VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
INSTITUTE OF COMPUTER SCIENCE
DEPARTMENT OF COMPUTATIONAL AND DATA MODELING

Software Engineering | Project Technical specification

Solar panel project

Area 1

Done by:

Arnas Vidžiūnas
Mindaugas Neseckas
Aleksandras Kuznecovas
Valentinas Straigis
Tomas Jefimovas

Supervisors:

Lekt. Virgilijus Krinickij
Lekt. Gediminas Rimša

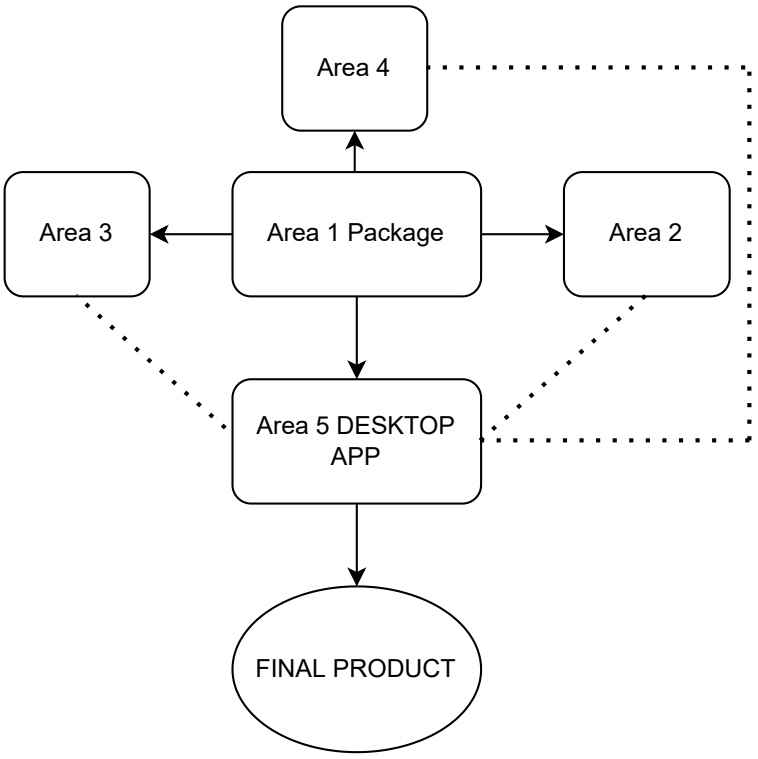
Vilnius
2022

Contents

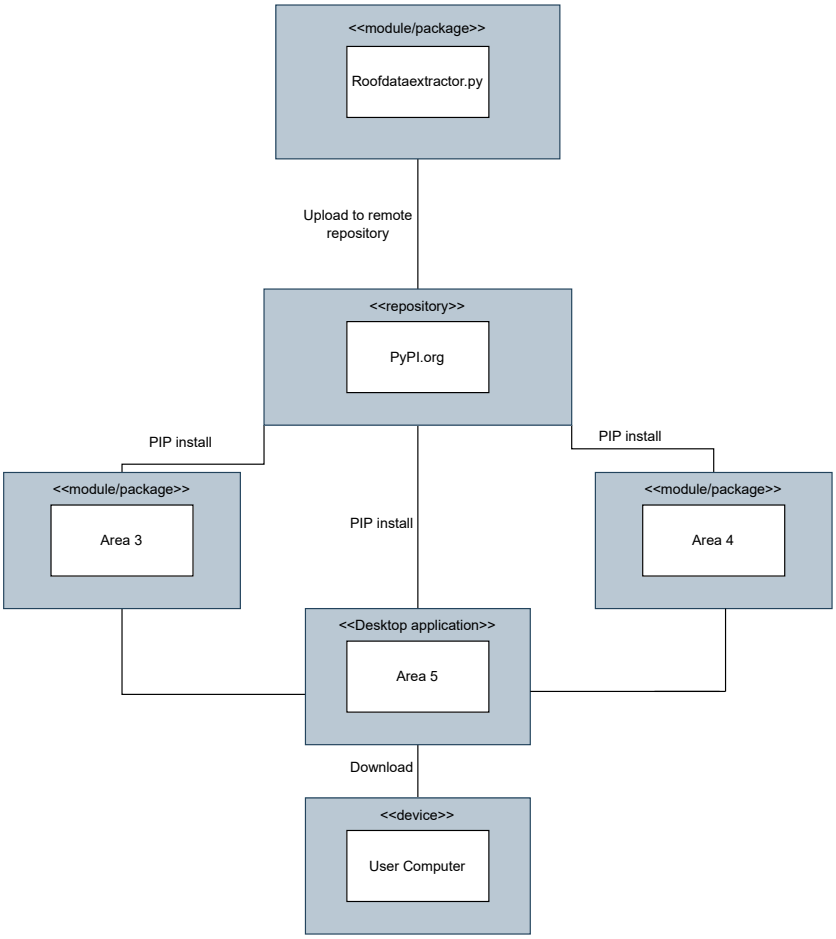
Overview

This is a technical specification for a package which will read any text file which has an XML format inside. The file will be provided by the user in an XML formatted file that would be processed and output a JSON file with filtered data. The provided file will contain roof's or collection of multiple roof panels measurements, coordinates, and other details about the roof. The package will include file and data validation