NTNU
Det skapende universitet

# Wind Load Module For CSi ETABS

**Eirik Aasved Holst**

Master i ingeniørvitenskap og IKT
Innlevert: juni 2015
Hovedveileder: Tor Guttorm Syvertsen, KT

Norges teknisk-naturvitenskapelige universitet
Institutt for konstruksjonsteknikk

NTNU
Department of Structural Engineering

# MASTER THESIS 2015

for

*Stud.techn. Eirik Aasved Holst*

## Wind Load Module for CSi ETABS

*Vindlastmodul for CSi ETABS*

## Background

Structural analysis require proper handling of all loads the structure is exposed to, including environmental loads like wind. In Norway, wind loads are determined from Eurocode 1991-1-4. The regulations are designed for hand calculations, but it is desirable to be able use a structural engineering software instead.

The software used in this thesis is CSi ETABS and the objective of the thesis is to develop an extension for CSi ETABS that provides wind loads in a semi-automatic fashion.

## Scope of Work

- Review of previous work and software
- Establish principles for import of structural model
- Establish principles for building envelope subjected to wind
- Develop methods for generating wind-induced loads according to Eurocode
- Develop methods for transferring loads to analysis
- Develop software extensions for CSi ETABS

## Deliverables

The thesis will result in an extension (called “Aeolus”) to CSi ETABS with a demo and a digital report. The demo and the digital report will be graded.

The report is to be handed in to the Department of Structural Engineering by June 10, 2015.

The thesis may be adjusted due to the progress of work and the interests of the student.

The report is to be organized in accordance with the current instructions (http://www.ntnu.no/kt/studier/masteroppgaven).

Supervisor is Professor Tor G. Syvertsen (torgsyv@gmail.com)

Contact at EDR&Medeso is Øystein Flakk (oystein.flakk@edrmedeso.com)

## Norwegian Abstract: Sammendrag

For korrekt dimensjonering av konstruksjoner er det viktig å medberegne miljøbelastninger, inkludert vindlaster. I Norge bestemmes vindlaster ved bruk av NS-EN 1991-1-4:2005+NA:2009. Regelverket er utformet for håndberegninger, og legger ikke til rette for bruk av datasystemer for effektiv lastberegning.

Denne masteroppgaven går ut på å designe og implementere en utvidelse til analyse-programmet CSi ETABS. Utvidelsen må kunne etablere en objektmodell basert på geometrien til en 3D-modell. Den må kunne inkludere terrengfaktorer og påføre modellen vindlaster i henhold til regelverket.

Denne rapporten forklarer bakgrunnen, teorien, fremgangsmåten og implementeringen til utvidelsen, heretter kalt Aeolus, etter den greske guden for vind.

# Abstract

It is crucial to include environmental loads in the design of a building. Codes and regulation are still in the age of hand calculation, but it is desirable to use an engineering software to generate environmental loads, including wind-incudes loads.

The extension created in this project may load a CAD-model/BIM, recognize the geometry, assign wind loads on the model according to NS-EN 1991-1-4:2005+NA:2009, and save the model for analysis. Its purpose is to augment the engineering process, not to automate it.

This report explains the theory, background, and implementation of wind load extension, hereafter called Aeolus, after the ruler of the winds in Greek mythology.

# Preface

This report, an extension developed for CSi ETABS and a demo of said extension is the result of the course TKT4915, Computational Mechanics, Master's Thesis. The work has been in cooperation with EDR&Medeso AS and is submitted in partial fulfilment of the requirements for the degree of Master of Science to NTNU Department of Structural Engineering.

I would like to thank EDR&Medeso for the opportunity to work with this project in addition to helping me and providing me what I needed when it came to software, licenses, and technical assistance.

In addition, I would like to thank Hege Auglænd for teaching and motivating me to use TDD in developing the extension.

Tor G. Syvertsen has been a great asset to my thesis, and I would like to thank him for his guidance and enthusiasm.

# Table of Contents

1 Introduction ..... 1
- 1.1 Background ..... 1
- 1.2 The idea ..... 1
- 1.3 Results ..... 1

2 Technologies ..... 2
- 2.1 CSi ETABS ..... 2
- 2.2 Application Programming Interface (API) ..... 3
- 2.3 Programming tools ..... 4

3 Geometry Recognition ..... 5
- 3.1 Wall Recognition ..... 6
- 3.2 Wall Partitioning ..... 12

4 Eurocode NS-EN 1991-1-4:2005+NA:2009 ..... 14
- 4.1 Assumptions ..... 16
- 4.2 Calculation model ..... 17

5 Examples ..... 19
- 5.1 Example 1: single wall ..... 19
- 5.2 Results of Example 1 ..... 20
- 5.3 Analytical calculation for example 1 ..... 22
- 5.4 Example 2: Box ..... 23

6 Software Development ..... 25
- 6.1 Implementation ..... 25
- 6.2 Extension vs Plug-in ..... 28
- 6.3 User Manual ..... 30
- 6.4 Code Complexity ..... 36
- 6.5 Unit Testing ..... 38

7 Discussion ..... 41
- 7.1 Conclusion ..... 42
- 7.2 Further Work ..... 43

8 Bibliography ..... 44

**Appendix A: Newspaper clippings ..... 48**

**Appendix B: Terrain Categories ..... 51**

**Appendix C: Class Overview for Aeolus ..... 53**

# Terms and Definitions

## Acronyms and abbreviations

| | |
|---|---|
| **ETABS** | Extended Three-dimensional Analysis of Building Systems. Software from Computers and Structures, Inc. [1]. |
| **DLL** | Dynamic-Link Library, data files that provides a mechanism for shared code and data [2]. |
| **API** | Application Programming Interface [3]. |
| **BIM** | Building Information Model |
| **TDD** | Test Driven Development [4]. |
| **FBD** | Free Body Diagram |

## Symbols

| | |
|---|---|
| $v_{b,0}$ | Fundamental value of the basic wind velocity [5]. |
| $v_m(z)$ | Mean wind velocity [5]. |
| $q_p(z)$ | Peak velocity pressure [5]. |
| $w_e(z)$ | External wind pressure [5]. |

## Notations

| | |
|---|---|
| $\vec{\boldsymbol{v}}$ | Vector notation |

## Software terms (in italic)

| | |
|---|---|
| *MethodName( [parameter type 1] [parameter name 1], [parameter type 2] [parameter name 2], ... )* | Object method from a code example. |
| *Model/View/Controller* | Components in the Model-View-Controller pattern [6]. |
| *Class/Object* | Object or a class from the code. |

# 1 Introduction

## 1.1 Background

Why are wind loads important? This question has a simple answer: Energy. Air moving at high speed has a considerable amount of kinetic energy. This energy may be utilized to generate electricity, but it can also cause severe damage. Every storm rolling in towards Western Norway may have the capability to take off roofs and leave towns without power. See Appendix A: Newspaper clippings for examples.

To prevent faulty designs of constructions, The European Committee for Standardization [7] develops Eurocodes [8], a set of technical regulations for structural design of constructions in the European Union. Norway is among the countries required to implement these Standards. However, the regulations are still in the age of hand calculations, and represent obstacles to a smooth flow of information between the various software components.

The motivation for this work has been to provide engineers with a tool for efficient calculation of wind-induced loads. The tool should be capable of exchanging information with other structural analysis software.

## 1.2 The idea

Based on geometry-related data from a CAD-model/BIM it should be possible to:

- Establish a structural model including characteristics of the surrounding terrain
- Generate wind-induced loads according to NS-EN 1991-1-4:2005+NA:2009 in a semi-automatic fashion

It is desirable that the system is interactive and iterative for convenient user control and presentation of the generated loads.

## 1.3 Results

Besides this report, the work has resulted in an extension to CSi ETABS for generation of wind-induced loads according to the Eurocode. Distribution of CSI ETABS and extensions is restricted, but a video recording demonstrates use of the developed software. The name of the extension is “Aeolus”, named after the ruler of winds in Greek Mythology [9].

# 2 Technologies

## 2.1 CSi ETABS

ETABS [1] is a software package for structural analysis and design of buildings developed by Computers & Structures, Inc. (CSi) based in Berkeley, CA. Among the well-known software from CSi are SAP2000 [10], CSiBridge [11] and ETABS. The name ETABS is an acronym for Extended Three-dimensional Analysis of Building Systems.

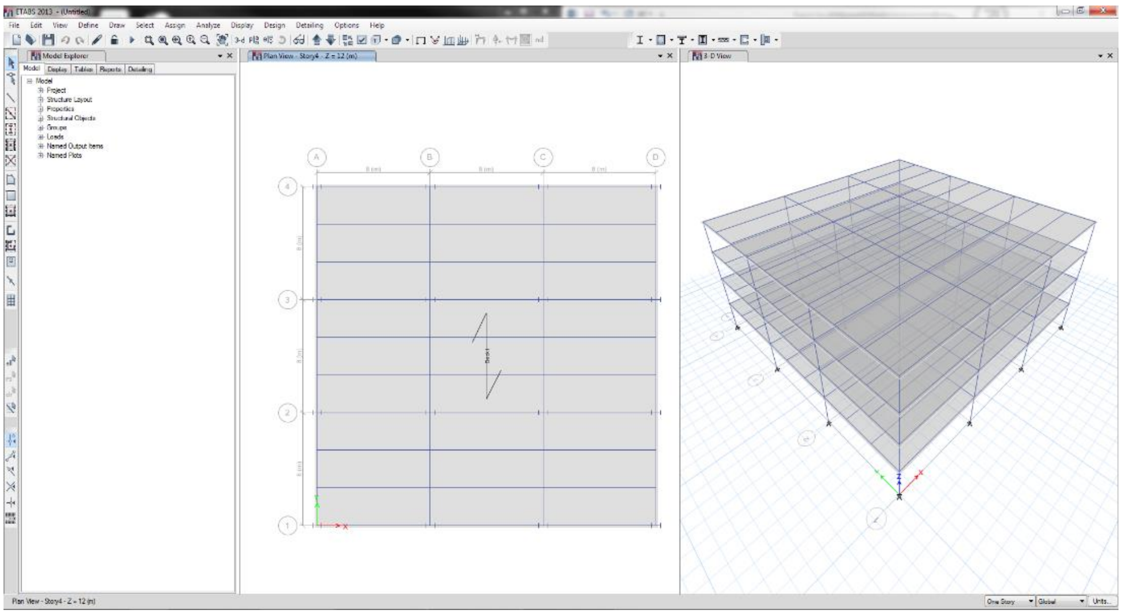

ETABS 2013 - (Untitled)
File Edit View Define Draw Select Assign Analyze Display Design Detailing Options Help
Model Explorer
Model Display Tables Reports Detailing
Model
Project
Structure Layout
Properties
Structural Objects
Groups
Loads
Named Output Items
Named Plots
Plan View - Story4 - Z = 12 (m)
3-D View
Plan View - Story4 - Z = 12 (m)
One Story
Global
Units...

Figure 2-1: Main views in CSi ETABS 2013 [47]. One planar view and one 3D-view. Here, one floor is modeled, then copied to the other three floors. Columns, beams and restraints are auto-generated

The concept of ETABS is based on stories, and it is dominantly used for regularly shaped high-rise buildings, often office-buildings. ETABS handles this by enabling the user to draw a planar floor and copy the floor to the number of stories needed. Each story may be edited individually if needed. Columns, walls, shafts and beams may be auto-generated. This enables quick and easy modelling of for instance skyscrapers like the world's tallest building, Burj Khalifa (formerly Burj Dubai) [12].

ETABS may perform both static and dynamic analysis, including the calculation of vibration modes and time-history analysis. It also supports material and geometrical nonlinearity.

ETABS has a wind load feature that is mainly used for stability analysis of structural systems and it is not specific enough to be used for smaller buildings and/or parts of structural systems. The current wind load feature will only apply wind loads at the nodes of the model, which excludes local bending moments. For more information about local bending moments, see Section 3.2. Wall Partitioning

## 2.2 Application Programming Interface (API)

CSi ETABS provides an Application Programming Interface (API) that allows engineers and developers to modify the software with extensions and bridges to third party software. The API is compatible with most major programming languages, including Visual Basic, C#, C++, FORTRAN, Python and MATLAB. However, the current version provides limited documentation. [13]

The API enables a rich real-time two-way link between an extension and ETABS, allowing access to models, complete control of execution, extraction, analysis, design information, all from within Aeolus. [14]

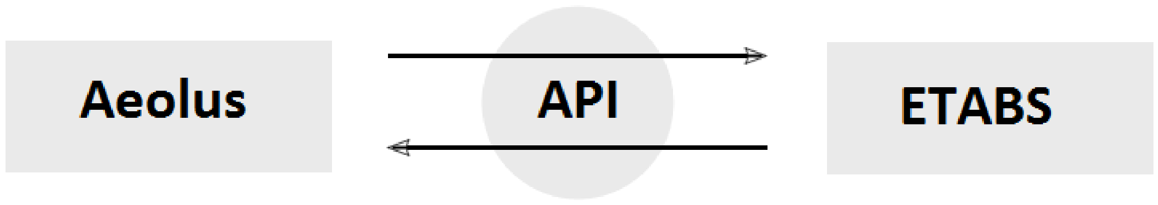


Figure 2-2: Aeolus and ETABS communicates through the provided API

### API restrictions

The main challenge regarding implementing an extension using the provided API is that one cannot access objects in the model. One may enquire unique names of objects and their properties, but not the object itself. Approaches to overcome this obstacle are discussed in Chapter 6. Software Development.

## 2.3 Programming tools

### Microsoft Visual Studio

The main programming tool used to create Aeolus is Microsoft Visual Studio [15], which is an IDE – Integrated Development Environment. This means that the tool includes a source code editor, a debugger, and a designer. The software is used to develop computer programs for Microsoft Windows, in addition to web sites, web application, and web services [16]. It includes a debugger to help fix errors in the code and a forms designer to create GUI applications. In addition, many other tools available have not been used in this thesis.

### Resharper

A plug-in for Visual Studio called Resharper [17] has been used to enhance the programming process. The plugin augments code editing with suggestions and corrections of compiler errors, runtime errors, redundancies, code smells [18], and other improvements while you type. [19]

### Programming language

The main programming language used is C# [20], which is general-purpose object-oriented programming language similar to Java, but developed by Microsoft. In addition, Visual Basic [21] was used to define macros and other minor tools for Microsoft Word [22] to simplify writing the report.

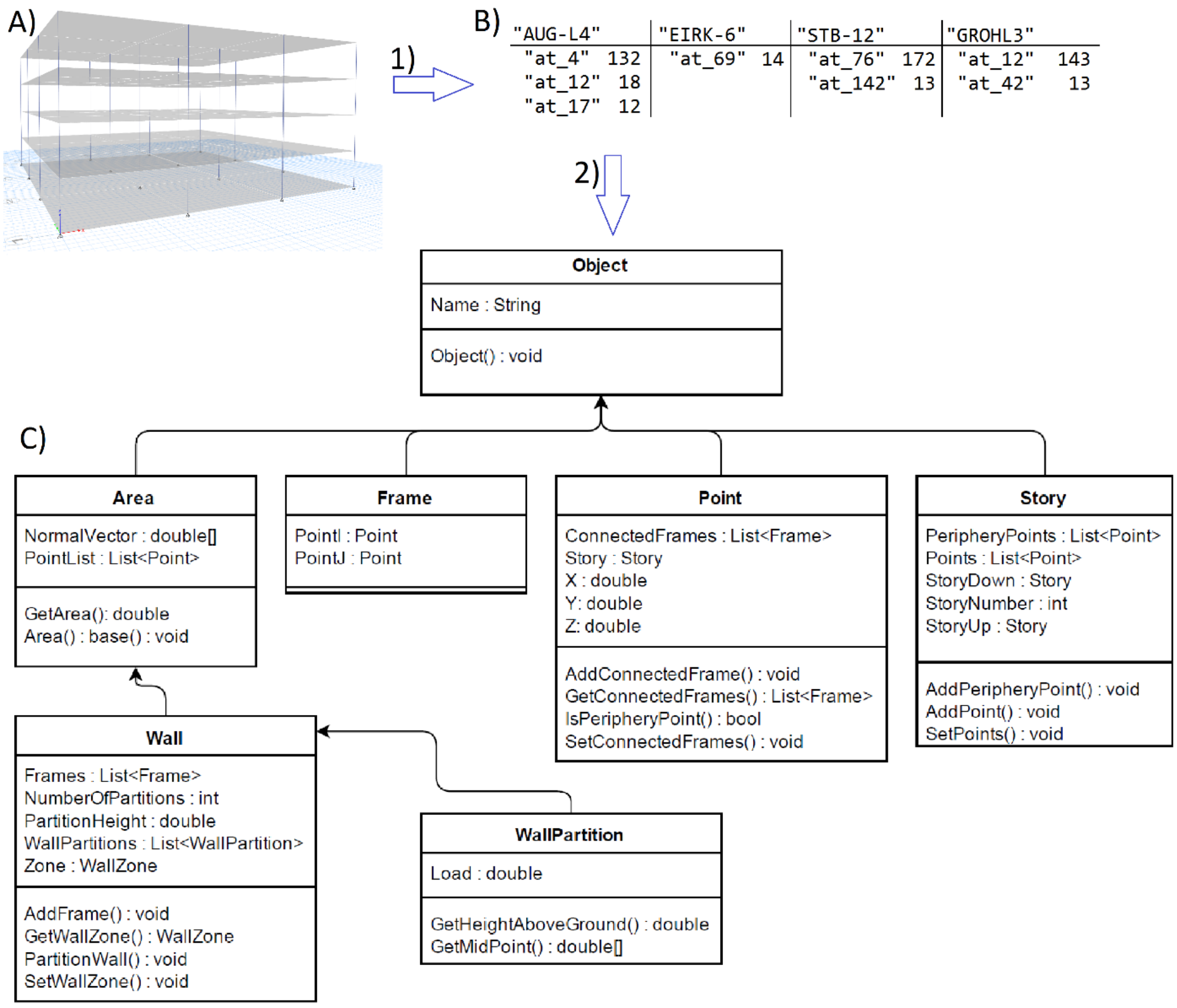
# 3 Geometry Recognition

A human looking at the image of a three-dimensional model will immediately make a mental model and establish some idea of the relations between different parts of the model. A single central processing unit (CPU) will never be able to do this because of its main disadvantage compared to humans: It can only observe one thing at a time. For that reason, geometry recognition may be challenging for a computer.

