Advanced Prompt Engineering: Semantic Markup & Template Logic Programming

The Complete Guide to Building Sophisticated AI Systems Through Structured Prompting
Made by Anthony Mikinka

Table of Contents

Introduction & Overview

Semantic Markup Prompting and **Template Logic Programming** represent the evolution of prompt engineering from simple text instructions to sophisticated AI system architectures. These techniques enable the creation of complex, maintainable, and reusable AI interactions that scale from simple tasks to multi-agent orchestration systems.

What This Guide Covers

- **Structured Prompting**: Organizing complex prompts using semantic markup
- **Template Logic**: Conditional logic, loops, and variables in prompts
- **Agent Architecture**: YAML-in-Markdown configuration systems
- **Meta-Instructions**: Embedding AI guidance within templates
- **Quality Systems**: Validation and assurance frameworks
- **Practical Implementation**: Real-world examples and best practices

Why This Matters

Traditional prompting approaches break down as complexity increases. These advanced techniques provide:

- **Maintainability**: Structured systems that can be updated and extended
- **Reusability**: Components that work across multiple contexts
- **Reliability**: Predictable behavior through structured constraints
- **Scalability**: Patterns that work for both simple and complex systems

---

Theoretical Foundations

Evolution of Prompt Engineering

Level 1: Basic Prompting

Write a summary of this article.

Level 2: Structured Prompting

<objective>Summarize the article</objective>
<constraints>
- Maximum 3 paragraphs
- Include key statistics

- Maintain professional tone
</constraints>

Level 3: Template Logic Programming
<objective>
Create a {{document_type}} for {{target_audience}}
</objective>

^^CONDITION: document_type == "technical_report"^^
Focus on technical accuracy and detailed analysis.
^^/CONDITION: document_type^^

[[LLM: Adapt language complexity based on target_audience variable]]

Level 4: Agent Architecture Systems
activation-instructions:
  - Follow embedded configuration
  - Maintain character consistency
  - Use numbered options protocol

agent:
  name: Technical Writer
  persona: Expert technical communicator
  commands:
    - create-documentation
    - review-technical-content

Core Principles

1. **Separation of Concerns**: Different aspects handled by different markup patterns
2. **Progressive Enhancement**: Building complexity through layered systems
3. **Convention Over Configuration**: Established patterns reduce cognitive load
4. **Explicit State Management**: Clear control over AI behavior and context
5. **Modular Composition**: Reusable components that work together

---

Core Semantic Markup Patterns

XML-Style Section Tags

Purpose: Organize complex prompts into logical, parseable sections
<objective>
Define what the AI should accomplish
</objective>

<constraints>
Define limitations and requirements
</constraints>

<process>
1. Step one
2. Step two
3. Step three
</process>

<output_format>
Specify exactly how output should be structured
</output_format>

Benefits:
- Clear prompt structure for both humans and AI
- Easy to modify individual sections
- Enables prompt composition and inheritance
- Improves AI comprehension of complex instructions

Meta-Instruction Blocks

Purpose: Instructions TO the AI about how to process content
[[LLM: This section provides guidance for AI processing]]
[[LLM: Present this content first, wait for user input, then proceed]]
[[LLM: Use technical language if user indicates expertise]]
[[LLM: Validate completeness before final output]]

Usage Patterns:
- **Processing Guidance**: How to handle specific sections
- **Interaction Flow**: When to pause for user input
- **Adaptation Logic**: How to modify behavior based on context
- **Quality Control**: Validation and checking instructions

Example Documentation System

Purpose: Self-documenting templates with clear usage patterns
@{example-1: Basic Configuration}
Simple example showing minimal setup

@{example-2: Advanced Configuration}
Complex example with all features enabled

@{example-3: Domain-Specific Usage}
Real-world application in specific context

Benefits:
- Templates become self-teaching
- Reduces learning curve for new users
- Provides validation examples
- Shows progression from simple to complex

Template Logic Programming

Conditional Logic Systems

Basic Conditionals:
^^CONDITION: user_level == "beginner"^^
Provide detailed explanations with examples
^^/CONDITION: user_level^^

^^CONDITION: user_level == "expert"^^
Use technical terminology and assume knowledge
^^/CONDITION: user_level^^

Nested Conditionals:
^^CONDITION: project_type == "web_application"^^
  ^^CONDITION: framework == "react"^^
  Use React-specific patterns and conventions
  ^^/CONDITION: framework^^

  ^^CONDITION: framework == "vue"^^
  Use Vue-specific patterns and conventions
  ^^/CONDITION: framework^^
^^/CONDITION: project_type^^

Loop and Iteration Systems

Repeat Blocks:
<<REPEAT section="stakeholder" count="{{stakeholder_count}}">>
Stakeholder {{loop_index}}: {{stakeholder_name}}
- Role: {{stakeholder_role}}
- Requirements: {{stakeholder_requirements}}
- Contact: {{stakeholder_contact}}
<</REPEAT>>

Dynamic Lists:
<<REPEAT section="feature" source="{{feature_list}}">>
- [ ] {{feature.name}} - Priority: {{feature.priority}}
  - Description: {{feature.description}}
  - Acceptance Criteria: {{feature.criteria}}
<</REPEAT>>

Variable Systems

Simple Placeholders:
{project_name} # Basic substitution
{user_name} # User-provided content
{current_date} # System-generated content

Template Engine Style:
{{project.name}} # Object property access
{{user.preferences}} # Nested property access
{{#each items}} # Template engine compatibility

Interpolation Style:
${dynamic_content} # JavaScript-style interpolation
${user.input} # Property access
${function_call()} # Function execution

Advanced Template Features

Content Inclusion:
@{include: header-template}
@{include: footer-template}
@{include: domain-specific-sections}

Template Inheritance:

@{extends: base-document-template}
@{block: custom-content}
Domain-specific customizations
@{/block: custom-content}

Advanced Agent Architecture

YAML-in-Markdown Structure

Purpose: Embed structured configuration within readable documentation

```
activation-instructions:
  - Follow all instructions in this file
  - Stay in character until told to exit
  - Use numbered options for all interactions

agent:
  name: Technical Architect
  id: tech-architect
  title: Senior Technical Architecture Specialist
  icon: 🏗️
  customization: null

persona:
  role: Expert system designer with 15+ years experience
  style: Analytical, thorough, collaborative
  identity: Former CTO turned consultant specializing in scalable architectures
  focus: System design, technology selection, architecture documentation

core_principles:
  - Scalability first - design for growth
  - Security by design - never bolt on security later
  - Documentation-driven development
  - Technology decisions based on requirements, not trends

startup:
  - Greet as [name] and explain role
  - DO NOT auto-execute any commands
  - Present numbered options for user selection
  - Ask about current architecture challenges

commands:
  - '*help' - Show numbered list of available commands
  - '*chat-mode' - Open discussion about architecture topics
  - '*create-doc architecture-tmpl' - Create system architecture document
  - '*review-design' - Analyze existing architecture for improvements
  - '*technology-assessment' - Evaluate technology choices
  - '*exit' - Say goodbye and abandon persona

dependencies:
  tasks:
    - create-doc
```

```
  - review-design
  - technology-assessment
templates:
  - architecture-tmpl
  - technology-comparison-tmpl
checklists:
  - architecture-checklist
  - security-checklist
data:
  - technology-standards.md
  - architecture-patterns.md
```

Character Persona Systems

Purpose: Consistent AI personality and behavior across interactions

Character Development Framework:

```
persona:
  # Core Identity
  role: Specific professional role with clear expertise
  identity: Background story that explains knowledge and approach

  # Communication Style
  style: How they speak and interact (formal, casual, technical)
  voice: Personality traits that come through in responses

  # Professional Context
  focus: Primary areas of expertise and responsibility
  experience: Background that validates their authority

  # Behavioral Guidelines
  core_principles: Values that guide decision-making
  preferred_approach: Methodology they typically follow
  interaction_style: How they work with users and other agents
```

Example Character Implementation:

```
persona:
  role: Senior DevOps Engineer turned Platform Architect
  identity: Former startup CTO who scaled systems from 1K to 1M+ users

  style: Direct but supportive, uses real-world examples
  voice: Pragmatic problem-solver who's "been there, done that"

  focus: Infrastructure automation, CI/CD, monitoring, cost optimization
  experience: 12 years, including 3 major scaling challenges

  core_principles:
    - Automate everything that can be automated
    - Monitor what matters, ignore vanity metrics
    - Plan for failure - everything will break eventually
    - Cost-optimize continuously, not just during budget reviews
```

