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

Software Engineering | Project Technical Specification

Solar design tool

Area 2

Done by:

Martin Martijan
Gerda Zykutė
Norvydas Martinka
Gustas Vasilevič

Supervisors:

Lekt. Virgilijus Krinickij
Lekt. Gediminas Rimša

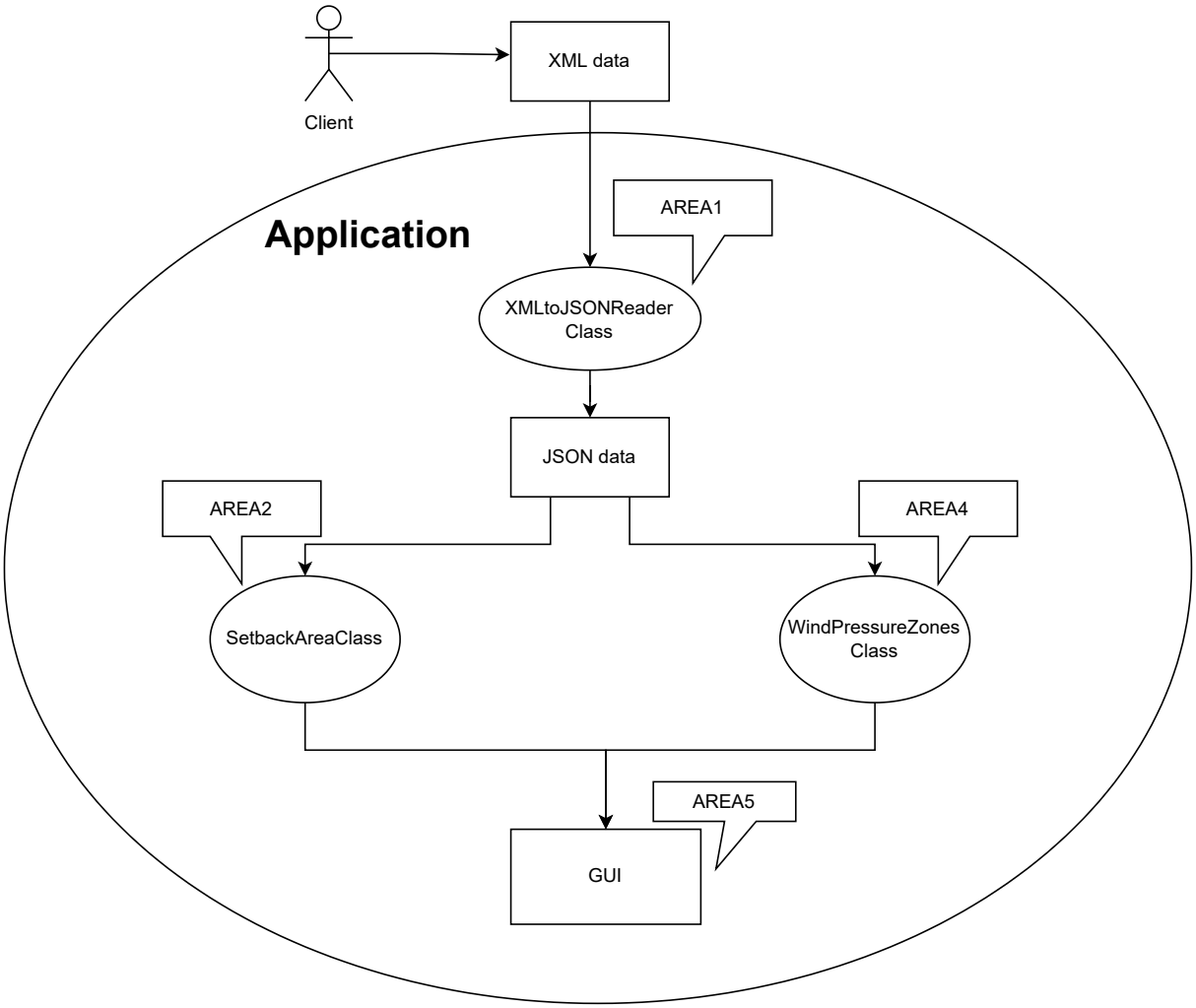
Vilnius
2022

Contents

1. Definitions And Acronyms

$JSON_2$ - JavaScript Object Notation
IDE_3 - Integrated development environment
DOD_4 - Definition of done
TDD_5 - Test driven development

