BNUMMET

VERSION 1

Code analysis

By: default

2023-03-26

CONTENT

INTRODUCTION

This document contains results of the code analysis of BNumMet.

CONFIGURATION

- Quality Profiles
  - Names: Sonar way [Python]; Sonar way [XML];
  - Files: AYZE7i3U2dITZFwgEli2.json; AYZE7jaV2dITZFwgEmPx.json;
- Quality Gate
  - Name: Sonar way
  - File: Sonar way.xml

SYNTHESIS

ANALYSIS STATUS

Reliability	Security	Security Review	Maintainability
A	A	A	A

QUALITY GATE STATUS

Quality Gate Status	Passed

Metric	Value
Reliability Rating on New Code	OK
Security Rating on New Code	OK
Maintainability Rating on New Code	OK
Coverage on New Code	OK
Duplicated Lines (%) on New Code	OK

METRICS

Coverage	Duplication	Comment density	Median number of lines of code per file	Adherence to coding standard
98.2 %	0.0 %	43.4 %	206.0	99.7 %

TESTS

Total	Success Rate	Skipped	Errors	Failures

BNumMet

95	**100.0 %**	**0**	**0**	**0**

DETAILED TECHNICAL DEBT

Reliability	Security	Maintainability	Total
-	-	0d 4h 5min	**0d 4h 5min**

METRICS RANGE

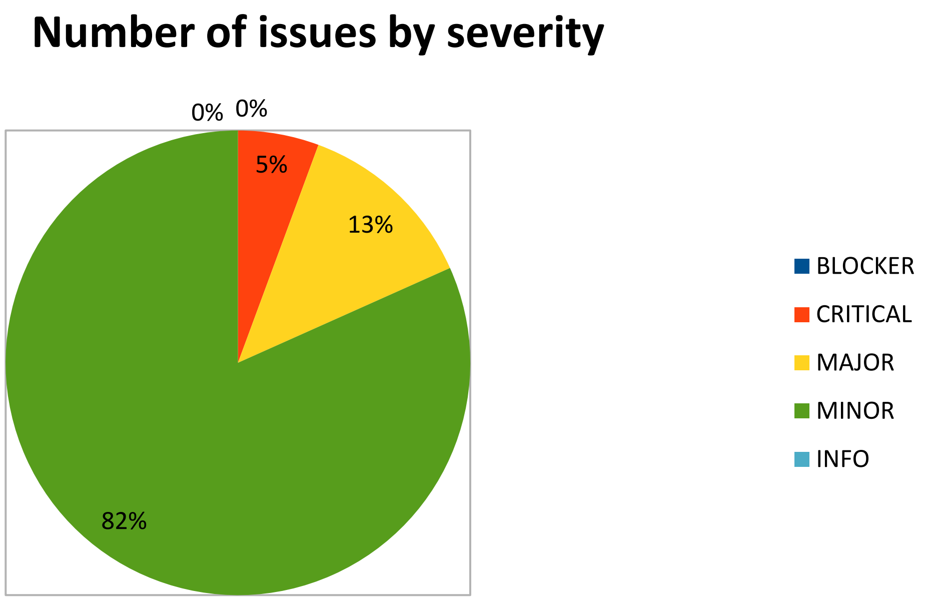
	Cyclomatic Complexity	Cognitive Complexity	Lines of code per file	Comment density (%)	Coverage	Duplication (%)
Min	0.0	0.0	0.0	22.2	95.5	0.0
Max	295.0	269.0	2028.0	74.1	100.0	0.0