The core problem is the establishment of relations between objects. Therefore, increasing the number of CPUs may speed up the process, but not alleviate the problem.

## What is geometry recognition?

The process of geometry recognition is creating an object model from the data available. In this context, Aeolus receives data from an ETABS model. The API, however, does not provide access to the ETABS object model, only to lists of attributes and values representing nodes, beams, etc. An object model contains all elements from the model as objects, including attributes, methods, and relations to other objects. Aeolus needs to create an object model from the data available.



```
"AUG-L4"       | "EIRK-6"      | "STB-12"        | "GROHL3"
  "at_4"   132 |  "at_69"  14  |  "at_76"   172  |  "at_12"   143
  "at_12"  18  |               |  "at_142"  13   |  "at_42"    13
  "at_17"  12  |               |                 |
```

Figure 3-1: A) Visual representation of a model from ETABS [25]. 1) Data generated from the ETABS-model via the API. B) A list of data received from ETABS. 2) Geometry recognition: Interpret the data and create an object model. C) Example of a class diagram created in the code

## 3.1 **Wall Recognition**

As seen in Figure 3-1: A) , walls are uncommon to include in the model, as their contribution to mass and stiffness matrices are negligible. This may make recognition of walls challenging, and it may require user input.

Since ETABS is based on stories, the recognizer is able split the problem into two parts:

1. Story periphery part: The recognizer locates the outline of each story.
2. Periphery connecting part: The recognizer connects the story peripheries to form walls.

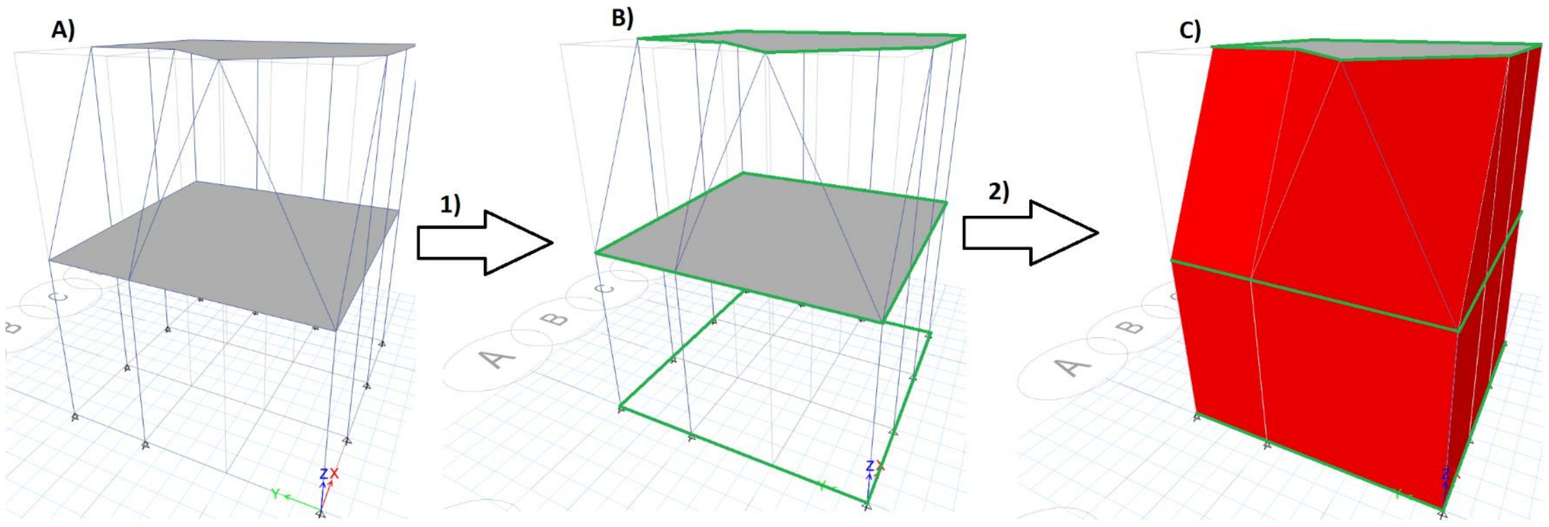


Figure 3-2: Strategy of wall recognition. From a model (A) the periphery of each story is locates (1). The peripheries are connected (2), defining the walls (C).

### Story periphery

The process of recognizing a story periphery may be described in the following two steps

### 1. Locate a source node on the story

The recognizer locates a periphery node on the story and set this node as a source node. A source node is defined as a node connected (directly or transitively) to another story and lies on the periphery of its own story. To locate it the recognizer loops through all nodes on the story and checks whether the first requirement for a source node is met, and picks the node with the smallest x coordinate (must be in the periphery).

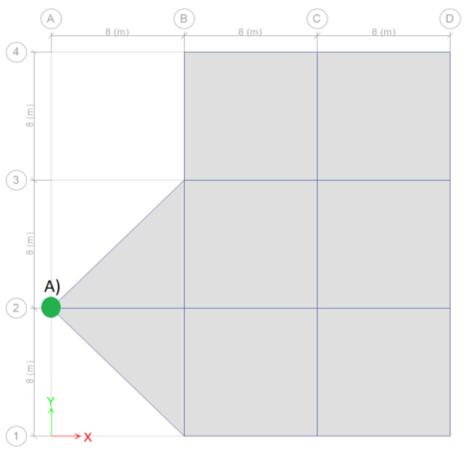


Figure 3-4: A) Source node on an example story in ETABS [51]

The base story will be a special case because the nodes on the base story are not necessary connected to each other, depending on the design of the base story. Therefore, the base story may not be traversed in the same manner. However, the base nodes are connected to nodes on the story above. Periphery nodes on the base story are defined as nodes directly below the periphery nodes on the story above.

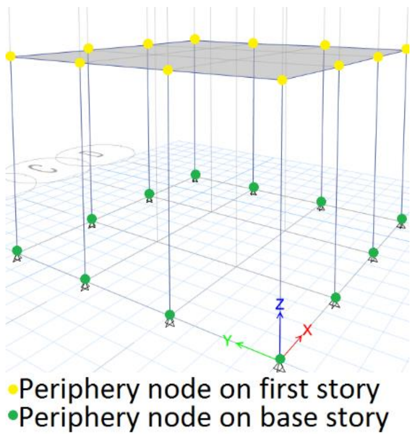


Figure 3-3: Base story periphery is defines by nodes directly below periphery nodes on the story above.

## 2. Traverse the periphery of a story from the source node

Starting from a source node on a story, the recognizer traverses the periphery of a story to locate the other periphery nodes on said story. This is done by “keeping right” when traversing. In other words, for each node in the path, the next node is found by following the frame with the lowest counter-clockwise angle from the previous frame. The traversing stops when the traverser reaches the source node.

Every periphery node has an attribute for its periphery node neighbor in the counter-clockwise direction. This attribute is set when the recognizer traverse a story, to create relations between the nodes.

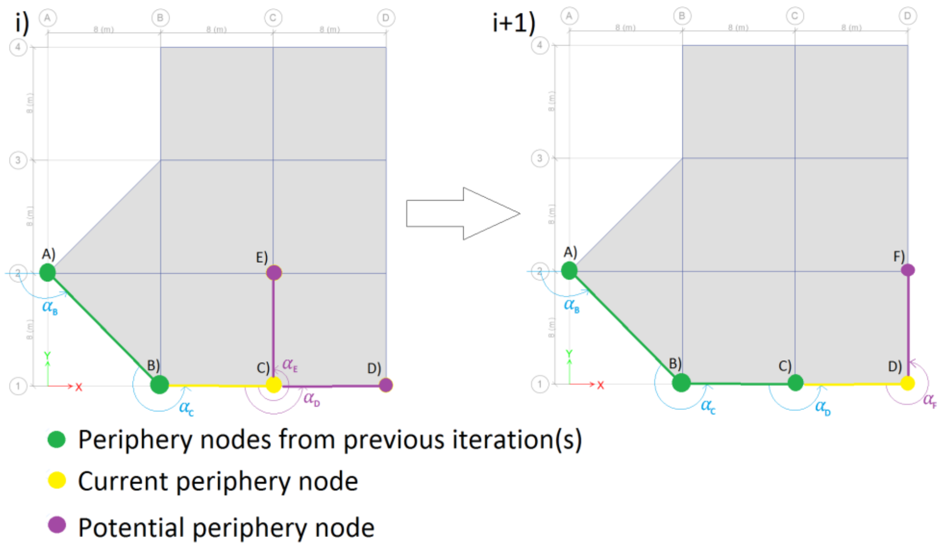


Figure 3-5: Example of traversing a story. At iteration i, the traverser has reached node C and will choose node D as next node because $\alpha_{BCD} < \alpha_{BCE}$

When all stories have a defined periphery, the stories may be connected with each other to create the walls. A possible approach is to define neighbor nodes to the story below for all stories but the base story (when a neighbor is set for a node object, the neighbors node attributes is updated). Unlike neighbors on the story, a node may have more than one neighbor to the story below or above, enabling triangular wall elements.

When the entire periphery of a model is well defined, wall elements may be added in an efficient manner. All node objects in the periphery of the model have attribute *neighbor* in any direction, providing the program the opportunity to loop through the entire periphery of the model. By following a set of rules when looping through the model, walls in all shapes are recognized. However, all wall elements are assumed planar. The recognizer uses the following algorithm to add wall objects:

```
foreach (var story in Storys)
{
      foreach (var peripheryNode in story.PeripheryNodes)
      {
            foreach (var neighborDown in peripheryNode.NeighborsDown)
            {
                  wallList.Add
                  (
                        GetWallFromPeripheryNode(peripheryNode,
                        neighborDown)
                  );
            }
      }
}
```

The method *GetWallFromPeripheryNode(Node peripheryNode, Node neighborDown)* uses the same principle as when finding periphery nodes on a story. It uses the information from the parameters and traverses a wall element periphery by always taking left. For detailed information, see Figure 3-7.

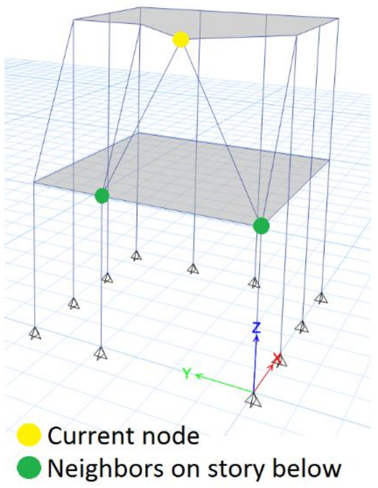


Figure 3-6: Node with node neighbors on story below

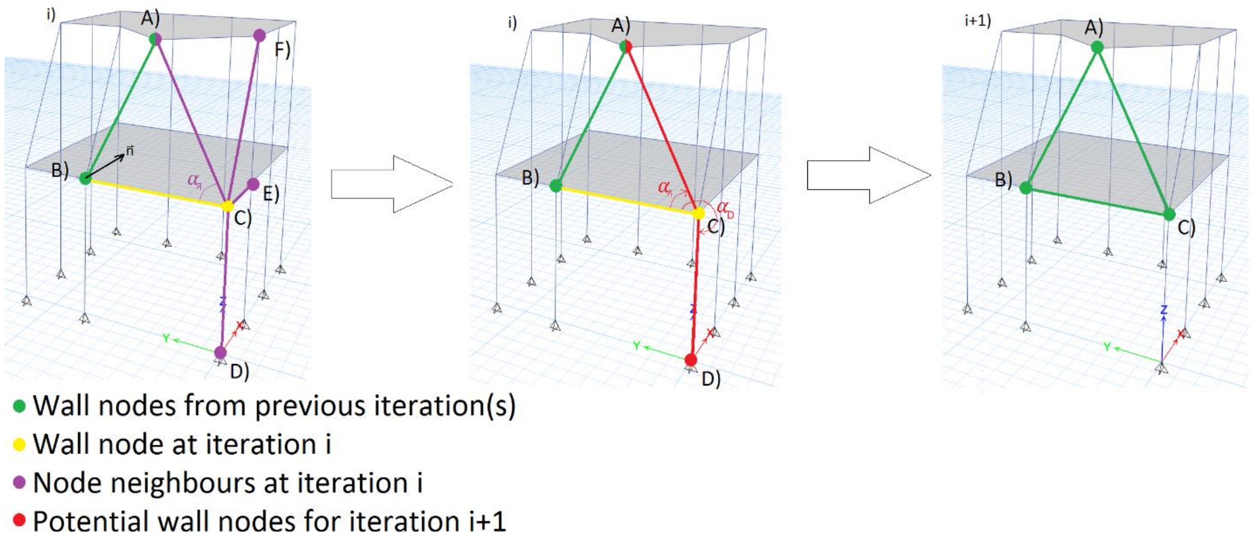

i)
A)
F)
B)
n
$\alpha_A$
E)
C)
D)
Z
Y
X
i)
A)
B)
$\alpha_A$
$\alpha_D$
C)
D)
i+1)
A)
B)
C)
● Wall nodes from previous iteration(s)
● Wall node at iteration i
● Node neighbours at iteration i
● Potential wall nodes for iteration i+1

Figure 3-7: Illustration of how a wall is found by traversing. The function that locates the wall takes two nodes as parameters: A source node (A) and its neighbor on the story below (B). First, three nodes need to be found do define the plane of the wall. This is done by starting from a source node (A), add the neighbor down (B), and then add the neighbor down's story neighbor in counter-clockwise direction (C). A normal vector defines the wall plane: $\vec{\mathbf{n}} = \overrightarrow{\mathbf{BA}} \times \overrightarrow{\mathbf{BC}}$.

The iteration loop is defined by always going to the next node, in the plane, which makes the smallest clockwise angle from the previous node. When the traverser reach node C it will choose node A as the next node since $\alpha_{BCA} < \forall\alpha_{BCD}$. The wall is defined when the traverser reaches the source node.

## User input

When a wall is created by Aeolus, it gets a property (in ETABS) called "*WindWall*". This is used as a reference when Aeolus calculates and applies the actual wind load. Every model object with said property is considered as a part of the model that is subject to wind-induced loading. This enables the user to edit the model after the geometry has been recognized, since the user has the ability to add and delete elements, and assign different properties to the elements. This process shifts the program's paradigm from "automation" to "augmentation", providing help and support for the user instead of trying to replace the user. For more information about user input and the workflow of Aeolus, see Section: 6.3. User Manual

## Unrestricted API

If the API had no restrictions and every object and method were accessible, there would be no need for the extension to create an object model. All objects would already have the methods and attributes needed for efficient geometry recognition. One could even imagine that the geometry recognition was excessive because the BIM received from ETABS would include the necessary information about geometry, walls, connections, restrictions, relations, and everything needed to add wind induced loads according to the Eurocode.

## 3.2 **Wall Partitioning**

In ETABS, a wall element is defined as a list of nodes in that run clockwise or counterclockwise around the wall element. It is possible to assign a distributed load to a wall element, but ETABS will lump these loads to the nodes of the wall elements, as these are the only information about the wall element. The lack of pressure forces on wall elements in ETABS is a severe limitation because if the columns carry the wall, the load transferred from the wall to the columns should introduce a distributed load along the column, not lumped loads at the nodes.

A viable approach to overcome this challenge is to divide the wall into horizontal strips to distribute the lumped loads along the columns, resulting in a discretized distributed load.

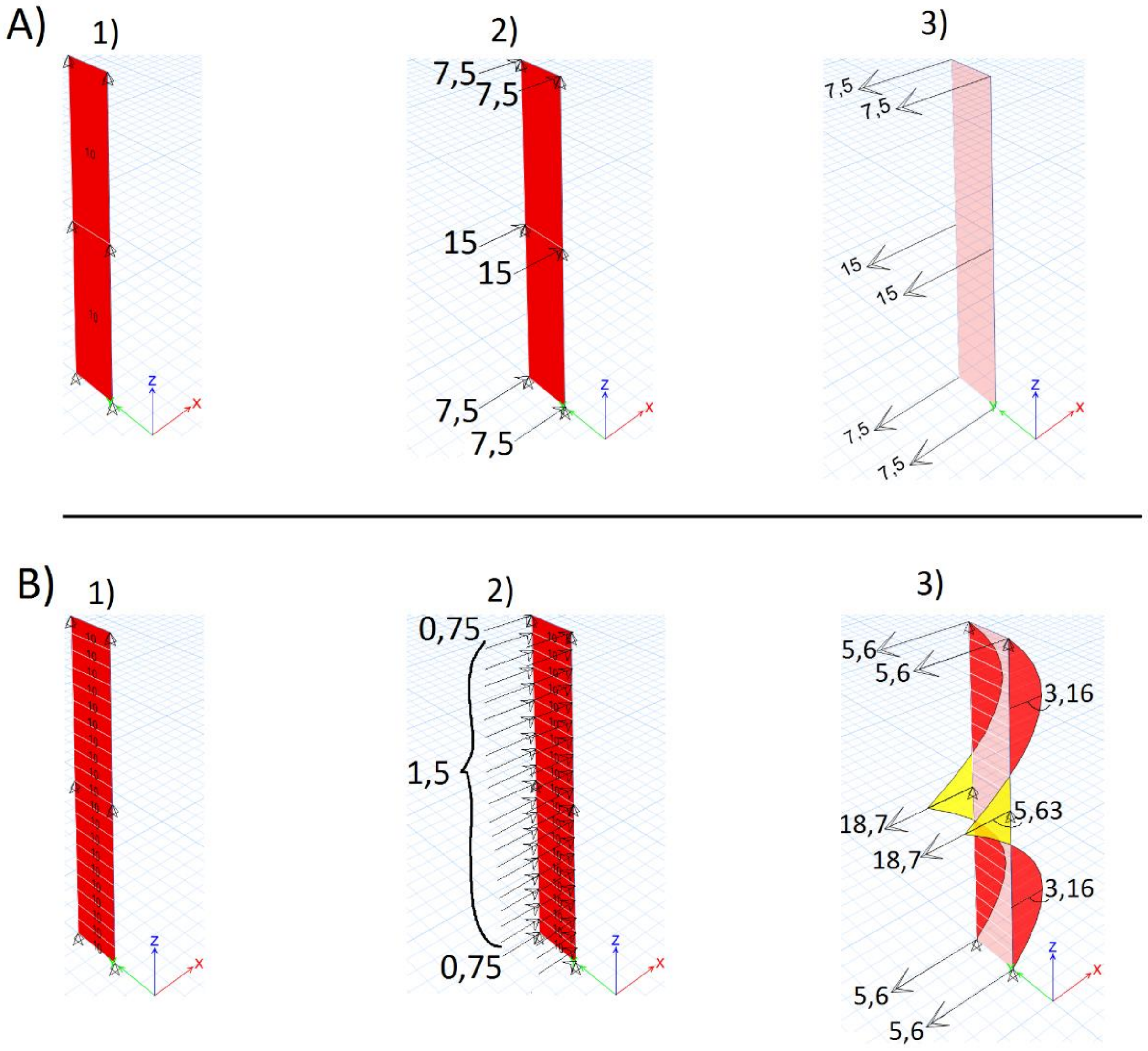


Figure 3-8: Explanation of wall partitioning. The figure shows one wall with two elements (A) and the same wall partitioned into twenty elements (B). The wall is six meters tall and one meter wide. The wall is supported at the base, at the middle and at the top. At (1) the distributed load is displayed (10 $Pa$). At (2) the distributed load is lumped to the corner of the wall elements. At (3) the reaction forces and moment diagram is displayed.