2. Overview



1 pic. area 2 part in global project, context diagram

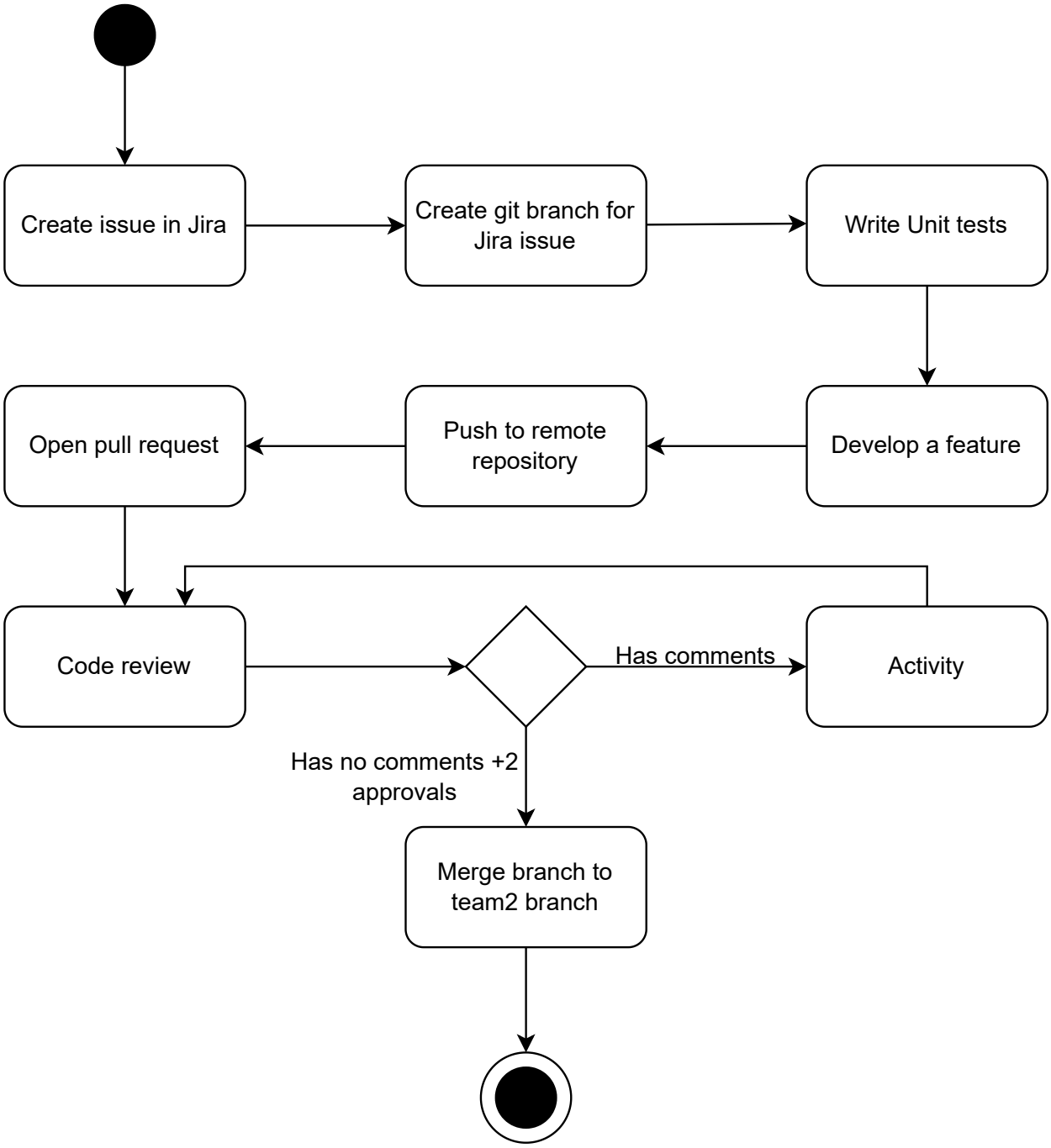
This is a technical specification for an application which would take JSON input, representing roof measurements and coordinates and calculate the best areas for dire ventilation setbacks and pathways on the building, returning the calculated in data array format. The output must comply with the requirements of the fire code and should maximise the area of solar panel placement.

3. Technologies And Tools

We chose Java because it is platform independent tool, and we work on different platforms. It is also OOP, which adds to the convenience of the creation of software, and we are well familiar with its features
For IDE we chose IntelliJ IDEA community edition because it is open source, supports VCS terminal, has easy code readability and testing frameworks also supports JSON schema.

For version control tools we chose Git and GitHub it is open source, has branching capabilities, local repositories. Easier to get great documentation, and easy to do code reviews.
For unit testing we chose Junit as it is open source, it provides assertions for testing expected results, annotations to identify test methods, and test runners for running tests. JUnit tests can be run automatically and they check their own results and provide immediate feedback
For planning and issue testing we chose, Jira. It is workflow-powered, it ensures top traceability, and it promotes a great integration of processes.