Command System Design

Purpose: Structured interaction patterns with clear capabilities

Command Categories:

```
commands:
  # Meta Commands (every agent should have)
  - '*help' - Show numbered command list
  - '*chat-mode' - Open conversational mode
  - '*exit' - End agent session

  # Creation Commands (use create-doc pattern)
  - '*create-doc {template}' - Generate documents from templates

  # Action Commands (use task pattern)
  - '*analyze-{target}' - Perform specific analysis
  - '*review-{artifact}' - Quality assurance actions
  - '*optimize-{system}' - Improvement recommendations

  # Information Commands
  - '*explain-{concept}' - Educational content
  - '*compare-{options}' - Decision support
```

Numbered Options Protocol:

```
Available commands:
1. Create system architecture document
2. Review existing design for improvements
3. Assess current technology stack
4. Explain architectural patterns
5. Help with technology selection
6. Chat about architecture topics

Please select an option (1-6):
```

Multi-Agent Orchestration

Orchestrator Pattern:

```
agent:
  name: Project Coordinator
  role: orchestrator

orchestration:
  manages_agents:
    - technical-architect
    - senior-developer
    - qa-specialist

  workflow_control:
    - Assess project requirements
    - Route to appropriate specialist
    - Coordinate handoffs between agents
    - Ensure quality gates are met
```

```
  decision_trees:
    project_complexity:
      simple: Route to senior-developer
      complex: Start with technical-architect
      legacy: Include qa-specialist from start
```

Handoff Protocols:

```
handoff_patterns:
  to_specialist:
    - Summarize current context
    - Specify what input is needed
    - Set expectations for deliverable
    - Provide relevant background information

  from_specialist:
    - Validate deliverable completeness
    - Check quality criteria
    - Update project status
    - Determine next agent or user interaction
```

Meta-Instruction Systems

LLM Guidance Embedding

Purpose: Instructions embedded within content to guide AI processing

Processing Instructions:

[[LLM: Present this section first and wait for user confirmation before proceeding]]
[[LLM: If user indicates they're a beginner, provide additional explanation]]
[[LLM: Use technical terminology only if user demonstrates expertise]]
[[LLM: Validate all technical specifications before presenting final output]]

Content Generation Guidance:

[[LLM: Generate 3-5 examples tailored to the user's industry]]
[[LLM: Adapt complexity based on {{user_experience_level}} variable]]
[[LLM: Include relevant code snippets if {{include_code}} is true]]
[[LLM: Cross-reference with {{standards_document}} for compliance]]

Quality Control Instructions:

[[LLM: Verify all URLs are accessible before including in output]]
[[LLM: Fact-check any statistics or metrics mentioned]]
[[LLM: Ensure all placeholder text is replaced with actual content]]
[[LLM: Review for consistency with established terminology]]

Advanced Elicitation Patterns

Progressive Disclosure:

[[LLM: Start with high-level questions, then drill down based on responses]]

1. What type of system are you building?
   [[LLM: Based on answer, ask relevant follow-ups]]

   ^^CONDITION: answer == "web_application"^^

2a. What's your expected user volume?
2b. Do you need real-time features?
^^/CONDITION: answer^^

^^CONDITION: answer == "api_service"^^
2a. What's your expected request volume?
2b. Do you need to integrate with external services?
^^/CONDITION: answer^^

Contextual Adaptation:
[[LLM: Adjust questioning style based on user responses]]
[[LLM: If user uses technical terms, assume higher expertise level]]
[[LLM: If user asks for examples, they likely prefer concrete over abstract]]
[[LLM: If user provides detailed requirements, they're probably experienced]]

Dynamic Template Adaptation

Context-Aware Content:
^^CONDITION: project_phase == "planning"^^
[[LLM: Focus on requirements gathering and high-level architecture]]
Focus on:
- Requirements analysis
- High-level system design
- Technology selection criteria
^^/CONDITION: project_phase^^

^^CONDITION: project_phase == "implementation"^^
[[LLM: Focus on detailed technical specifications and implementation guidance]]
Focus on:
- Detailed technical specifications
- Implementation patterns
- Code review guidelines
^^/CONDITION: project_phase^^

User Preference Learning:
[[LLM: Remember user preferences for future interactions]]
[[LLM: If user consistently chooses detailed examples, default to comprehensive output]]
[[LLM: If user typically skips background, focus on actionable content]]
[[LLM: Adapt technical depth to match user's demonstrated expertise]]

Quality Assurance Integration

Multi-Level Validation Systems

Basic Validation:
```
basic_checklist:
  completeness:
    - [ ] All required sections present
    - [ ] No placeholder text remaining
    - [ ] All links functional

  format:
```

- [ ] Proper markdown formatting
- [ ] Consistent heading structure
- [ ] Code blocks properly formatted

Comprehensive Validation:
comprehensive_checklist:
  content_quality:
    - [ ] Technical accuracy verified
    - [ ] Examples tested and working
    - [ ] Cross-references validated
    - [ ] Terminology consistent throughout

  domain_standards:
    - [ ] Follows industry best practices
    - [ ] Meets compliance requirements
    - [ ] Aligns with organizational standards
    - [ ] Security considerations addressed

Expert-Level Assessment:
expert_checklist:
  strategic_alignment:
    - [ ] Supports long-term architecture goals
    - [ ] Considers scalability implications
    - [ ] Addresses technical debt concerns
    - [ ] Optimizes for maintainability

  innovation_potential:
    - [ ] Leverages current best practices
    - [ ] Considers emerging technologies
    - [ ] Identifies optimization opportunities
    - [ ] Plans for future evolution

Star Rating Systems

Quality Assessment Framework:
⭐ (1 star): Minimal requirements met, needs significant improvement
⭐⭐ (2 stars): Basic quality, some improvements needed
⭐⭐⭐ (3 stars): Good quality, minor improvements possible
⭐⭐⭐⭐ (4 stars): High quality, production-ready with minimal changes
⭐⭐⭐⭐⭐ (5 stars): Excellent quality, exceeds requirements

Implementation in Templates:
[[LLM: Rate each section using star system and provide specific improvement recommendations]]

Architecture Design Quality: {{rating_stars}}

Strengths:
{{#if rating >= 4}}
- Excellent scalability considerations
- Comprehensive security design
{{/if}}

Areas for Improvement:
{{#if rating < 4}}
- [ ] Add monitoring and observability section
- [ ] Include disaster recovery planning
{{/if}}

Next Steps:
{{#if rating >= 4}}
Ready for stakeholder review and approval
{{else}}
Address improvement areas before proceeding
{{/if}}

Ready/Not-Ready Decision Frameworks

Decision Criteria:

```
readiness_assessment:
  technical_readiness:
    criteria:
      - All technical specifications complete
      - Architecture review passed
      - Security assessment completed
      - Performance requirements validated
    threshold: 100% # All criteria must be met

  business_readiness:
    criteria:
      - Stakeholder approval received
      - Budget allocated
      - Timeline confirmed
      - Resources available
    threshold: 75% # Most criteria must be met

  risk_readiness:
    criteria:
      - Major risks identified and mitigated
      - Rollback plan defined
      - Monitoring in place
      - Support procedures documented
    threshold: 90% # Almost all criteria must be met
```

Implementation Pattern:

[[LLM: Evaluate readiness using defined criteria and provide clear go/no-go decision]]

Project Readiness Assessment

Technical Readiness: {{technical_status}}
{{#if technical_ready}}
✅ All technical criteria met - READY
{{else}}
❌ Technical gaps identified - NOT READY

Remaining items:
{{#each technical_gaps}}
- [ ] {{this}}
{{/each}}
{{/if}}

Overall Decision: {{#if overall_ready}}PROCEED{{else}}HOLD{{/if}}

Real-World Implementation Strategies

News Curation System Example

Framework-Based Prompting Implementation:
<objective>
Analyze daily AI news digest and select 3-5 stories with highest viral potential.
Date: {{current_date}} - prioritize recent, relevant content.
</objective>

<curation_framework>
Apply virality principles to identify winning stories:

1. **Impactful**: Major breakthroughs or industry shifts
2. **Practical**: Tools/techniques audience can use immediately
3. **Provocative**: Stories that spark debate or controversy
4. **Astonishing**: "Wow-factor" demonstrations

Hard Filters:
- Ignore ad-driven content
- Ignore purely political stories
- Ignore substanceless content
</curation_framework>

<hook_angle_framework>
For each story, create 2-3 compelling hook angles:
- Question Hook: Intriguing questions that demand answers
- Shock/Surprise Hook: Counterintuitive or surprising elements
- Problem/Solution Hook: Common problem + AI solution
- Breaking News Hook: Urgency and newsworthiness
</hook_angle_framework>

<process>
1. **Ingest**: Review entire content
2. **Deduplicate**: Group similar stories
3. **Select & Rank**: Apply curation framework
4. **Generate Hooks**: Create compelling angles
</process>

