# SonarSolidity Docs

## Architecture – Configuration

a) Build from scratch (use of sonar-custom-plugin https://github.com/SonarSource/sonar-custom-plugin-example)

Note: Class that implements org.sonar.api.Plugin must declare implementation of extensions provided by a plugin.

b) in place CI (Sonar Cloud) & .travis.yml

Use of Antlr4 to build the Parser and the Lexer.

### Notes on antlr4:

1. Use the Solidity.g4 file to generate the grammar.
2. In details: antlr4 needs a .sol file with all the possible cases produced by the grammar and the run-tests.sh script provided by the repo: https://github.com/solidityj/solidity-antlr4
3. Rules in ANTLR4 stand for Parser Rules.
4. Under SolidityParser: ANTLR4 has declared all the nodes that exist in the Parse Tree (concrete syntaxt tree).
5. SourceUnitContext is the root of the parse tree(Constituency-based parse trees).
6. From any node to access its ancestors, you can use the methods of the corresponding …Context class.
7. More detailed examples lie in SolidityParsingPhaseTest file.
8. Nodes are typed.
9. For parser do not use TokenStream to initialize SolidityParser, use CommonTokenStream!
10. ANTLR4 stores comments single-line and structured to channel HIDDEN by default.
11. ParserRuleContext class provides the function getToken(index, position) which can be very helpful (e.g. for else case of if, etc).

Code example to search for a **specific node type** class on ANTLR4 (here **StatementContext**):

```
Suppose List<FunctionDefinitionContext> funList = … ;
        funList.stream()
        .map(FunctionDefinitionContext::block)
        .filter(Objects::nonNull)
        .forEach(x -> {
        for (ParseTree travers : x.children) {
         if (StatementContext.class.isInstance(travers))
         // do sth
        }
```

### Syntax Highlighting

In SolidityKeywords.java I have defined 2 arrays of Strings.
String[] KEYWORDS includes all the reserved keywords of solidity, even some that are deprecated but still used, like 'throw'.
String[] KEYWORD_TYPES includes all the primitive types used by solidity.
Tokens of KEYWORDS array are highlighted as KEYWORD.
Tokens of KEYWORD_TYPES array are highlighted as KEYWORD_LIGHT.
String literals are highlighted as STRING.
The rest of the literals (Version, Boolean, Decimal Number, Hex Number, Number Unit, Hex) are highlighted as CONSTANT.
SyntaxHighlightingVisitor is responsible to highlight comments correctly.

### Basic Metrics

MetricsVisitor class computes the metrics for lines of code, lines of comments, number of statements, number of functions, number of Contracts and Cpd.

### Ruling Mechanism

relies upon RuleContext interface, which is implemented by the private class TestRuleContext, responsible for reporting for unit tests and by the SolidityRuleContext, responsible for reporting on SonarQube.

## Some important details about Solidity/ Ethereum (Smart Contracts):

A contract has functionality (functions) and state (data).

Magic Global Variables: msg, tx, block

Anyone in the world has access to a smart contract (set/ get) the state.
But there are access restrictions that can be configured.
-address type is a 160 bit value, storing external contracts, keypairs.
-public generates a function that can be accessed outside of the contract.
-mapping(address => uint) public balances; complex datatype maps addresses to uint, like hashtables. *** use more advanced datatype

-event Sent(address from, address to, uint amount); blockchain nodes listen for those events.
e.g. to listen to an event: call the constructor of the corresponding contract and then call Sent(): Coin.Sent().
-Gas, upon creation each txn is charged with **gas.** Its purpose is to limit the amount of work required to execute the txn and pay for this execution (gas_price * gas ).
**Storage, key-value** store, each account has persistent memory area.
**Memory**, cleared instance for each message call. ** Memory is more COSTLY, the larger it grows.
EVM is a stack machine, Access to the stack is limited to the top 16 elements.
Logs feature is used in Solidity to implement maps.
Selfdestruct, code of a contract is removed from the blockchain.
Function Types: internal(by default) ones can be called inside the current contract; external functions can be passed via and returned from external function calls.

## Solidity Rules implemented:

## Common Rules(keep track even for those that don't apply to Solidity)

1. Empty File DONE
2. MultiLine Comments not empty
3. Empty FUNCTIONS should be removed DONE
4. Contracts should not be empty DONE
5. String literals should not be concatenated NOT FOR SOLIDITY
6. Boolean checks should not inverted DONE
7. Constants should come first in equality tests NOT FOR SOLIDITY
8. Variables not shadowed
9. An open curly brace should be at the end of the line DONE
10. Cognitive Complexity Rule DONE
11. Unused local variable
12. Unused function parameters
13. Parameter names should not duplicate the names of their methods DONE
14. Variables should be defined in the blocks where they are used

## Solidity Specific Rules

1. Pragma (Solidity) version should be the latest
2. Use string only for dynamically allocated data (NOT YET)
3. Function state mutability can be restricted to pure, if the function does not read/modify the state
4. Specify constructor's visibility either public or internal
5. Defining constructors as functions with the same name as the contract
6. is deprecated use "constructor(...){...}"
7. Pragma does not exist
8. Deprecated suicide function
9. Avoid tx.Origin (it will be deprecated, the contract will not work well with multisig wallets, it can lead to security issues) https://github.com/ethereum/solidity/issues/683
10. For security reasons avoid sha3 in favor of keccak256