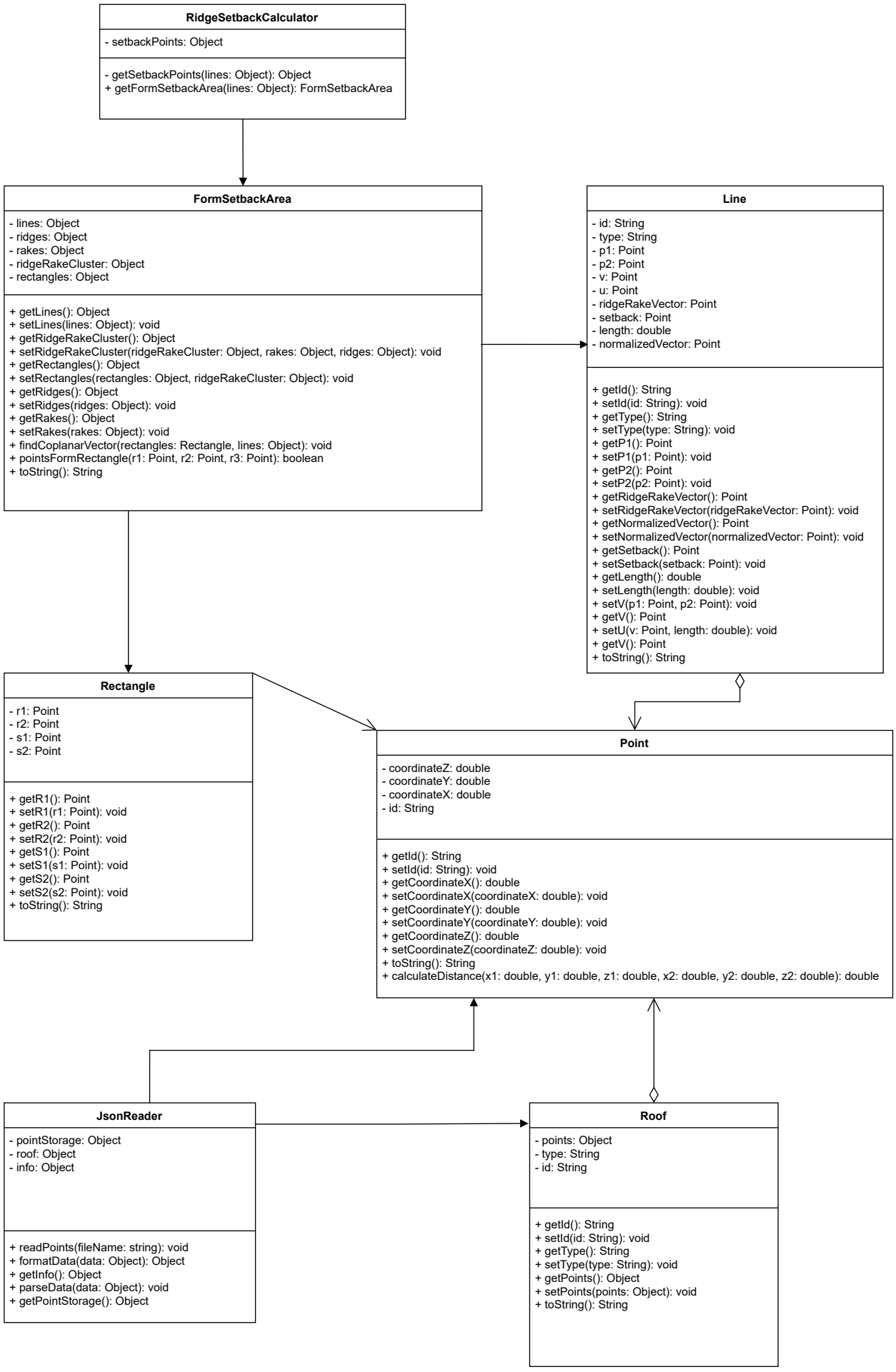
4. Workflow



2 pic. activity diagram

- Creating an issue in Jira. Issues are separated in such way that they can all be done in a week or less. Describing what needs to be done for this specific issue and specifying definition of done (DOD).
- Creating a separate branch for Jira issue. The agreed naming convention - issue id combined with short description of an issue, all words separated with underscores.
- Writing Unit tests if needed for the issue. We are following test driven development approach, therefore tests come before development.
- Development of a feature.
- Pushing changes to a remote repository.
- Opening a pull request for the branch when development is done and the code passes all tests.
- Code review of the pull request has to be done by at least two people from the team.
- Merging of the branch to the main branch of team 2. This can only be done when at least two approvals for the pull request are given.