A wall is partitioned by attaching a rectangular plane around the wall element, and dividing this plane into a reasonable amount of vertical strips. These strips define the wall partitions. See figure below for details. The default setting is that the walls are connected to the *columns*; however, the user may choose to connect the walls to either columns, beams or nodes.

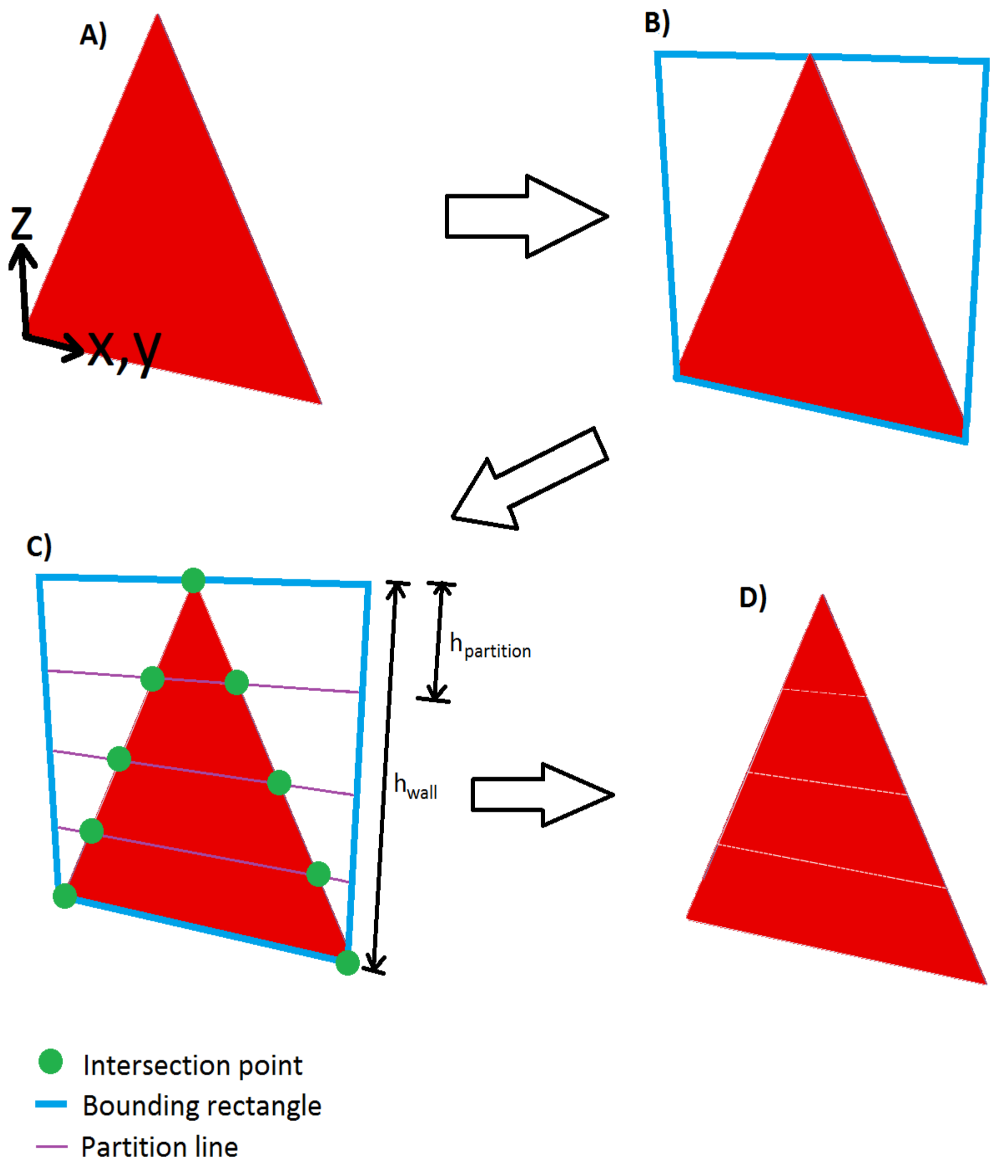


Figure 3-9: Wall partitioning into horizontal strips with equal height. A wall element (A) forms a basis for a bounding rectangle (B) that is used to partition the wall element. A reasonable number* of partition lines is created (C) to achieve the coordinates of the points that defines the wall partition corner points. These points are is defined as the intersection points from the partition lines or the bounding rectangle and the wall frames. When all intersection points are established, the wall can easily be partitioned (D).

*reasonable number: A value calculated from partition height, which given by the user. The default value for partition height is 500 mm.

# 4 Eurocode NS-EN 1991-1-4:2005+NA:2009

Eurocodes [8] are a set of harmonized technical rules developed by the European Committee for Standardization [7] for structural design of construction works in the European Union. Its purpose is to provide a basis for construction and engineering contract specifications. It is mandatory for European public work to obey the regulations and they are intended to become the de facto standard for the private sector. [8]

There are about 300 published EN standards [23]. However, ten of those standards are published as The Eurocodes [8]. Each of the ten codes, except EN 1990, are divided into a number of parts covering specific aspects of the subject. In total there are 58 EN Eurocode parts distributed in the ten Eurocodes [24]

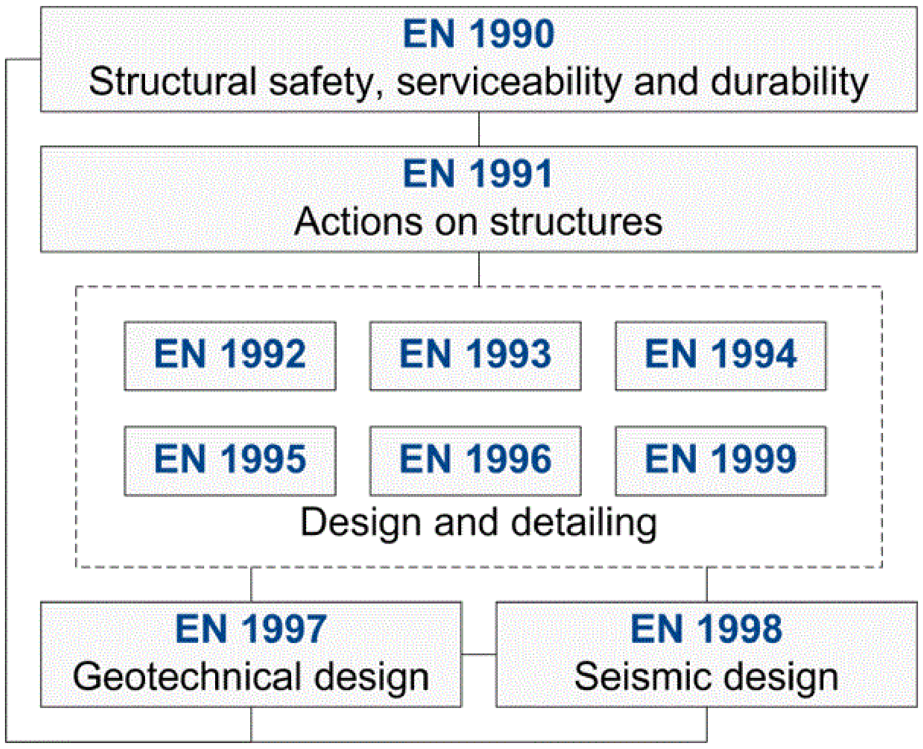


Figure 4-1: Links between the ten Eurocodes [48].

## Eurocode naming

The name NS-EN 1991-1-4:2005+NA:2009 gives information about what the standard is about. 1991 is *actions on structures*. 1-4 is *general wind actions*. 2005 is when the general document was updated and NA:2009 means the national annex was updated in 2009.

## Eurocode complexity

Facts about the extent of the Eurocodes: The 58 parts of the ten Eurocodes consist of about 5077 pages in total [25], averaging 87.5 pages per part. That is without national annexes. The wind action standard is 149 pages long and contains 126 equations and 105 figures and tables. The national annex is 27 pages and contains an additional (mainly overriding) 49 equations and 16 tables and figures.

Generally, all this information could be straightforward to implement in software, the Eurocodes are strongly biased towards hand calculation. The lack of generalization, extensive use of figures and tables make implementation in computer code difficult. However, this issue is outside the scope of this report.

## 4.1 Assumptions

The Eurocodes make assumptions to ease calculation of wind loads. This thesis makes some assumptions of its own as well, introduced in this section.

### Eurocode assumptions

- Buildings are lower than 200 m. Bridges have spans less than 200 m.
- Fundamental value of basic wind velocity $v_{b,0}$ is based on zip-coded locations.
- Rectangles, triangles, and circles may define the building surface geometry.

### Thesis assumptions

- The structure is static (including $B^2 = 1.0,\ R^2 = 0.0$).

  Where:

  | | |
  |---|---|
  | $B^2$ | Background factor, allowing for the lack of full correlation of the pressure on the surface structure. |
  | $R^2$ | Resonance factor, allowing for turbulence in the resonance with the vibration mode. |

- The wind is coming from North, South, East, or West.
- The walls are piecewise planar.
- The loads from walls are transferred to the columns by default (the user may choose otherwise).
- The building is closed (no wall openings).

## 4.2 Calculation model

The general external wind pressure is obtained in four steps:

1. Read $v_{b,0}$ from a table, according to zip code provided by the user.

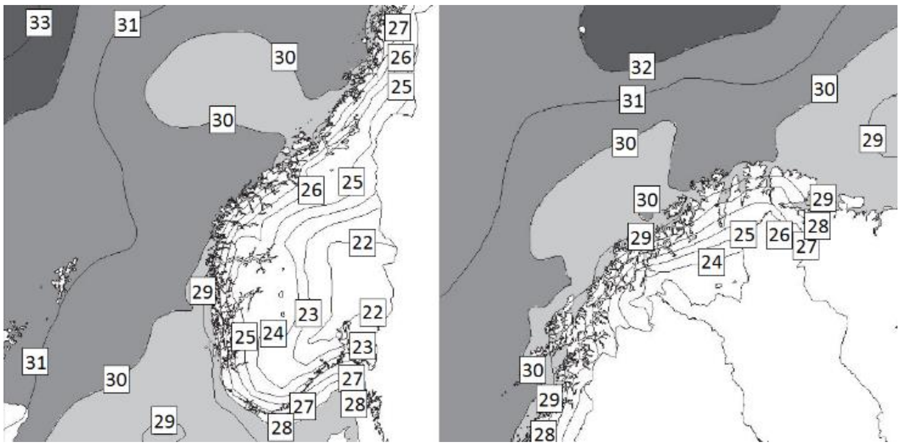


Figure 4-2: Basic wind velocity based on location in Norway for a 50 year return period and terrain category 2. [5]

2. Calculate mean wind velocity $v_m$ as a function of $v_{b,0}$, height above ground, and terrain category. The latter is a factor provided by the user that represents the amount of obstacles in the terrain. See Appendix B: Terrain Categories and Table B-1 regarding terrain categories.

$$v_m = c_r(z) * c_0(z) * v_b$$

Where:

| | |
|---|---|
| $c_r(z)$ | Roughness factor depending on terrain category and height above ground<br>$c_r(z) = k_r \ln\left(\frac{z}{z_0}\right)$ |
| $k_r$ | Terrain factor defined by the terrain category |
| $z_0$ | Roughness length defined by the terrain category |
| $c_0(z)$ | Orography factor ($= 1.0$ when ground incline $< 3\%$) |
| $v_b$ | Basic wind velocity ($= v_{b,0}$ unless special case, e.g. far out at sea or at extreme heights) |
| NB! | Height $z$ is set to $z_{min}$, a value defined by the terrain category, if $z < z_{min}$ |

3. Calculate peak velocity pressure $q_p$ as a function of mean wind velocity, air density and a peak velocity factor

$$q_p(z) = \frac{1}{2}\rho v_m^2(z) * [1 + 2 * k_p * I_v(z)]$$

Where:

| | |
|---|---|
| $\rho$ | Air density with default value $\rho = 1.25\frac{kg}{m^3}$ |
| $[1 + 2k_pI_v(z)]$ | Peak velocity factor with a magnitude usually around $1.5 - 3.5$. |

| | |
|---|---|
| $k_p$ | Peak factor defined as 3.5 in the Norwegian National Annex of the Eurocode [5] |
| $I_v(z)$ | Turbulence intensity that is proportional to $1/\ln(\frac{z}{z_0})$.For more information, see NA 4.4 of the Eurocode [5] |

4. Multiply the peak velocity pressure with a pressure coefficient to get the external pressure $w_e(z)$.

$$w_e(z) = q_p(z) * c_{pe}$$

Where:

| | |
|---|---|
| $c_{pe}$ | External pressure coefficient depending on the zone the wall is in, see below. |

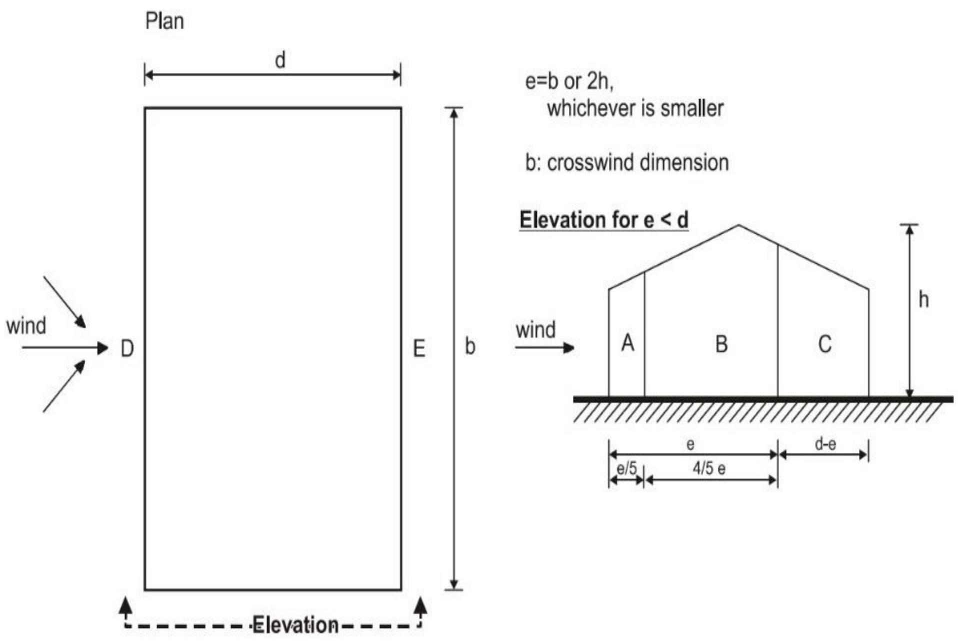


Figure 4-3: Wall zones for vertical walls

Table 4-1: Conservative values of external pressure coefficients for vertical walls of rectangular plan buildings [5]

| **Zone** | **A** | **B** | **C** | **D** | **E** |
|---|---|---|---|---|---|
| $c_{pe}$ | -1.4 | -1.1 | -0.5 | +1.0 | -0.7 |

# 5 Examples

To verify the results from the Aeolus, this section gives two examples where results obtained with ETABS are compared with an analytical solution. The first example calculates shear force, bending moments and deflection of a single wall exposed to wind load. The second example verifies zone allocation of a rectangular model.

## 5.1 Example 1: single wall

This section gives an example of wind calculation on a single wall with ETABS compared to an analytical solution. The task is to calculate the shear force, bending moments, and deflection of the wall.

### Technical information:

- Wall height: $H = 10\ m$
- Wall width: $b = 1\ m$
- Basic wind velocity: $v_b = 22\ m/s$
- Terrain category: Category 2:
    - $\boldsymbol{k_r} = 0.19$
    - $\boldsymbol{z_0} = 0.05\ m$
    - $\boldsymbol{z_{min}} = 4\ m$
- $E = 200.000\ MPa$
- $I = 63.72 * 10^7 mm^4$

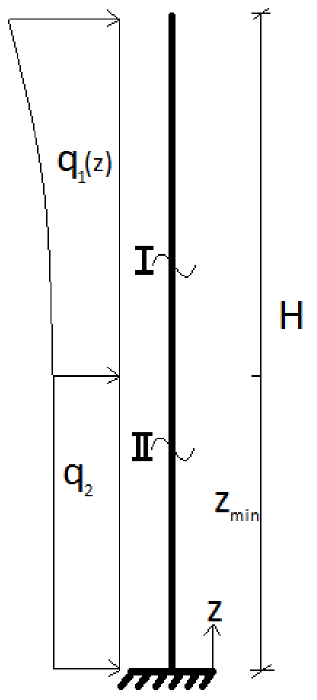


Figure 5-1: Example wall. Constant distributed wind load ($q_2$) for heights under $z_{min}$ and a varying distributed wind load ($q_1$) for heights over $z_{min}$. The roman numerals I and II indicate the breaks needed to compute the shear force and bending moments.

