The SPARC Universal Program Development Workflow

This workflow leverages the SPARC multi-agent orchestration (roomodes) and deep research capabilities to systematically transform an idea into a fully functional, well-documented, and maintainable software application. It emphasizes modularity, security, testability, and adherence to best practices outlined in the Prompt Engineering guide.

Core Principles (To be enforced throughout):

1. **Modularity:** Break down problems into the smallest logical, independent units. Enforce file size limits (< 500 lines).
2. **Environment Safety:** Absolutely NO hard-coded secrets, API keys, or environment-specific values in code, specs, or docs. Use configuration abstractions and secure management (DevOps, Security Reviewer).
3. **Testability:** Design for testing from the start (TDD). Ensure adequate coverage.
4. **Clear Communication:** Use precise instructions, provide examples (few-shot), specify output formats (JSON/Markdown where appropriate), and prefer positive instructions over constraints.
5. **Iteration & Feedback:** Development is cyclical. Revisit phases as needed based on testing, feedback, and refinement.
6. **Documentation:** Maintain clear, concise documentation at all stages (Specs, Architecture, Code Comments, User Docs).
7. **SPARC Orchestration:** Use the SPARC mode (slug: sparc) as the primary interface to delegate tasks via new_task to the specialized modes. Ensure all sub-tasks conclude with attempt_completion.
8. **Leverage Research:** Utilize the Deep Research mode exhaustively in the initial phases and whenever new complex requirements or technical hurdles arise.

The Workflow Phases:

Phase 0: Conception & Vision Clarification

- **Goal:** Define the core idea, target audience, and high-level objectives of the program.
- **User Input:** Describe the program you want to build in natural language. What problem does it solve? Who is it for? What are the must-have features?
- **Activities:**

1. User provides the initial vision to the SPARC orchestrator.
2. SPARC might use the Ask mode (slug: ask) to help the user refine the vision, clarify ambiguities, and frame the initial request.

- **Primary Modes:** SPARC, Ask.
- **Output:** A concise, clear statement summarizing the program's purpose, target audience, and key goals.

Phase 1: Deep Research & Comprehensive Specification

- **Goal:** Gather *all* necessary information and translate it into a detailed, actionable specification. This is CRITICAL.
- **Activities:**
    1. **Initiate Deep Research:** Task the SPARC orchestrator to use the Deep Research mode (slug: deep-research). Provide the program goal statement and specific areas to investigate (e.g., target user workflows, existing solutions, potential APIs/libraries, algorithms, data structures, security considerations, compliance requirements, UI/UX paradigms, scalability concerns, potential tech stacks).
        - *Prompting Deep Research:* Be specific. Ask for structured output (e.g., "Research authentication methods for web apps targeting enterprise users, covering SSO, MFA, passwordless options. Provide pros/cons, security implications, and implementation complexity for each. Cite sources.") Use its recursive capabilities to dive deeper into identified knowledge gaps.
    2. **Synthesize Research:** Review the structured output from Deep Research.
    3. **Write Specifications:** Task SPARC to use the Specification Writer (slug: spec-pseudocode). Feed it the synthesized research findings and the refined program vision.
        - Instruct it to create modular .md files covering: Functional Requirements, Non-Functional Requirements (performance, security, scalability), User Stories/Use Cases, Edge Cases, Constraints, Data Models (initial draft), UI/UX flow outlines, and TDD anchors/pseudocode for core logic modules.
- **Primary Modes:** SPARC, Deep Research, Specification Writer.

- **Inputs:** Program goal statement, User refinements, Specific research questions.
- **Outputs:**
    - Comprehensive research documentation (organized file structure as defined in Deep Research mode).
    - Detailed, modular Specification Documents (phase_number_name.md files).

Phase 2: Architecture & System Design

- **Goal:** Define the high-level structure, components, interactions, data flow, and technology stack.
- **Activities:**
    1. Task SPARC to use the Architect mode (slug: architect). Provide the full set of Specification Documents.
    2. Instruct Architect to design a scalable, secure, modular architecture. Define service boundaries, API contracts, data flow diagrams (e.g., using Mermaid syntax), component responsibilities, and database schema (if applicable, collaborating with Supabase Admin later).
    3. Select the primary technology stack based on research and requirements.
- **Primary Modes:** SPARC, Architect. Potentially Supabase Admin (slug: supabase-admin) for initial schema consultation if using Supabase.
- **Inputs:** Specification Documents.
- **Outputs:** Architecture diagrams (.md or diagram files), API definitions, Data model specifications, Technology stack choices documented.

Phase 3: Implementation (Iterative Cycles - TDD Driven)

- **Goal:** Build the program features incrementally based on specs and architecture, ensuring quality through testing.
- **Activities (Repeated for each logical feature/module):**
    1. **Task Breakdown:** User/SPARC identifies a small, implementable feature/module from the specs.

