**Prism Viewer Application - Methodology and Screening Report**

**1. Introduction**

This report outlines the approach used to implement the Prism Viewer Application for the screening task. The application allows users to view 3D models of rectangular prisms and compute their surface area and volume, alongside providing a 3D viewer for model visualization. The application incorporates both Conda and PIP as package managers and includes unit tests to ensure accurate calculations.

**2. Methodology**

**2.1 Application Structure**

The Prism Viewer Application is developed in Python and uses the following key tools:

* **PyQt5**: For creating the user interface.
* **pythonocc-core**: For 3D visualization of rectangular prisms.
* **SQLite**: For storing prism dimension data.
* **NumPy**: For efficient data handling and calculations.

When a prism is selected from a dropdown menu, the application retrieves its dimensions from an SQLite database. Calculations for surface area and volume are performed using the PrismCalculator class, while the 3D model is rendered through OpenCascade’s 3D rendering engine.

**3. Unit Tests and Test Strategy**

Unit tests were created to confirm the accuracy of the surface area and volume calculations for the rectangular prism.

**3.1 Selected Tests**

The unit tests are defined in the test\_prism\_calculator.py file and cover the following areas:

* **Surface Area Calculation**: Verifies that the surface area is correctly computed using the formula:

Surface Area=2×(Length×Width+Width×Height+Length×Height)\text{Surface Area} = 2 \times (\text{Length} \times \text{Width} + \text{Width} \times \text{Height} + \text{Length} \times \text{Height})Surface Area=2×(Length×Width+Width×Height+Length×Height)

* **Volume Calculation**: Verifies that the volume is accurately calculated with the formula:

Volume=Length×Width×Height\text{Volume} = \text{Length} \times \text{Width} \times \text{Height}Volume=Length×Width×Height

Each test uses sample prism dimensions to validate that the output matches known correct values, providing self-verification by comparing results against these expected values.

**3.2 Code Example**

Below is an example of the unit test implementation:

python

import unittest

from prism\_calculator import PrismCalculator

class TestPrismCalculator(unittest.TestCase):

def test\_surface\_area(self):

length, width, height = 3.0, 4.0, 5.0

expected\_surface\_area = 94.0

result = PrismCalculator.surface\_area(length, width, height)

self.assertEqual(result, expected\_surface\_area)

def test\_volume(self):

length, width, height = 3.0, 4.0, 5.0

expected\_volume = 60.0

result = PrismCalculator.volume(length, width, height)

self.assertEqual(result, expected\_volume)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

These tests ensure the core functionalities (surface area and volume calculations) are reliable and produce correct outputs.

**4. Conda Packaging Challenges**

**4.1 Packaging with Conda**

The application was packaged using Conda with the following steps:

1. **Creating a Conda Recipe**: A meta.yaml file was created to specify dependencies, including Python, PyQt5, NumPy, and pythonocc-core.
2. **Building the Package**: Using conda-build, a .tar.bz2 package file was generated.
3. **Testing**: The package was tested by installing it in a clean environment to ensure proper functionality.

**4.2 Challenges Encountered**

Several challenges arose during packaging:

* **Dependency Management**: Managing dependencies, particularly for pythonocc-core, required adjustments to the Conda Forge channels to prevent the application from retrieving incorrect package versions.
* **Path Issues with Executables**: Using the default method to create .exe files led to hard-coded paths, causing runtime errors in different environments. This was resolved by using Python’s gui\_scripts entry point, eliminating the need for .exe files and enabling direct launching of Python scripts.

**5. Conclusion**

The Prism Viewer Application successfully meets the design requirements, providing a graphical interface for both computation and visualization of rectangular prisms. The unit tests confirm the accuracy of surface area and volume calculations, and the application is distributed via Conda and PIP for easy access.

**6. References**

* [PyQt5 Documentation](https://www.riverbankcomputing.com/static/Docs/PyQt5/)
* [pythonocc-core Documentation](http://www.pythonocc.org/)
* [Conda Documentation](https://docs.conda.io/projects/conda/en/latest/)