Python Framework for Wind Turbines Enabling Test Automation of MoWiT

Johannes Fricke[1] Marcus Wiens[1] Niklas Requate[1]
Mareike Leimeister[1]

[1]Fraunhofer IWES, Fraunhofer Institute for Wind Energy Systems, Germany,
{johannes.fricke, marcus.wiens, niklas.requate, mareike.leimeister}@iwes.fraunhofer.de

Abstract

The development and simulation of engineering systems, especially wind turbines, is becoming increasingly complex and elaborate. At the Fraunhofer Institute for Wind Energy Systems (IWES), the in-house tool MoWiT (Modelica library for Wind Turbines) is being developed for load simulation. MoWiT is based on the modeling language Modelica and is constantly evolving. It is, thus, also becoming more and more enhanced. This results in an increased need for automation for the complex simulation setups and a need for quality assurance of simulation code used. Test automation is used to always ensure the quality of the code. The automation of various simulations and the test automation for the load simulation code are provided by PyWiT (Python Framework for Wind Turbines), which will be presented here in more detail.

Keywords Modelica, MoWiT, Python, Wind Turbines, Test Automation

1 Introduction

To represent different operating conditions of wind turbine systems, hundreds of input parameters are used for the various load simulations. Additionally, the simulation models for realistic estimation of loads and their influence on the wind turbine are getting more and more complex. This paper presents the Python Framework for Wind Turbines (PyWiT), which is developed at the Fraunhofer Institute for Wind Energy Systems (IWES). PyWiT further automates the load simulations written in Modelica using the Modelica Library for Wind Turbines (MoWiT) – also developed at Fraunhofer IWES. In addition, PyWiT is used for test automation of the load simulations to be able to automatically detect unsuitable or incorrectly implemented models and, thus, contribute to the quality control of MoWiT in an automated manner. The procedure for such a test automation is described in this paper and explained based on some examples.

1.1 Motivation

In the load simulation of wind turbines, numerous cases of environmental conditions are simulated in conjunction with different operating conditions of the wind turbines. In the design of wind turbines, for example, these include the so-called "design load cases", which comprise several hundred to thousands of simulations for a single wind turbine. In addition to normal operation under various wind conditions (speeds, profiles over the height, inclined flows, turbulence) and – if offshore – also wave and current conditions, these also include fault load cases in which the turbine shuts down under certain conditions and extreme weather conditions, such as gusts, extreme waves, and strong currents. Furthermore, to reduce dependence on chance, all simulations are run with multiple seeds for wind and waves. The combination of all these parameters results in a high number of simulations, which can only be performed with reasonable effort through automation.

All the systems in MoWiT interact with each other, which means that, when the code is changed in one model in MoWiT, there can be changes in the results for many or all the system parts at the same time.

1.2 Problem Description

As explained previously numerous combinations of input parameters (e.g., wind speed, angle of attack, seed for creating the wind fields, turbulence, etc.) are used in the load simulation of wind turbines. This results in a high three- or four-digit number of simulations for a single turbine. The combination of all these parameters by hand is therefore only possible with great effort and automation, with respect to both combinatorics and the execution of the simulations, as well as further processing of the simulation results (error checking, sorting, filtering, further evaluations) is inevitable.

Due to the strong coupling within MoWiT, even small changes or code optimizations, as well as new implementations of models, can quickly lead to different results in many parts of the entire wind energy system model. To prevent unwanted changes, e.g., due to unsuitable models or incorrect implementation, tests are implemented that automatically compare reference results

of the simulations with results of the current implementation. Thereby, the effects of changes in the code in different subsystems of the wind turbine, as well as in the different output parameters (forces, moments, generator power, etc.) can be detected and evaluated.

This paper introduces the Python Framework for Wind Turbines PyWiT and explains its structure in more detail. It is described how the program is structured, as well as how and for which practical applications, in particular for the test automation of MoWiT, it is used.