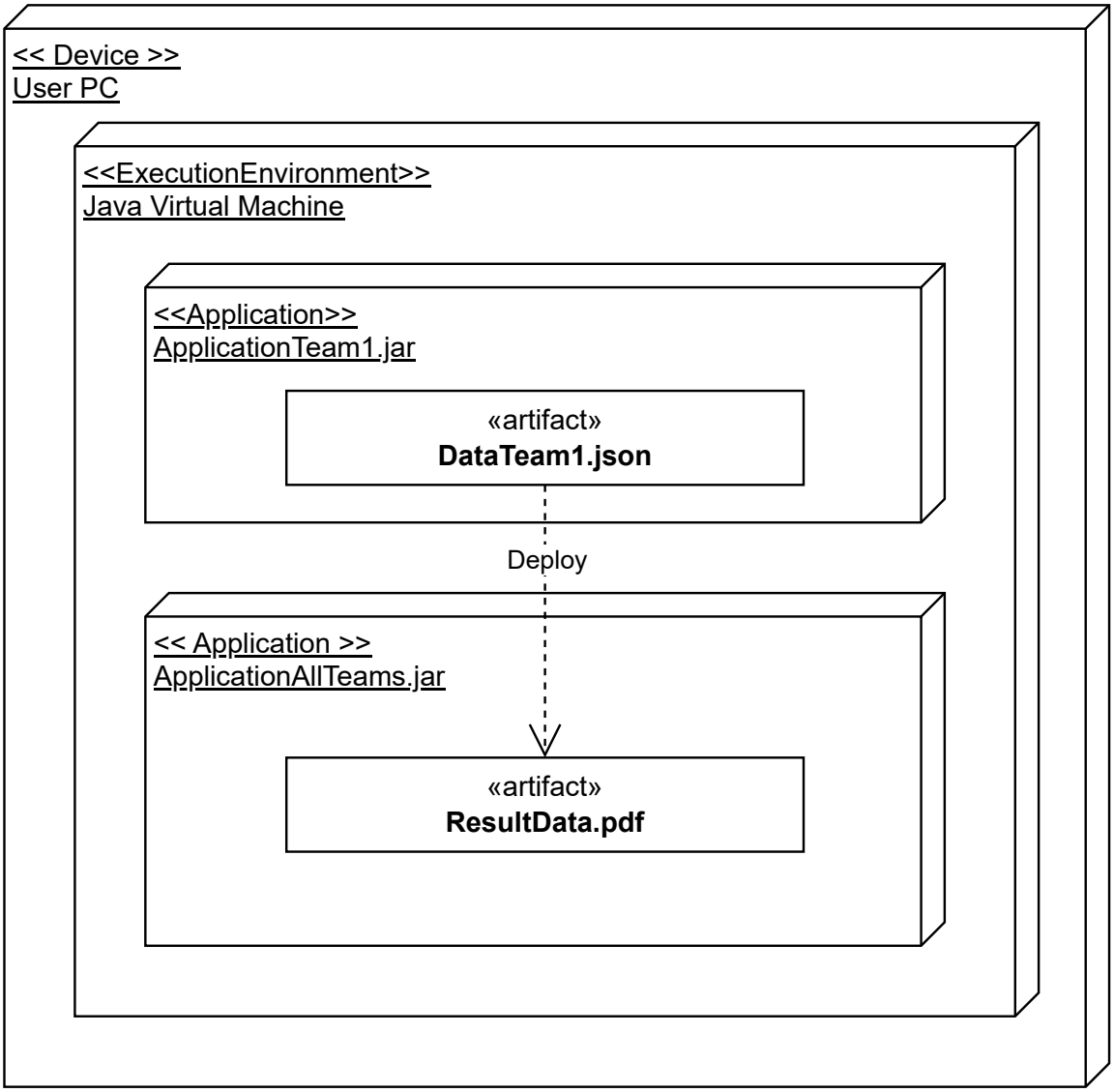
5. Structural Aspects



3 pic. class diagram

- JsonReader - a complex method used to validate Json input as safe to use and sufficient to provide meaningful output (has all necessary information for calculations).
- Roof - a class representing a roof. Consists of many lines and points.
- Point - a class representing a point on the roof. Has ID and the coordinates.
- Line - a class representing a line on the roof. Has ID, the starting and ending Point where line begins and ends.
- Rectangle - a class used for intermediary calculations when we want to find the line which will form a setback area. Rectangle is formed out of 4 Points, 2 of which are points of a particular ridge, and other 2 points are setback points 3feet away from ridge on the rake.
- FormSetbackArea - the main calculation class where we connect the correct setback points and add them to the list of lines on the roof.

6. Dynamic Aspects



4. pav deployment diagram

- The application receives input in Json format.
- Json input should be transformed into an object of class Roof.
- Class Roof contains method FormSetbackArea. It runs them at the time of creation of class instance
- Data array is formed from the modified Roof class and sent as an output.

7. Testing

Test driven development (TDD) strategy will be used in this projects. Requirements will be covered using unit tests written with JUnit. Tests will mainly cover the three main methods of the project - FormSetbackArea, calculateSecurityPathway and validateJsonInput.

Main test cases by version:

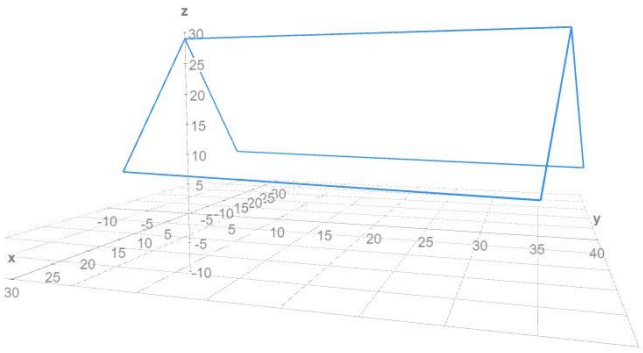
- Version 0.1.0 Calculations for a roof with one ridge and no obstacles.
- Version 0.2.0 Calculations for a roof with multiple ridges and no obstacles.
- Version 0.3.0 Calculations for a roof with multiple ridges and obstacles.
- Version 0.4.0 Validation of incoming Json input.

