert code]

Example: Architecture Design Persona

Act as a principal solutions architect who specializes in designing high-scale, resilient systems. You have deep expertise in:

- Distributed systems patterns
- Data consistency models in distributed environments
- Cloud-native architecture
- Cost optimization at scale
- Resilience engineering

Design approach:
[Insert design challenge]

Pro Move

Create a shared library of 5-10 engineering personas your team can use consistently across different AI tools and tasks.

🔗 3. Multi-Stage Prompt Chains

Complex engineering tasks often benefit from breaking the problem into sequential steps:

Example: Legacy Code Modernization Chain

1. Analysis Stage:

Analyze this legacy code to identify:

- Core functionality and business logic
- Current architecture and patterns
- Technical debt and code smells
- Test coverage assessment

[Insert legacy code]

2. Strategy Stage (using output from step 1):

Based on this analysis, develop a modernization strategy that:

- Prioritizes components for refactoring
- Suggests appropriate modern patterns
- Outlines a phased approach
- Identifies risks and mitigation strategies

[Insert analysis from step 1]

3. Implementation Stage (for highest priority component):

Create a refactored implementation of this component that:

- Preserves existing functionality
- Applies modern patterns
- Improves testability
- Maintains or improves performance

[Insert original component and strategy from step 2]

Pro Move

Map out 2-3 prompt workflows for your most complex processes, where output from one prompt feeds directly into the next.

🛠️ 4. Prompt Refinement Techniques

Iterating on prompts is essential for optimal results:

Technique: Critique and Revise

You've provided this solution:
[Insert AI's previous response]

- Please critique your own solution, considering:
- Have you fully addressed all requirements?
- Are there edge cases you missed?
- Is the approach optimal for performance/readability/maintainability?
- Have you introduced any potential bugs or security issues?

Then provide a revised solution addressing these critiques.

Technique: Multiple Perspectives

You've provided this architectural approach:
[Insert AI's previous response]

- Now, analyze this approach from multiple perspectives:
- As a security architect concerned with data protection
- As a DevOps engineer responsible for deployment and monitoring
- As a product manager concerned with feature extensibility
- As a junior developer who will need to implement and maintain this

Identify potential concerns from each perspective and suggest refinements to address
them.

Pro Move

Create a handful of standard follow-up prompts your team can use to critique and improve their initial AI responses.

Everyday habits.

8. How to Make AI Part of Your Team's Daily Workflow

EXECUTIVE SUMMARY: To maximize AI's impact, embed it into your engineering processes, build internal knowledge-sharing systems around it, and develop team capabilities for effective AI collaboration. This requires technical integration, culture change, and thoughtful governance.

🔄 1. Engineering Workflow Integration

Code Review Process:

- Automate first-pass reviews with AI for style, conventions, and security scanning
- Train reviewers to use AI for deeper analysis of complex changes
- Track common AI-identified issues to improve engineering standards

Planning Process:

- Use AI to analyze requirements for completeness and edge cases
- Generate technical specifications from user stories
- Create test plans that cover identified edge cases

Documentation:

- Automatically generate initial documentation from code
- Schedule regular documentation refreshes synchronized with releases
- Use AI to identify documentation gaps based on code changes

🏗️ 2. Building an Internal AI Knowledge Hub

Components:

- Prompt library organized by engineering task
- Custom-tuned models for company-specific domains
- Integration scripts for development environments
- Usage analytics to track effectiveness
- Prompt version control and improvement tracking

Governance:

- AI usage policies and security guidelines
- Review process for adding prompts to the official library
- Training materials for effective AI collaboration

🌱 3. Fostering an AI-Native Engineering Culture

Accountability Framework:

- Clear guidelines on AI-assisted code ownership
- Code review practices that account for AI generation
- Attribution practices for AI contributions

Training Strategy:

- Onboarding module on effective AI collaboration
- Prompt engineering workshops for engineers
- Regular sharing of AI success patterns and pitfalls

Ethical Considerations:

- Guidelines for avoiding bias amplification
- Responsible AI usage principles
- Transparency about AI involvement in development

Rolling out AI.

9. Your Step-by-Step Playbook for Building an AI-Native Team

EXECUTIVE SUMMARY: Implement in phases, starting with foundational capabilities and working toward a transformed engineering organization. Begin with the highest-leverage, lowest-risk prompts like PR reviews and documentation to build confidence before tackling more complex areas.

Phase 1: Foundation (1-2 months)

- Establish AI selection criteria and security guidelines
- Implement basic PR review and documentation prompts
- Set up baseline metrics for impact tracking

Phase 2: Process Integration (2-3 months)

- Integrate AI into formal engineering workflows
- Develop custom prompts for your specific domain and stack
- Create internal knowledge base of effective prompts

Phase 3: Advanced Applications (3-6 months)

- Implement multi-stage prompt chains for complex tasks
- Develop custom-tuned models for company-specific needs
- Build automated systems for continuous prompt improvement

Phase 4: Culture Transformation (6+ months)

- Transform engineering processes around AI capabilities
- Establish AI expertise as a core engineering competency
- Develop unique AI-enabled competitive advantages

Where to start?

10. Quick Start Guide

Today: Start with the PR Review Synthesizer prompt during your next code review

This Week: Create standardized context templates for your specific systems

This Month: Implement the Documentation Generator and Legacy System Knowledge Extraction prompts

Next Quarter: Build your internal prompt library and begin formal AI integration into engineering processes

Seb Hall
CEO & Founder @ Cloud Employee

Final Thoughts.

11. Conclusion: The Future CTO Will Be AI-Native

The difference between basic AI usage and strategic AI integration is the difference between incremental improvements and transformative engineering capabilities. This guide provides a framework for moving beyond individual productivity gains to organization-wide transformation of your software development lifecycle.

For engineering leaders ready to build the future of AI-enhanced development, this is just the beginning of the journey.

Want a free consultation with a fellow CTO to discuss AI usage and strategy in your organisation? We're all in it together, so why don't you book a time with one of our CTO partners today.

Book a CTO Consultation →

CloudEmployee