2 Material and Methods

In this section the used materials and methods are presented. After a short introduction to MoWiT, on which the framework PyWiT is based, PyWiT is explained in more detail. Beside the rough program flow, the different modular designed functions are described.

2.1 Modelica Library for Wind Turbines

MoWiT, which is available free of charge for academic use, is developed in-house at Fraunhofer IWES as a completely object-oriented simulation tool for fully coupled aero-hydro-servo-elastic simulations of wind turbines on- and offshore, with bottom-fixed or even floating substructures. It is written in the open-source object-oriented and equation-based modeling language Modelica, which can be used for various engineering systems to solve multi-physics problems. Detailed information on the development of MoWiT, as well as on the structure and components of this library can be found in the literature (Leimeister et al.2020; Leimeister and Thomas. 2017; Thomas et al. 2014; Strobel et al. 2011).

2.2 Python Framework for Wind Turbines

PyWiT, developed in-house at Fraunhofer IWES, is a program coded in Python to manage simulations of wind energy systems modeled in MoWiT. These simulations are executed in Dymola (Dymola2021). The current PyWiT version consists of six modules:

- Input Module,
- Experiment Generator (incl. Design of Experiments),
- Package Creator,
- Wind Field Generator,
- Simulation Manager, and
- Post-processing.

The single modules are organized in three main groups “Input Generation”, “Simulation Manager”, and “Post-processing”, as it is shown in Figure 1 by the simplified activity diagram. It shows the possible processing paths depending on the input point and decisions (indicated by the diamond symbols) for a single input file. A simplified presentation for one input file was chosen since the additional loops for multiple files are unnecessary for the understanding of the structure. The three main groups represent the code structure and the responsibility of the specific code. Additionally, the diagram shows the intersection between the main groups. The single groups are explained in more detail in the following subsections.

