Strengthening development workflows by graphically communicating elements of software design

Rafael Mudafort
IDEAS HPC Best Practices
June 12, 2024
Funding and staffing cycles
About Me


OpenFAST

KiteFAST

FLORIS

Other contributions
- ERF
- WEIS
- ExaWind
- OpenTurbine
- JOSS Reviewer
Implicit design in software

Singleton

![Singleton Diagram](https://sourcemaking.com/design_patterns)

Object Pool

![Object Pool Diagram](https://sourcemaking.com/design_patterns)

Prototype

![Prototype Diagram](https://sourcemaking.com/design_patterns)

Factory

![Factory Diagram](https://sourcemaking.com/design_patterns)

https://sourcemaking.com/design_patterns
Invest and scale

Abstract template design pattern

FLORIS template design pattern

1:1 ➞ 1:N
# Designed components of software

<table>
<thead>
<tr>
<th>Component</th>
<th>Developer</th>
<th>User</th>
<th>Program Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>System-level architecture</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Package, module, subroutine architecture</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design principles, overarching themes</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Data structures and data flow</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Effective usage workflows</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Developer coordination and processes</td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
UML: Unified Modeling Language
The Unified Modeling Language (UML) is a family of graphical notations, backed by single metamodel, that help in describing and designing software systems, particularly software systems built using the object-oriented (OO) style.
Table of Contents
Chapter 1: Introduction to UML
Chapter 2: Development Process
Chapter 3: Class Diagrams
Chapter 4: Sequence Diagrams
Chapter 5: Class Diagrams Advanced
Chapter 6: Object Diagrams
Chapter 7: Package Diagrams
Chapter 8: Deployment Diagrams
Chapter 9: Use Cases
Chapter 10: State Machine Diagrams
Chapter 11: Activity Diagrams
Chapter 12: Communication Diagrams
Chapter 13: Composite Structures
Chapter 14: Component Diagrams
Chapter 15: Collaborations
Chapter 16: Interaction Overview Diagrams
Chapter 17: Timing Diagrams
Appendix: UML Versions

Key features:
• Development Processes chapter
• When to use X for each diagram
• Short: 178 pages total
• Acts as a Reference and Explanation
• 1st Edition has a chapter on design
UML: Unified Modeling Language
UML: Unified Modeling Language

Class

Use Case

Sequence

State

Package

Activity

Deployment

Interaction
UML: Unified Modeling Language
UML: Unified Modeling Language

Class

Sequence

Package

Deployment

Use Case

State

Activity

Interaction
UML: Unified Modeling Language

Class

Sequence

Package

Deployment

Use Case

State

Activity

Interaction
UML: Unified Modeling Language

Class

Sequence

Package

Deployment

Use Case

State

Activity

Interaction

NREL | 15
UML: Unified Modeling Language

Class

Sequence

Package

Deployment

Use Case

State

Activity

Interaction
UML: Unified Modeling Language

- **Class**
- **Sequence**
- **Package**
- **Deployment**

- **Use Case**
- **State**
- **Activity**
- **Interaction**
UML: Unified Modeling Language

Class

Sequence

Package

Deployment

Use Case

State

Activity

Interaction
UML: Class Diagram Model

**Class**
- **Name**
  - attributes (member variables)
  - methods (member functions)
    - `protected_method()`
    - `private_method()`

**Abstract class**
- **Name**
  - virtual method() method()
  - {Abstract}

**Inheritance (is-a) relationship**
- **Base**
  - Derived2 is-a Base

**Object**
- `classname: objectname`

**Aggregation and Composition (has-a) relationship**
- **Whole**
  - Part
  - Whole has Part as a part; lifetimes might be different; Part might be shared with other Wholes. (aggregation)

- **Whole**
  - Part
  - Whole has Part as a part; lifetime of Part controlled by Whole, Part objects are contained in one Whole object. (composition)

**Association (uses, interacts-with) relationship**
- **A**
  - A's role

- **B**
  - B's role

- **Navigability** - can reach B starting from A
UML: A note on perspective
UML: Class Diagram

Example

**Class**

- **Name**
  - attributes
    - (member variables)
  - methods
    - (member functions)
    - public_method() + public_method()
    - # protected_method()
    - - private_method()

**Abstract class**

- **Name**
  - (Abstract)
  - virtual_method()

**Inheritance (is-a) relationship**

- **Base**
- **Derived1**
- **Derived2**

**Object**

- **classname: objectname**

**Aggregation and Composition (has-a) relationship**

- **Whole**
  - Whole has Part as a part; lifetimes might be different; Part might be shared with other Whole.
  - (aggregation)