11. Access Restriction Rules: use modifiers pattern: In the beginning the required condition is checked. Afterwards the execution jumps back to the initial function. This behavior is indicated by an underscore (_;)

12. Guard Check Rule: The desired behavior of a smart contract would be to check for all required circumstances and only proceed if everything is as intended. This rule ensures that all parameters of public/external functions that are not pure or view are validated with require()
13. String Comparison Rule: for string comparison it is suggested to use keccak256 to check if the hashes of the strings are equal.
14. Gain access to data stored outside of the blockchain through an Oracle. The contract requesting info from an outside source, assembles a query and sends it in a txn to the oracle. The contract needs to have a callback function, so that oracle can deliver the result of the query. Include a check in the callback function so that only the specified oracle could call it
15. Transfer Ether Securely: since there is no semantics information yet. I raise an issue each time a send() function is called
16. Randomnness: generating a random number, taking into account block, could introduce serious security vulnerabilities
17. Lower Gas Bytes Rule: developers should use bytes instead of byte[], it's cheaper in terms of gas consumption
18. Checks Effects Rule: When handing over control flow to an external entity is is important to guard your public functions.
    Specifically, issue is raised when external interactions; transfer, or send is not the last step of a public function
19. Tight Variable Packing: Optimize gas consumption when storing or loading statically-sized variables
20. "var" keyword is deprecated, two cases variable declaration and tuple assignment

Creating a new Rule:
1. Add Check class under: solidity-checks e.g. AvoidTxOriginCheck.java
2. Add Test Class
3. add test.sol file that creates Noncompliant and Compliant examples of the rule implemented
4. Add the RULE_KEY of the Check class in the List of RuleKeyList class
5. Add the Check class in the list in CheckList class
6. Add the files needed to define the rule under the folder structure: solidity-plugin/src/main/resources/org/sonar/l10n/solidity/rules/solidity
7. Increment the test check in SolidityRulesDefinitionTest(fail-safe mechanism)

**Release** a new version of SonarSolidity:
1. Create new release in github with the correct version and check that the correct version exists in the MANIFEST.MF of the jar file created.
2. Plan next release (change version in pom file)

## Importing External Linters:

Import Solium linter (https://github.com/duaraghav8/Solium)

Use Solium release version 1.0.0

The reports from Solium have the underlined form:

```
file6.sol
  6:34        warning      Avoid using 'now' (alias to 'block.timestamp').
security/no-block-members
  9:8  warning      Provide an error message for require().
error-reason
  14:8        warning      Provide an error message for require().
error-reason

✖ 3 warnings found.
```

To create a report from Solium you have to install Solium:

```
npm install -g solium
```

Then in the root directory of the Dapp execute:

```
solium --init
solium -f file.sol > report.out

or in case of folder: solium -d contracts/ > report.out
```

After the report is created successfully, to import it to SonarQube go to Administration-> General Settings, Solidity and set the property sonar.solidity.solium.reportPaths to report.out.

At last, you have to add the property "sonar.solidity.solium.reportPaths" to the SonarQube Scanner configuration (sonar-project.properties) or

```
sonar-scanner -Dsonar.projectKey=SolidityFile6 -Dsonar.sources=.
-Dsonar.java.binaries=. -Dsonar.solidity.solium.reportPaths=report.out -X
```

If I have more than one report files to import then:

```
-Dsonar.solidity.solium.reportPaths="report1.out","report2.out"
```

Now let’s deep into the code

AbstractExternalReportSensor implements org.sonar.api.batch.sensor.Sensor and is the abstract class responsible for importing reports from Solium external linter.

SoliumReportSensor is the class that extends AbstractExternalReportSensor and is responsible for reporting the issues from Solium linter(version 1.0.0).

There is one java.util.Pattern named SOLIUM_FILE_REGEX, to match the filename and another Pattern, SOLIUM_LINE_REGEX that recognizes issues from Solium.
Also, the Pattern SOLIUM_SPECIAL_CHARACTERS is used in order to remove some special characters that were produced from Solium and make no sense for a txt file.

## Adding repositories for ITs:

In the repository sonar-solidity, under the module solidity-its there is the submodule solidity-test-sources. This repo is used for IT testing.
To add a new repository in solidity-test-sources:

1. git config --local core.autocrlf false
2. git clone --depth 1 git@github.com:repo/repo-api.git repo-api
3. rm -rf repo-api/.git
4. Commit changes in the submodule and push
5. Commit in the parent repo (include the commit in the submodule)
6. git push --recurse-submodules=on-demand

## Other external linters

1. Solium(imported)
2. Solhint (https://protofire.github.io/solhint/)
3. Manticore (https://github.com/trailofbits/manticore)
4. Mythril (https://sonarcloud.io/dashboard?id=mythril, https://github.com/ConsenSys/mythril)
5. Oyente https://github.com/melonproject/oyente
6. Solidity-Coverage https://github.com/sc-forks/solidity-coverage
7. SolGraph https://github.com/raineorshine/solgraph

## Reading Sources/ Interesting github repositories

- https://github.com/trailofbits/not-so-smart-contracts
- https://eprint.iacr.org/2016/633.pdf
- https://arxiv.org/pdf/1806.01143.pdf
- https://fravoll.github.io/solidity-patterns/
- https://consensys.github.io/smart-contract-best-practices/recommendations/
- https://www.comp.nus.edu.sg/%7Eloiluu/papers/oyente.pdf
- Tutorial https://github.com/androlo/solidity-workshop