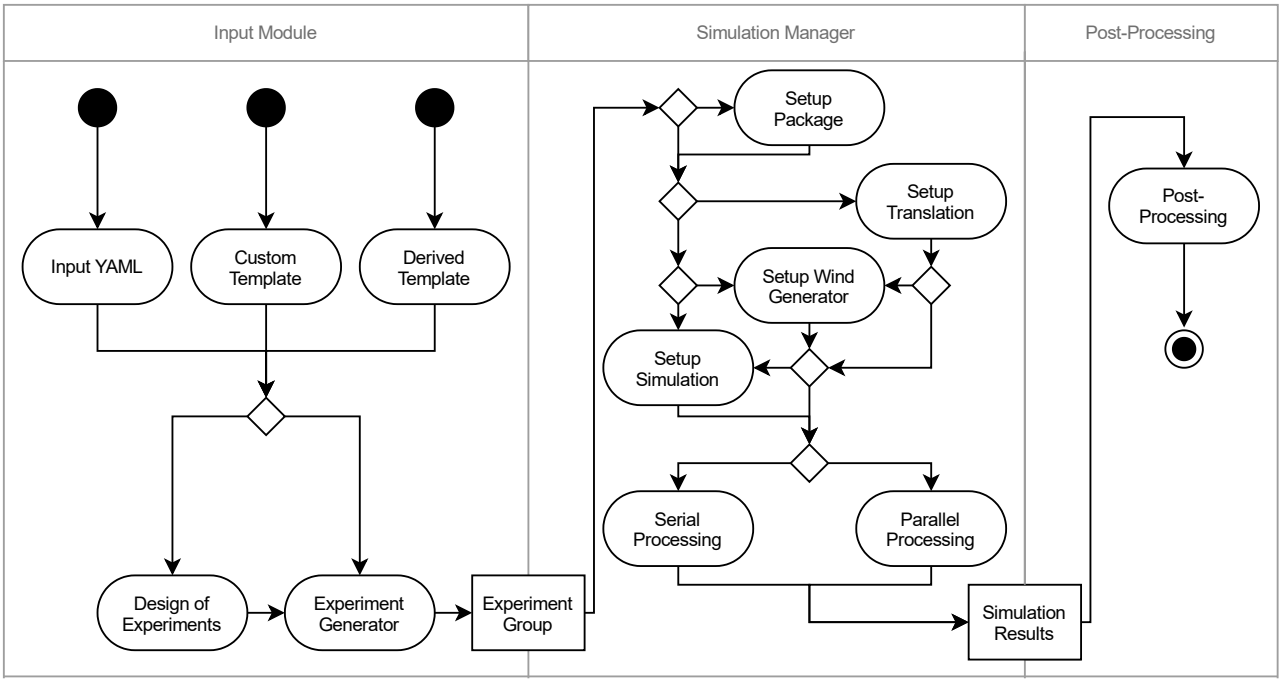

Figure 1 Simplified activity diagram for PyWiT

DOI 10.3384/ecp21181403

Input Manager

PyWiT is controlled by YAML files, which define a structured format. A structured input file, which contains all input parameters necessary for the simulations, can be created for the user without programming knowledge. PyWiT is started by specifying the input file(s) and the available processors. The input files are processed by the Experiment Generator into experiment groups, which are handled by the Simulation Manager. The Experiment Generator is responsible for sorting the data from the input file and parameterizing the input objects. Each input file results into one experiment group, which defines one set of simulations for a single model. A single simulation is represented by a data object, which contains all necessary information for independent simulation. Furthermore, there are three different types of input for the Input manager according to Figure 1. The usual YAML input specifies a fixed number of simulations by varying parameter lists. A single simulation is defined based on the common index for all lists, which leads to the requirement of equal length for all varying parameter lists. An example for the varying parameter lists is given in Listing 1.

Listing 1. Input Parameters

```
windSpeed: [9, 11, 13]

yawAngle: [-8, 0, 8]

randomSeed: [1, 2, 3]
```

This would result in three simulations, where the first configuration is given by the first element of every list. Derived templates offer simulation setups for, e.g., wind speed ranges or other commonly simulated cases.

Every input file has the option of using the Design of Experiments module. The Design of Experiments is used to expand the input files by creating the varying parameter lists from the combination of small parameter lists to obtain a large number of simulations. Currently, the Design of Experiments is based on the pyDOE2 library (PyPi.org2021) and only combinatorial designs are considered. The module offers designs, which are commonly used in wind turbine simulations. A simple example is given in Listing 2.