8. Error Handling

- In case incorrectly formatted input was provided, the application should not break and should return a response to the sender indicating the problem and suggesting to send corrected version of the input.
- In case insufficient data for calculations is provided as an input, the application should not break and should not perform any calculations. It should return a response to the sender indicating the problem and suggesting to send corrected version of the input.
- In case security pathways and/or ventilation pathways can not be found, the application should not break and should return a response to the sender indicating the problem and suggesting to send corrected version of the input.

9. Calculations

1. Let's say we have the simplest gable roof. It consists of 6 points in 3D space,forming a ridge, 4 rakes and 2 eaves.



2. We have the coordinates of each of the points. With that we can find:

2.1) Vector from one of the point to another (ridge to rake bottom):

v = (a; b; c) = (x - x_0; y - y_0; z - z_0)

2.2) Distance between two points (lenght of the rake):

$$|v| = \sqrt{((x - x_0)^2; (y - y_0)^2; (z - z_0)^2} = \sqrt{a^2 + b^2 + c^2}$$

2.3) Point on the rake, which is the needed setback from the ridge:

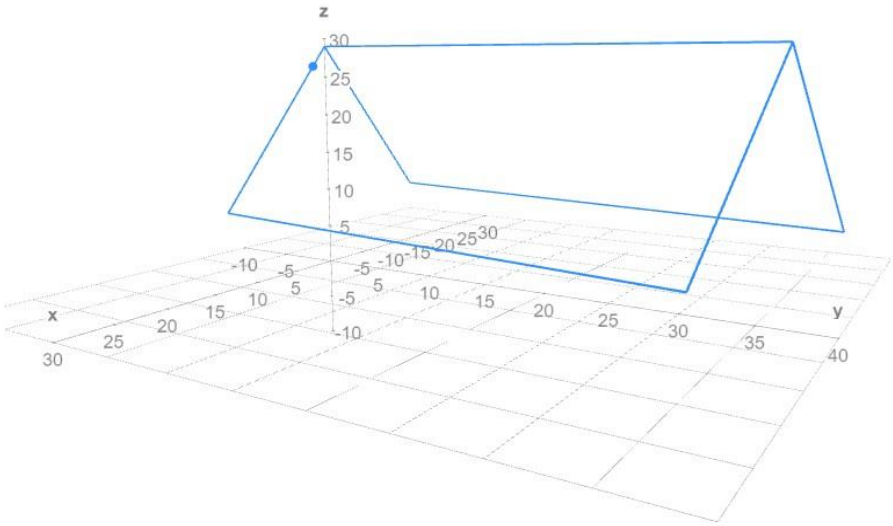
Mark the necessary distance as f (the 3 feet in our dimension).

Normalize the vector v to u:

$$u = \frac{v}{|v|} = (\frac{a}{\sqrt{a^2+b^2+c^2}} \frac{b}{\sqrt{a^2+b^2+c^2}}; \frac{cf}{\sqrt{a^2+b^2+c^2}})$$

Find the setback coordinates:

$$(x_0; y_0; z_0) + fu = ((\frac{af}{\sqrt{a^2+b^2+c^2}} + x_0; \frac{bf}{\sqrt{a^2+b^2+c^2}} + y_0; \frac{cf}{\sqrt{a^2+b^2+c^2}} + z_0))$$



2.4) The complement point, which is on the other side of the ridge.

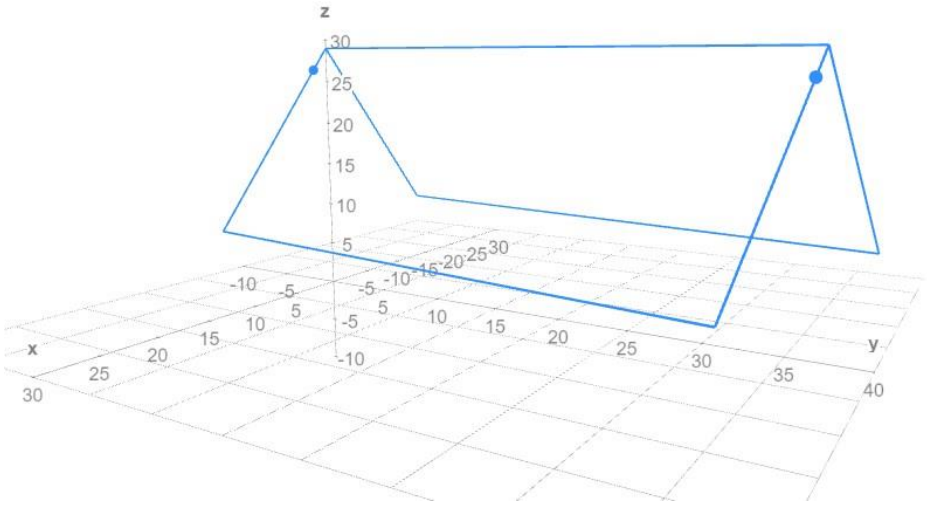
Find the distance of the ridge:

$$|v_{ridge}| = \sqrt{((x - x_0)^2; (y - y_0)^2; (z - z_0)^2} = \sqrt{a^2 + b^2 + c^2}$$