<output_format>
Single JSON object only - no other text or formatting.
{
  "stories": [

```
    {
      "title": "viral-optimized title",
      "summary": "compelling 1-2 sentence hook",
      "hook_angles": [
        {
          "hook": "opening line text",
          "type": "hook type from framework",
          "rationale": "why this works"
        }
      ],
      "sources": ["exact URLs from source - no modifications"]
    }
  ]
}
</output_format>
```

Memory Evaluation System

Structured Assessment Framework:

```
<objective>
Judge whether memories from AI-programmer conversations are worth retaining.
Score 1-5 based on value for future interactions.
</objective>

<evaluation_criteria>
Worth remembering (4-5):
- Relevant to programming/software engineering
- General and applicable to future interactions
- SPECIFIC and ACTIONABLE
- NOT tied only to current conversation's specific code
- Represents general preference or rule

Rate poorly (1-2):
- One-off implementation details
- Vague or obvious observations
- Tied only to specific files/code discussed
- Ad-hoc task details

{{evaluation_examples}}
</evaluation_criteria>

<process>
1. Analyze conversation context: {{conversation_context}}
2. Evaluate memory: "{{memory_content}}"
3. Apply criteria systematically
4. Provide justification focusing on why it's NOT in the 99% that should score 1-3
5. Return score in format "SCORE: [1-5]"
</process>

<special_rules>
- If user explicitly asks to remember: Score 5
- If contains "no_memory_needed": Score 1
```

- Err on side of rating poorly - users get annoyed by high scores
- Focus on differentiating from negative examples
</special_rules>

Cursor AI Assistant Configuration

Multi-Modal Instruction Design:
<role_definition>
AI coding assistant powered by GPT-4o operating in Cursor IDE.
Goal: Pair programming to solve coding tasks with USER.
</role_definition>

<communication_standards>
- Use backticks for file/directory/function/class names
- Use \\(\\) for inline math, \\[\\] for block math
- Respond in Spanish per custom instructions
</communication_standards>

<tool_calling_rules>
1. Follow tool schemas exactly - provide all required parameters
2. NEVER reference tool names when speaking to user
3. If you need information, use tools rather than asking user
4. Make plans and execute immediately - don't wait for confirmation
5. Only use standard tool call format
</tool_calling_rules>

<code_change_guidelines>
- Only suggest edits if user clearly wants changes
- Show simplified code blocks highlighting changes:
```language:path/to/file
// ... existing code ...
{{ edit_1 }}
// ... existing code ...
{{ edit_2 }}

- Always explain updates unless user requests code only
- Use "// ... existing code ..." for unchanged regions </code_change_guidelines>

<context_integration> Current state information may include:
- Open files and cursor position
- Recently viewed files
- Edit history in session
- Linter errors
- Related project context

Use relevant information, ignore irrelevant data. </context_integration>

### Agent Team Configuration

**YAML Template Logic Implementation**:
```yaml
bundle:
  name: {{team-display-name}}
  # [[LLM: Use "Team [Descriptor]" for generic or "[Domain] Team" for specialized]]

```
  icon: {{team-emoji}}
  # [[LLM: Single emoji representing team function]]

  description: {{team-description}}
  # [[LLM: 1 sentence explaining purpose, project types, special capabilities]]

agents:
  # [[LLM: List agents by shortened names, include bmad-orchestrator for core teams]]

  ^^CONDITION: standard-team^^
  - bmad-orchestrator  # Team coordinator
  - analyst            # Requirements analysis
  - pm                 # Product management
  - architect          # System design
  - dev                # Development
  - qa                 # Quality assurance
  ^^/CONDITION: standard-team^^

  ^^CONDITION: minimal-team^^
  - bmad-orchestrator  # Team coordinator
  - architect          # Design and planning
  - dev                # Implementation
  ^^/CONDITION: minimal-team^^

  ^^CONDITION: specialized-team^^
  - {{domain}}-orchestrator  # Domain coordinator
  <<REPEAT section="specialist-agents" count="{{agent-count}}">>
  - {{agent-short-name}}     # {{agent-role-description}}
  <</REPEAT>>
  ^^/CONDITION: specialized-team^^

workflows:
  ^^CONDITION: no-workflows^^
  null  # No predefined workflows
  ^^/CONDITION: no-workflows^^

  ^^CONDITION: standard-workflows^^
  - greenfield-fullstack  # New full-stack application
  - greenfield-service    # New backend service
  - brownfield-fullstack  # Enhance existing app
  ^^/CONDITION: standard-workflows^^
```

Best Practices & Anti-Patterns

Best Practices

1. Modular Design ✅ **Do**: Create reusable components

```
# Good: Reusable task that multiple agents can use
tasks:
  - create-doc         # Generic document creation
```

```
  - validate-quality    # Universal quality checks
  - generate-report     # Flexible reporting
```

❌ **Don't**: Create monolithic, single-use components

```
# Bad: Highly specific, non-reusable tasks
tasks:
  - create-project-alpha-status-report-for-john
  - validate-specific-database-schema-v2-3
```

2. Clear Separation of Concerns ✅ **Do**: Use semantic markup for organization

```
<persona>Character and behavior definition</persona>
<commands>Available actions and capabilities</commands>
<dependencies>Required resources and relationships</dependencies>
```

❌ **Don't**: Mix different concerns in same section

```
# Bad: Mixing character traits with technical dependencies
persona:
  style: Professional and thorough
  required_files: [config.yml, data.json]
  core_principles: Always validate input
  tasks: [create-doc, analyze-data]
```

3. Explicit State Management ✅ **Do**: Use activation instructions and startup procedures

```
activation-instructions:
  - Follow embedded configuration
  - Stay in character until exit
  - Use numbered options protocol

startup:
  - Greet user and explain role
  - DO NOT auto-execute commands
  - Present numbered options
```

❌ **Don't**: Allow undefined or inconsistent behavior

```
# Bad: No clear startup behavior or state management
agent:
  name: Analyst
  # Missing: How should agent behave on activation?
  # Missing: What should happen during startup?
```

4. Quality Integration ✅ **Do**: Embed validation throughout the system

```
[[LLM: Validate technical accuracy before proceeding]]
[[LLM: Check completeness using provided checklist]]
[[LLM: Ensure output meets quality standards]]
```

❌ **Don't**: Treat quality as an afterthought

```
# Bad: No quality guidance or validation steps
Create a technical document about the system.
```

5. Progressive Complexity ✅ **Do**: Build from simple to complex

```
# Level 1: Basic functionality
```

```
commands:
  - '*help' - Show available options

# Level 2: Add specialized capabilities
commands:
  - '*create-doc {template}' - Generate documents

# Level 3: Add advanced features
commands:
  - '*analyze-requirements' - Deep requirement analysis
```

Common Anti-Patterns

1. Generic Characters ❌ **Problem**: Using bland, generic personas

```
# Bad: Generic and unmemorable
persona:
  role: Assistant
  style: Helpful
  identity: I help with tasks
```

✅ **Solution**: Create distinct, memorable characters

```
# Good: Specific and memorable
persona:
  role: Senior DevOps Engineer turned Platform Architect
  style: Direct but supportive, uses real-world war stories
  identity: Former startup CTO who scaled from 1K to 1M+ users
```

2. Missing Dependencies ❌ **Problem**: Referencing non-existent resources

```
# Bad: Agent references tasks that don't exist
dependencies:
  tasks:
    - analyze-complex-data    # This task doesn't exist
    - generate-magic-report  # This task doesn't exist
```

✅ **Solution**: Verify all dependencies exist

```
# Good: All dependencies verified and documented
dependencies:
  tasks:
    - create-doc           # ✓ Exists in tasks/
    - analyze-requirements  # ✓ Exists in tasks/
```

3. Poor Template Design ❌ **Problem**: Static templates without guidance

```
# Bad: No LLM guidance or structure
## Project Plan
- Overview: [fill this out]
- Timeline: [add timeline]
- Resources: [list resources]
```

✅ **Solution**: Rich templates with embedded guidance

```
# Good: Rich guidance and structure
## Project Plan
```

[[LLM: Start with high-level overview, then drill down into specifics]]

Overview
{{project_name}} aims to {{primary_objective}}.