Listing 2. List created by Design of Experiments

```
windSpeed: [9, 9, 9, 11, 11, 11, 13, 13, 13]

yawAngle: [-8, 0, 8, -8, 0, 8, -8, 0, 8]

randomSeed: [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

The parameter lists for wind speed and yaw angle from Listing 1 are combined in a “full factorial” design, which is the combination of each list elements of the first list with all the other lists elements. This leads to nine simulations. Additionally, using a “copy-stretch” design, the random seed variable list from Listing 1 has been copied to the appropriate length. Initially, the random seed list has three elements and is therefore appended twice to itself to reach a length of nine.

Simulation Manager

The Simulation Manager takes care of the four different stages of processing for the experiment groups:

- Create Package,
- Translation,
- Generate Wind Fields,
- Simulation,

The processing of the experiment groups is separated into two steps. First, the Simulation Manager splits all Experiment Groups according to their runtime flags, so that each of the four stages described above are performed one after the other. The stages Wind Field Generation, Translation, and Simulation can be performed in parallel. The Package Creation is not parallelized, since it needs only a computing time in the range of milliseconds.

It is possible to execute these steps independently of each other if the prerequisites for the specific step are already fulfilled. For example, it is possible to perform the Translation step (possibly with the following steps) without creating a package, if a package already exists. It is also possible to create only wind fields if they are to be used for other purposes than the simulation with PyWiT. These wind fields are created by automatically writing input files for TurbSim (Bonnie J. Jonkman2009), which generates wind fields from these input files.

Furthermore, the step for creating packages implies that the dependencies of all modules containing the physical relationships of a wind turbine are determined in MoWiT and all functions necessary for the simulation of a particular model are copied together so that their dependency is maintained. This package can then be translated with Dymola, i.e., the physical interdependencies are compiled into C code and result in a translated model. This model already contains all the necessary input parameters, which, however, can be replaced by means of PyWiT to run different simulations based on the same model.

Subsequently, the physical equations are solved iteratively by Dymola. The results of each simulation are MAT files, which contain time series for all previously defined output parameters, as well as log files, which contain information about set parameters and possible errors in the simulation.

In addition to common time-series simulations, it is also possible to perform modal analyses and subsequently create Campbell diagrams.

Post-processing

The post-processing is constantly extended and includes at the current level:

- Sorting, moving, and renaming of simulation results,
- Conversion of simulation results to CSV or NETCDF data format,
- Clustering and averaging of specific time ranges for entry into an Excel spreadsheet for verification of results,
- Creating plots of different time series, probability density functions (PDFs), and power spectral densities (PSDs),
- Comparison if two time series are identical, and
- Fast Fourier transformations of different time series.

All Post-Processing modules described herein can be executed in connection with or independently of simulations. Furthermore, all these modules run independently as a stand-alone version and can be used for already existing files or new simulation results.

Test Automation

Through the various methods presented in this section, PyWiT can be used for test automation of MoWiT in addition to automatically performing simulations for further development of wind turbines. The nature of the input through YAML files allows the setup of different tests of the models created in MoWiT, for which a package is automatically generated, compiled, and simulated. Afterwards, the different Post-processing modules can be used to check the results of the test simulations and compare them with reference results if necessary. For example, it is possible to store a simulation file that has been validated as a reference in the MoWiT structure and to define this as a reference in the input file. The test model is then simulated and, first, the log files of the test simulation are checked for errors. Depending on the test, the time series of the test simulation can then be compared with the time series of the reference simulation. It is also possible to create different plots or statistical data based on the comparison. The comparison between the reference results and the new simulation results are generated automatically, but the decision whether changes to the code must be made based on these comparisons is currently the responsibility of an experienced engineer.