2. **Write Failing Tests:** Task SPARC to use Tester (TDD) (slug: tdd) to write comprehensive (unit, potentially integration) tests for the feature *first*, based on the spec. These tests should fail initially.
3. **Implement Code:** Task SPARC to use the appropriate coding mode (slug: code for general logic, slug: supabase-admin for database interactions/RLS/functions, slug: mcp-integration for external service connections) to write the *minimum* amount of code required to make the tests pass. Emphasize clean architecture principles and config abstractions.
4. **Refactor:** Once tests pass ("Green"), task SPARC to use Tester (TDD) or Code to refactor the newly written code and tests for clarity, efficiency, and adherence to best practices, ensuring tests still pass.
5. **Security Check (Optional but recommended):** Periodically, or for critical modules, task SPARC to use Security Reviewer (slug: security-review) to scan the new code.

- **Primary Modes:** SPARC (Orchestration), Tester (TDD), Code, Supabase Admin, MCP Integration, Security Reviewer.
- **Inputs:** Specification module, Architecture diagrams/contracts, Existing codebase/tests.
- **Outputs:** Implemented feature code, Passing test suite for the feature, Refactored/clean code.

Phase 4: Integration & End-to-End Testing

- **Goal:** Combine individually implemented modules into a cohesive system and verify their interactions.
- **Activities:**
    1. Task SPARC to use the System Integrator (slug: integration) to connect the various modules according to the architecture.
    2. Task SPARC (potentially guiding TDD or Code) to write and execute integration tests and end-to-end tests that cover key user workflows across module boundaries.
    3. If tests fail or bugs are found, task SPARC to use the Debugger (slug: debug) to identify and fix the issues, looping back to Phase 3 if necessary.
- **Primary Modes:** SPARC, System Integrator, Tester (TDD), Debugger.

- **Inputs:** Implemented code modules, Architecture definitions.
- **Outputs:** Integrated application build, Integration/E2E test suite results, Bug fixes.

Phase 5: Documentation Creation

- **Goal:** Generate comprehensive documentation for end-users and developers. (Can often run parallel to Phases 3 & 4).
- **Activities:**
    1. Task SPARC to use the Documentation Writer (slug: docs-writer). Provide specs, architecture documents, and potentially code comments/context.
    2. Instruct it to generate: User Manuals, API Documentation, Setup/Installation Guides, Configuration Guides, Architecture Overview (if not already sufficiently covered). Keep docs modular and in .md format.
- **Primary Modes:** SPARC, Documentation Writer.
- **Inputs:** Specs, Architecture docs, Codebase (for API docs generation possibly), User stories.
- **Outputs:** Set of Markdown documentation files.

Phase 6: Deployment & Infrastructure Setup

- **Goal:** Deploy the integrated and tested application to the target environment(s).
- **Activities:**
    1. Define infrastructure requirements (servers, databases, serverless functions, domains, etc.).
    2. Task SPARC to use the DevOps mode (slug: devops). Provide the integrated codebase, infrastructure requirements, and target environment details.
    3. Instruct DevOps to: provision infrastructure, set up CI/CD pipelines, securely configure environment variables (using secret managers, NEVER hardcoding), deploy the application, set up monitoring hooks.
    4. Task SPARC to have Security Reviewer audit deployment configurations.
- **Primary Modes:** SPARC, DevOps, Security Reviewer.
- **Inputs:** Integrated codebase, Test results, Infrastructure requirements, Target environment details.

- **Outputs:** Deployed application, Configured CI/CD pipeline, Infrastructure-as-code scripts (optional), Monitoring setup.

Phase 7: Post-Deployment Monitoring & Maintenance

- **Goal:** Observe the live application, gather feedback, identify issues, and perform routine maintenance.
- **Activities:**
    1. Task SPARC to use the Deployment Monitor (slug: post-deployment-monitoring-mode) to actively watch logs, metrics, and uptime. Configure alerts for anomalies.
    2. Collect user feedback.
    3. When bugs are reported or detected, task SPARC to use Debugger to diagnose.
    4. Feed bug fixes back into the loop (Phase 3 -> 4 -> 6).
- **Primary Modes:** SPARC, Deployment Monitor, Debugger.
- **Inputs:** Live application metrics/logs, User feedback, Alerting rules.
- **Outputs:** Performance reports, Alert notifications, Diagnosed bugs, Maintenance patches/updates deployed.

Phase 8: Refinement & Optimization

- **Goal:** Continuously improve the application's performance, cost-efficiency, security, and maintainability based on ongoing monitoring and evolving requirements.
- **Activities:**
    1. Analyze monitoring data and user feedback for improvement opportunities.
    2. Task SPARC to use the Optimizer (slug: refinement-optimization-mode) to refactor bottlenecks, improve resource utilization, or enhance modularity.
    3. Task SPARC to use Security Reviewer for periodic security audits of the live system and codebase.
    4. Feed approved optimizations and security enhancements back into the loop (Phase 3 -> 4 -> 6).

- **Primary Modes:** SPARC, Optimizer, Security Reviewer, Deployment Monitor.
- **Inputs:** Monitoring data, Performance analysis, User feedback, Security audit findings, New feature requests (which might trigger a loop back to Phase 1).
- **Outputs:** Optimized code/architecture, Enhanced security posture, Improved performance/cost-efficiency metrics, New features (if applicable).

This comprehensive workflow provides a repeatable blueprint. The user's primary role is to provide the initial vision, specific requirements, feedback during iterations, and final approval, while leveraging the SPARC orchestrator to manage the execution across the specialized roomodes. The Deep Research mode is the critical enabler for tackling *any* program by ensuring the necessary domain knowledge is gathered upfront.