in order to protect the final program from malicious scripts and other dangerous content. Our area's goal is to extract required data which is specified by other teams.

Our area's package highly contributes to other areas' work. We extract and filter certain data from the given XML formatted file and output it to the JSON file in order to make other areas' work less complicated. Our package is used in every other area of the project.



Context Diagram



Deployment Diagram

Technologies and Tools

The following technologies will be used to build and test the project:

Programming language - Python (Recommended version: 3.10, Minimum version: 3.6)
Unit testing - UnitTest
IDE - VS Code
Version Control tool - Git and GitHub
Project management and issue tracking - Jira
Documentation writing tool - Overleaf LaTeX

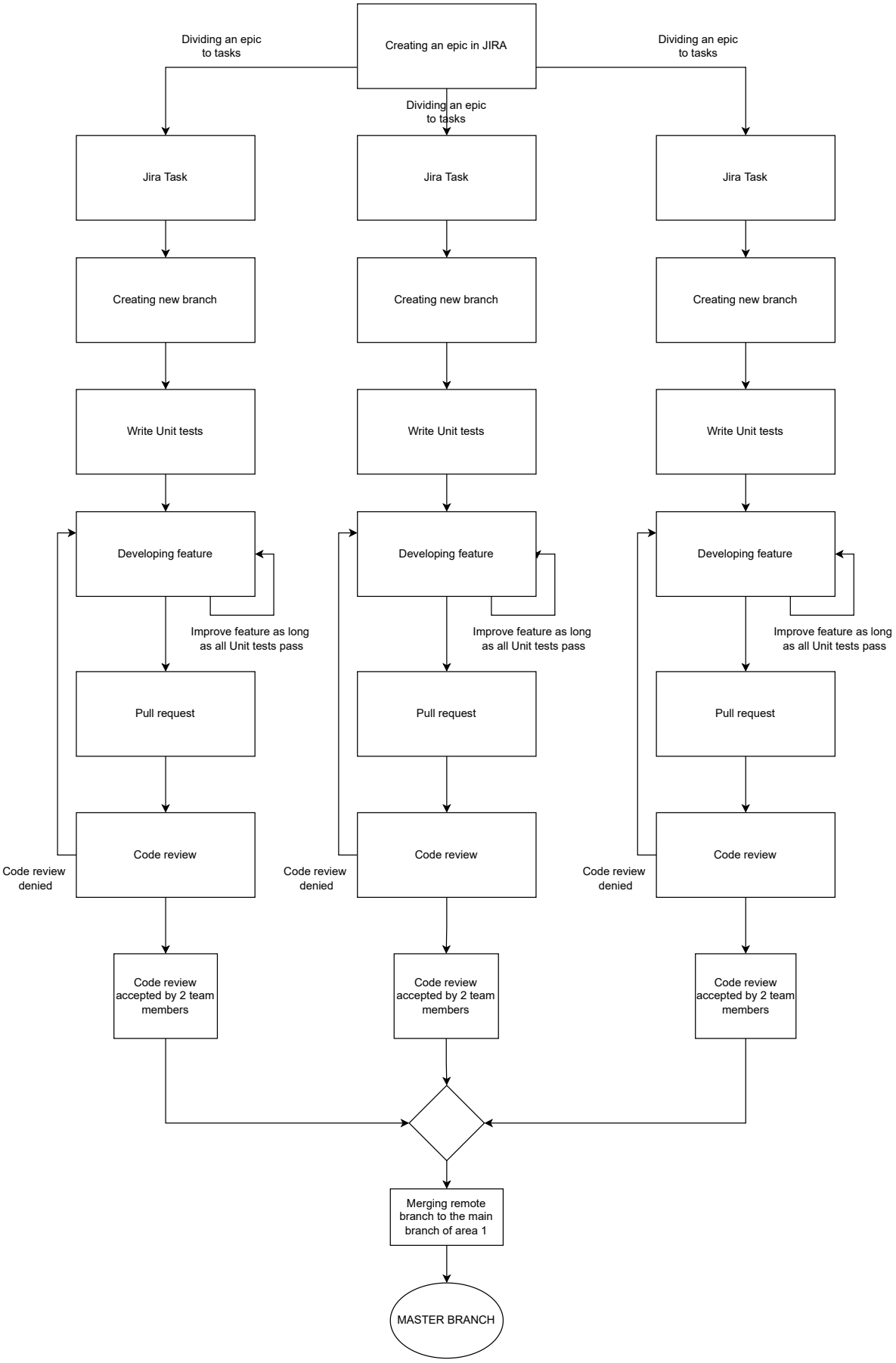
PIP (package installer for Python) libraries used in the project:

- **XMLJSON** - converts XML into Python dictionary structures (trees, like in JSON) and vice-versa.
- **DefusedXML** - workarounds and fixes for denial of service and other vulnerabilities in Python's XML libraries.
- **UnitTest** - builtin for unit tests.

Workflow

- Creating an epic in Jira. The epic is divided into smaller tasks, which are separated between each team member.
- Creating a new branch for that Jira task.
- If needed, writing the unit test, which firstly fails. We agreed to use test driven deployment.
- Develop an improvement for the package (develop a feature). Every change has to pass the unit tests.
- Opening a pull request for the development branch (main of area 1) if everything is done and the code passes all the tests.
- Code review of the pull request. The code review has to be done by at least 2 people.
- If the feature passes two code reviews, merging this branch to the main branch of area 1.

  comment: the graph provided below is not full, since we wanted to have it nicely fit in the page. There should be 2 more identical columns on the sides.



Workflow Diagram

Testing

The code will be tested using two methods:

Manually checking the given XML file and comparing it to our JSON file:
We are comparing the given XML file with our output JSON file manually because the automated test could fail in this situation. We delete unnecessary XML elements and output only the main elements for the roof measurements. Deleting this elements could fail the automated tests.

Using Unit tests:
The main help we get from using unit tests is that we can instantly see if the given XML file is not broken, before doing the manual testing. This helps us to reduce possibility of mistakes in manual testing and in our final code.

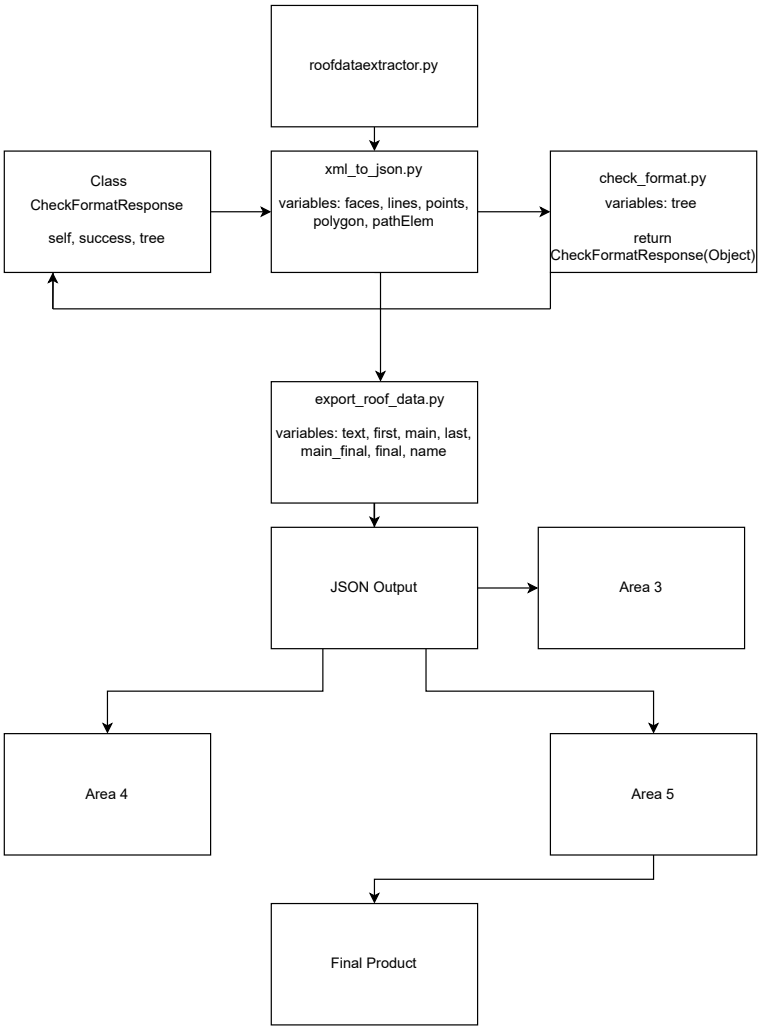
Written Unit Tests by code version:

- **Version 0.1** - Tests are throwing a lot of different file extensions to see if the package reads only XML files.
- **Version 0.2** - Tests are throwing a lot of different file extensions with XML format inside to see if the package reads only XML formatted files.
- **Version 0.3** - Tests checks if the provided XML file has roof measurements inside.
- **Version 1.0** - Tests help us to compare XML file content and JSON file output.