The test automation can either be started manually, which is useful, for example, when a new model is written in MoWiT before it is merged from the develop branch into the master branch. In addition, it is possible to run certain tests automatically on a regular basis (under Windows, for example, through the task scheduler) to obtain regular information about the quality of MoWiT.

3 Results

As described in the previous section, PyWiT offers extensive possibilities for automating simulations and various tasks around these simulations. Through the Design of Experiments, it is possible to create various combinations of input parameters by minimal user input and to run simulations with them. Different Post-processing modules can be used to evaluate the simulations on the one hand and to test the MoWiT code on the other hand. Thus, PyWiT extends the existing MoWiT tool and automates otherwise labor-intensive and error-prone tasks, leading to higher quality results and reduction of working time and, hence, costs in wind turbine development.

An example of an evaluation from test automation is shown in Figure 2.

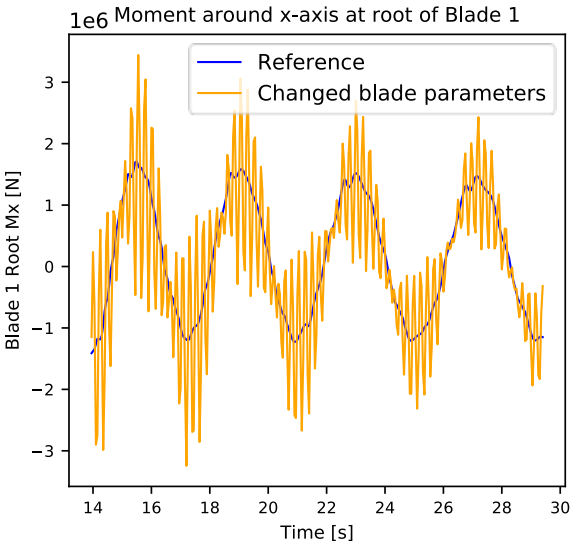

Figure 2 Comparison of the moment around the x-axis on a blade of a reference model and a model with shifted aerodynamic axis in the blade

Here, two models with identical input parameters were compared. They differ in the position of the aerodynamic axis in the blades of the wind turbine, which has an influence on almost all result variables. As an example, the moment in the root of a blade is shown here. In this case, the blue line is considered the reference model, which means that those results come from a valid simulation. The orange line is compared to that valid simulation, to see if the code still works reasonable. The plot is created automatically, while the interpretation of the results is made by hand. The mean values of the two values only have small differences, but the changed aerodynamic axis leads to higher-frequency oscillations that are superimposed on the reference signal. Such a figure, or one like it, can be evaluated by an experienced engineer to determine if there are errors in the code and where they may lie.

4 Discussion

The presented PyWiT framework eases the setup of various simulation studies. Adjustments of simulation

DOI 10.3384/ecp21181403

setups are simplified and sped up by taking advantage of object-oriented programming methods for the definition of simulation cases. This is especially useful in the setup of simulation series according to standards, e.g., (IEC2019; DNV GL. 2016). Along with the specific standard, the simulation series can be defined by derived YAML templates from class instances, which are initialized by specific wind turbine parameters. Common variable definitions are distributed to different simulation cases by class inheritance. Specific wind conditions can be implemented as single classes and used as modules in the simulation series class.

Furthermore, high flexibility can be maintained by implementation in individual packages. By the separation of Experiment Generator and Simulation Manager the coupling between code blocks is reduced. This makes code adjustments more efficient. The Experiment Generator produces single experiments, which can be analyzed and thereby the errors in the setup are reduced. An example would be the settings for wind conditions like the wind shear. They can be specified for the generated wind fields or directly in MoWiT but should be only set in one of these options. The Design of Experiments for this framework is carried out often for variables with more than two levels. Since a full factorial design leads to a high number of simulations, a combinatorial design offers a possibility to reduce the simulation amount. However, a combinatorial design could lead to bias in a simulation series, when correlating influences are combined in the design.

Implementation of a simulation framework can be a complicated task. Our example shows one way of structuring the required steps from the simulation definition to the execution. A guideline for other developers/researchers would be to develop code, which can be changed without much effort. The reduction of coupling in the code, aim for high flexibility, and coding principles like DRY (Wilson et al.2014) are helping to achieve that goal.

5 Outlook

PyWiT is already at a stage of development where it can take over many tasks automatically; however, it is still in constant further development. In addition to various modules for Post-processing, also for the extension of test automation, the applications Load Simulation Verification, Distributed Computing with HTCondor, and Optimization, which will be explained in the following, are already ongoing or planned. At the end, further planned advancements are briefly discussed.