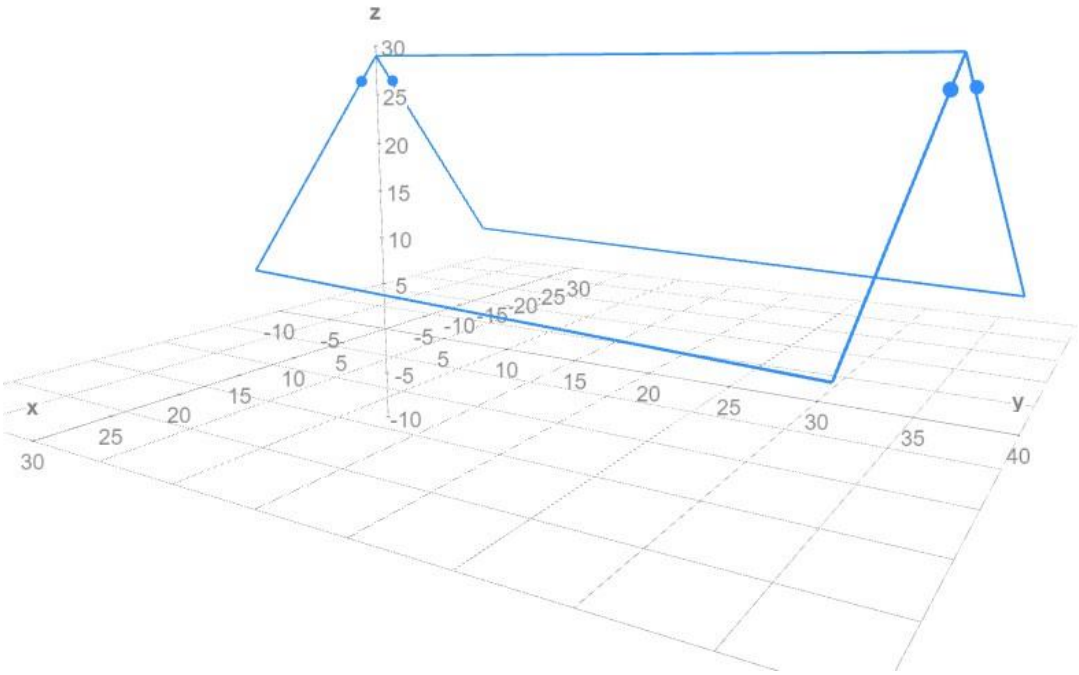
We find the complement coordinates by adding |vridge| to the y_0

$$(x_0; y_0; z_0) + fu = ((\frac{af}{\sqrt{a^2+b^2+c^2}} + x_0; \frac{bf}{\sqrt{a^2+b^2+c^2}} + y_0 + |v_{ridge}|; \frac{cf}{\sqrt{a^2+b^2+c^2}} + z_0)$$

***In case of miscalculation repeat step 2.3 with according points. Might not work because of different angle - better to calculate just as in step 2.3**



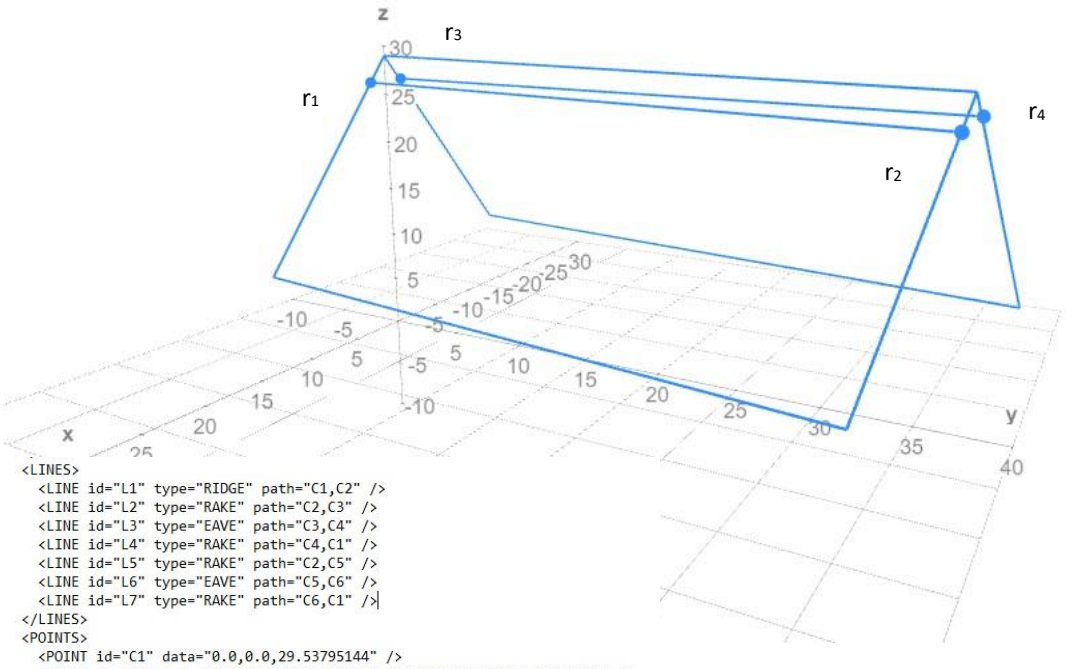
2.5) Repeat steps 2.3 and 2.4 with the opposite side of the rakes.