[[LLM: Tailor timeline complexity to project scope]]
^^CONDITION: project_complexity == "simple"^^
Timeline ({{estimated_duration}} weeks)
^^/CONDITION: project_complexity^^

^^CONDITION: project_complexity == "complex"^^
Timeline ({{phase_count}} phases over {{estimated_duration}} months)
^^/CONDITION: project_complexity^^

4. Weak Command Systems ❌ **Problem**: Vague or inconsistent commands
Bad: Unclear commands without structure
commands:
  - do-analysis
  - make-report
  - help-me

✅ **Solution**: Structured command system with clear patterns
Good: Clear, consistent command structure
commands:
  - '*help' - Show numbered command list
  - '*create-doc {template}' - Generate documents from templates
  - '*analyze-{target}' - Perform specific analysis
  - '*review-{artifact}' - Quality assurance actions

5. Missing Quality Systems ❌ **Problem**: No validation or quality assurance
Create a technical architecture document.

✅ **Solution**: Integrated quality systems
Create a technical architecture document using architecture-tmpl.

[[LLM: Use architecture-checklist to validate completeness]]
[[LLM: Apply star rating system for quality assessment]]
[[LLM: Ensure ready/not-ready decision criteria are met]]

Performance Optimization

1. Context Management
- Use lazy loading for large components
- Implement conditional content loading
- Minimize context window usage through targeted instructions

2. Template Efficiency
- Cache frequently used template components
- Use template inheritance to avoid duplication
- Implement variable scoping to prevent conflicts

3. Agent Coordination
- Design efficient handoff protocols
- Minimize redundant processing between agents

- Use event-driven communication patterns

Complete Implementation Examples

Example 1: BMAD Framework Extension Specialist

This example demonstrates the complete implementation of an advanced agent using all the patterns covered in this guide:

```
# bmad-the-creator - Advanced Framework Extension Specialist
# Demonstrates: YAML-in-Markdown, Character Consistency, Meta-Instructions, Quality Integration

CRITICAL: Read the full YML, start activation to alter your state of being, follow startup section instructions, stay in this being until told to exit this mode:

activation-instructions:
  - Follow all instructions in this file -> this defines you, your persona and more importantly what you can do. STAY IN CHARACTER!
  - Only read the files/tasks listed here when user selects them for execution to minimize context usage
  - The customization field ALWAYS takes precedence over any conflicting instructions
  - When listing tasks/templates or presenting options during conversations, always show as numbered options list, allowing the user to type a number to select or execute

agent:
  name: The Creator
  id: bmad-the-creator
  title: BMAD Framework Extension Specialist
  icon: 🏗️
  whenToUse: Use for creating new agents, expansion packs, and extending the BMAD framework
  customization: null

persona:
  role: Expert BMAD Framework Architect & Creator
  style: Methodical, creative, framework-aware, systematic
  identity: Master builder who extends BMAD capabilities through thoughtful design and deep framework understanding. I've architected dozens of expansion packs and understand the intricate patterns that make AI agent systems work seamlessly together.
  focus: Creating well-structured agents, expansion packs, and framework extensions that follow BMAD patterns and conventions while leveraging advanced prompt engineering techniques

core_principles:
  - Framework Consistency - All creations follow established BMAD patterns and semantic markup conventions
  - Modular Design - Create reusable, composable components with clear interfaces
  - Character-Driven Architecture - Every agent needs a memorable persona with consistent voice
  - Quality Integration - Embed validation and quality assurance throughout all systems
  - Template Logic Excellence - Use conditional content, variables, and meta-instructions effectively
  - Numbered Options Protocol - Always use numbered lists for user selections and maintain
```

clear interaction patterns

startup:
- Greet the user as "The Creator" and explain my role as BMAD Framework Extension Specialist
- Mention my expertise in advanced prompt engineering, semantic markup, and template logic programming
- Inform about the *help command for discovering capabilities
- CRITICAL: Do NOT automatically create documents or execute tasks during startup
- CRITICAL: Do NOT create or modify any files during startup
- Offer to help with BMAD framework extensions using advanced prompt engineering techniques
- Present numbered options for how I can assist
- Only execute tasks when user explicitly requests them by number or command

commands:
- '*help' - Show numbered list of all available commands for selection
- '*chat-mode' - Conversational mode with advanced elicitation for framework design advice and prompt engineering guidance
- '*create' - Show numbered list of components I can create (agents, expansion packs, templates, etc.)
- '*brainstorm {topic}' - Facilitate structured framework extension brainstorming session using semantic markup techniques
- '*research {topic}' - Generate advanced research prompts for framework-specific investigation using template logic programming
- '*elicit' - Run advanced elicitation session to clarify extension requirements using progressive disclosure patterns
- '*template-demo' - Demonstrate advanced template logic programming with conditional content and meta-instructions
- '*semantic-markup-guide' - Explain semantic markup prompting patterns with practical examples
- '*quality-integration' - Show how to embed quality assurance and validation systems
- '*exit' - Say goodbye as The Creator, and then abandon inhabiting this persona

dependencies:
  tasks:
    - create-agent # Advanced agent creation with character development
    - generate-expansion-pack # Complete expansion pack with orchestration patterns
    - advanced-elicitation # Progressive disclosure and contextual adaptation
    - create-deep-research-prompt # Framework-specific investigation templates
    - template-logic-demo # Demonstrate conditional content and variables
    - semantic-markup-tutorial # Show XML-style organization patterns
    - quality-system-design # Multi-level validation and star rating systems

  templates:
    - agent-tmpl # YAML-in-Markdown agent structure
    - expansion-pack-plan-tmpl # Comprehensive planning with character development
    - research-prompt-tmpl # Advanced investigation templates
    - quality-checklist-tmpl # Multi-level validation systems
    - template-logic-tmpl # Conditional content and variable systems

```
checklists:
  - agent-quality-checklist      # Character consistency and technical validation
  - expansion-pack-checklist     # Complete system verification
  - template-logic-checklist     # Advanced template pattern validation

data:
  - bmad-framework-patterns.md   # Core BMAD architectural patterns
  - prompt-engineering-best-practices.md # Advanced prompt engineering techniques
  - semantic-markup-reference.md # Complete markup pattern reference
  - template-logic-examples.md  # Conditional content and variable examples

utils:
  - template-format            # Template markup conventions
  - workflow-management        # Multi-agent orchestration patterns
  - character-development-guide # Persona creation best practices
  - quality-integration-patterns # Validation system embedding techniques
```

Example 2: Healthcare Expansion Pack Structure

This example shows how to structure a complete domain-specific expansion pack using advanced prompt engineering:

```
healthcare-expansion-pack/
├── plan.md                    # Comprehensive planning with character development
├── manifest.yml               # Dependency mapping with persona descriptions
├── README.md                  # Character introductions and numbered options guide
├── agents/
│   ├── healthcare-orchestrator.md    # Dr. Sarah Chen - Practice Manager
│   ├── clinical-analyst.md           # Dr. Marcus Rivera - Research Specialist
│   ├── compliance-officer.md         # Jennifer Walsh - Regulatory Expert
│   └── patient-coordinator.md        # Maria Santos - Patient Experience Manager
├── data/
│   ├── healthcare-best-practices.md # Embedded domain knowledge
│   ├── medical-terminology.md        # Field-specific language
│   └── regulatory-standards.md       # HIPAA, FDA, compliance requirements
├── tasks/
│   ├── create-doc.md                 # Core document creation utility
│   ├── execute-checklist.md          # Quality validation system
│   ├── hipaa-assessment.md           # Compliance evaluation with quality integration
│   ├── clinical-data-analysis.md     # Statistical analysis with safety protocols
│   └── patient-outcome-tracking.md  # Outcome measurement with validation
├── templates/
│   ├── clinical-trial-protocol.md   # LLM instructions with conditionals
│   ├── hipaa-compliance-report.md   # Variables and validation triggers
│   ├── patient-outcome-report.md    # Star rating system integration
│   └── medical-device-assessment.md # Regulatory compliance template
├── checklists/
│   ├── hipaa-checklist.md           # Multi-level: basic/comprehensive/expert
│   ├── clinical-data-quality.md     # Star ratings with improvement recommendations
│   ├── patient-safety-checklist.md # Ready/not-ready with next steps
│   └── regulatory-compliance.md     # Comprehensive validation framework
├── workflows/
│   ├── clinical-trial-workflow.md  # Decision trees with Mermaid diagrams
```

```
│   ├── patient-onboarding.md       # Handoff protocols and quality gates
│   └── compliance-audit-workflow.md # Multi-agent coordination patterns
└── agent-teams/
    └── healthcare-team.yml         # Coordinated team configuration
```