5.1 Load Simulation Verification

Load Simulation Verification is a methodology, applied at Fraunhofer IWES (Huhn and Popko2020), for comparing different load simulation codes, such as FAST (Jason Jonkman2018; DNV GL, 2018) or Bladed (DNV GL, 2018; DNV GL, 2018), etc.), with MoWiT. In addition to various plots with time series, PSDs, and PDFs, the focus here is on an XLSX file in which various sensors are available for different input parameters. This XLSX file is reviewed and evaluated by an experienced engineer and any deviations in the simulations are evaluated.

An automatic execution of the simulations in MoWiT for the verification and a subsequent creation of the different plots, as well as the filling of the XLSX file, is in the alpha version and is currently being tested in different projects.

5.2 Distributed Computing with HTCondor

In order to efficiently use computing resources, which are often distributed over several computers, a distributed computing system has been developed at Fraunhofer IWES based on the open-source software HTCondor (HTCondor2021). This system distributes the simulations to the available computing resources and manages the simulations that are entered into the queue system by different users. The already tested system can easily be extended by further computing resources. An integration of the system for distributed simulation in PyWiT is currently under development.

5.3 Optimization

The characteristic of PyWiT of automatically executing tasks facilitates its usage for mathematical optimization for a broad range of problems. One can make use of the various existing open-source optimization libraries in Python. Optimizing wind turbine parameters was already possible and extensively used in a previous version of the framework (Leimeister. 2019; Leimeister et al.2021). These range from the design optimization of the entire wind energy system or specific components, such as the floating support structure, to controller tuning optimization tasks or further applications. The objectives are mostly related to cost minimization, load reduction, or performance improvement (Leimeister et al.2020; Leimeister et al.2020), but can also address system or component scaling (Leimeister et al. 2019), as well as reliability-based design optimization targets (Leimeister and Kolios2021).

A major advantage of PyWiT for the further use for solving optimization problems is the modular design. Various simulations with different parameters used as optimization parameters can be combined with various objective functions and constraints which are defined in the Post-processing module. Therefore, holistic and multi-disciplinary applications are possible through combination for realizing any kind of optimization problem, e.g., multi-objective optimization of different components or consideration of fatigue and extreme loads.

As for each optimization problem and corresponding considered system certain optimization algorithms are more suitable than others, the large number of different optimization platforms, tools, and algorithms must be

exploited when aiming for solving diverse optimization problems. However, each optimization algorithm has certain parameters and options to be specified. Therefore, a modular and standardized interface to different Python optimization libraries is currently developed at Fraunhofer IWES. It allows the easy use of different optimizers by utilizing the same input definitions for the optimization problem, comprising optimization parameters, objectives, and constraints. PyWiT will be connected to the interface in future.

5.4 Further Planned Advancements

Further advancements will include the fully automated creation of different design load cases in a separate module. The module will make use of the Design of Experience module and contain the required parametrization of turbine models and simulation setup. This will replace the current template-based approach. Another step includes further implementations for Post-processing (DNV GL, 2018), which partially already exist in separate tools. These comprise, among others, fatigue load evaluation, using Rainflow Counting and Miner's rule, as well as other load evaluation methods.

6 Conclusions

In this paper, the Python Framework for Wind Turbines PyWiT developed at Fraunhofer IWES is described. It was shown that PyWiT already automates many tasks around the simulation of wind turbines and, thus, makes the development of wind turbines faster and cheaper. Besides the automated input of many parameters by the Design of Experiments, PyWiT can also be used for test automation of the simulation code MoWiT to guarantee the quality of the code. PyWiT is already used for many tasks at Fraunhofer IWES, including load verification, but is under constant development, especially in Post-processing.

