/code-maturity-assessor

Source: `~/.claude/skills/tob-building-secure-contracts/skills/code-maturity-assessor/SKILL.md`

---

name: code-maturity-assessor
description: Systematic code maturity assessment using Trail of Bits' 9-category framework. Analyzes codebase for arithmetic safety, auditing practices, access controls, complexity, decentralization, documentation, MEV risks, low-level code, and testing. Produces professional scorecard with evidence-based ratings and actionable recommendations.

Code Maturity Assessor

Purpose

Systematically assesses codebase maturity using Trail of Bits' 9-category framework. Provides evidence-based ratings and actionable recommendations.

Framework: Building Secure Contracts - Code Maturity Evaluation v0.1.0

How This Works

Phase 1: Discovery

Explores the codebase to understand:

- Project structure and platform
- Contract/module files
- Test coverage
- Documentation availability

Phase 2: Analysis

For each of 9 categories, I'll:

- **Search the code** for relevant patterns
- **Read key files** to assess implementation
- **Present findings** with file references
- **Ask clarifying questions** about processes I can't see in code
- **Determine rating** based on criteria

Phase 3: Report

Generates:

- Executive summary
- Maturity scorecard (ratings for all 9 categories)
- Detailed analysis with evidence
- Priority-ordered improvement roadmap

Rating System

- **Missing (0)**: Not present/not implemented
- **Weak (1)**: Several significant improvements needed
- **Moderate (2)**: Adequate, can be improved
- **Satisfactory (3)**: Above average, minor improvements
- **Strong (4)**: Exceptional, only small improvements possible

Rating Logic:

- ANY "Weak" criteria → **Weak**
- NO "Weak" + SOME "Moderate" unmet → **Moderate**
- ALL "Moderate" + SOME "Satisfactory" met → **Satisfactory**
- ALL "Satisfactory" + exceptional practices → **Strong**

The 9 Categories

I assess 9 comprehensive categories covering all aspects of code maturity. For detailed criteria, analysis approaches, and rating thresholds, see ASSESSMENT_CRITERIA.md.

Quick Reference:

1. ARITHMETIC

- Overflow protection mechanisms
- Precision handling and rounding
- Formula specifications
- Edge case testing

2. AUDITING

- Event definitions and coverage
- Monitoring infrastructure
- Incident response planning

3. AUTHENTICATION / ACCESS CONTROLS

- Privilege management
- Role separation
- Access control testing

- Key compromise scenarios

4. COMPLEXITY MANAGEMENT

- Function scope and clarity
- Cyclomatic complexity
- Inheritance hierarchies
- Code duplication

5. DECENTRALIZATION

- Centralization risks
- Upgrade control mechanisms
- User opt-out paths
- Timelock/multisig patterns

6. DOCUMENTATION

- Specifications and architecture
- Inline code documentation
- User stories
- Domain glossaries

7. TRANSACTION ORDERING RISKS

- MEV vulnerabilities
- Front-running protections
- Slippage controls
- Oracle security

8. LOW-LEVEL MANIPULATION

- Assembly usage
- Unsafe code sections
- Low-level calls
- Justification and testing

9. TESTING & VERIFICATION

- Test coverage
- Fuzzing and formal verification
- CI/CD integration
- Test quality

For complete assessment criteria including what I'll analyze, what I'll ask you, and detailed rating thresholds (WEAK/MODERATE/SATISFACTORY/STRONG), see ASSESSMENT_CRITERIA.md.

Example Output

When the assessment is complete, you'll receive a comprehensive maturity report including:

- **Executive Summary**: Overall score, top 3 strengths, top 3 gaps, priority recommendations
- **Maturity Scorecard**: Table with all 9 categories rated with scores and notes
- **Detailed Analysis**: Category-by-category breakdown with evidence (file:line references)
- **Improvement Roadmap**: Priority-ordered recommendations (CRITICAL/HIGH/MEDIUM) with effort estimates

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

Assessment Process

When invoked, I will:

1. **Explore codebase**
   - Find contract/module files
   - Identify test files
   - Locate documentation
2. **Analyze each category**
   - Search for relevant code patterns
   - Read key implementations
   - Assess against criteria
   - Collect evidence
3. **Interactive assessment**
   - Present my findings with file references
   - Ask about processes I can't see in code
   - Discuss borderline cases
   - Determine ratings together
4. **Generate report**
   - Executive summary
   - Maturity scorecard table
   - Detailed category analysis with evidence
   - Priority-ordered improvement roadmap

Rationalizations (Do Not Skip)

Rationalization	Why It's Wrong	Required Action
"Found some findings, assessment complete"	Assessment requires evaluating ALL 9 categories	Complete assessment of all 9 categories with evidence for each
"I see events, auditing category looks good"	Events alone don't equal auditing maturity	Check logging comprehensiveness, testing, incident response processes
"Code looks simple, complexity is low"	Visual simplicity masks composition complexity	Analyze cyclomatic complexity, dependency depth, state machine transitions
"Not a DeFi protocol, MEV category doesn't apply"	MEV extends beyond DeFi (governance, NFTs, games)	Verify with transaction ordering analysis before declaring N/A
"No assembly found, low-level category is N/A"	Low-level risks include external calls, delegatecall, inline assembly	Search for all low-level patterns before skipping category
"This is taking too long"	Thorough assessment requires time per category	Complete all 9 categories, ask clarifying questions about off-chain processes
"I can rate this without evidence"	Ratings without file:line references = unsubstantiated claims	Collect concrete code evidence for every category assessment
"User will know what to improve"	Vague guidance = no action	Provide priority-ordered roadmap with specific improvements and effort estimates

---

Report Format

For detailed report structure and templates, see REPORT_FORMAT.md.

Structure:

1. **Executive Summary**
   - Project name and platform
   - Overall maturity (average rating)
   - Top 3 strengths
   - Top 3 critical gaps
   - Priority recommendations
2. **Maturity Scorecard**
   - Table with all 9 categories
   - Ratings and scores
   - Key findings notes
3. **Detailed Analysis**
   - Per-category breakdown
   - Evidence with file:line references
   - Gaps and improvement actions

4. **Improvement Roadmap**
   - CRITICAL (immediate)
   - HIGH (1-2 months)
   - MEDIUM (2-4 months)
   - Effort estimates and impact

Ready to Begin

Estimated Time: 30-40 minutes

I'll need:

- Access to full codebase
- Your knowledge of processes (monitoring, incident response, team practices)
- Context about the project (DeFi, NFT, infrastructure, etc.)

Let's assess this codebase!

Revision #4
Created 2026-02-18 08:40:03 UTC by John
Updated 2026-05-31 20:01:38 UTC by John