# SMART CONTRACT & QA DOCUMENTATION

**Contrato:** Calculator.sol , Structure.sol
**Auditor / QA:** QA Engineer (Blockchain / EVM)
**Tipo de auditoría:** Manual Review + Análisis Funcional QA
**Versión de Solidity:** 0.8.30
**Licencia:** LGPL-3.0-only
**Estado:** Documentación QA - Nivel Básico

**Fecha: 31/01/2026**

---

# 1. Resumen General

Este documento consolida la **documentación técnica y de QA** de dos contratos inteligentes desarrollados en Solidity, con el objetivo de **mostrar estructura, sintaxis, tipos de datos, eventos y comportamientos funcionales básicos** sobre la EVM.

El enfoque del trabajo está orientado a la comprensión progresiva donde llegaremos a utilizar recursos avanzados los cuales aún no se encuentran en línea, siendo este el primer proyecto donde trabajamos sobre:

1. Estructura de contratos
2. Flujo de ejecución
3. Uso de modificadores
4. Manejo de estado
5. Observabilidad mediante eventos
6. Diseño de casos de prueba QA

---

# 1.1 Objetivo de validación

- Correcta estructura de contratos Solidity
- Uso apropiado de tipos de datos (uint, int, bool, string, arrays, mappings)
- Comportamiento funcional esperado
- Emisión correcta de eventos
- Persistencia y atomicidad del estado
- Comprensión de visibilidad (public, internal)

- Documentación QA clara y trazable

---

## 1.2 Conclusión ejecutiva

Los contratos cumplen el objetivo definido, presentan una estructura clara y permite documentar de forma efectiva casos de prueba, flujos de ejecución y conceptos de Solidity.

El proyecto es adecuado como primer ejemplo dentro de una serie progresiva de contratos más complejos

---

# 2. Alcance (Scope)

### 2.1 Incluido

- Licencia SPDX
- Pragma Solidity fijo
- Variables de estado
- Tipos de datos básicos y avanzados
- Modificadores
- Eventos
- Funciones públicas e internas
- Persistencia del estado
- Atomicidad de transacciones
- Casos de prueba QA Manuales

### 2.2 Excluido

- Optimización avanzada de gas
- Control de acceso (roles)
- Pausabilidad
- Upgradeability
- Ataques avanzados (reentrancy, MEV)
- Testing automatizado

Todo lo excluido está fuera del alcance por diseño, no por omisión.

# 3. Metodología

- Revisión manual línea por línea
- Análisis de flujo de ejecución
- Validación de estados previos y posteriores
- Simulación de escenarios:
    1. Escenarios válidos
    2. Escenarios inválidos
    3. Edge cases
- Validación contra buenas prácticas Solidity ≥ 0.8.x
- Enfoque QA Funcional

---

# Contrato 1: Calculator.sol

# 4. Análisis Arquitectónico

## 4.1 Licencia

- SPDX definida y válida
- Cumple estándares
- Estado: Conforme
- Riesgo: Bajo

---

## 4.2 Versión de Solidity

- Versión fija: pragma solidity `0.8.30`
- Protecciones nativas contra overflow / underflow
- Estado: Conforme
- Riesgo: Bajo

---

# 5. Análisis de Variables de Estado

## Result

- Tipo: `uint256`
- Persistente en storage
- Inicialización explícita (10)
- Getter público automático

**Uso:** Demostrar persistencia y mutabilidad del estado.

---

# 6. Modificadores

## `checkNumber(uint256)`

### Función

- Restringe la ejecución a un valor exacto (10)
- Demuestra uso de revert
- Aplica control previo a la ejecucion

**Uso:** Mostrar flujo de control de modifiers.

---

# 7. Eventos

| A | B | C |
|---|---|---|
| **Evento** | **Propósito** | **Estado** |
| ***Addition*** | Logging de suma | **Conforme** |
| ***Subtraction*** | Logging de resta | **Conforme** |
| ***Multiplication*** | Logging de cambios de estado | **Conforme** |

Estado: Conforme

# 8. Análisis de Funciones

**`addition (uint256, uint256)`**

- Cálculo determinista
- No modifica estado
- Emite evento

**`subtraction (uint256, uint256)`**

- Uso de función interna
- Underflow protegido por Solidity 0.8.x

**`subtraction2 (uint256, uint256)`**

- Demuestra uso de enteros con signo

**`multiply (uint256)`**

- Modifica el estado persistente
- Sin validación de rango

**`multiply2 (uint256)`**

- Uso combinado de modifier + estado

# 9. Atomicidad

Todas las funciones:

- Ejecución completa o revert
- Sin estados intermedios
- Cumplen principios EVM
- Persistencia correcta post-transacción

# Contrato 2: Structure.sol

Structure.sol tiene como objetivo **demostrar la estructura completa de un contrato Solidity**, Incluyendo:

- Tipos de datos primitivos
- Arrays
- Mappings
- Estructs
- Modificadores
- Eventos
- Funciones con retorno

# 10. Componentes

**Variables Numericas:**

- **uint256, uint16, uint8, uint32**
- Uso de límites máximos
- Operaciones entre tipos

**Strings:**

- Almacenamiento de texto en storage

**Booleanos:**

- Estados lógicos simples

**Arrays:**

- Array estático con valores iniciales

**Mapping:**

- Asociación clave-valor (addres → uint)

**Struct:**

- Definición de estructura de datos compuesta

# 11. Modificador

**Modifier1:**

- Control previo a ejecución
- Ejemplo de validación lógica

# 12. Evento

**Multiplier:**

- Registro del resultado de una operación

# 13. Función **Multiplier (uint256, uint256)**

- Uso de modifier
- Retorno explícito
- Emisión de evento
- Flujo claro y secuencial

# 14. Estructura del proyecto

**/smart-contract-structure/**

```
│
├── contracts/
│   ├── Calculator.sol
│   └── Structure.sol
│
├── docs/
        └── smart contract_&_qa_documentation_es.pdf
        └── smart contract_&_qa_documentation_in.pdf
        ├── qa_test_cases.pdf
        └── report_es.pdf
        └── report_in.pdf
├── README.md
└── LICENSE
```

# 15. QA Checklist

- SPDX
- Pragma fijo
- Visibilidad explícita
- Uso de pure / internal
- Eventos presentes
- Atomicidad
- Compilación correcta

# 16. Conclusión Final

**Estado: OK - Nivel Base**

**Este repositorio:**

- Es claro
- Es progresivo
- Es comprensible para QA y developers
- Está Documentado