- **Part**

**Association (uses, interacts-with) relationship**

- **A**
  - A's role
- **B**
  - B's role

**Navigability** - can reach B starting from A

- **A**
- **B**

**WCompFloris**

- **LEGEND**: str
- **LINE_PLOT_COLOR**: str
- **LINE_PLOT_LINESTYLE**: str
- **LINE_PLOT_MARKER**: str
- **deflection_model_string**: str
- **deflection_parameters**: Optional[dict]
- **ri**: FlorisInterface
- **floris_dict**: dict
- **hub_height**: float
- **rotor_diameter**: float
- **velocity_deficit_model_string**: str
- **velocity_deficit_parameters**: dict
- **yaw_angles**: ndarray

- **AEP()**
  - horizontal_contour(wind_direction: float, resolution: tuple): WakePlane
  - streamwise_profile_plot(wind_direction: float, y_coordinate: float, xmin: float, xmax: float)
  - vertical_profile_plot(wind_direction: float, x_coordinate: float, y_coordinate: float, zmin: float, zmax: float)
  - xsection_contour(wind_direction: float, resolution: tuple, x_coordinate: float): WakePlane
  - xsection_profile_plot(wind_direction: float, x_coordinate: float, ymin: float, ymax: float)
UML: Class Diagram

Example

Aggregation and Composition (has-a) relationship
- Whole has Part as a part; lifetimes might be different; Part might be shared with other Whole objects. (aggregation)
- Whole has Part as a part; lifetime of Part controlled by Whole, Part objects are contained in one Whole object. (composition)

Association (uses, interacts-with) relationship
- Navigability - can reach B starting from A

WCompBase
- LEGEND: str
- LINE_PLOT_COLOR: str
- hub_height
- rotor_diameter
- AEP(): float
- horizontal_contour(wind_direction: float, resolution: tuple): WakePlane
- streamwise_profile_plot(wind_direction: float, y_coordinate: float, xmin: float, xmax: float)
- vertical_profile_plot(wind_direction: float, x_coordinate: float, y_coordinate: float, zmax: float)
- xsection_contour(wind_direction: float, resolution: tuple, x_coordinate: float): WakePlane
- xsection_profile_plot(wind_direction: float, x_coordinate: float, ymin: float, ymax: float)
UML: Class Diagram

**Class**
- **Name**
- **attributes** (member variables)
- **methods** (member functions) + public_method() # protected_method() - private_method()

**Abstract class**
- **Name**
- virtual_method() method()

**Object**
-classname: objectname

**Inheritance (is-a) relationship**
- **Base**
- Derived1
- Derived2
- Derived2 is a Base

**Aggregation and Composition (has-a) relationship**
- **Whole**
- Part
- Whole has Part as a part; lifetimes might be different; Part might be shared with other Wholes. (aggregation)
- **Whole**
- Part
- Whole has Part as a part; lifetime of Part controlled by Whole, Part objects are contained in one Whole object. (composition)

**Association (uses, interacts-with) relationship**
- **A**
- A's role
- **B**
- B's role
- Navigability - can reach B starting from A

**Example**

```
WCompBase

LEGEND: str
LINE_PLOT_COLOR: str
hub_height
rotor_diameter

AEP(): float
horizontal_contour(wind_direction: float, resolution: tuple) : WakePlane
streamwise_profile_plot(wind_direction: float, y_coordinate: float, xmin: float, xmax: float)
vertical_profile_plot(wind_direction: float, x_coordinate: float, ymin: float, ymax: float)

WCompFois

LEGEND: str
LINE_PLOT_COLOR: str
LINE_PLOT_LINESTYLE: str
LINE_PLOT_MARKER: str
deflection_model_string: str
deflection_parameters: Optional[dict]

f: float
flons_dict: dict
hub_height
rotor_diameter
velocity_deficit_model_string: str
velocity_deficit_parameters: dict
yaw_angles: ndarray

AEP(): float
horizontal_contour(wind_direction: float, resolution: tuple) : WakePlane
streamwise_profile_plot(wind_direction: float, y_coordinate: float, xmin: float, xmax: float)
vertical_profile_plot(wind_direction: float, x_coordinate: float, ymin: float, ymax: float)
excitation_contour(wind_direction: float, resolution: tuple, x_coordinate: float, ymax: float)
excitation_profile_plot(wind_direction: float, x_coordinate: float, ymin: float, ymax: float)
```
### UML: Class Diagram

#### Class
- Name
- attributes (member variables)
- methods (member functions) + public_method() + protected_method() - private_method()

#### Abstract class
- Name (Abstract)
- virtual method()
- method()

#### Inheritance (is-a) relationship
- Base
- Derived1
- Derived2
- Derived2 is-a Base

#### Aggregation and Composition (has-a) relationship
- Whole
- Part
- Whole has Part as a part; lifetimes might be different; Part might be shared with other Whole.
  (aggregation)