Structural Aspects

The main components of the module are the main - roofdataextractor.py file, xml-to-json.py, check-format.py and export-roof-data.py. The ”roofdataextractor” is the main file, which calls all the functions. Secondly, check-format checks if the XML file is well-formatted and if there are not any malicious scripts. Moreover, we have xml-to-json, which converts everything from the given XML file to JSON format. Finally, the last file/class is export-roof-data, which deletes unnecessary components of the roof measurements. The output is JSON file with only the useful measurements for the other project's areas to continue the work.

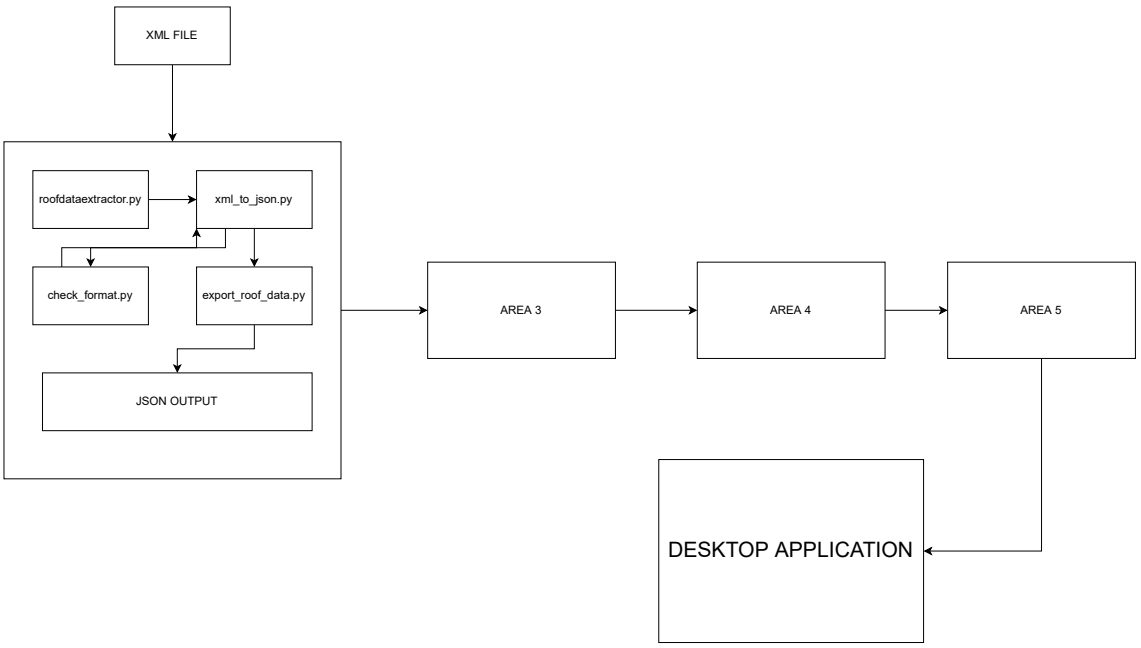
The ”Roof-Data-Extractor” will run with Python (3.6 or above). The module will be uploaded to the ”pypi.org” page, so other teams could easily adapt our code.



Structural-Classes Diagram

Dynamic Aspects

- The module receives one file as an input.
- Firstly, the module checks if the file is not malicious.
- If the given file is safe, the module calls XmlToJson, which instantly calls checkFormat function.
- If the given file is not formatted as an XML file, the module prints: “The file is not well formatted ”.
- If the given file is formatted as an XML file, the module continues reading that file.
- If the given file is formatted as an XML file and after the reading it founds, that the XML file is written with no roof measurements it prompts an error to change the input file.
- If the file is well formatted, with no malicious scripts, the module continues its reading and calling export-roof-data.py.
- The module outputs a JSON file as a result.



Dynamic Diagram (how our code work to achieve larger goal)

Error Handling

- In case of incorrect data, the module will prompt an error. The user will be asked to fix it, rewrite or simply change the file.

- The module will cycle through the content of the provided file to check if it is written in an XML format. For example, the provided file could have any extension like *.txt, *.pptx, *.docx, but the content is written in an XML format, so the provided file will be accepted.
- The module will terminate itself and prompt errors if the malicious scripts are detected in the provided file.
- If the provided file is empty, the module will prompt an error and ask the user to fix the file.
- If the module can not handle an unexpected operation it will prompt an unexpected error.