## 5.2 Results of Example 1

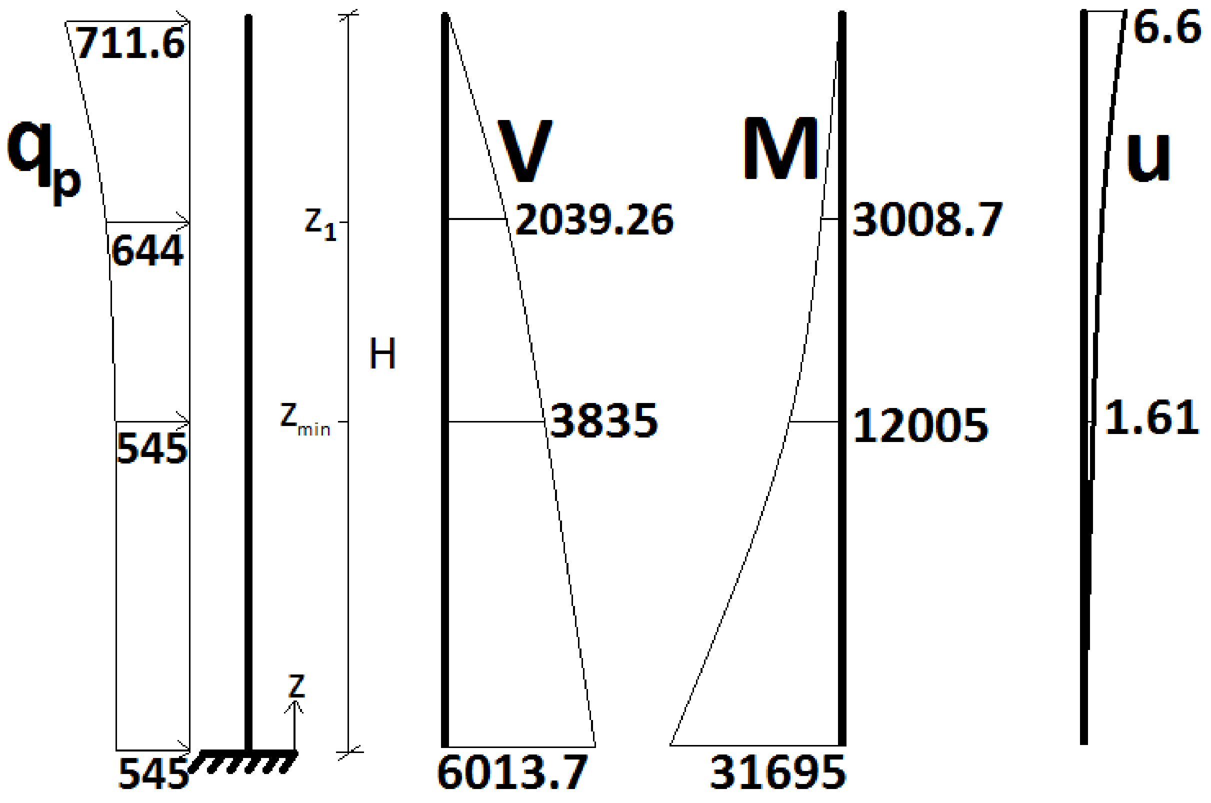


Figure 5-2: Results from the analytical solution. Units: $q_p$: *[*Pa*]*, V: *[*N*]*, M: *[*Nm*]*, u: *[*mm*]*. $z_1$ is halfway from $z_{min}$ to H. ($z_1 = z_{min} + \frac{H - z_{min}}{2}$)

Table 5-1: Comparing values from Analytical methods and from ETABS

| Value | Analytical results | ETABS and Aeolus results | Absolute difference | Relative difference |
|---|---|---|---|---|
| $q_p(z_{min})$ $[Pa]$ | 544.6 | 545 | 0.40 | 0.07 % |
| $q_p(z_1)$ $[Pa]$ | 544.6 | 545 | 0.40 | 0.07 % |
| $q_p(H)$ $[Pa]$ | 711.6 | 706 | 5.60 | 0.79 % |
| $M_b$ $[Nm]$ | 30715.9 | 31422.5 | 406.60 | 2.30 % |
| $M(z_{min})$ $[Nm]$ | 11015.6 | 11126.4 | 110.80 | 1.01 % |
| $M(z_1)$ $[Nm]$ | 3008.7 | 3084.8 | 16.10 | 0.54 % |
| $F_b$ $[N]$ | 6015.06 | 5973.9 | 41.16 | 0.68 % |
| $V(z_{min})$ $[N]$ | 3835.06 | 3795.1 | 39.96 | 1.04 % |
| $V(z_1)$ $[N]$ | 2039.26 | 2022.1 | 17.16 | 0.84 % |
| $u(z_{min})$ $[mm]$ | 1.59 | 1.60 | 0.01 | 0,63 % |
| $u(top)$ $[mm]$ | 6.29 | 6.40 | 0.11 | 1,75 % |

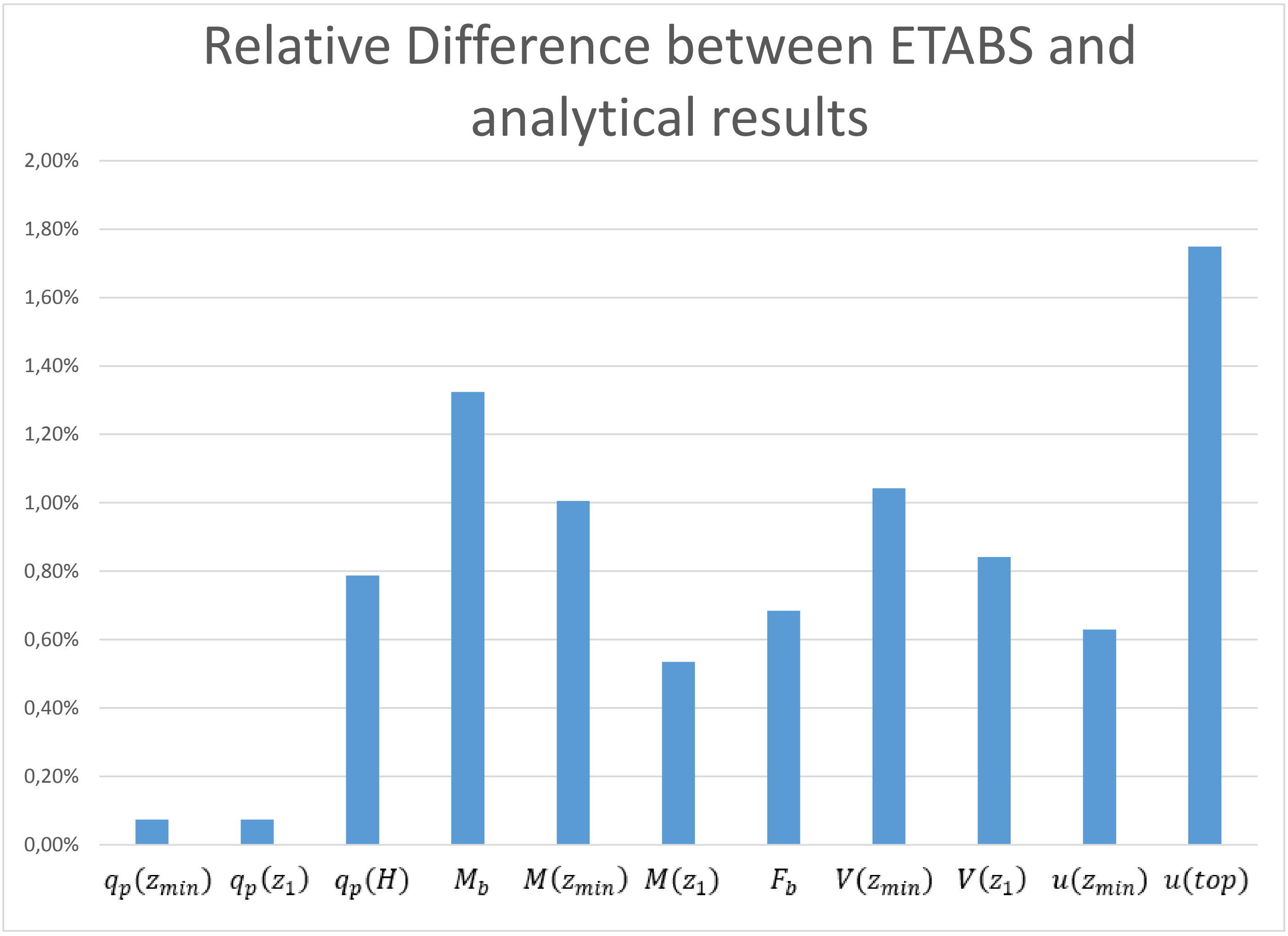


Figure 5-3: Relative difference chart

## Different results because of partially linearized load

The main reason for the difference in results from Aeolus and an analytical approach is that Aeolus linearizes the wind load. In Aeolus, the wind load is calculated for a given number of locations (Default: Every half meter, user may override.) and the distributed load is defined as a linear interpolation between the calculated values. This gives a good approximation to an analytical approach, and it saves a considerable amount of CPU time.

## 5.3 Analytical calculation for example 1

Wind force:

$$q_p(z) = \frac{1}{2}\rho k_r^2 v_b^2 \begin{cases} \ln^2\left(\frac{z}{z_0}\right)\left(1 + \frac{7}{\ln\left(\frac{z}{z_0}\right)}\right) & for\ z_{min} \le z \le H \\ \ln^2\left(\frac{z_{min}}{z_0}\right)\left(1 + \frac{7}{\ln\left(\frac{z_{min}}{z_0}\right)}\right) & for\ 0 \le z \le z_{min} \end{cases}$$

### Shear forces, bending moments, and deflection:

Starting from the top, defining $\tilde{z} = H - \tilde{z} \rightarrow q(z) = q(H - \tilde{z})$

**Section 1 ($0 \le \tilde{z} \le H - z_{min}$):**

$$V_1(\tilde{z}) \int_0^{\tilde{z}} q_1(H - \tilde{z})dz$$

$$= \frac{1}{2}\rho k_r^2 v_b^2 \left\{\left[-2(H-\tilde{z}) - 7\tilde{z} - 5(H-\tilde{z})\ln\left(\frac{H-\tilde{z}}{z_0}\right) - (H-\tilde{z})\ln^2\left(\frac{H-\tilde{z}}{z_0}\right)\right] - H\left[-2 - 5\ln\left(\frac{H}{z_0}\right) - \ln^2\left(\frac{H}{z_0}\right)\right]\right\}$$

$$M_1(\tilde{z}) = \int_0^{\tilde{z}} q_1(H - \tilde{z})\tilde{z}dz$$

$$= \frac{1}{4}\rho k_r^2 v_b^2 \left\{\left[(\tilde{z}^2 - H^2)\ln^2\left(\frac{H-\tilde{z}}{z_0}\right) - 2(2H^2 + H\tilde{z} - 3\tilde{z}^2)\ln\left(\frac{H-\tilde{z}}{z_0}\right) + 7H^2 - 4H\tilde{z} - 3\tilde{z}^2\right] - \left[-H^2\ln^2\left(\frac{H}{z_0}\right) - 4H^2\ln\left(\frac{H}{z_0}\right) + 7H^2\right]\right\}$$

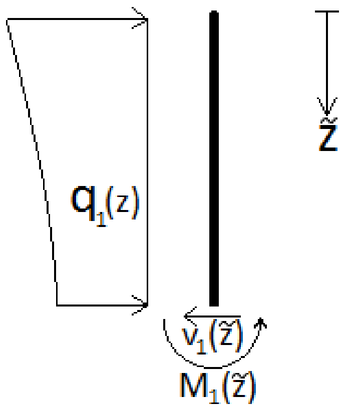


Figure 5-5: FBD of Section I ($0 \le \tilde{z} \le H - z_{min}$)

**Section 2 ($H - z_{min} \le \tilde{z} \le H$)**

$$V_2(\tilde{z}) = V_1(H - \tilde{z}) + q_2(\tilde{z} - (H - z_{min}))$$

$$M_2(\tilde{z}) = M_1(H - \tilde{z}) + V_1(H - \tilde{z})(\tilde{z} - (H - z_{min})) + \frac{1}{2}q_2(\tilde{z} - (H - z_{min}))^2$$

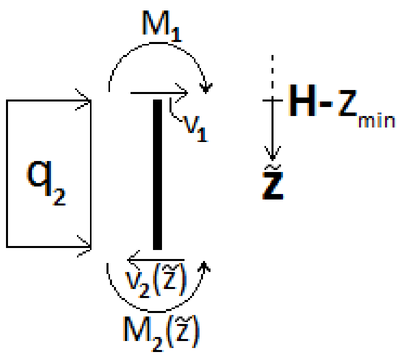


Figure 5-4: FBD of Section II ($H - z_{min} \le \tilde{z} \le H$)

**Deflection**:

$$u(z) = \frac{1}{EI}\int\int M\,dz^2$$

## 5.4 Example 2: Box

This example explains and verifies the zone allocation and the external pressure coefficients provided by Aeolus. For more information about construction zones, see Figure 4-3 and Table 4-1.

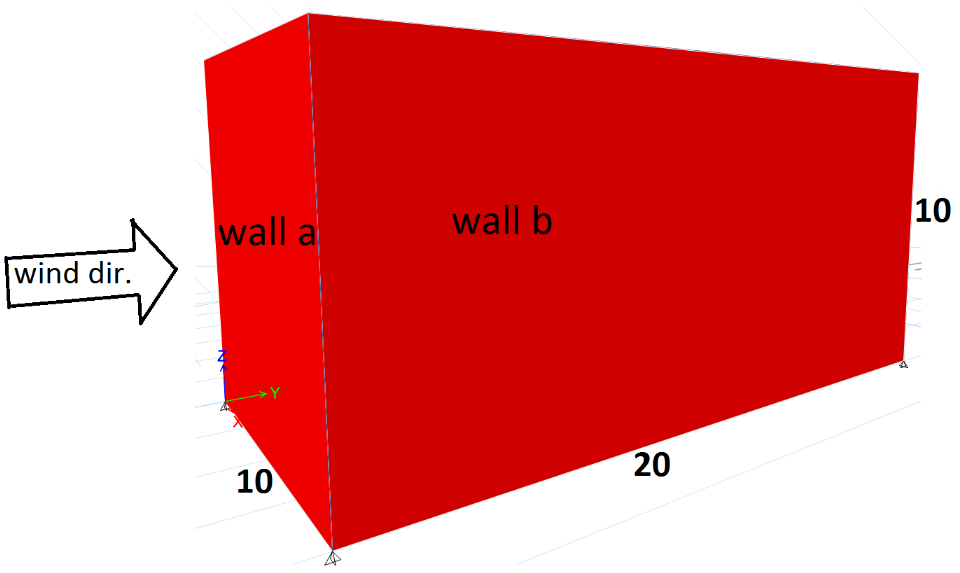


Figure 5-6: A box, ten meter high, ten meter wide, and twenty meters deep. The wind is coming straight towards wall a.

The wall parallel to the wind direction (wall b) is twenty meters long and ten meter high. It should be divided into three vertical zones: A, B, and C, according to the following rules:

- Variables:
  - $e = \min \begin{cases} \text{width of the construction} \\ 2 * \text{height of the construction} \end{cases} = min \begin{cases} 10\ m \\ 20\ m \end{cases} = 10\ m$
  - $d = \text{depth of the construction} = 20\ m$
- Zone A: The part of the wall from the start until $e/5$ in the wind direction.
- Zone B: The part of the wall from $e/5$ until $e$ in the wind direction.
- Zone C: The part of the wall from $d - e$ until the end of the wall in the wind direction.

This gives the following zones:

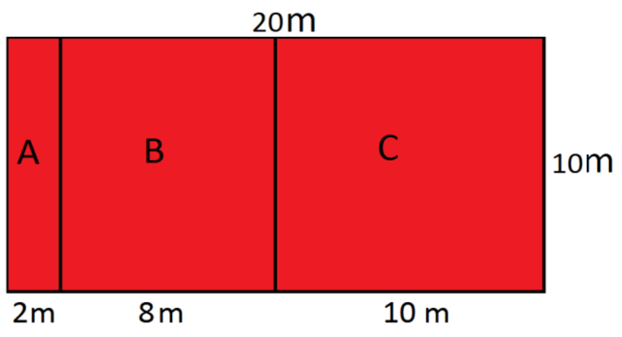


Figure 5-7: Zones of the wall parallel to the wind direction (wall b in Figure 5-6).