3. Creating new lines with found points(just as in firstly provided xml):

$l_1 = (r_1; r_2)$ = path

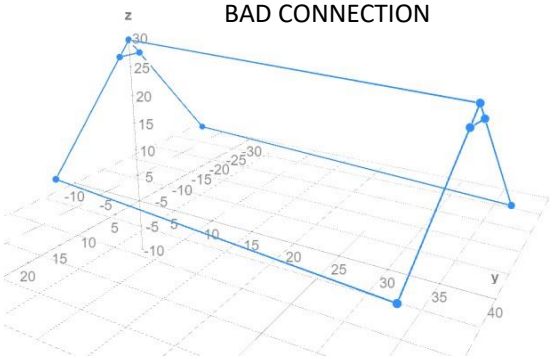
$l_2 = (r_3; r_4)$ = path



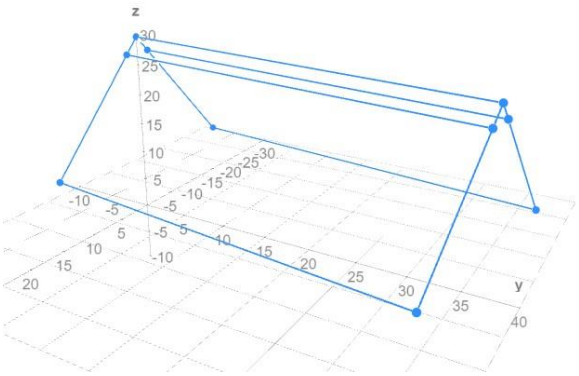

```
<LINES>
  <LINE id="L1" type="RIDGE" path="C1,C2" />
  <LINE id="L2" type="RAKE" path="C2,C3" />
  <LINE id="L3" type="EAVE" path="C3,C4" />
  <LINE id="L4" type="RAKE" path="C4,C1" />
  <LINE id="L5" type="RAKE" path="C2,C5" />
  <LINE id="L6" type="EAVE" path="C5,C6" />
  <LINE id="L7" type="RAKE" path="C6,C1" />
</LINES>
<POINTS>
  <POINT id="C1" data="0.0,0.0,29.53795144" />
  <POINT id="C2" data="4.697802960025147,37.42631535828113,29.53795144" />
  <POINT id="C3" data="17.390205760058013,35.833147199004884,9.84" />
  <POINT id="C4" data="12.692402800032868,-1.593168162330985,9.84" />
  <POINT id="C5" data="-7.994599840007722,39.01948351755738,9.84" />
  <POINT id="C6" data="-12.692402800032868,1.593168159276247,9.84" />
  <POINT id="C7" data="-3.6193651999766003,27.230786319896577,18.7856588" />
  <POINT id="C8" data="-3.4151130399433893,28.858017279654 74,18.7856588" />
```

Our job here is to make sure we connect the right setbacks points:

BAD CONNECTION



GOOD CONNECTION



Although it seems like an easy task intuitively, for a program it is not so obvious. The problem is that the program doesn't have that 3d model – it only has 4 points with coordinates, and it has to decide which of these points have to be connected.
These are the steps to do so:

3.1) Find all rakes that connect to a particular ridge. We can have more than one ridge on a roof, and to that one ridge there are 4 rakes connecting to it.
Here is a visualization:

List ridges [ridge1, ridge2, ridge3];

List rakes [rake1, rake2, ..., rake18];

In order to group a rake with ridge, we need a double cycle. In first one, we scan through the list of ridges, and in that for loop we find all of the 18 rakes that have the same point as a ridge. If it does, then we will add it to a cluster of ridge and rake points.

3.2) Form a list of 6 points for each cluster (2 ridge points + 4 setback points).
Here we need to create a cluster of points of lines, where rakes connect to a particular ridge. In a simple example, every ridge has 4 rakes, so a cluster will be consisted of 6 lines in total.
However, we need to keep in mind that there might be more than one ridge on a roof, so we need to save these clusters in a two dimensional list: first dimension is the size of the amount of ridges on the roof (how many ridges = how many clusters), and the second dimension is for storing 6 lines in that cluster.
Here is a visualization (amounts of lines are taken as example):

List ridges [ridge1, ridge2, ridge3];

List rakes [rake1, rake2, …, rake18];

List cluster [[r1,r2, s1, s2, s3, s4], [r1, r2, s1, s2, s3, s4], [r1, r2, s1, s2, s3, s4]]