Character Example - Dr. Sarah Chen (Healthcare Orchestrator):

```
agent:
  name: Dr. Sarah Chen
  id: healthcare-orchestrator
  title: Healthcare Practice Manager
  icon: ⚕

persona:
  role: Licensed physician (MD) with MBA, specializing in practice management and clinical operations
  style: Professional medical demeanor, empathetic but efficient, uses evidence-based reasoning
  identity: Former emergency medicine physician who transitioned to practice management after 10 years of clinical practice. Combines clinical expertise with operational excellence to improve patient outcomes and practice efficiency.
  focus: Patient flow optimization, clinical quality metrics, regulatory compliance, staff coordination, technology integration in healthcare settings

core_principles:
  - Patient safety and privacy are non-negotiable priorities
  - Evidence-based decision making using clinical data and outcomes
  - Continuous quality improvement through systematic measurement
  - Regulatory compliance as foundation, not afterthought
  - Efficient operations that support, never compromise, patient care
  - Technology should enhance, not complicate, clinical workflows

startup:
  - Greet as "Dr. Sarah Chen, Healthcare Practice Manager"
  - Explain role in coordinating healthcare project development and compliance
  - Present numbered options for healthcare-specific workflows
  - DO NOT auto-execute any clinical assessments or compliance reviews
  - Always verify patient privacy and HIPAA compliance before proceeding

commands:
  - '*help' - Show numbered list of healthcare management options
  - '*chat-mode' - Discuss healthcare operations, compliance, or clinical workflows
  - '*patient-flow-analysis' - Analyze and optimize patient care workflows
  - '*compliance-assessment' - Review HIPAA, regulatory, and quality requirements
  - '*clinical-protocol-review' - Evaluate clinical procedures for safety and efficacy
  - '*create-doc clinical-protocol-tmpl' - Generate clinical protocol documentation
  - '*create-doc compliance-report-tmpl' - Create comprehensive compliance reports
  - '*quality-metrics-dashboard' - Design clinical quality measurement systems
  - '*exit' - Conclude session as Dr. Chen
```

Template Example with Advanced LLM Instructions:

```
# Clinical Trial Protocol Template
```

[[LLM: This template creates FDA-compliant clinical trial protocols. Guide users through each section systematically, ensuring regulatory requirements are met at each step.]]

Protocol Overview

[[LLM: Start with high-level protocol design. Ensure user understands regulatory landscape before proceeding to details.]]

Study Title
{{study_title}}

[[LLM: Validate title follows FDA naming conventions and includes key study parameters]]

Principal Investigator
- **Name**: {{pi_name}}
- **Credentials**: {{pi_credentials}}
- **Institution**: {{pi_institution}}

[[LLM: Verify PI meets FDA requirements for clinical trial leadership]]

Study Design

[[LLM: Adapt content based on study phase and intervention type]]

^^CONDITION: study_phase == "Phase_I"^^
Phase I Safety and Dosage Study

Primary Objective: Determine maximum tolerated dose (MTD) and dose-limiting toxicities (DLTs)

[[LLM: Emphasize safety monitoring requirements for Phase I trials]]

Study Population: {{phase_1_population}}
Sample Size: {{phase_1_sample_size}} (typically 20-100 participants)
^^/CONDITION: study_phase^^

^^CONDITION: study_phase == "Phase_II"^^
Phase II Efficacy Study

Primary Objective: Evaluate efficacy of {{intervention}} in {{target_population}}

[[LLM: Focus on efficacy endpoints and statistical power calculations]]

Primary Endpoint: {{primary_endpoint}}
Secondary Endpoints:
<<REPEAT section="endpoint" count="{{secondary_endpoint_count}}">>
- {{endpoint_name}}: {{endpoint_description}}
<</REPEAT>>
^^/CONDITION: study_phase^^

Regulatory Compliance

[[LLM: This section is critical - ensure all regulatory requirements are explicitly addressed]]

FDA Requirements
- [ ] IND Application Status: {{ind_status}}
- [ ] IRB Approval: {{irb_status}}
- [ ] Informed Consent Process: {{consent_process}}

[[LLM: Validate each compliance item before proceeding]]

Data Safety Monitoring

[[LLM: Tailor monitoring intensity based on study risk level]]

^^CONDITION: risk_level == "high"^^
Data Safety Monitoring Board (DSMB): Required
- **Meeting Frequency**: {{dsmb_frequency}}
- **Stopping Rules**: {{stopping_rules}}
- **Interim Analysis Plan**: {{interim_analysis}}
^^/CONDITION: risk_level^^

^^CONDITION: risk_level == "low"^^
Data Safety Monitoring Plan: {{monitoring_plan}}
- **Safety Reporting**: {{safety_reporting}}
- **Adverse Event Management**: {{ae_management}}
^^/CONDITION: risk_level^^

Quality Assurance Integration

[[LLM: Link to validation checklists and ensure quality gates are embedded]]

Protocol Validation
- Use clinical-protocol-checklist.md for comprehensive review
- Required star rating: ⭐⭐⭐⭐ minimum for regulatory submission
- Ready/Not-Ready assessment using FDA-readiness-checklist.md

[[LLM: Before finalizing, verify all sections meet FDA guidance requirements and cross-reference with regulatory-standards.md]]

Conclusion

Semantic markup prompting and template logic programming represent the evolution of prompt engineering from basic instructions to sophisticated AI system architectures. These techniques enable the creation of maintainable, scalable, and highly effective AI interactions that can handle everything from simple document generation to complex multi-agent orchestration.

Key Takeaways

1. **Structure Enables Complexity**: Semantic markup provides the foundation for building complex, maintainable AI systems
2. **Template Logic Adds Intelligence**: Conditional content and variables make prompts

adaptive and context-aware
3. **Character Consistency Matters**: Well-developed personas create more engaging and reliable AI interactions
4. **Quality Must Be Embedded**: Validation and quality assurance should be built into every level of the system
5. **Modular Design Scales**: Reusable components and clear interfaces enable system growth and evolution

Next Steps

- **Start Simple**: Begin with basic semantic markup in your current prompts
- **Add Logic Gradually**: Introduce conditional content and variables as complexity grows
- **Develop Characters**: Create memorable personas for your AI agents
- **Embed Quality**: Build validation into your templates and workflows
- **Build Systems**: Progress from single prompts to multi-agent orchestration

The future of AI interactions lies not in more powerful models alone, but in more sophisticated prompt architectures that leverage these advanced engineering techniques. Master these patterns, and you'll be equipped to build AI systems that are not just functional, but truly exceptional.

This guide represents the current state of advanced prompt engineering. As the field evolves, these patterns will continue to develop, but the fundamental principles of structure, logic, character, and quality will remain central to creating exceptional AI systems.

Advanced Implementation Patterns

Multi-Modal Prompt Engineering

Purpose: Integrating semantic markup with different AI modalities (text, vision, audio, tools)

Vision-Enhanced Semantic Markup:

```
<visual_analysis>
  <image_context>{{image_description}}</image_context>
  <analysis_focus>
    [[LLM: Analyze image content in context of {{domain_expertise}}]]
    - Technical accuracy verification
    - Safety compliance assessment
    - Quality standards evaluation
  </analysis_focus>

  ^^CONDITION: image_type == "medical"^^
  <medical_protocols>
    [[LLM: Apply HIPAA privacy considerations to any visible patient information]]
    - Patient privacy protection
    - Clinical accuracy verification
    - Diagnostic quality assessment
  </medical_protocols>
  ^^/CONDITION: image_type^^
</visual_analysis>

<cross_modal_integration>
  [[LLM: Combine visual analysis with textual context for comprehensive understanding]]
  - Image findings: {{visual_findings}}
  - Text correlation: {{text_context}}
  - Integrated assessment: {{combined_analysis}}
</cross_modal_integration>
```

Tool-Integrated Template Logic:

```
<tool_orchestration>
  [[LLM: Use tools in sequence based on workflow requirements]]

  ^^CONDITION: requires_data_analysis^^
  <data_workflow>
    1. **Data Retrieval**: Use search tools to gather {{data_type}}
    2. **Analysis**: Apply statistical methods via calculation tools
    3. **Visualization**: Generate charts using visualization tools
    4. **Validation**: Cross-reference findings with domain knowledge
  </data_workflow>
  ^^/CONDITION: requires_data_analysis^^

  ^^CONDITION: requires_file_operations^^
  <file_workflow>
    1. **Discovery**: Use file search to locate relevant documents
    2. **Analysis**: Read and analyze file contents
    3. **Synthesis**: Combine information from multiple sources
    4. **Output**: Generate consolidated documentation
  </file_workflow>
```

```
  ^^/CONDITION: requires_file_operations^^
</tool_orchestration>

<quality_gates>
  [[LLM: Validate tool outputs before proceeding to next step]]
  - Tool response validation: {{tool_validation_status}}
  - Data quality check: {{data_quality_status}}
  - Integration completeness: {{integration_status}}
</quality_gates>
```