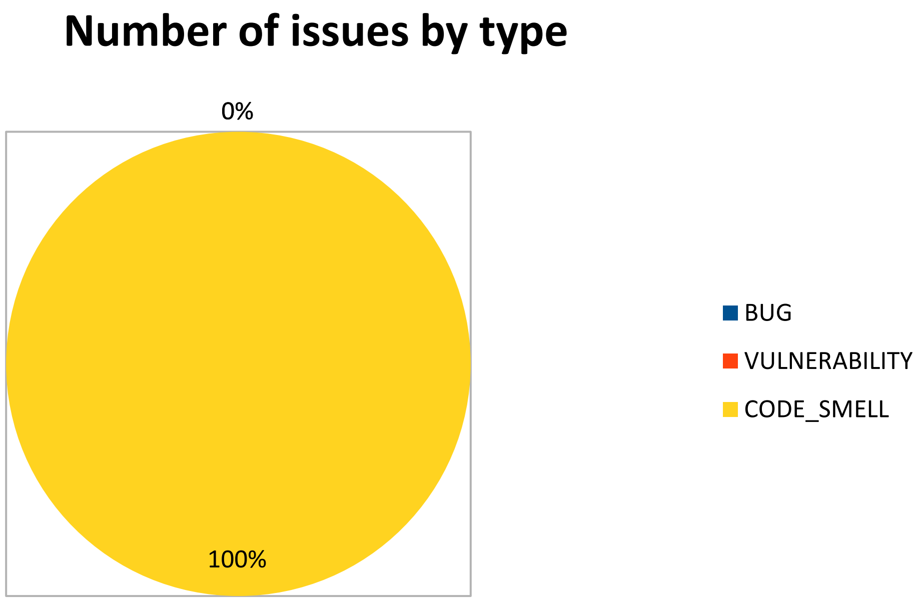
VOLUME

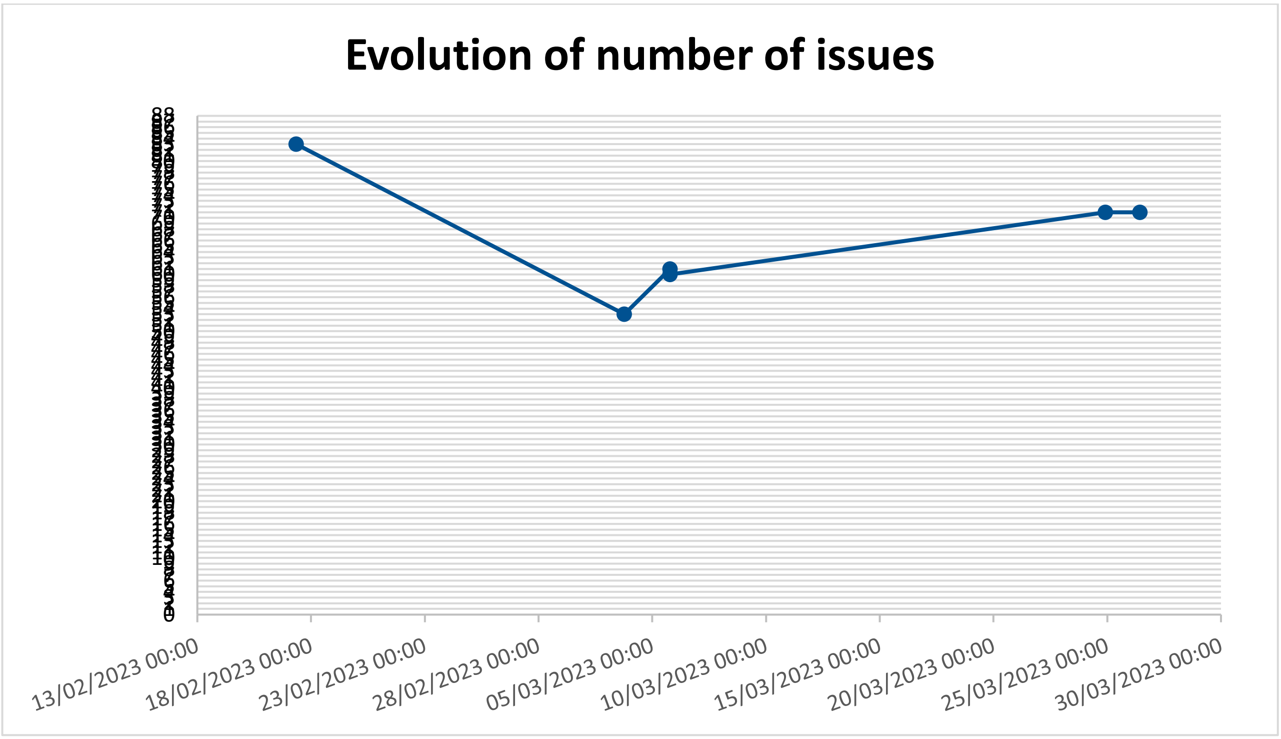
Language	Number
Python	2028
Total	2028

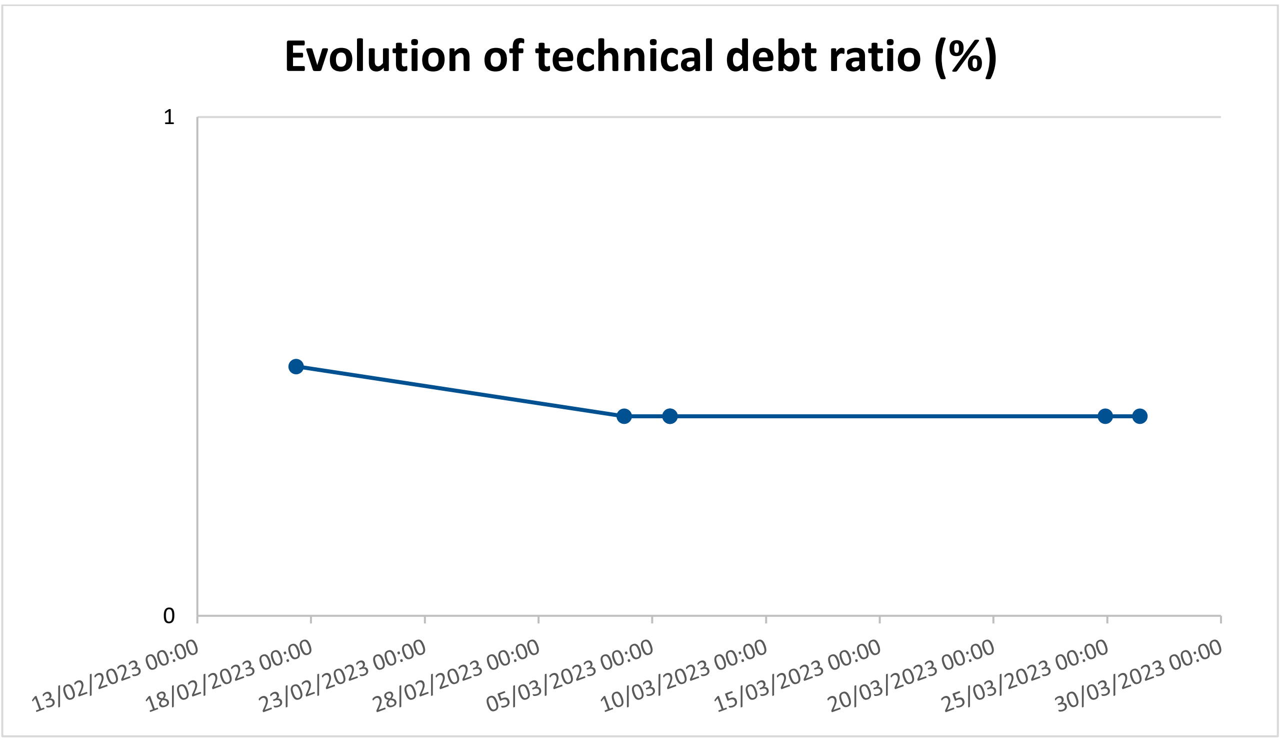
ISSUES

CHARTS






Evolution of number of issues
0
13/02/2023 00:00
18/02/2023 00:00
23/02/2023 00:00
28/02/2023 00:00
05/03/2023 00:00
10/03/2023 00:00
15/03/2023 00:00
20/03/2023 00:00
25/03/2023 00:00
30/03/2023 00:00


Evolution of technical debt ratio (%)
1
0
13/02/2023 00:00
18/02/2023 00:00
23/02/2023 00:00
28/02/2023 00:00
05/03/2023 00:00
10/03/2023 00:00
15/03/2023 00:00
20/03/2023 00:00
25/03/2023 00:00
30/03/2023 00:00