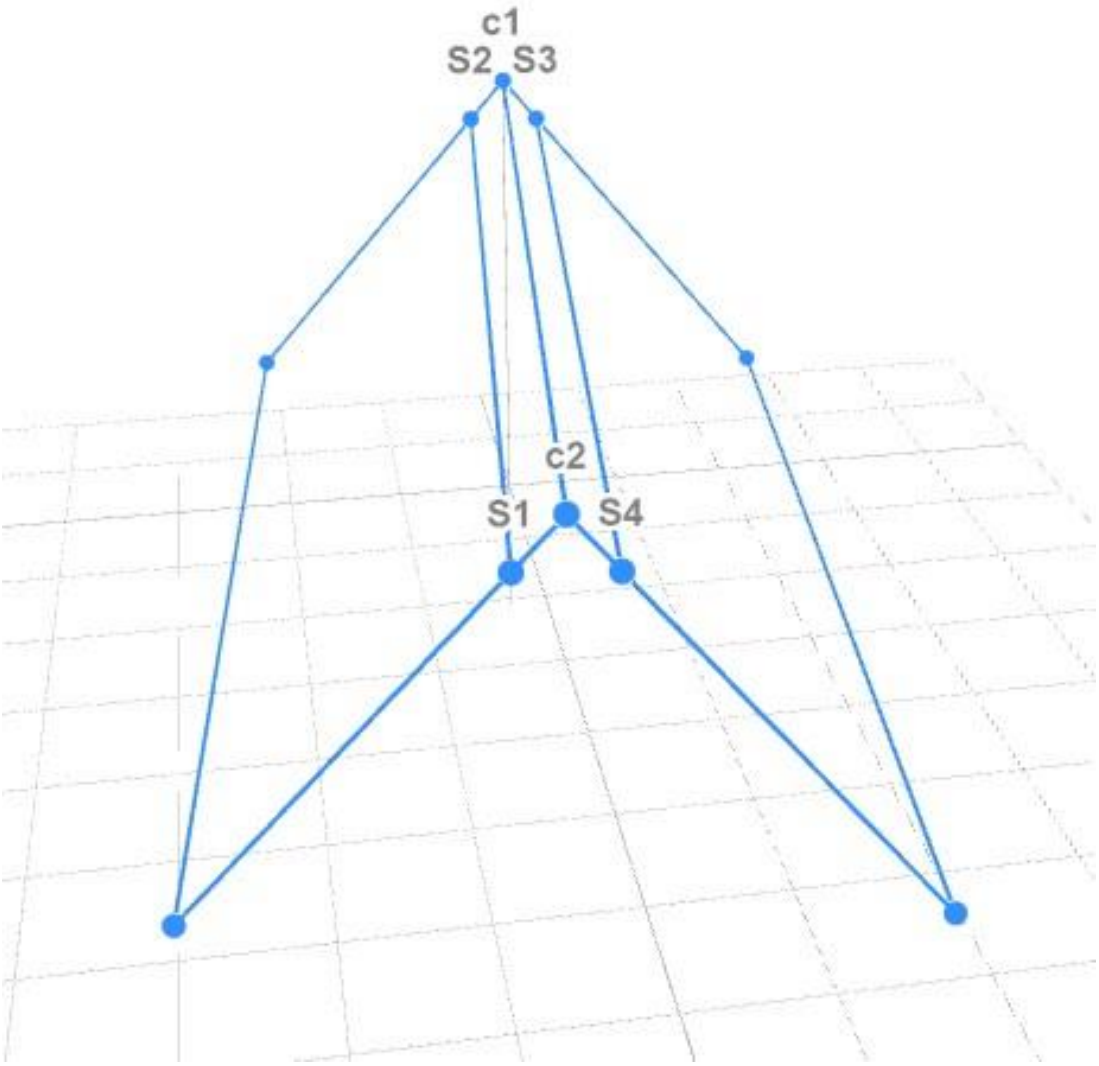
***Here r1 and r2 are ridge points, and points s1-4 are setbacks points of the rakes that connect to that ridge**
That ridge and rake cluster of points will be very usefull later, because for every object (ridge, or small roof) we have all the information needed to draw the lines between setback points.

3.3) Find 2 possible rectangles for each cluster, which always consists of 2 ridge points. This is an intermediary step. Out of 6 points in a cluster, we can form 2 rectangles that will both have ridge points. In this example those rectangles are:

rect1=[c1, c2, S2, S1] rect2=[c1, c2, S4, S3]

We save that information in a similar double array as our cluster. The first dimension again is the size of ridge amount, and second dimension will always contain two rectangles. It looks like that:

List rectangles[[rect1, rect2], [rect1, rect2], [rect1, rect2]]



3.4)Add the setback points in a rectangle to all line list. Every rectangle has 4 vertices. So in that rectangle we need to connect points, that don't have the ridge points in them. In our example for first rectangle we connect a line between S2 and S1, and in second between S4 and S3. Here is an example of such addition:

lines.add(new Lines("L"+(lines.size()+1), "SETBACK_LINE", rectangle.getS1(), rectangle.getS2()));

***Here lines is a list of all lines (ridges, rakes and eaves). first parameter is its ID (starts with L) second parameter is its type (ridge, rake, eave or setback_line) third parameter is its first point fourth parameter is its second point**