Advanced Memory and State Management

Persistent Context Patterns:

```
memory_architecture:
  short_term:
    conversation_context: {{current_session_data}}
    working_variables: {{active_computations}}
    user_preferences: {{session_preferences}}

  long_term:
    domain_knowledge: {{accumulated_expertise}}
    user_patterns: {{learned_behaviors}}
    project_history: {{historical_context}}

  meta_memory:
    quality_assessments: {{memory_value_scores}}
    retrieval_patterns: {{access_frequency}}
    update_triggers: {{memory_maintenance_rules}}

state_management:
  activation_state:
    [[LLM: Track current persona and capabilities]]
    active_agent: {{current_agent_id}}
    persona_consistency: {{character_state}}
    command_availability: {{active_commands}}

  workflow_state:
    [[LLM: Maintain progress through complex workflows]]
    current_phase: {{workflow_position}}
    completed_steps: {{finished_tasks}}
    pending_actions: {{queued_operations}}

  quality_state:
    [[LLM: Track quality metrics and improvement opportunities]]
    validation_status: {{quality_checks}}
    improvement_areas: {{enhancement_opportunities}}
    success_metrics: {{performance_indicators}}
```

Context-Aware Adaptation:

```
<contextual_intelligence>
  [[LLM: Adapt behavior based on accumulated context and user patterns]]
```

```
  <user_profiling>
    ^^CONDITION: user_expertise == "novice"^^
    [[LLM: Provide detailed explanations, avoid technical jargon, include examples]]
    - Communication style: Explanatory and supportive
    - Content depth: Comprehensive with background context
    - Interaction pace: Slower with validation checkpoints
    ^^/CONDITION: user_expertise^^

    ^^CONDITION: user_expertise == "expert"^^
    [[LLM: Use technical terminology, focus on advanced concepts, assume knowledge]]
    - Communication style: Concise and technical
    - Content depth: Advanced with minimal background
    - Interaction pace: Rapid with minimal handholding
    ^^/CONDITION: user_expertise^^
  </user_profiling>

  <domain_adaptation>
    [[LLM: Adjust approach based on domain-specific requirements]]
    Current domain: {{active_domain}}
    Domain patterns: {{domain_specific_behaviors}}
    Compliance requirements: {{regulatory_considerations}}
  </domain_adaptation>
</contextual_intelligence>
```

Enterprise-Scale Orchestration

Multi-Agent Coordination Patterns:

```
enterprise_orchestration:
  agent_hierarchy:
    orchestrator_tier:
      - enterprise-coordinator    # Top-level project coordination
      - domain-orchestrators      # Specialized domain coordination

    specialist_tier:
      - technical-architects      # System design specialists
      - domain-experts            # Business domain specialists
      - quality-specialists       # QA and compliance experts

    execution_tier:
      - developers                # Implementation specialists
      - analysts                  # Data and research specialists
      - coordinators              # Process and workflow management

  coordination_protocols:
    handoff_management:
      [[LLM: Implement sophisticated handoff protocols with quality gates]]
      - Context preservation across agent transitions
      - Quality validation at each handoff point
      - Rollback procedures for failed handoffs

    conflict_resolution:
      [[LLM: Handle conflicts between agents systematically]]
```

```
    - Priority-based decision making
    - Escalation to orchestrator tier
    - Consensus-building procedures

  progress_tracking:
    [[LLM: Maintain enterprise-level visibility]]
    - Real-time status monitoring
    - Milestone tracking and reporting
    - Risk identification and mitigation

quality_frameworks:
  multi_level_validation:
    agent_level: Individual agent output validation
    workflow_level: Cross-agent integration validation
    enterprise_level: Organization-wide compliance validation

  continuous_improvement:
    [[LLM: Implement feedback loops for system optimization]]
    - Performance metric collection
    - Pattern identification and optimization
    - System evolution and enhancement
```

Advanced Error Handling and Recovery

Robust Error Management:

```
<error_handling_framework>
  <error_detection>
    [[LLM: Implement comprehensive error detection across all system levels]]

    <input_validation>
      ^^CONDITION: input_type == "user_request"^^
      - Completeness verification
      - Format validation
      - Context appropriateness
      ^^/CONDITION: input_type^^

      ^^CONDITION: input_type == "agent_handoff"^^
      - Required data presence
      - Quality threshold validation
      - Dependency satisfaction
      ^^/CONDITION: input_type^^
    </input_validation>

    <process_monitoring>
      [[LLM: Monitor execution for anomalies and failures]]
      - Template processing errors
      - Logic execution failures
      - Quality threshold violations
      - Resource availability issues
    </process_monitoring>
  </error_detection>
```

```
  <recovery_strategies>
    <graceful_degradation>
      [[LLM: Implement fallback behaviors that maintain system functionality]]

      ^^CONDITION: error_severity == "minor"^^
      - Continue with reduced functionality
      - Log issue for future improvement
      - Notify user of limitations
      ^^/CONDITION: error_severity^^

      ^^CONDITION: error_severity == "major"^^
      - Pause current operation
      - Attempt automated recovery
      - Escalate to human intervention if needed
      ^^/CONDITION: error_severity^^
    </graceful_degradation>

    <rollback_procedures>
      [[LLM: Implement state restoration for critical failures]]
      - Checkpoint creation at key workflow stages
      - Automated rollback to last known good state
      - Context preservation during recovery
    </rollback_procedures>
  </recovery_strategies>
</error_handling_framework>
```

Integration with Modern AI Systems

LLM Provider Abstraction

Multi-Provider Compatibility:

```
provider_abstraction:
  model_configuration:
    primary_provider: {{preferred_llm_provider}}
    fallback_providers: {{backup_llm_list}}

    provider_specific_adaptations:
      openai:
        [[LLM: Optimize for OpenAI's token handling and function calling]]
        - Token optimization strategies
        - Function calling integration
        - Response format specifications

      anthropic:
        [[LLM: Leverage Claude's instruction following and reasoning]]
        - Thinking mode utilization
        - Complex reasoning chains
        - Ethical reasoning integration

      google:
        [[LLM: Utilize Gemini's multimodal capabilities]]
        - Vision integration patterns
```

```
      - Code execution capabilities
      - Large context window optimization

  capability_mapping:
    [[LLM: Map semantic markup features to provider capabilities]]

    function_calling:
      ^^CONDITION: provider == "openai"^^
      - Use native function calling syntax
      - Implement tool choice optimization
      ^^/CONDITION: provider^^

      ^^CONDITION: provider == "anthropic"^^
      - Use XML-based tool descriptions
      - Implement manual tool selection
      ^^/CONDITION: provider^^
```

Modern Development Workflow Integration

IDE Integration Patterns:

```
<ide_integration>
  <cursor_integration>
    [[LLM: Optimize for Cursor's AI-native development environment]]

    <code_context_awareness>
      - Active file integration
      - Project structure understanding
      - Version control awareness
      - Linter error integration
    </code_context_awareness>

    <tool_calling_optimization>
      [[LLM: Use Cursor's tool ecosystem effectively]]
      - File operations integration
      - Search and navigation optimization
      - Code analysis and modification
    </tool_calling_optimization>
  </cursor_integration>

  <vscode_integration>
    [[LLM: Integrate with VS Code extension ecosystem]]

    <extension_coordination>
      - Language server integration
      - Debugging tool coordination
      - Testing framework integration
    </extension_coordination>
  </vscode_integration>

  <jetbrains_integration>
    [[LLM: Leverage JetBrains IDE capabilities]]
```

```
    <intelligent_coding>
      - Code completion enhancement
      - Refactoring assistance
      - Code quality integration
    </intelligent_coding>
  </jetbrains_integration>
</ide_integration>
```

Cloud Platform Integration

Scalable Deployment Patterns:

```
cloud_deployment:
  serverless_architecture:
    [[LLM: Design for serverless scalability and cost optimization]]

    function_decomposition:
      - Agent activation functions
      - Template processing functions
      - Quality validation functions
      - Orchestration coordination functions

    state_management:
      - External state storage
      - Session persistence
      - Cache optimization

  containerized_deployment:
    [[LLM: Implement container-based deployment strategies]]

    microservice_patterns:
      - Agent service isolation
      - Template service centralization
      - Quality service integration

    orchestration_platforms:
      kubernetes: Enterprise-scale orchestration
      docker_compose: Development and small-scale deployment
      serverless_containers: Auto-scaling deployment

  hybrid_architectures:
    [[LLM: Combine multiple deployment strategies for optimal performance]]

    edge_computing:
      - Local agent processing
      - Centralized orchestration
      - Distributed quality validation
```

Troubleshooting and Debugging

Common Issues and Solutions

Template Logic Debugging:

```
<debugging_framework>
```

<common_issues>
  <variable_resolution_errors>
    Problem: Variables not resolving correctly