References

Bonnie J. Jonkman (2009) "Turbsim User's Guide." Technical Report. *National Renewable Energy Laboratory*.

DNV GL (2016-10) *Loads and Site Conditions for Wind Turbines (Standard DNVGL-ST-0437)*. November. DNV GL AS, www.dnvgl.com/. Accessed 9 Oct. 2018.

--- (2018-01) *Bladed Theory Manual: Version 4.9*. 1 Jan. 2018.

--- (2018-01) *Bladed User Manual: Version 4.9*. 1 Jan. 2018.

Dymola: (Dynamic Modeling Laboratory) (2021). Dassault Systèmes, 2021. Accessed 16 Apr. 2021.

HTCondor: High Throughput Computing (2021). University of Wisconsin, Madison, USA, 2021. Accessed 16 Apr. 2021.

Huhn, Matthias L., and Wojciech Popko (2020) "Best Practice for Verification of Wind Turbine Numerical Models." *Journal of Physics: Conference Series,* vol. 1618, p. 52026. doi:10.1088/1742-6596/1618/5/052026.

IEC (2019-02) *Wind Energy Generation Systems - Part 1: Design Requirements (International Standard IEC 61400-1:2019-02)*. 4.0th ed. International Electrotechnical Commission. International Standard.

Jason Jonkman (2018-01) *NWTC Information Portal (FAST)*. 1 Jan. 2018, nwtc.nrel.gov/FAST. Accessed 21 Apr. 2021.

Leimeister, Mareike (2019-03) "Python-Modelica Framework for Automated Simulation and Optimization." *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6,* 2019, March 4-6, 2019, Regensburg, Germany, The 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019. Linköing University Electronic Press, 2019pp. 51–58. Linköping Electronic Conference Proceedings.

Leimeister, Mareike, et al. (2020) *A Fully Integrated Optimization Framework for Designing a Complex Geometry Offshore Wind Turbine Spar-Type Floating Support Structure.*

Leimeister, Mareike, and Athanasios Kolios (2021) "Reliability-Based Design Optimization of a Spar-Type Floating Offshore Wind Turbine Support Structure." *Reliability Engineering & System Safety,* vol. Document 32009L0028, no. 4, p. 107666. doi:10.1016/j.ress.2021.107666.

Leimeister, Mareike, Athanasios Kolios, and Maurizio Collu (2020) "Development and Verification of an Aero-Hydro-Servo-Elastic Coupled Model of Dynamics for FOWT, Based on the MoWiT Library." *Energies,* vol. 13, no. 8, p. 1974. doi:10.3390/en13081974.

--- (2021) "Development of a Framework for Wind Turbine Design and Optimization." *Modelling,* vol. 2, no. 1, pp. 105–28. doi:10.3390/modelling2010006.

Leimeister, Mareike, Athanasios Kolios, and Maurizio Collu, and Philipp Thomas (2019-06) "Larger MW-Class Floater Designs Without Upscaling? A Direct Optimization Approach." *ASME 2019 38th International Conference on Ocean, Offshore and Arctic Engineering: Volume 1: Offshore Technology; Offshore Geotechnics*, June 9-14, 2019, Glasgow, Scotland, UK, ASME 2019 38th International Conference on Ocean, Offshore and Arctic Engineering. American Society of Mechanical Engineers, 2019.

--- (2020) "Design Optimization of the OC3 Phase IV Floating Spar-Buoy, Based on Global Limit States." *Ocean Engineering,* vol. 202, no. 1, p. 107186. doi:10.1016/j.oceaneng.2020.107186.

Leimeister, Mareike, and Philipp Thomas (2017-05) "The OneWind Modelica Library for Floating Offshore

Wind Turbine Simulations with Flexible Structures." *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*, May 15-16, 2017, Prague, Czech Republic, The 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017. Linköping University Electronic Press, 2017pp. 633–42. Linköping Electronic Conference Proceedings.

PyDOE2 1.3.0 (2021-04) PyPi.org. 1 Jan. 2021, pypi.org/project/pyDOE2/. Accessed 20 Apr. 2021.

Strobel, M., et al. (2011-03) "The OnWind Modelica Library for OffshoreWind Turbines - Implementation and First Results." *Proceedings from the 8th International Modelica Conference, Technical Univeristy, Dresden, Germany, March 20-22, 2011*, March 20-22, 2011, Dresden, Germany, The 8th International Modelica Conference, Technical Univeristy, Dresden, Germany, March 20-22, 2011. Linköping University Electronic Press, 2011pp. 603–09. Linköping Electronic Conference Proceedings.

Thomas, Philipp, et al. (2014-03) "The OneWind Modelica Library for Wind Turbine Simulation with Flexible Structure - Modal Reduction Method in Modelica." *Proceedings of the 10th International Modelica Conference, Lund, Sweden, March 10-12, 2014*, March 10-12, 2014, Lund, Sweden, the 10th International Modelica Conference, Lund, Sweden, March 10-12, 2014. Linköping University Electronic Press, 2014pp. 939–48. Linköping Electronic Conference Proceedings.

Wilson, Greg, et al. (2014) "Best Practices for Scientific Computing." *PLoS biology,* vol. 12, no. 1, e1001745. *Guidelines for effective coding. Write programs for people, not computers. Let the computer do the work. Make incremental changes. Don't repeat yourself (or others). Plan for mistakes. Optimize software only after it works correctly. Document design and purpose, not mechanics. Collaborate.*, doi:10.1371/journal.pbio.1001745.