The basic wind velocity is 22 $m/s$ and the terrain category is Category 0. This gives the following peak wind velocity (below $\boldsymbol{z_{min}} = 4m$): $\boldsymbol{q_p(z_{min})} = 679.89\ Pa$. For information about how this value is calculated, see Chapter 4.2. Calculation model, and the previous example. Aeolus presents the following results:

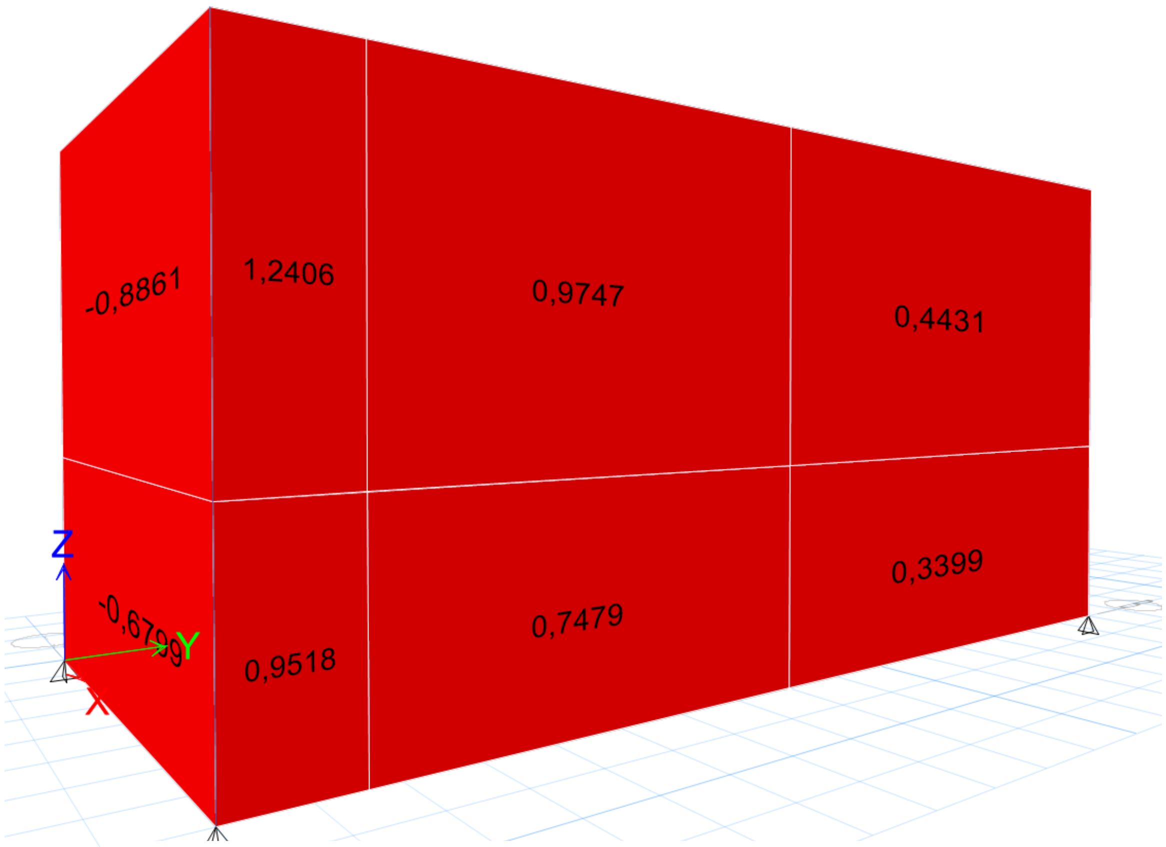


Figure 5-8: Wind loads applied to the ETABS model by Aeolus. The wall parallel to the wind direction is divided into three vertical zones, and two horizontal zones. The horizontal zone split is at z = $z_{min}$. The vertical zone splits is at y = 2 m, and y = 10 m. Unis are in kPa.

Table 5-2: Zones and loads of walls below $z_{min}$ of the example model

| **Zone** | **External Pressure Coefficient, $c_{pe}$** | **pressure calculated analytically, $q_p(z_{min})$** | **Load $w_e = q_p(z_{min}) * c_{pe}$** | **Wind load from Aeolus** |
|---|---|---|---|---|
| D | 1.0 | 679.89 $Pa$ | 679.89 $Pa$ | -679.9 $Pa$ |
| A | -1.4 | 679.89 $Pa$ | $-$ 951.846 $Pa$ | 951.8 $Pa$ |
| B | -1.1 | 679.89 $Pa$ | $-$ 747.879 $Pa$ | 749.9 $Pa$ |
| C | -0.5 | 679.89 $Pa$ | $-$ 339.945 $Pa$ | 339.9 $Pa$ |

As seen in the table above, the results from Aeolus and from and analytical solution corresponds (with different sign due to different conventions) and thus verifies the zone allocation for this type of structures.

# 6 Software Development

## 6.1 Implementation

The main challenge regarding implementing Aeolus using the API is that one cannot access objects in the model. This section describes an approach to overcome this challenge.

The crucial part of the implementation was to establish an object model. To do this, new objects where created to mimic the unobtainable model objects from ETABS. Information about all objects from the ETABS model is available, through the API, to create a well-defined object model.

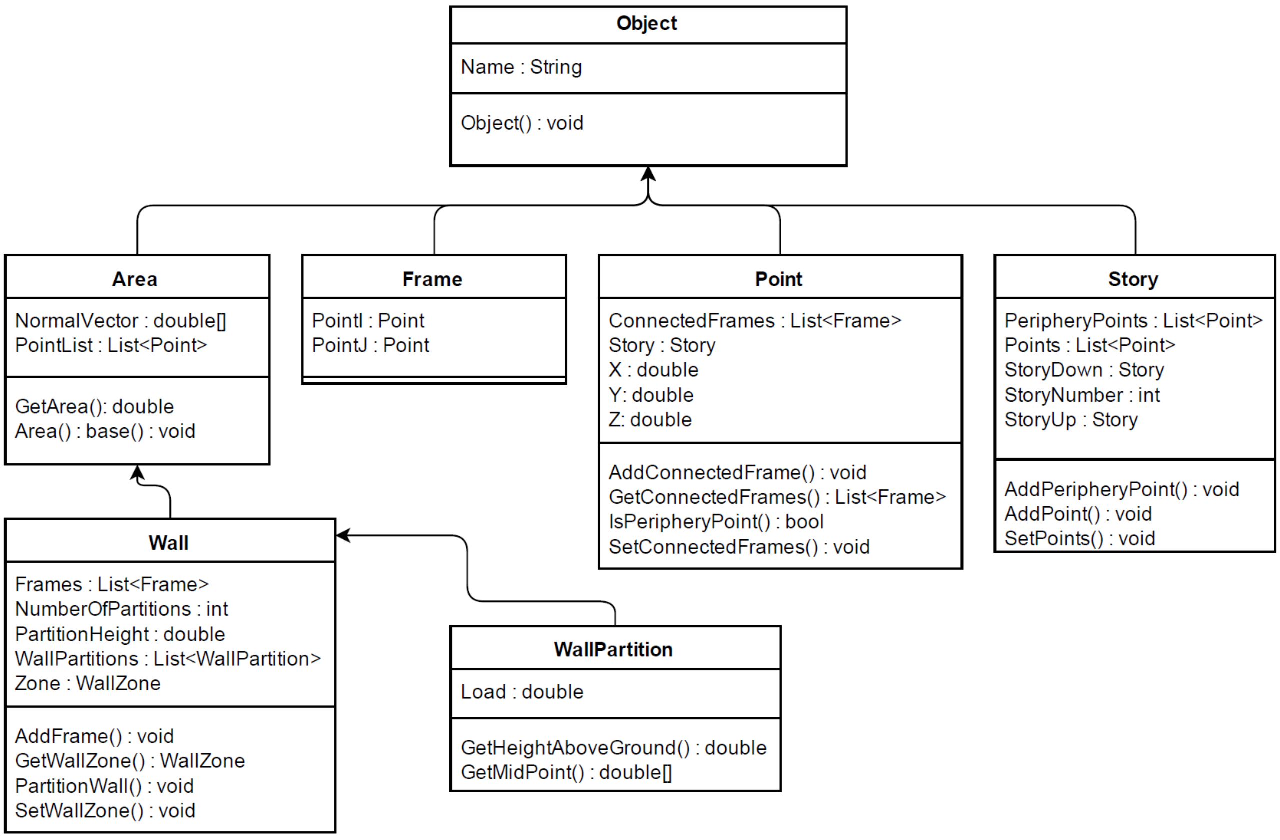


Figure 6-1: Class diagram for an example object model created in the code of Aeolus

To enable communication between Aeolus and ETABS, the Aeolus project in Microsoft Visual Studio [15] imported a DLL file called “ETABS2013.dll”. This file, distributed by CSi may provide a two-way real-time direct link between third party extensions and ETABS.

## Architecture

The architecture of the Aeolus is based on the pattern "Model-View-Controller (MVC)" [6], which separates the user interface from the data and the data controller.

- The *View* requests information from the *Model* to generate an output representation to the user.
- The *Model* contains all the data from an ETABS model and notifies its associated *View* when there has been a change in its state. This notification allows the *View* to produce updated output.
- The *Controller* moves and manipulates data from the *Model* and sends commands to update the *Model*'s state.

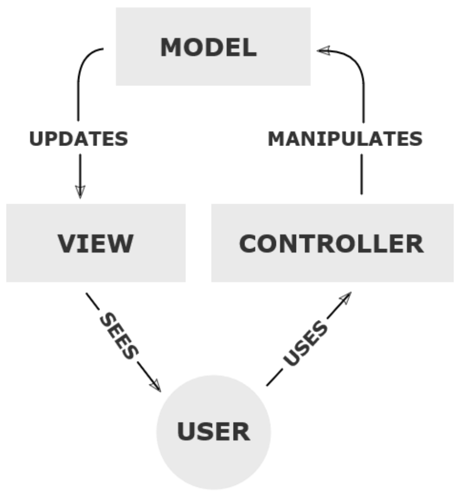


Figure 6-2: Typical collaboration of MVC components [6]

The *Controller* component of the architecture contains three classes: *Eurocode*, *Recognizer*, and *CSiInstance*. A *CSiInstance* object is instantiated when the user opens a model and starts communicating with ETABS. It contains and controls both the *Eurocode* class and the *Recognizer* class. See Appendix C: Class Overview for information about the implemented classes.

When the user sends a command to recognize geometry, a *Recognizer* object is instantiated and immediately executes the methods to detect and create *Point* objects, *Frame* objects, *Wall* objects etc. based on the ETABS model. These model objects form the basis for the *Model* component of the architecture.

When the user sends a command to assign wind loads, a *Eurocode* object is instantiated within the *CSiInstance* and uses the information *Recognizer* got from the *Model* component to calculate wind loads according to the regulations. The calculated wind loads are assigned to the ETABS model.

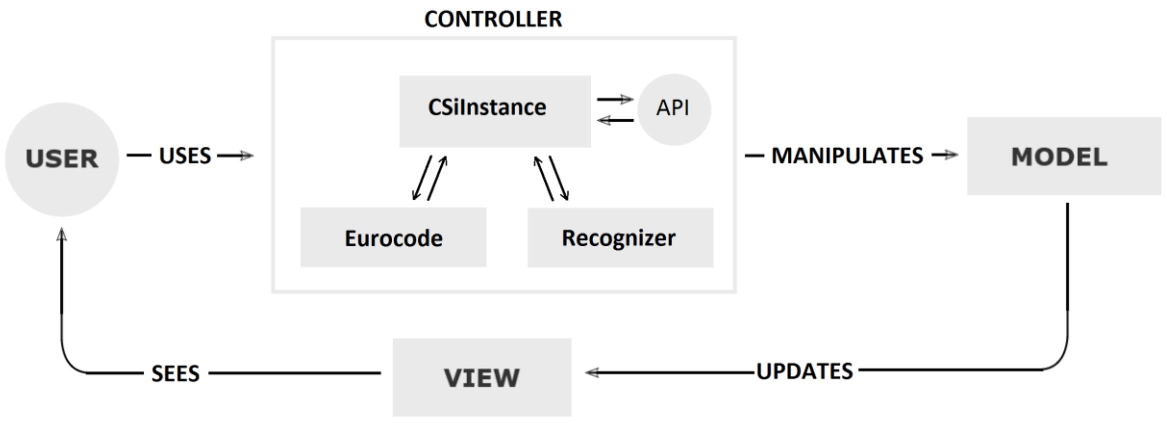


Figure 6-3: Detailed view of the Controller-component of the MVC-pattern.

The *Recognizer* object contains the logic for geometry recognition, but does not communicate directly with an ETABS model. The *CSiInstance* class sends and requests information to and from the ETABS model through the API. This information is passed on to the *Recognizer* for geometry recognition and to the *Eurocode* for load calculation. This architecture is highly maintainable and enables efficient unit testing. See Chapter: 6.5. Unit Testing for more details.

## 6.2 Extension vs Plug-in

The current version of ETABS supports use of extensions, but not plug-ins, but what it the difference? Both extensions and plug-ins are computer programs that depend on an existing software application. Both are intended to expand the usability and/or extend the capability of the application [26].

### Examples of extensions and plug-ins

Common examples of plug-ins are the well-known internet browser plug-ins such as Adobe Flash Player [27], Apple QuickTime Player [28], and the Java plug-in [29]. Historically, plug-ins are used as bridges between a main application and a third party software, like the link between a web browser and Adobe Flash Player, enabling the user to play flash videos etc. within the browser.

Figure 6-4: To be able to play the YouTube [50] video "Rick Astley – Never Gonna Give You Up", the internet browser needs a flash player plug-in.

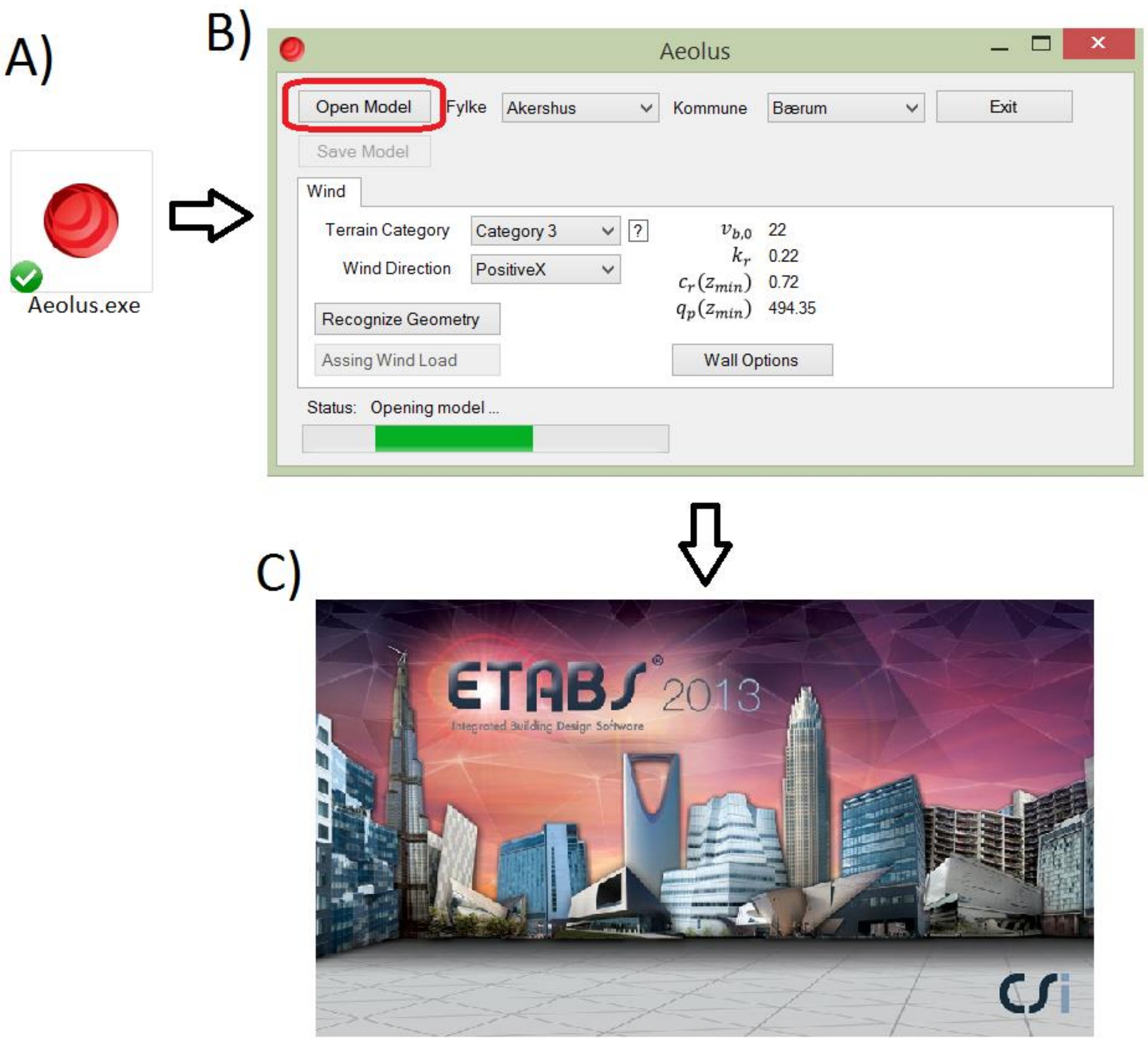
Common examples of extensions are browser toolbars [30] and modifications to computer games, designed to alter the content of the game. Among the well-known computer games that started as an extension are Counter Strike [31], Defense of the Ancients (DotA) [32], and Team Fortress [33]. Extensions are made for the main application, and are only and intended to expand that application.

### Difference between an extension and a plug-in to ETABS

What separates an extension from a plug-in, *in the ETABS convention*, is primarily when it is loaded. An extension to ETABS is loaded before ETABS starts, and only one extension may run at a time. A plug-in may be loaded at any time during the execution ETABS, and several plug-ins may run simultaneously. The ETABS API does not currently support plug-ins, but it will in a future release.

## Extension of ETABS

ETABS must by executed *through* an extension to use it. This means that extensions for ETABS are stand-alone executable programs with the ability to start an instance of ETABS from the extension itself. For more information about the workflow of the wind load extension, see Section: Workflow.