ISSUES COUNT BY SEVERITY AND TYPE

Type / Severity	INFO	MINOR	MAJOR	CRITICAL	BLOCKER
BUG	0	0	0	0	0
VULNERABILITY	0	0	0	0	0
CODE_SMELL	0	58	9	4	0

ISSUES LIST

Name	Description	Type	Severity	Number
String literals should not be duplicated	Duplicated string literals make the process of refactoring error-prone, since you must be sure to update all occurrences. On the other hand, constants can be referenced from many places, but only need to be updated in a single place. Noncompliant Code Example With the default threshold of 3: def run(): prepare("this is a duplicate") # Noncompliant - "this is a duplicate" is duplicated 3 times execute("this is a duplicate") release("this is a duplicate") Compliant Solution ACTION_1 = "action1" def run(): prepare(ACTION_1) execute(ACTION_1) release(ACTION_1) Exceptions No issue will be raised on: duplicated string in decorators strings with less than 5 characters strings with only letters, numbers and underscores @app.route("/api/users/", methods=['GET', 'POST', 'PUT']) def users(): pass @app.route("/api/projects/", methods=['GET', 'POST', 'PUT']) # Compliant def projects(): pass	CODE_SMELL	CRITICAL	1
Cognitive Complexity of functions should not be too high	Cognitive Complexity is a measure of how hard the control flow of a function is to understand. Functions with high Cognitive Complexity will be difficult to maintain. See Cognitive Complexity	CODE_SMELL	CRITICAL	3
Sections of code should not be commented out	Programmers should not comment out code as it bloats programs and reduces readability. Unused code should be deleted and can be retrieved from source control history if required.	CODE_SMELL	MAJOR	7
Function names should comply with a naming convention	Shared coding conventions allow teams to collaborate efficiently. This rule checks that all function names match a provided regular expression. Noncompliant Code Example With the default provided regular expression:	CODE_SMELL	MAJOR	2

	^[a-z_][a-z0-9_]*$ def MyFunction(a,b): ... Compliant Solution def my_function(a,b): ...			
Method names should comply with a naming convention	Sharing some naming conventions is a key point to make it possible for a team to efficiently collaborate. This rule allows to check that all method names match a provided regular expression. Noncompliant Code Example With default provided regular expression: ^[a-z_][a-z0-9_]*$ class MyClass: def MyMethod(a,b): ... Compliant Solution class MyClass: def my_method(a,b): ...	CODE_SMELL	MINOR	6
Field names should comply with a naming convention	Sharing some naming conventions is a key point to make it possible for a team to efficiently collaborate. This rule allows to check that field names match a provided regular expression. Noncompliant Code Example With the default regular expression ^[_a-z][_a-z0-9]*$: class MyClass: myField = 1 Compliant Solution class MyClass: my_field = 1	CODE_SMELL	MINOR	28
Local variable and function parameter names should comply with a naming convention	Shared naming conventions allow teams to collaborate effectively. This rule raises an issue when a local variable or function parameter name does not match the provided regular expression. Exceptions Loop counters are ignored by this rule. for i in range(limit): # Compliant print(i)	CODE_SMELL	MINOR	24

SECURITY HOTSPOTS

SECURITY HOTSPOTS COUNT BY CATEGORY AND PRIORITY

Category / Priority	LOW	MEDIUM	HIGH
LDAP Injection	0	0	0
Object Injection	0	0	0
Server-Side Request Forgery (SSRF)	0	0	0
XML External Entity (XXE)	0	0	0
Insecure Configuration	0	0	0
XPath Injection	0	0	0
Authentication	0	0	0
Weak Cryptography	0	0	0
Denial of Service (DoS)	0	0	0
Log Injection	0	0	0
Cross-Site Request Forgery (CSRF)	0	0	0
Open Redirect	0	0	0
Permission	0	0	0
SQL Injection	0	0	0
Encryption of Sensitive Data	0	0	0
Traceability	0	0	0
Buffer Overflow	0	0	0
File Manipulation	0	0	0
Code Injection (RCE)	0	0	0

BNumMet

Cross-Site Scripting (XSS)	0	0	0
Command Injection	0	0	0
Path Traversal Injection	0	0	0
HTTP Response Splitting	0	0	0
Others	0	0	0

SECURITY HOTSPOTS LIST