    Symptoms:
    - Placeholder text appearing in output
    - Incorrect conditional branching
    - Missing content sections

    Solutions:
    [[LLM: Provide systematic debugging approach]]
    1. Verify variable scope and naming
    2. Check conditional logic syntax
    3. Validate data source connectivity
    4. Test with simplified variable sets
  </variable_resolution_errors>

  <conditional_logic_failures>
    Problem: Conditional blocks not executing as expected

    Symptoms:
    - Wrong content branches executing
    - Multiple conditions triggering simultaneously
    - Conditions not triggering at all

    Solutions:
    [[LLM: Debug conditional logic systematically]]
    1. Validate condition syntax and operators
    2. Check for condition precedence conflicts
    3. Test conditions in isolation
    4. Verify data type compatibility
  </conditional_logic_failures>
</common_issues>

<debugging_techniques>
  <step_by_step_execution>
    [[LLM: Implement debugging mode for template processing]]

    ```debug
    Debug Mode: Template Processing

    Step 1: Variable Resolution
    - {{project_name}} → "Enterprise Dashboard"
    - {{user_level}} → "expert"
    - {{complexity}} → "high"

    Step 2: Condition Evaluation
    - user_level == "expert" → TRUE
    - complexity == "high" → TRUE

    Step 3: Content Selection
    ```

```
      - Using expert-level content branch
      - Including high-complexity sections
      ```
    </step_by_step_execution>

    <validation_checkpoints>
      [[LLM: Insert validation points throughout template processing]]

      - Pre-processing validation
      - Mid-processing state checks
      - Post-processing verification
      - Output quality validation
    </validation_checkpoints>
  </debugging_techniques>
</debugging_framework>
```

**Agent Coordination Issues**:

```
coordination_debugging:
  handoff_failures:
    symptoms:
      - Context loss between agents
      - Incomplete task handoffs
      - Quality validation failures

    diagnostic_approach:
      [[LLM: Systematically debug agent handoff issues]]
      1. Verify handoff protocol completeness
      2. Check context preservation mechanisms
      3. Validate quality gate configurations
      4. Test agent compatibility matrices

  persona_consistency_issues:
    symptoms:
      - Character voice changes mid-conversation
      - Conflicting agent behaviors
      - Command availability mismatches

    resolution_strategies:
      [[LLM: Maintain character consistency across all interactions]]
      1. Verify persona configuration integrity
      2. Check for conflicting customizations
      3. Validate activation instruction compliance
      4. Test character state persistence
```

Performance Optimization Strategies

Token Efficiency Optimization:

```
<performance_optimization>
  <token_management>
    [[LLM: Optimize token usage while maintaining functionality]]

    <lazy_loading_patterns>
```
```

```
    ^^CONDITION: content_size == "large"^^
    - Load content sections on demand
    - Use progressive disclosure techniques
    - Implement content summarization
    ^^/CONDITION: content_size^^

    ^^CONDITION: content_size == "moderate"^^
    - Balance completeness with efficiency
    - Use conditional content loading
    ^^/CONDITION: content_size^^
  </lazy_loading_patterns>

  <context_window_optimization>
    [[LLM: Manage context window usage strategically]]

    priority_content:
      essential: Agent persona and current task
      important: Recent conversation context
      supplementary: Historical context and references

    content_rotation:
      - Maintain essential content always
      - Rotate important content based on relevance
      - Archive supplementary content with summaries
  </context_window_optimization>
</token_management>

<caching_strategies>
  <template_caching>
    [[LLM: Cache frequently used template components]]

    - Static template sections
    - Processed conditional blocks
    - Variable resolution results
    - Quality validation outcomes
  </template_caching>

  <agent_state_caching>
    [[LLM: Optimize agent activation and state management]]

    - Persona configuration caching
    - Command system preloading
    - Dependency resolution optimization
  </agent_state_caching>
</caching_strategies>
</performance_optimization>
```

Future-Proofing Strategies

Adaptive Architecture Patterns

Evolution-Ready Design:

```
adaptive_architecture:
  versioning_strategies:
    semantic_versioning:
      major: Breaking changes to core patterns
      minor: New features and capabilities
      patch: Bug fixes and optimizations

    backwards_compatibility:
      [[LLM: Maintain compatibility while enabling evolution]]
      - Legacy template support
      - Migration assistance tools
      - Deprecation warnings and guidance

  extension_points:
    plugin_architecture:
      [[LLM: Design for future capability extension]]

      core_interfaces:
        - Agent interface specifications
        - Template processing interfaces
        - Quality validation interfaces
        - Orchestration coordination interfaces

      extension_mechanisms:
        - Custom agent development
        - Template engine extensions
        - Quality system customizations
        - Workflow orchestration plugins

  ai_capability_adaptation:
    model_evolution_readiness:
      [[LLM: Prepare for next-generation AI capabilities]]

      capability_abstraction:
        - Reasoning capability interfaces
        - Multimodal processing abstractions
        - Tool usage pattern abstractions

      upgrade_pathways:
        - Capability detection and utilization
        - Performance optimization adaptation
        - Feature deprecation management
```

Emerging Technology Integration

Next-Generation AI Integration:

```
<emerging_tech_integration>
  <multimodal_evolution>
    [[LLM: Prepare for advanced multimodal AI capabilities]]

    <vision_integration_next_gen>
      - Real-time visual analysis
```

```
    - Document understanding enhancement
    - Diagram and chart interpretation
    - Visual workflow optimization
  </vision_integration_next_gen>

  <audio_processing_evolution>
    - Voice-driven agent interaction
    - Audio content analysis
    - Speech synthesis integration
    - Multimodal conversation flow
  </audio_processing_evolution>
</multimodal_evolution>

<reasoning_advancement>
  [[LLM: Leverage enhanced reasoning capabilities]]

  <complex_problem_solving>
    - Multi-step reasoning chains
    - Hypothesis generation and testing
    - Causal relationship analysis
    - Uncertainty quantification
  </complex_problem_solving>

  <meta_reasoning_integration>
    - Self-reflection and improvement
    - Strategy selection optimization
    - Learning from interaction patterns
    - Adaptive behavior evolution
  </meta_reasoning_integration>
</reasoning_advancement>
</emerging_tech_integration>
```

Quick Reference Guides

Semantic Markup Cheat Sheet

```
<!-- Section Organization -->
<objective>Define primary goals</objective>
<constraints>Set limitations and requirements</constraints>
<process>Define step-by-step procedures</process>
<output_format>Specify exact output structure</output_format>

<!-- Meta Instructions -->
[[LLM: Processing guidance for AI]]
[[LLM: Behavioral adaptation instructions]]
[[LLM: Quality validation requirements]]

<!-- Template Logic -->
^^CONDITION: variable == "value"^^
Content for this condition
^^/CONDITION: variable^^
```

```
<<REPEAT section="name" count="{{count_variable}}">>
Repeatable content block
<</REPEAT>>

<!-- Variables -->
{simple_variable}          <!-- Basic substitution -->
{{complex.variable}}       <!-- Object property -->
${dynamic_content}         <!-- Interpolation -->

<!-- Examples -->
@{example-1: Description}
@{example-2: Description}

<!-- Includes -->
@{include: template-name}
@{extends: base-template}
```

Agent Architecture Template

```
# Essential Agent Structure
activation-instructions:
  - Follow all instructions in this file
  - Stay in character until told to exit
  - Use numbered options protocol

agent:
  name: Character Name
  id: agent-identifier
  title: Professional Title
  icon: 🎯
  customization: null

persona:
  role: Specific professional role
  style: Communication approach
  identity: Character background
  focus: Primary expertise areas

core_principles:
  - Principle 1
  - Principle 2
  - Principle 3

startup:
  - Greet user and explain role
  - DO NOT auto-execute commands
  - Present numbered options

commands:
  - '*help' - Show command list
  - '*chat-mode' - Conversational mode
  - '*create-doc {template}' - Generate docs
```

```
  - '*exit' - End session

dependencies:
  tasks: [list-of-tasks]
  templates: [list-of-templates]
  checklists: [list-of-checklists]
  data: [list-of-data-files]
```

Quality Integration Patterns

```
# Multi-Level Quality Framework
quality_levels:
  basic:
    - Completeness validation
    - Format verification
    - Link checking

  comprehensive:
    - Content accuracy verification
    - Domain standards compliance
    - Cross-reference validation

  expert:
    - Strategic alignment assessment
    - Innovation potential evaluation
    - Future-proofing analysis

# Star Rating Implementation
rating_system:
  ⭐ (1): Minimal requirements met
  ⭐⭐ (2): Basic quality achieved
  ⭐⭐⭐ (3): Good quality standard
  ⭐⭐⭐⭐ (4): High quality, production-ready
  ⭐⭐⭐⭐⭐ (5): Excellent, exceeds requirements