Fylke: Akershus
Kommune: Bærum

Wind

Terrain Category: Category 3

Wind Direction: PositiveX

| | |
|---|---|
| $v_{b,0}$ | 22 |
| $k_r$ | 0.22 |
| $c_r(z_{min})$ | 0.72 |
| $q_p(z_{min})$ | 494.35 |

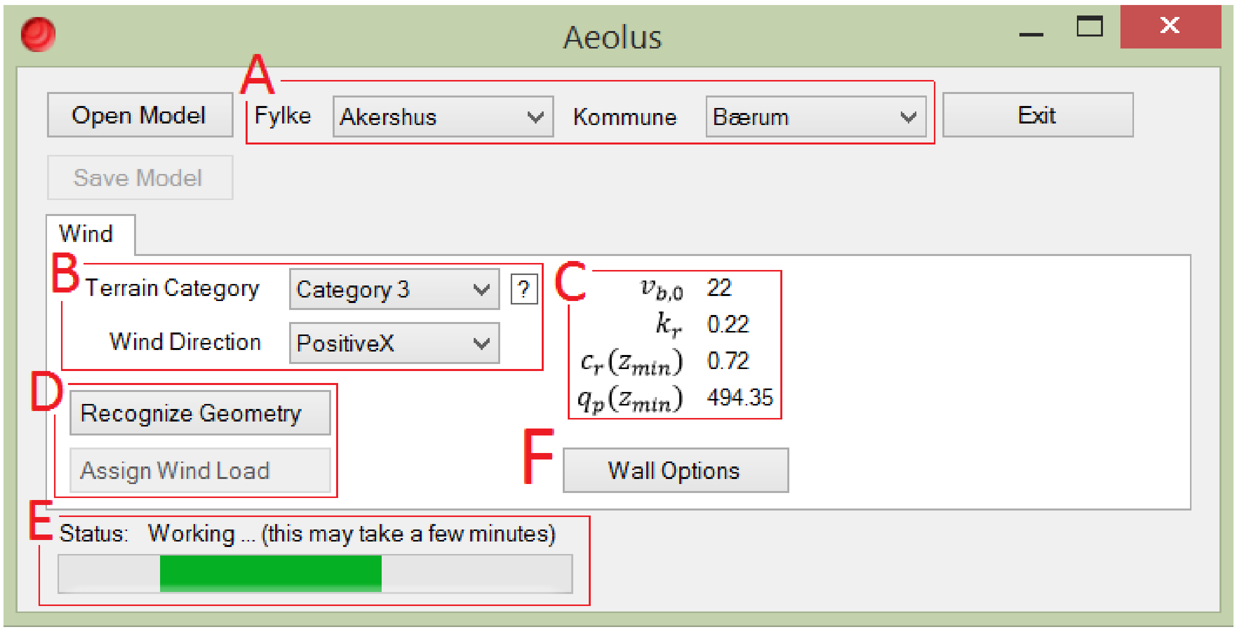
Status: Opening model ...

Figure 6-5: Aeolus is loaded as a stand-alone program (A). From within Aeolus, the user may load an instance of ETABS (B) by selecting “Open Model”. The instance of ETABS will the loaded (C) with the extension Aeolus.

## 6.3 **User Manual**

The features of Aeolus are described in this section. A more detailed user manual for ETABS in general is available online [34].

The purpose of the main view of Aeolus is to be intuitive and as simple as possible for the user to use. It has the user interface [35] of a general form [36] from Microsoft Windows.



Fylke: Akershus

Kommune: Bærum

Terrain Category: Category 3

Wind Direction: PositiveX

| | |
|---|---|
| $v_{b,0}$ | 22 |
| $k_r$ | 0.22 |
| $c_r(z_{min})$ | 0.72 |
| $q_p(z_{min})$ | 494.35 |

Status: Working ... (this may take a few minutes)

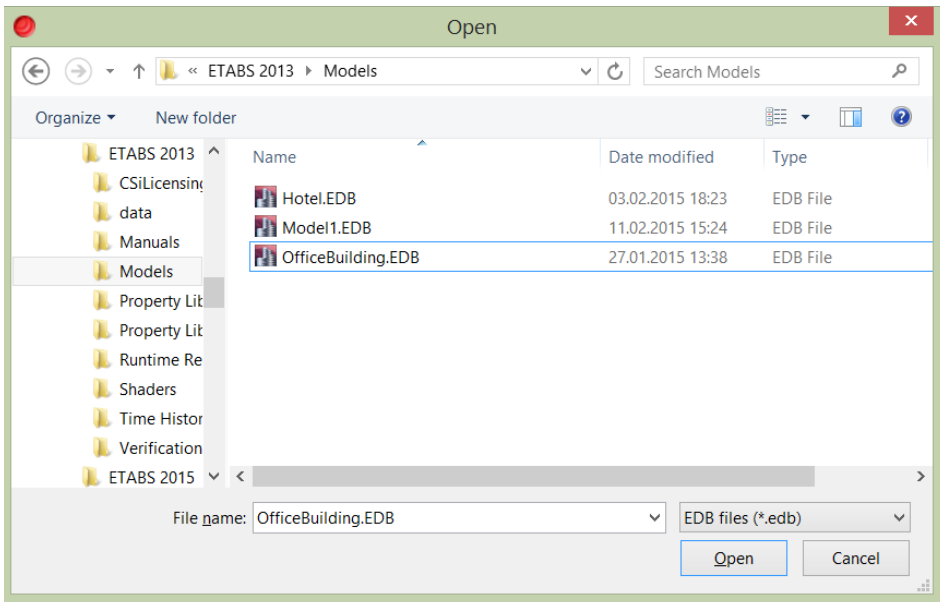
Figure 6-6: The view is composed of five parts: (A) Location, (B) Environment, (C) Wind load information, (D) User control and communication with ETABS, (E) Status area that gives feedback to the user on progress, and (F) opens a new dialog where the user may select options for the walls.

## Workflow

Since Aeolus is an extension, it is started as a stand-alone executable program. The user workflow is described in the following steps

### 1. Open Model

The communication with ETABS begins when the user press the "Open Model" button, which opens an instance of ETABS where the user is asked to choose a model file to open. This step in the workflow is not required to be the first step, but may be executed at any time during the usage Aeolus.



| Name | Date modified | Type |
|---|---|---|
| Hotel.EDB | 03.02.2015 18:23 | EDB File |
| Model1.EDB | 11.02.2015 15:24 | EDB File |
| OfficeBuilding.EDB | 27.01.2015 13:38 | EDB File |

File name: OfficeBuilding.EDB
EDB files (*.edb)

Figure 6-7: An open file dialog is presented to the user when communication between Aeolus and ETABS is established. The user chooses a model file to open. Model files have the file extension .EDB

### 2. Select Location

From a dropdown menu, the user selects county and municipality, to provide the program with a Zip Code. This is used to read a fundamental value of the basic wind velocity, $v_{b,0}$, from table NA.4(901.1) from the Eurocode [5]. For more information about the Eurocode and its implementation, see Chapter: 4.2. Calculation model. If the user wants a special case (e.g. not location specific construction, higher-than-normal wind velocity, etc.), the basic wind velocity may be overridden.

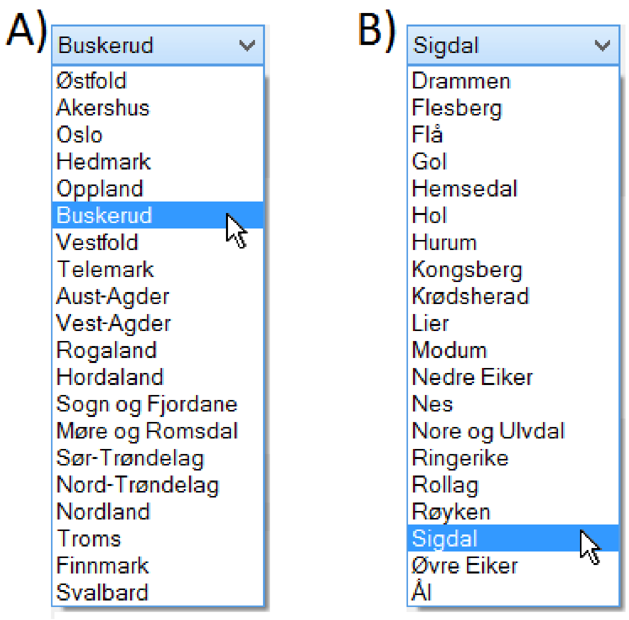


Figure 6-8: Example of selecting a location. At (A) the user chooses a county, and at (B) the user chooses a municipality within the selected county. This sends a zip-code to the program that is used to read the basic wind velocity from a table.

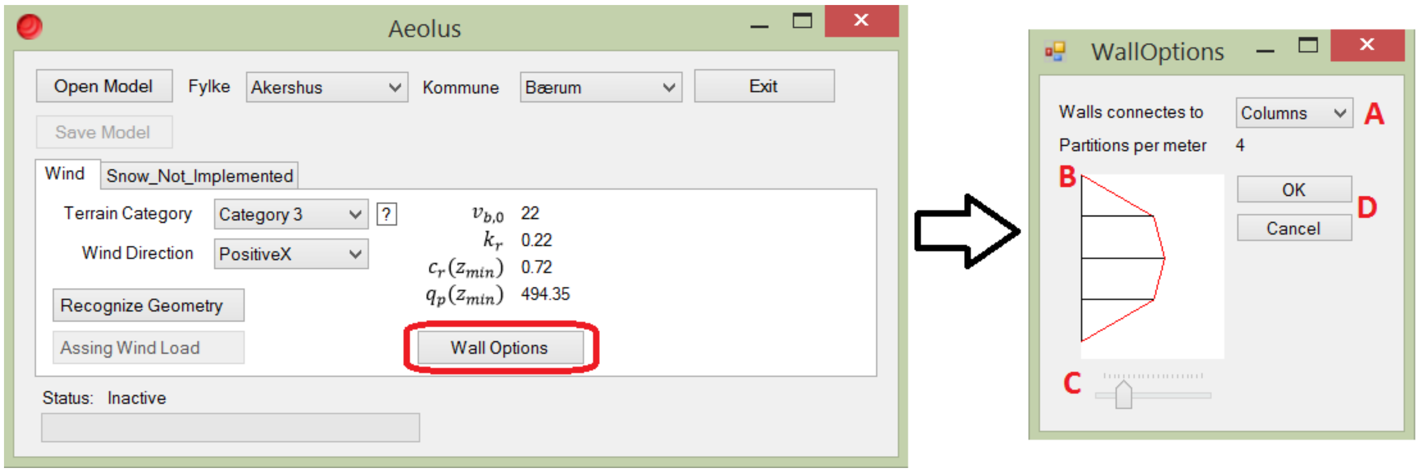
## 3. Select environmental factors

There are two environmental factors the user need to provide:

1. Terrain Category. A factor representing the terrain and possible obstacles for the wind. See Appendix B: Terrain Categories and Table B-1 for more information regarding terrain categories. The default terrain category is Category 0.
2. Wind direction, which is restricted to four directions; North, East, South, and West. The default wind direction is West, and the building is located accordingly in the first quadrant with the positive x towards east. Further releases of Aeolus may include wind direction from any directions, and determination of the “worst” wind angle, i.e. the wind angle causing most harm to the building.

## 4. Select Wall Options

The user may choose options for that structural elements the walls are connected to (columns, beams, or nodes), and how many partitions the walls should be divided into.



Fylke: Akershus
Kommune: Bærum
Terrain Category: Category 3
Wind Direction: PositiveX

| | |
|---|---|
| $v_{b,0}$ | 22 |
| $k_r$ | 0.22 |
| $c_r(z_{min})$ | 0.72 |
| $q_p(z_{min})$ | 494.35 |

Status: Inactive

Walls connectes to: Columns
Partitions per meter: 4

Figure 6-9: Wall options dialog. The user may choose if the walls are connected to columns (default), beams or nodes (A). The number of partitions is selected using a slider (C). An example moment diagram is displayed, reflected by the number of partitions. The user may choose to accept or to cancel the options chosen (D).

## 5. Recognize geometry

The process of geometry recognition is the part of the workflow that consumes most CPU time. The runtime for the geometry recognition is proportional to $O(m * n^2)$, where $m$ is the number of frames (beams, columns, or braces) in the model and $n$ is the number of nodes in the model [37]. However, even for a large model, the time taken for the program to recognize the geometry should be less than five minutes. For details regarding geometry recognition, see Chapter: 3. Geometry Recognition.

After the geometry recognition, the user may override what is presented. If, for instance, a wall element is not where it is supposed to be, the user may delete said wall element and add a correct one. This is possible due to a property, *"WindWall"*, set to all wall elements created by Aeolus. The wind load assigner uses this property to define a set of walls to be included in the load calculation. When a user add a new wall, the user may set the "*WindWall*" property for said wall, thus including it in the set of wall elements.

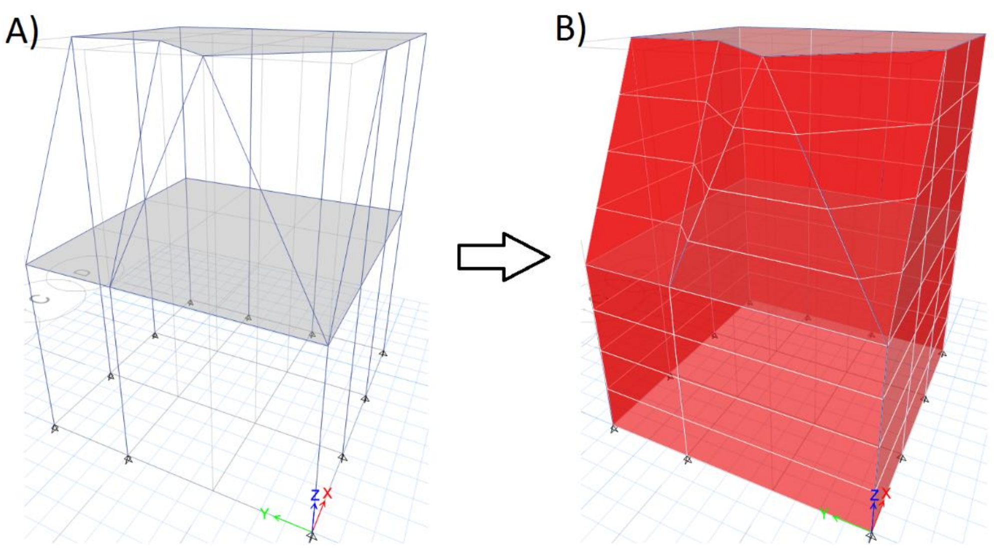


Figure 6-10: Geometry Recognition completed. At (A) the model have no walls, at (B) walls have been located, added to the ETABS model, and partitioned to a reasonable number of horizontal strips

## 6. Assign Wind Loads

This is a straightforward procedure where the load assigner calculates wind loads according to regulations and assigns it to the model elements. For more information about the regulations, see Chapter: 4. Eurocode NS-EN 1991-1-4:2005+NA:2009

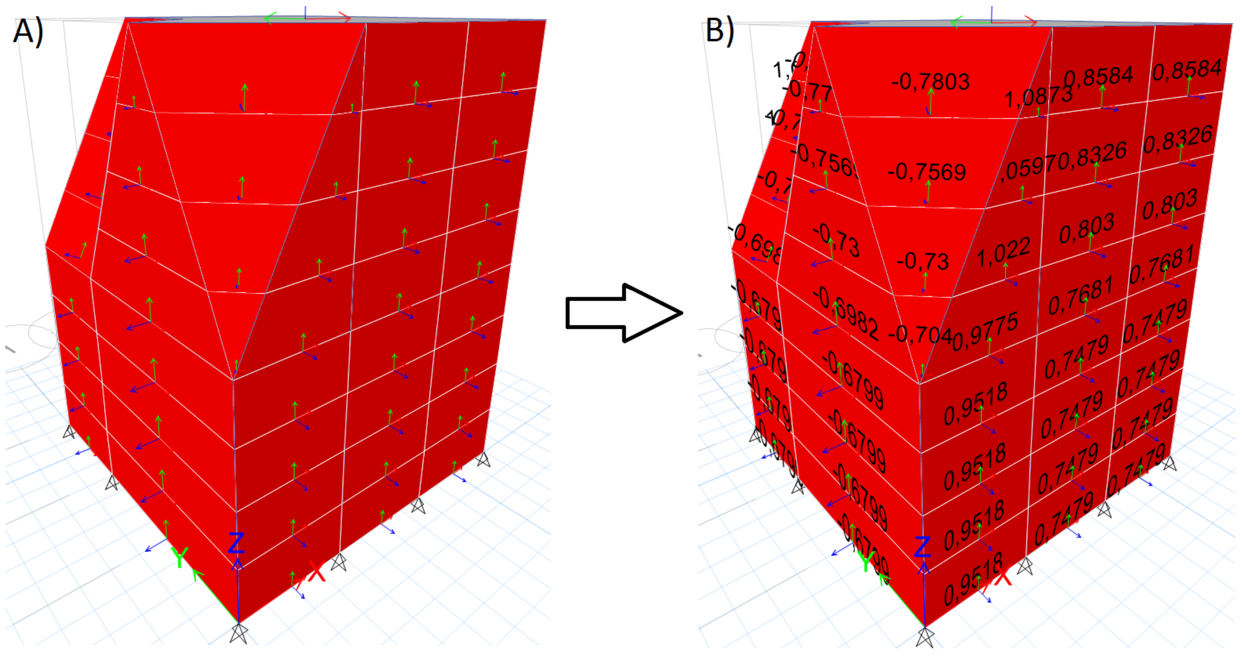
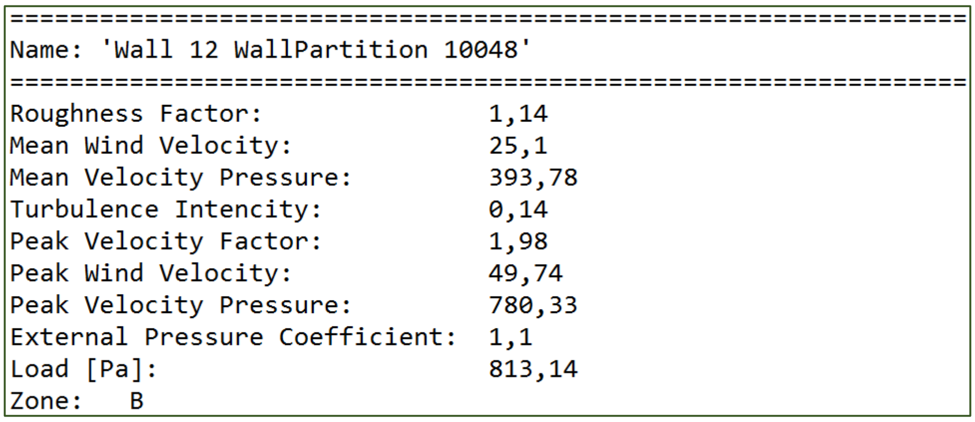


Figure 6-11: Load assigned to all wall elements according to regulations. A model without loads (A), and a model with wind loads assigned (B) (values are in kPa). The local axes on each wall element contains a blue arrow, which is the walls normal vector that points in the direction of the load (by convention, this is outwards). Here, $v_{b,0} = 22$ and terrain category is 0.

