/guidelines-advisor

Source: `~/.claude/skills/tob-building-secure-contracts/skills/guidelines-advisor/SKILL.md`

---

name: guidelines-advisor description: Smart contract development advisor based on Trail of Bits' best practices. Analyzes codebase to generate documentation/specifications, review architecture, check upgradeability patterns, assess implementation quality, identify pitfalls, review dependencies, and evaluate testing. Provides actionable recommendations.

Guidelines Advisor

Purpose

Systematically analyzes the codebase and provides guidance based on Trail of Bits' development guidelines:

1. **Generate documentation and specifications** (plain English descriptions, architectural diagrams, code documentation)
2. **Optimize on-chain/off-chain architecture** (only if applicable)
3. **Review upgradeability patterns** (if your project has upgrades)
4. **Check delegatecall/proxy implementations** (if present)
5. **Assess implementation quality** (functions, inheritance, events)
6. **Identify common pitfalls**
7. **Review dependencies**
8. **Evaluate test suite and suggest improvements**

Framework: Building Secure Contracts - Development Guidelines

---

How This Works

Phase 1: Discovery & Context

Explores the codebase to understand:

- Project structure and platform
- Contract/module files and their purposes
- Existing documentation
- Architecture patterns (proxies, upgrades, etc.)
- Testing setup
- Dependencies

Phase 2: Documentation Generation

Helps create:

- Plain English system description
- Architectural diagrams (using Slither printers for Solidity)
- Code documentation recommendations (NatSpec for Solidity)

Phase 3: Architecture Analysis

Analyzes:

- On-chain vs off-chain component distribution (if applicable)
- Upgradeability approach (if applicable)
- Delegatecall proxy patterns (if present)

Phase 4: Implementation Review

Assesses:

- Function composition and clarity
- Inheritance structure
- Event logging practices
- Common pitfalls presence
- Dependencies quality
- Testing coverage and techniques

Phase 5: Recommendations

Provides:

- Prioritized improvement suggestions
- Best practice guidance
- Actionable next steps

Assessment Areas

I analyze 11 comprehensive areas covering all aspects of smart contract development. For detailed criteria, best practices, and specific checks, see ASSESSMENT_AREAS.md.

Quick Reference:

1. **Documentation & Specifications**
   - Plain English system descriptions
   - Architectural diagrams
   - NatSpec completeness (Solidity)
   - Documentation gaps identification
2. **On-Chain vs Off-Chain Computation**
   - Complexity analysis
   - Gas optimization opportunities
   - Verification vs computation patterns

3. **Upgradeability**
   - Migration vs upgradeability trade-offs
   - Data separation patterns
   - Upgrade procedure documentation
4. **Delegatecall Proxy Pattern**
   - Storage layout consistency
   - Initialization patterns
   - Function shadowing risks
   - Slither upgradeability checks
5. **Function Composition**
   - Function size and clarity
   - Logical grouping
   - Modularity assessment
6. **Inheritance**
   - Hierarchy depth/width
   - Diamond problem risks
   - Inheritance visualization
7. **Events**
   - Critical operation coverage
   - Event naming consistency
   - Indexed parameters
8. **Common Pitfalls**
   - Reentrancy patterns
   - Integer overflow/underflow
   - Access control issues
   - Platform-specific vulnerabilities
9. **Dependencies**
   - Library quality assessment
   - Version management
   - Dependency manager usage
   - Copied code detection
10. **Testing & Verification**
    - Coverage analysis
    - Fuzzing techniques
    - Formal verification
    - CI/CD integration
11. **Platform-Specific Guidance**
    - Solidity version recommendations
    - Compiler warning checks
    - Inline assembly warnings
    - Platform-specific tools

For complete details on each area including what I'll check, analyze, and recommend, see ASSESSMENT_AREAS.md.

Example Output

When the analysis is complete, you'll receive comprehensive guidance covering:

- System documentation with plain English descriptions
- Architectural diagrams and documentation gaps
- Architecture analysis (on-chain/off-chain, upgradeability, proxies)
- Implementation review (functions, inheritance, events, pitfalls)
- Dependencies and testing evaluation
- Prioritized recommendations (CRITICAL, HIGH, MEDIUM, LOW)
- Overall assessment and path to production

For a complete example analysis report, see EXAMPLE_REPORT.md.

Deliverables

I provide four comprehensive deliverable categories:

1. System Documentation

- Plain English descriptions
- Architectural diagrams
- Documentation gaps analysis

2. Architecture Analysis

- On-chain/off-chain assessment
- Upgradeability review
- Proxy pattern security review

3. Implementation Review

- Function composition analysis
- Inheritance assessment
- Events coverage
- Pitfall identification
- Dependencies evaluation
- Testing analysis

4. Prioritized Recommendations

- CRITICAL (address immediately)
- HIGH (address before deployment)
- MEDIUM (address for production quality)
- LOW (nice to have)

For detailed templates and examples of each deliverable, see DELIVERABLES.md.

Assessment Process

When invoked, I will:

1. **Explore the codebase**
   - Identify all contract/module files
   - Find existing documentation
   - Locate test files
   - Check for proxies/upgrades
   - Identify dependencies
2. **Generate documentation**
   - Create plain English system description
   - Generate architectural diagrams (if tools available)
   - Identify documentation gaps
3. **Analyze architecture**
   - Assess on-chain/off-chain distribution (if applicable)
   - Review upgradeability approach (if applicable)
   - Audit proxy patterns (if present)
4. **Review implementation**
   - Analyze functions, inheritance, events
   - Check for common pitfalls
   - Assess dependencies
   - Evaluate testing
5. **Provide recommendations**
   - Present findings with file references
   - Ask clarifying questions about design decisions
   - Suggest prioritized improvements
   - Offer actionable next steps

Rationalizations (Do Not Skip)

Rationalization	Why It's Wrong	Required Action
"System is simple, description covers everything"	Plain English descriptions miss security-critical details	Complete all 5 phases: documentation, architecture, implementation, dependencies, recommendations
"No upgrades detected, skip upgradeability section"	Upgradeability can be implicit (ownable patterns, delegatecall)	Search for proxy patterns, delegatecall, storage collisions before declaring N/A
"Not applicable" without verification	Premature scope reduction misses vulnerabilities	Verify with explicit codebase search before skipping any guideline section
"Architecture is straightforward, no analysis needed"	Obvious architectures have subtle trust boundaries	Analyze on-chain/off-chain distribution, access control flow, external dependencies
"Common pitfalls don't apply to this codebase"	Every codebase has common pitfalls	Systematically check all guideline pitfalls with grep/code search
"Tests exist, testing guideline is satisfied"	Test existence ≠ test quality	Check coverage, property-based tests, integration tests, failure cases
"I can provide generic best practices"	Generic advice isn't actionable	Provide project-specific findings with file:line references
"User knows what to improve from findings"	Findings without prioritization = no action plan	Generate prioritized improvement roadmap with specific next steps

---

Notes

- I'll only analyze relevant sections (won't hallucinate about upgrades if not present)
- I'll adapt to your platform (Solidity, Rust, Cairo, etc.)
- I'll use available tools (Slither, etc.) but work without them if unavailable
- I'll provide file references and line numbers for all findings
- I'll ask questions about design decisions I can't infer from code

---

Ready to Begin

What I'll need:

- Access to your codebase
- Context about your project goals
- Any existing documentation or specifications
- Information about deployment plans

Let's analyze your codebase and improve it using Trail of Bits' best practices!

Revision #4
Created 2026-02-18 08:40:03 UTC by John
Updated 2026-05-31 20:01:40 UTC by John