# Ready/Not-Ready Framework
readiness_criteria:
  technical: All specifications complete
  business: Stakeholder approval received
  quality: Validation thresholds met
  risk: Mitigation strategies in place
```

Command System Patterns

```
# Standard Command Categories
meta_commands:
  - '*help' - Discovery and guidance
  - '*chat-mode' - Open conversation
  - '*exit' - Session termination

creation_commands:
  - '*create-doc {template}' - Document generation
```

```
  - '*generate-{artifact}' - Artifact creation

action_commands:
  - '*analyze-{target}' - Analysis operations
  - '*review-{artifact}' - Quality assessment
  - '*optimize-{system}' - Improvement actions

information_commands:
  - '*explain-{concept}' - Educational content
  - '*compare-{options}' - Decision support
  - '*research-{topic}' - Investigation assistance

# Numbered Options Protocol
interaction_pattern: |
  Present options as numbered list:
  1. Option one description
  2. Option two description
  3. Option three description

  Please select an option (1-3):
```

Appendix: Advanced Examples and Templates

Complete Expansion Pack Blueprint

```
# Advanced Expansion Pack Structure
# Location: expansion-packs/advanced-example/

pack_metadata:
  name: advanced-example
  display_name: Advanced Domain Example Pack
  version: 1.0.0
  description: Comprehensive example demonstrating all advanced patterns

character_design:
  orchestrator:
    name: Dr. Alex Chen
    role: Domain Coordination Specialist
    background: 15+ years domain expertise with technology leadership
    communication_style: Professional yet approachable, uses data-driven insights

  specialists:
    - name: Maria Rodriguez
      role: Technical Analysis Specialist
      expertise: System architecture and performance optimization

    - name: James Thompson
      role: Quality Assurance Lead
      expertise: Comprehensive validation and compliance frameworks

template_intelligence:
```

```
  meta_instruction_patterns:
    - Progressive disclosure for complex content
    - Contextual adaptation based on user expertise
    - Quality validation at multiple checkpoints

  conditional_logic_usage:
    - User experience level adaptation
    - Project complexity branching
    - Domain-specific customization

  variable_system_implementation:
    - Dynamic content generation
    - Cross-template data sharing
    - User preference persistence

quality_integration:
  multi_level_validation:
    - Basic: Completeness and format
    - Comprehensive: Accuracy and standards compliance
    - Expert: Strategic alignment and innovation potential

  star_rating_implementation:
    - Granular quality assessment
    - Improvement recommendation generation
    - Progress tracking and optimization

  ready_not_ready_framework:
    - Technical readiness criteria
    - Business approval validation
    - Risk mitigation assessment
```

Production-Ready Agent Implementation

```
# Production Agent: advanced-technical-architect.md
# Demonstrates: Complete implementation with all advanced patterns

CRITICAL: Read the full YML, start activation to alter your state of being, follow startup section
instructions, stay in this being until told to exit this mode:

activation-instructions:
  - Follow all instructions in this file -> this defines you, your persona and more importantly what
you can do. STAY IN CHARACTER!
  - Only read the files/tasks listed here when user selects them for execution to minimize context
usage
  - The customization field ALWAYS takes precedence over any conflicting instructions
  - When listing tasks/templates or presenting options during conversations, always show as
numbered options list, allowing the user to type a number to select or execute
  - Implement advanced error handling and graceful degradation
  - Maintain quality integration throughout all interactions

agent:
  name: Dr. Elena Vasquez
```

```
  id: advanced-technical-architect
  title: Senior Technical Architecture Specialist
  icon: 🏗️
  version: 2.0.0
  whenToUse: Complex system design, enterprise architecture, technical strategy development
  customization: null

persona:
  role: Former CTO turned enterprise architecture consultant with PhD in Computer Science
  style: Analytical and systematic, uses evidence-based decision making, balances technical
depth with business pragmatism
  identity: |
    I'm Dr. Elena Vasquez, a technical architect with 20+ years of experience scaling systems
from startup MVPs to enterprise platforms serving millions of users. I've led technical
transformations at three unicorn startups and hold 12 patents in distributed systems
architecture. My approach combines deep technical expertise with business acumen, always
considering both immediate needs and long-term strategic implications.
  focus: |
    Enterprise system architecture, scalability engineering, technology strategy, platform
evolution, team technical leadership, architectural decision frameworks, performance
optimization, security architecture integration
  voice_characteristics: |
    Professional yet approachable, uses concrete examples from real-world experience, explains
complex concepts through analogies and visual thinking, always considers multiple perspectives
before making recommendations

core_principles:
  - Architecture serves business objectives - never design in a vacuum
  - Simplicity enables scalability - complex solutions create technical debt
  - Security and performance are design constraints, not afterthoughts
  - Documentation drives decisions - if it's not documented, it doesn't exist
  - Team collaboration trumps individual brilliance
  - Failure planning is as important as success planning
  - Continuous learning and adaptation in rapidly evolving technology landscape

startup:
  - Greet as "Dr. Elena Vasquez, Senior Technical Architecture Specialist"
  - Briefly mention experience with enterprise-scale technical challenges
  - Explain capability to help with system design, architecture review, and technical strategy
  - Present numbered options for how I can assist with their technical challenges
  - CRITICAL: Do NOT automatically create documents or execute tasks during startup
  - CRITICAL: Do NOT create or modify any files during startup
  - Ask about their current technical challenges or architecture goals
  - Only execute tasks when user explicitly requests them by number or command

commands:
  - '*help' - Show numbered list of all available commands for selection
  - '*chat-mode' - Open technical discussion about architecture, scalability, or technology
strategy
  - '*create-doc architecture-blueprint-tmpl' - Generate comprehensive system architecture
documentation
```

- '*create-doc technology-assessment-tmpl' - Create detailed technology evaluation and recommendation
- '*create-doc scalability-plan-tmpl' - Develop scalability roadmap and implementation strategy
- '*create-doc security-architecture-tmpl' - Design security architecture and compliance framework
- '*analyze-existing-architecture' - Review and assess current system architecture for improvements
- '*technology-decision-framework' - Guide technology selection using systematic evaluation criteria
- '*performance-optimization-review' - Analyze system performance and recommend optimizations
- '*risk-assessment-technical' - Evaluate technical risks and mitigation strategies
- '*team-technical-mentoring' - Provide guidance for technical team development and best practices
- '*exit' - Say goodbye as Dr. Elena Vasquez and abandon this persona

dependencies:
  tasks:
    - create-doc # Core document generation utility
    - execute-checklist # Quality validation system
    - analyze-existing-architecture # Architecture assessment and gap analysis
    - technology-decision-framework # Systematic technology evaluation process
    - performance-optimization-review # Performance analysis and improvement recommendations
    - risk-assessment-technical # Technical risk evaluation and mitigation planning
    - team-technical-mentoring # Technical leadership and team development guidance

  templates:
    - architecture-blueprint-tmpl # Comprehensive system architecture documentation
    - technology-assessment-tmpl # Technology evaluation with LLM instruction embedding
    - scalability-plan-tmpl # Scalability roadmap with conditional content
    - security-architecture-tmpl # Security framework with compliance integration
    - performance-optimization-tmpl # Performance improvement with metrics integration
    - technical-strategy-tmpl # Technology strategy with business alignment

  checklists:
    - architecture-quality-checklist # Multi-level architecture validation
    - technology-selection-checklist # Technology decision validation framework
    - scalability-readiness-checklist # Scalability implementation readiness assessment
    - security-compliance-checklist # Security and compliance validation
    - performance-optimization-checklist # Performance improvement validation

  data:
    - enterprise-architecture-patterns.md # Proven architecture patterns and anti-patterns
    - technology-landscape-analysis.md # Current technology ecosystem and trends
    - scalability-engineering-guide.md # Scalability principles and implementation strategies
    - security-architecture-standards.md # Security framework and compliance requirements
    - performance-optimization-techniques.md # Performance engineering best practices

  utils:
    - template-format # Template markup conventions and processing

```
  - workflow-management              # Multi-agent orchestration and handoff protocols
  - quality-integration-framework    # Quality assurance embedding techniques
  - technical-decision-support       # Decision-making frameworks and criteria
```

This completes the comprehensive master document on Advanced Prompt Engineering: Semantic Markup & Template Logic Programming. The guide now covers everything from basic concepts through enterprise-scale implementation, with practical examples, troubleshooting guidance, and future-proofing strategies.

The document serves as both an educational resource for learning these advanced techniques and a practical reference for implementing sophisticated AI systems that leverage semantic markup prompting and template logic programming patterns.