The load assigner writes a log containing information about every calculation made. The user is able to trace back all variables used for every wall element to verify the results. If the user disagrees with the results, it may be manually overridden in ETABS.



```
================================================================
Name: 'Wall 12 WallPartition 10048'
================================================================
Roughness Factor:                1,14
Mean Wind Velocity:              25,1
Mean Velocity Pressure:          393,78
Turbulence Intencity:            0,14
Peak Velocity Factor:            1,98
Peak Wind Velocity:              49,74
Peak Velocity Pressure:          780,33
External Pressure Coefficient:   1,1
Load [Pa]:                       813,14
Zone:   B
```

Figure 6-12: Snippet from a log showing values for variables used in calculating the load on a wall element

## 7. Save model

The user may save the model with wind loads assigned for further analysis and design. It is also possible to remove all changes to the model and save it as it was.

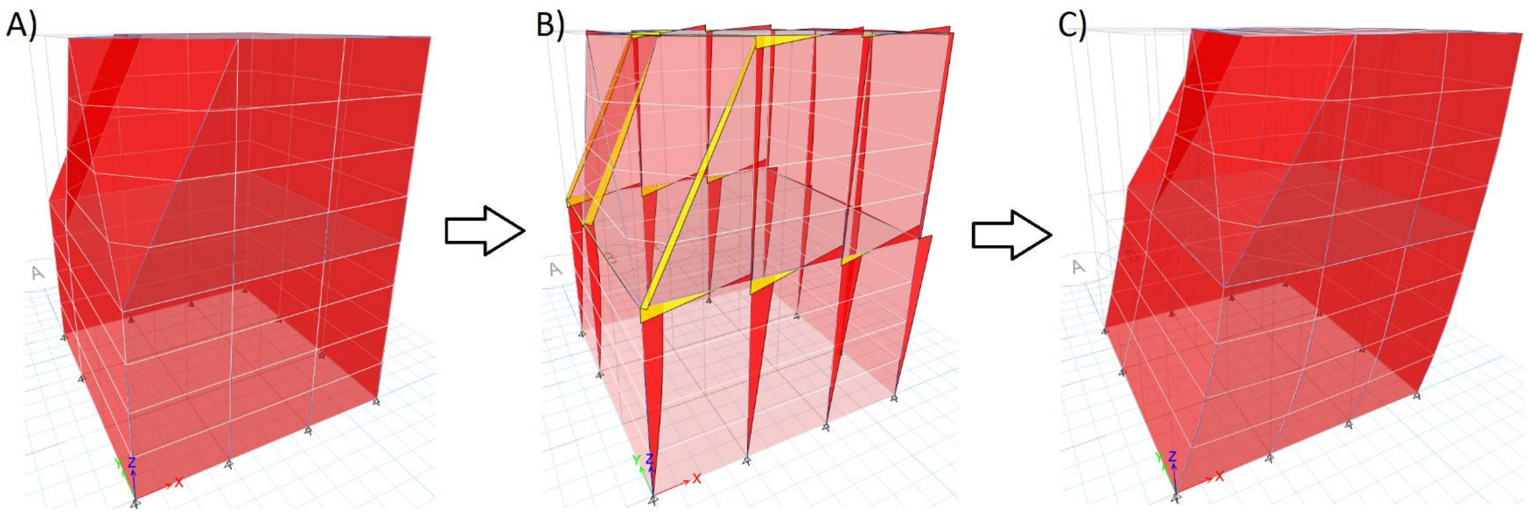


Figure 6-13: Analysis of a model affected by wind loads. At (A), the model is displayed without deformation. At (B), the model is displayed with moment diagram in the X-Z-Plane. At (C), the model is displayed with exaggerated deformations

## 6.4 Code Complexity

It is crucial to write understandable, maintainable, reusable, and robust code. With this in mind, Aeolus was written using Test Driven Development (TDD) and the focus was to make a program that other developers might develop further without struggling to understand the code. For more information about TDD, see Chapter: 6.5 Unit Testing.

Aeolus was written using an Object-Oriented paradigm [38] to reflect the structure of systems “in the real world”. This means the coded objects interacts with each other and simulate objects in the physical world.

### Code metrics

Code metrics [39] is a set of software measures that provide developers better insight into the code they are developing. Microsoft has developed the software measures for Visual Studio.

1. **Maintainability Index**

This index value between 0 and 100 represents the relative ease of maintaining the code. Higher values mean better maintainability. Aeolus has a Maintainability Index of 90, which is considered *highly maintainable*.

2. **Cyclomatic Complexity**

This value represents the number of different code paths in the flow of the program. Lower values mean better code, in respect to comprehensibility and code readability. Aeolus has an average Cyclomatic Complexity for each method of 2.3, which are considered simple and easy to understand [40]. However, there are certain methods in the code with a Cyclomatic Complexity of >20, meaning they might need an effort to understand. These methods are commented in the code to help describe their functions.

3. **Class Coupling**

A number representing the amount of code coupling to unique classes through parameters, local variables, return types, method calls, generic instantiations, and fields. Good software design dictates that types and methods should have high cohesion and low coupling. High coupling indicates a design that is difficult to reuse and maintain because of its many interdependencies to other types.

An analogy to class coupling in the physical world could be a cell phone. If the battery is soldered to the phone and unable to replace, the phone has tight coupling and thereby expensive and difficult to maintain by replacing a defect battery. A loosely coupled phone would allow effortlessly changing the phone battery.

Aeolus is written such that every class may be replaced and edited independently without causing errors on other classes, by extensive use of interfaces [41] and “The Law of Demeter” [42]. The latter is an object-oriented software design principle, which can be summarized as: A method of an object may only call methods of the object itself, an argument of the method, any object created within the method, or any direct property/field of the object [43].

The motivation for a low class coupling index is better maintainability. Just like a phone battery can be replaced on a loosely coupled cell phone, a class in a loosely coupled code may be edited or even removed without causing compiler errors at other classes. This simplifies maintaining the code significantly because a developer may maintain one part of the code as an independent unit.

## 4. Lines of Code

Aeolus consist of about 2600 lines of code, which does not say anything about how much time spend writing the code or how good the program is. It is simply a value of how many lines of code there are. However, the number indicates that the extension is not written overnight. Six hundred of the 2600 lines are tests that make sure the other 2000 lines do what they are supposed to do. A net value of code lines produced per hour spent on developing the extension is estimated to about $6.5\,\frac{code\ lines}{hour}$, corresponding to 400 hours of coding.

## 6.5 **Unit Testing**

Aeolus is implemented using Test Driven Development (TDD) [4], which is a software development process that relies on the repetition of a very short development cycle:

1. Write an (initially failing) automated unit test case that defines a desired improvement or a new function.
2. Produce the minimum amount of code to pass that test
3. Refactor the new code to acceptable standards

The reason why developers write the unit tests before writing the code, is to force a focus on the requirements of the program before writing the code. To write a test, the developer must clearly understand the feature's specification and requirements. A unit test is defined by three elements: What to test, what to test with, and what to expect.

Example of a test is shown in Figure 6-14. The example test demonstrates testing of the method *GetAngle(Frame f1, Frame f2)* in the class *GeometryLogic*. The methods function is to return the counter-clockwise angle between two frames (f1 and f2) when traversing from f1 to f2. The test uses two orthogonal frames as input to the method, where f1 points east, and f2 points north. The expected result from *GetAngle()* is $3\pi/2$, which is checked with the method *Assert.AreEqual()*. The *Assert.AreEqual()*-method also takes an acceptable delta as input. The test passes if the two values $correctAngle$ and $actualAngle$ are closer to each other than the value *acceptableDelta*.

### **"Arrange, Act, Assert"**

The design pattern used for unit testing in this thesis is the commonly used "Arrange, Act, Assert"-pattern [44]. The purpose of the pattern is to make unit test easy to read, easy to understand, and thereby easy to maintain.

Each test is divided into three functional sections [45]:

1. Arrange all necessary preconditions and inputs.
2. Act on the object or method under test.
3. Assert that the expected result have occurred.

```
[TestMethod]
public void GetAngle_OrthogonalFrames_3TimesHalfPi()
{
    //Arrange
    /*
    *          (j3)
    *            |
    *          f2
    *            |
    *(j1)--f1--(j2)
    */
    _testModel.Points = new List<Point>()
    {
        new Point("j1", 0, 0, 0, 0),
        new Point("j2", 1, 0, 0, 0),
        new Point("j3", 1, 1, 0, 0)
    };
    _testModel.Frames = new List<Frame>()
    {
        new Frame
        (
            "f1",
            _testModel.Points[0],
            _testModel.Points[1]
        ),
        new Frame
        (
            "f2",
            _testModel.Points[1],
            _testModel.Points[2]
        )
    };
    const double correctAngle = 3*Math.PI/2;
    const double acceptableDelta = 0.0001;

    //Act
    var actualAngle = GeometryLogic.GetAngle
    (
        _testModel.Frames[0],
        _testModel.Frames[1]
    );

    //Assert
    Assert.AreEqual(correctAngle, actualAngle, acceptableDelta);
}
```

Figure 6-14: Code example of a unit test. The name of the test is composed of the three components of a unit test *NameOfMehtod_InputToMethod_ExpectedOutput()*.

## Advantages of TDD

One of the main advantages of TDD is efficient debugging. If a developer changes anything in the code at any time that causes an error, the tests will provide information about where and what the error is. The error feedback loop is short. This enables efficient debugging, which is important for code maintainability. It also tells the developer “yes, your code does indeed work”. It will confirm that what you have coded actually does what it is supposed to do, and thus help developers sleep at night.

Another advantage that may be the most important one is that it forces the developer to think before he/she writes the code. There has to be a meaning behind the work, and the specifications have to be in place *before* writing the code.

TDD is a simple method, but extremely powerful. It encourages developers in the start-up face of a large project (or when new functionality to an existing project should be implemented) to divide the problems and challenges into smaller parts, which can be solved one by one. This is why it is called unit testing. The purpose is to build a project by many small units that may be tested one by one.

## Disadvantages of TDD

One may argue that the time invested in writing tests is a downside of the development method. In a small project with few people, test writing may not be worth the effort, but in a complex project involving dozens of coders, time will probably be saved by writing tests. In this case, it was a personal choice to use TDD, because other developers may develop the project further, and they will benefit from the tests.

Another aspect of TDD is that it hampers design change of the project during the development, making the developing process less agile. The tests may have to be rewritten if the requirements or the architecture of the project changes. However, with proper planning of the project, this should not be an issue.

# 7 Discussion

This chapter discusses different aspects of Aeolus and the development process and gives some concluding remarks. It also gives recommendations for further work.

## Why an Extension?

Section 6.2. Extension vs Plug-in states that ETABS has to be launched *through* Aeolus and only one extension to ETABS may run at a time. It would be preferable if Aeolus was a plug-in, enabling it to run within (a running instance of) ETABS and enabling use of other plug-ins (simultaneously). However, the current release of ETABS API does not support plug-ins, but future releases of the API will. Conversion from an extension to a plug-in should be straightforward.

## Challenges faced when developing the extension

The main challenges when developing a semi-automatic wind load module is geometry recognition (see Chapter 3. Geometry Recognition). There is another rather enervating challenge: The extensiveness of the Eurocode (discussed in Eurocode complexity). Equations from the Eurocode are logical and straightforward to convert to a software. However, the many *figures* represent a challenge because they are intended for a human interpretation. Engineering judgements and adjustments are required when a figure are applied to a particular structure, and is hence a poor candidate for automated computations [46].

However, there is no indication that anything in the Eurocode is impossible to convert to a software which intension is to *augment* the engineering environment, not to automate it.

## An alternative to a non-digital Eurocode?

One may argue that a Eurocode for wind loads is unnecessary. A viable alternative is to model the building in a software and run a CFD-analysis [47]. This will give accurate results, is fairly quick, and will reduce human errors. The European Committee for Standardization could consider converting the entire set of Eurocodes to a software package, but that discussion is outside the scope of this report.

## 7.1 **Conclusion**

The main product of this thesis work is Aeolus, an extension for CSi ETABS that generates wind loads on a BIM according to NS-EN 1991-1-4:2005+NA:2009. It is not 100% finished and ready for professional use. However, connection to ETABS, geometry recognition of building walls, and wind load calculation for walls have been completed and tested. Aeolus works well and gives good results. For more information about what remains, see Chapter: 7.2. Further Work. The most important are roof recognition and corner effects.

This report and a demo video of Aeolus are additional results of this thesis work. These explain theoretical background for Aeolus, how Aeolus is used, how it works, and how the results may be interpreted.

As shown in Chapter 5. Examples, the results provided by Aeolus are satisfactory. The results correspond well with an analytical solution and the loads applied to a model are as expected. The analysis is accurate, and the usage of Aeolus is efficient.

## 7.2 Further Work

Aeolus is not completely finished; there are still several things that need to be done before it may be distributed to the engineering community. The essentials are listed in the table below. In addition, Aeolus should be distributed to a testing group for user feedback.

Table 7-1: Essential further work

| Element | Description |
|---|---|
| Plug-in conversion | Convert Aeolus from an Extension to a plug-in to enable execution of Aeolus within a running instance of ETABS and execution of other plug-ins simultaneously. This process should be straightforward when CSi enables development of plug-ins to ETABS. |
| Roof recognition | A routine in the geometry recognition should be created to enable the recognition of model roofs. |
| Rounded parts | The wall objects should include an attribute giving the relative angle to the adjacent wall object, providing information about whether the wall is part of a rounded region of the model |
| Composite buildings | Aeolus should be able to handle models that have parts that act as windscreens for other parts, e.g. several wings or multi-towered models. |
| Other environmental loads | Aeolus could be a part of a larger system of semi-automatic environmental load generators, that includes snow loads, earthquake loads, etc. |

# 8 Bibliography

[1] "ETABS," Csi, [Online]. Available: http://www.csiamerica.com/products/etabs. [Accessed 10 October 2014].

[2] "Dynamic-Link Library," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Dynamic-link_library. [Accessed 08 April 2014].

[3] "Application Programming Interface," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Application_programming_interface. [Accessed 10 April 2015].

[4] "Test Driven Development," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Test-driven_development. [Accessed 10 April 10].

[5] NS-EN 1991-1-4:2005+NA:2009, Brussels: European Committee for Standardization.

[6] "Model-View-Controller," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller. [Accessed 08 April 2014].

[7] "European Committee for Standardization," Wikipedia, [Online]. Available: https://www.cen.eu/Pages/default.aspx. [Accessed 11 March 2015].

[8] "Eurocode," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Eurocode. [Accessed 11 March 2015].

[9] "Aeolus," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Aeolus. [Accessed 22 May 2015].

[10] "SAP2000," CSi, [Online]. Available: http://www.csiamerica.com/products/sap2000. [Accessed 30 October 2014].

[11] "CSiBridge," Computers & Structures, Inc., [Online]. Available: http://www.csiamerica.com/products/csibridge. [Accessed 06 May 2015].

[12] W. F. Baker, "Design and construction of the world's talles building: The Burj Dubai," CENews, [Online]. Available: http://cenews.com/article/7709/design_and_construction_of_the_world_acute_s_tallest_building__the_burj_dubai. [Accessed 30 October 2014].

[13] E. A. Holst, "Wind Load Module for CSi ETABS," Project Assignment, NTNU, Department of Structural Engineering, Trondheim, 2014.

[14] "Application Programming Interface," Computers and Structures, Inc, [Online]. Available: https://www.csiamerica.com/application-programming-interface. [Accessed 10 April 2015].

[15] "Visual Studio," Microsoft Corporation, 2014.

[16] "Microsoft Visual Studio," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Microsoft_Visual_Studio. [Accessed 23 April 2015].

[17] "Resharper," Jetbrains, [Online]. Available: https://www.jetbrains.com/resharper/,. [Accessed 23 April 2015].

[18] "Code Smell," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Code_smell. [Accessed 13 May 2015].

[19] "Resharper extension," [Online]. Available: https://visualstudiogallery.msdn.microsoft.com/EA4AC039-1B5C-4D11-804E-9BEDE2E63ECF. [Accessed 23 April 2015].

[20] "C Sharp (Programming Language)," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/C_Sharp_%28programming_language%29. [Accessed 23 April 2015].

[21] "Visual Basic," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Visual_Basic. [Accessed 23 April 2015].

[22] "Microsoft Word," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Microsoft_Word. [Accessed 23 April 2015].

[23] "List of EN standards," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/List_of_EN_standards. [Accessed 09 April 2015].

[24] "EN Eurocode Parts," European Comittee of Stanradization, [Online]. Available: http://eurocodes.jrc.ec.europa.eu/showpage.php?id=13. [Accessed 09 April 2015].

[25] "Public Safety Standards of the European Union," Public.Resource.Org, [Online]. Available: https://law.resource.org/pub/eur/manifest.eur.html. [Accessed 09 April 2015].

[26] "Software extension," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Software_extension. [Accessed 04 May 2015].

[27] "Adobe Flash Player," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Adobe_Flash_Player. [Accessed 04 May 2015].

[28] "QuickTime," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/QuickTime. [Accessed May 04 2015].

[29] "Java applet," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Java_applet. [Accessed 04 May 2015].

[30] "Browser Toolbar," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Browser_toolbar. [Accessed 04 May 2015].

[31] "Counter Strike," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Counter-Strike. [Accessed 04 May 2015].

[32] "Defense of the Ancients," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Defense_of_the_Ancients. [Accessed 05 May 2015].

[33] "Team Fortress Classic," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Team_Fortress_Classic. [Accessed 04 May 2015].

[34] "ETABS User Guide," Computers And Structures, Inc, [Online]. Available: http://docs.csiamerica.com/manuals/etabs/User%27s%20Guide.pdf. [Accessed 08 April 2015].

[35] "User Interface," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/User_interface. [Accessed 08 May 2015].

[36] "Windows Forms," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Windows_Forms. [Accessed 08 May 2015].

[37] "Big-O Notation," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Big_O_notation. [Accessed 2015 April 28].

[38] "Object-Oriented Programming," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Object-oriented_programming. [Accessed 04 May 2015].

[39] "Code Metrics Values," Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/bb385914.aspx. [Accessed 04 May 2015].

[40] "Cyclomatic Complexity Ranges," StackExchange, [Online]. Available: http://programmers.stackexchange.com/questions/194061/cyclomatic-complexity-ranges. [Accessed 04 May 2015].

[41] "Interface (Programming)," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Interface_%28computing%29. [Accessed 04 May 2015].

[42] "Law of Demeter," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Law_of_Demeter. [Accessed 04 May 2015].

[43] "High Cohesion Loos Coupling," The Bojan's Blog, 08 April 2015. [Online]. Available: http://thebojan.ninja/2015/04/08/high-cohesion-loose-coupling/. [Accessed 05 May 2015].

[44] "Why and what is Arrange Act Assert," ArrangeActAssert, [Online]. Available: http://www.arrangeactassert.com/why-and-what-is-arrange-act-assert/. [Accessed 05 May 2015].

[45] "Arrange Act Assert," Jeff Grigg, [Online]. Available: http://c2.com/cgi/wiki?ArrangeActAssert. [Accessed 05 May 2015].

[46] T. Syvertsen and R. Sandvik, "Computers and Building Codes - Enemies Forever?," International Association for Bridge and Structural Engineering (IABSE) 13th Congress, Helsinki, June 1988.

[47] "Computational Fluid Dynamics," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Computational_fluid_dynamics. [Accessed 07 May 2015].

[48] "Eurocode Links," European Comitee for Standardization, 2009.

[49] "Intelligent code completion," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Intelligent_code_completion. [Accessed 23 April 2015].

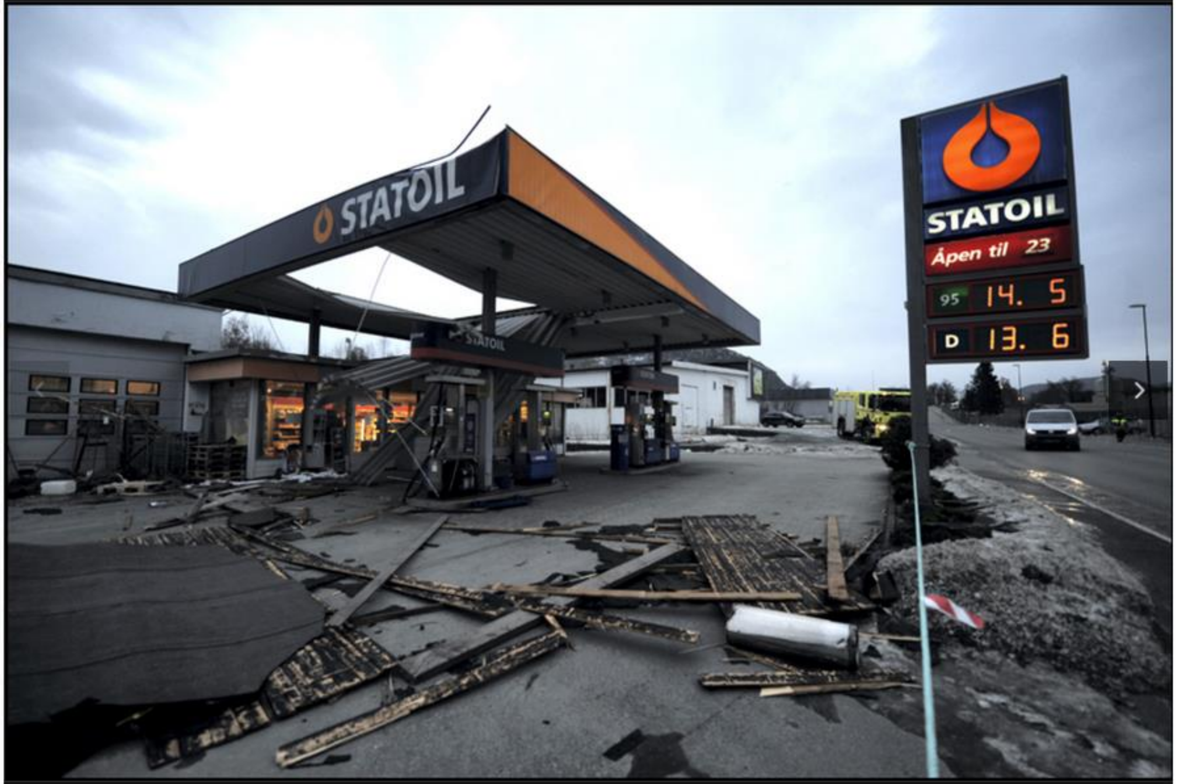
[50] "YouTube," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/YouTube. [Accessed 04 May 2015].

[51] *ETABS 2013,* Berkley, CA: Computers and Structures, Inc, August 2013.

# Appendix A: Newspaper clippings

## Kraftig vind tok med seg taket på Statoil-stasjonen

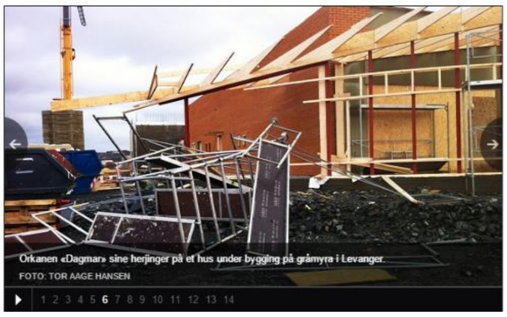
**** Flere tak på bygg forsvinner i Jørpeland**
**** Politiet i Rogaland: - Gjenstandene flyr**



*TAKET BLÅST BORT: Slik så taket på Statoil-stasjonen ut på Jørpeland lille julaften. Foto: TOR INGE JØSSANG / STAVANGER AFTENBLAD*

## Her herjet «Dagmar» fra seg

Ekstremværet «Dagmar» har satt sine spor. Sjekk omfanget av uværet her og hvor det ble registrert orkaner.



Orkanen «Dagmar» sine herjinger på et hus under bygging på gråmyra i Levanger.
FOTO: TOR AAGE HANSEN

## Vinden tok taket

Øylo Gjestgiveri i Vang er tydelig preget etter stormen i helgen. Deler av taket er blåst til side.

## Slik gikk det med den splitter nye hytta da vinden tok tak

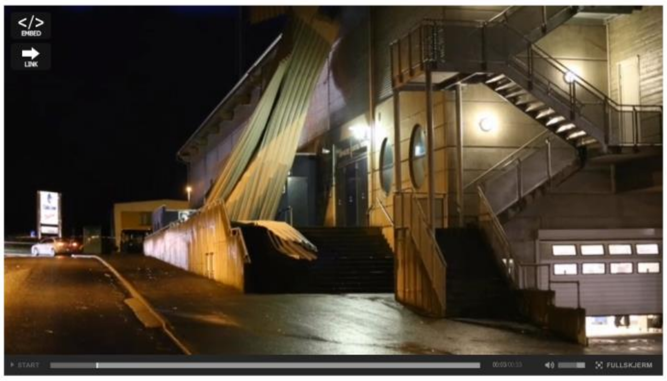
**Bjørn Håvik trodde ikke sine egne øyne da han så hva vinden hadde tatt med seg i Lindesnes.**

Hvem som eier hytta som er tatt av vinden, er fortsatt uvisst. FOTO: Bjørn Håvik

## «Ole» herjer: 30.000 mennesker uten strøm

30.000 personer er uten strøm etter at ekstremværet «Ole» har herjet i Midt-Norge, Nordland og Troms. – Det verste som kunne skje har skjedd. Vi har feil på stort sett alle hovedlinjene, sier sjef i kraftselskap.

18.000 I LOFOTEN UTEN STRØM: Opprydningsarbeidet på Vareide i Flakstad kommune er i gang, manskaper fra Lofotkraft er ute å rydder stolper av veien så bergingsmannskaper kan komme til et vogntog som har blåst av veien.
FOTO: TOMMY JOHANSEN



## Vinden tok deler av tribunetaket

Mye å gjøre for politi og brannvesen som følge av uværet, men ingen alvorlige hendelser. Strømbrudd flere steder. Fortsatt noen kanselleringer av båt- og ferjeruter, tirsdag morgen.

HELT NYTT: Taket ble lagt sommeren 2014. - Taket er helt nytt, så det må ha vært enorme krefter i sving, sier Unni Arnøy.
**FOTO: RUNE CHRISTOPHERSEN**

Uværet slo til mot Lista tidligere i dag med voldsomme bølger. Mange skuelystne tok turen ned til sjøen for å se på naturkreftenes herjinger.
FOTO: Anders Martinsen

## 168.000 uten strøm etter ekstremværets hjeringer

**Ekstremværet Nina har ført til omfattende skader og strømbrudd en rekke steder på kysten fra Østfold til Nordvestlandet.**

*Det regnar rett inn i stova på det to år gamle huset til Rune Gerhardsen. I 14-tida laurdag vart taket røska av, som følgje av den kraftige vinden. FOTO: PRIVAT*

## "Nina" tok taket til Rune

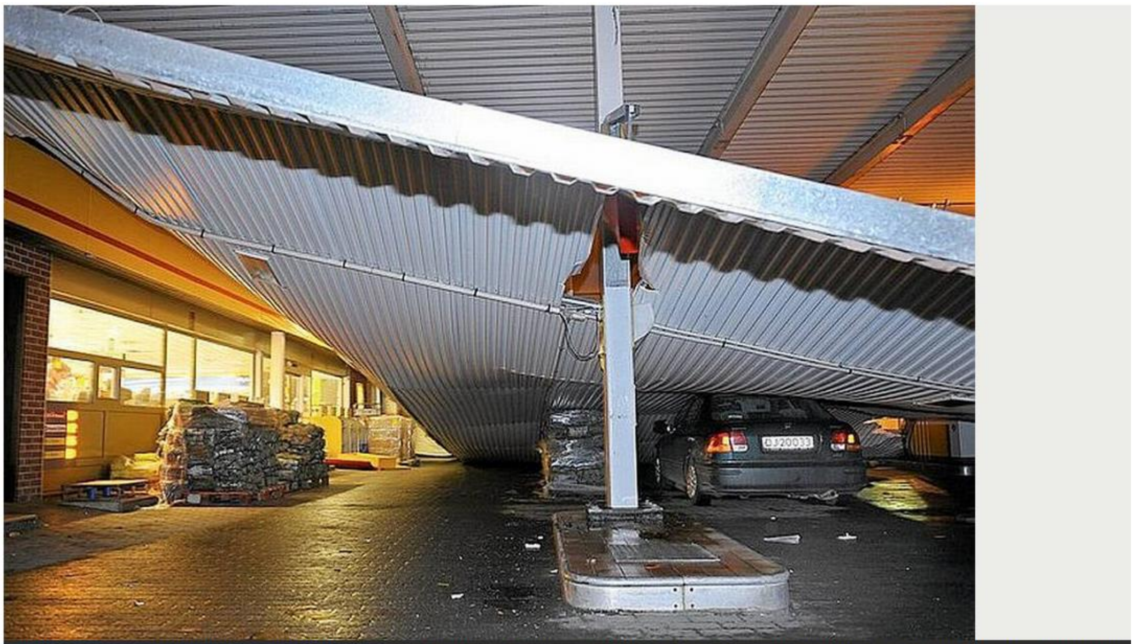
Det regnar rett inn i stova, og skadene kan passera millionen.

Marius Knutsen marius@sunnhordland.no

**Relaterte artiklar**

Vinden tok hallporten

– Verste uvêret på 20 år


Bølgeblikk og lysarmatur gjorde skade på drivstoffpumpene og en parkert bil, men ingen mennesker ble skadd da taket på en bensinstasjon utenfor Stavanger ble tatt av vinden. FOTO: Fredrik Refvem/Stavanger Aftenblad

## Vinden tok tak

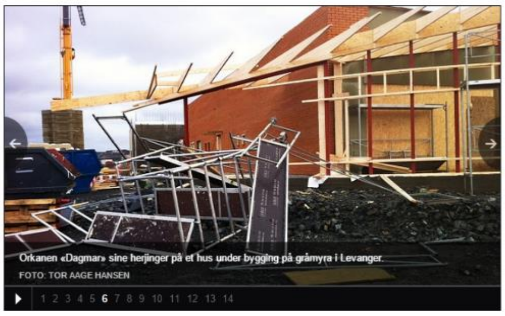
**Taket på en Shell-stasjon utenfor Stavanger måtte gi tapt for vinden i natt.**

Mens vinden forårsaker materielle skader på Vestlandet, skaper snøværet vanskelige kjøreforhold på Sørlandet. Her i Kristiansand oppfordres folk til å la bilen stå dersom de kan. Foto: Tor Erik Schrøder / NTB scanpix

## Vinden tok flere tak på Vestlandet

## Her herjet «Dagmar» fra seg

Ekstremværet «Dagmar» har satt sine spor. Sjekk omfanget av uværet her og hvor det ble registrert orkaner.


Orkanen «Dagmar» sine herjinger på et hus under bygging på gråmyra i Levanger.
FOTO: TOR AAGE HANSEN

Teltene på Lindahlplan fikk hard medfart i den kraftige vinden søndag
Foto: FOTO: Vigdis Kittang Ramstad

# Appendix B: Terrain Categories

**Category 0**
Sea or costal area exposed to the open sea

**Category 1**
Lakes or flat and horizontal area with negligible vegetation and without obstacles

**Category 2**
Area with low vegetation such as grass and isolated obstacles (trees, buildings) with separations of at least 20 obstacle heights

**Category 3**
Area with regular cover of vegetation or buildings or with isolated obstacles with separations of maximum 20 obstacle heights (such as villages, suburban terrain, permanent forest)

**Category 4**
Area in which at least 15 % of the surface is covered with buildings and their average height exceeds 15 m

Table B-1: Terrain Categories and terrain parameters from the Table 5.1 in the Norwegian National Annex of the Eurocode [5]

| Terrain Category | $k_r$ | $z_0$ | $z_{min}$ |
|---|---|---|---|
| 0 | 0,16 | 0,003 | 2 |
| 1 | 0,17 | 0,01 | 2 |
| 2 | 0,19 | 0,05 | 4 |
| 3 | 0,22 | 0,3 | 8 |
| 4 | 0,24 | 1,0 | 16 |

Where:

| | |
|---|---|
| $k_r$ | Terrain factor |
| $z_0$ | Roughness length |
| $z_{min}$ | Minimum height ($z_{min} < z < 200\ m$) |

# Appendix C: Class Overview for Aeolus

| General classes | Function |
|---|---|
| *View* | Contains attributes and methods for user input and output. |
| *CSiInstance* | Serves as a controller for the *Recognizer,* the *API* and the *Eurocode* objects. |
| *Recognizer* | Contains method for geometry recognition. Gets data from the ETABS model through the *API* class and creates model objects based on said data. |
| *AreaObjectHandler* | Uses information for the *Recognizer* class to create *Wall* objects and to partition said *Wall* objects. |
| *Eurocode* | Contains functionality from NS-EN 1991-1-4:2005+NA:2009, including necessary methods for wind load calculation for all *Wall* elements |
| *API* | Contains methods to communicate with ETABS, including, but not limited to *GetPointList(), RefreshView(), AddPointToModel(Point p),* and *DrawAreaObjectByPoints(List<Point> points).* |
| *ConstantHolder* | Static class that contains data several classes need access to. Includes, but not limited to, Terrain Category, Construction Height, Basic Wind Velocity, and Wind Direction |
| *DataReader* | Contains methods for reading a data file containing location based information such as basic wind velocity |
| *GeometryLogic* | Static class that contains basic geometry methods, including, but not limited to *GetNormalVector(Area a), GetVectorRotated(double[] v, double yaw, double pitch, double roll), IsWindwards(Wall w),* and *GetAngle(Frame f1, Frame f2).* |
| *LoadAssigner* | Contains methods to add wind load patterns to the model and wind load to all area objects. |
| *LogWriter* | Contains methods to log all logic that is done. Every calculation may be monitored in retrospect. Every variable for every calculation is systematically logged. |

| Model object classes | Function |
|---|---|
| *Object* | Contains one attribute: Name. Every model object class inherits from this class |
| *Point* | Contains five attributes: X, Y, Z, Story, and ConnectedFrames. Contains methods to get connected frames and to add connected frames |
| *PeripheryPoint* | Subclass of *Point*. Contains four attributes: NeighborsDown, NeighborsUp, NeighborClockwise, and NeighborCounterClockwise |

|  |  |
|---|---|
| *Frame* | Contains two attributes: Point1 and Point2. Contains a method to get the other point when one point is known. |
| *Area* | Contains two attributes: PointList and NormalVector. Contains a method to get the area of the *Area* object. |
| *Wall* | Inherits from *Area*. Containing methods include, but are not limited to, *GetWallZone(), PartitionWall(), and GetCoordinates().* |
| *WallPartition* | Inherits from *Wall*. Contains attributes Load, centroidX, centroidY and centroidZ. |
| *Story* | Contains attributes PeripheryPoints, StoryUp, StoryDown, and StoryNumber. |

| **Unit Test classes** | **(Contains unit tests for the class given in the unit test class name)** |
|---|---|
| *GeometryLogicTests* |  |
| *EurocodeTests* |  |
| *RecognizerTests* |  |
| *AreaObjectHandlerTests* |  |
| *WallTester* |  |