- Whole
- Part
- Whole has Part as a part; lifetime of Part controlled by Whole, Part objects are contained in one Whole object.
  (composition)

#### Association (uses, interacts-with) relationship
- A
- B
- A's role
- B's role
- Navigability - can reach B starting from A
- A → B

---

### Example

- **WCompBase**
  - LEGEND: `str`
  - LINE_PLOT_COLOR: `str`
  - hub_height
  - rotor_diameter
  - `AEP(): float`
  - horizontal_contour(wind_direction: `float`, resolution: `tuple`): `WakePlane`
  - streamwise_profile_plot(wind_direction: `float`, y_coordinate: `float`, xmin: `float`, xmax: `float`): `WakePlane`
  - vertical_profile_plot(wind_direction: `float`, x_coordinate: `float`, y_coordinate: `float`, xmin: `float`, xmax: `float`): `WakePlane`
  - xsection_contour(wind_direction: `float`, resolution: `tuple`, x_coordinate: `float`): `WakePlane`
  - xsection_profile_plot(wind_direction: `float`, x_coordinate: `float`, ymin: `float`, ymax: `float`): `WakePlane`

- **WCompFloris**
  - deflection_model_string: `str`
  - deflection_parameters: `Optional[dict]`
  - fi: `FlorInterface`
  - flors_dict: `dict`
  - velocity_deficit_model_string: `str`
  - velocity_deficit_parameters: `dict`
  - yaw_angles: `ndarray`
**UML: Class Diagram**

**Class**
- **Name**
  - attributes (member variables)
  - methods (member functions) + public_method() # protected_method() - private_method()

**Abstract class**
- **Name**
  - virtual method()
  - method()

**Object**
- **classname: objectname**

**Inheritance (is-a) relationship**
- **Base**
  - Derived1
  - Derived2
  - Derived2 is-a Base

**Aggregation and Composition (has-a) relationship**
- **Whole**
  - Part (aggregation)
- **Whole**
  - Part (composition)

**Association (uses, interacts-with) relationship**
- **A's role**
- **B's role**

**Example**

```
WakeProfile
  - values: x1
  - resolution:Tuple values x1 x2

WCompBase
WCompFloris
WCompFoxes
WCompPyWake
```

Navigability - can reach B starting from A
UML: Sequence Diagram

Floris \rightarrow \text{Solver} \rightarrow \text{DeflectionModel} \rightarrow \text{DeficitModel} \rightarrow \text{CombinationModel}

- \text{Floris} \rightarrow \text{Solver} \rightarrow \text{DeflectionModel} \rightarrow \text{DeficitModel} \rightarrow \text{CombinationModel}
- \text{Floris} \rightarrow \text{Solver} \rightarrow \text{DeflectionModel} \rightarrow \text{DeficitModel} \rightarrow \text{CombinationModel}

\text{loop}
- \text{function()}
  - \text{deflection field} \rightarrow \text{function()}
  - \text{velocity field} \rightarrow \text{Function()}

\text{Wind direction}

\text{Rotor grid only}
UML: Sequence Diagram
Use UML and anything else to get your message out there!
Diagramming in the development workflow
Documentation as Code (Docs as Code) refers to a philosophy that you should be writing documentation with the same tools as code:

• Issue Trackers
• Version Control (Git)
• Plain Text Markup (Markdown, reStructuredText, Asciidoc)
• Code Reviews
• Automated Tests

This means following the same workflows as development teams, and being integrated in the product team. It enables a culture where writers and developers both feel ownership of documentation, and work together to make it as good as possible.

Write the Docs
https://www.writethedocs.org/guide/docs-as-code/
UML: Class Diagrams

**Example**

**INCORRECT**

**Aggregation and Composition (has-a) relationship**
- **Whole** has **Part** as a part; lifetimes might be different; Part might be shared with other wholes. (aggregation)
- **Whole** has **Part** as a part; lifetime of Part controlled by Whole, Part objects are contained in one Whole object. (composition)

**Association (uses, interacts-with) relationship**
- A's role
- B's role

Navigability - can reach B starting from A
In Practice: Automated Tools

Treat diagrams like other software infrastructure:
• Incorporate it into the development process
• Build tools, trust them, and lean on them heavily
• Automate – let the computers do the hard work

- Graphviz
- Mermaid
- doxygen
- pylint
- pyreverse
In Practice: Doxygen

https://www.doxygen.nl

Doxygen is a widely-used documentation generator tool in software development. It automates the generation of documentation from source code comments, parsing information about classes, functions, and variables to produce output in formats like HTML and PDF. By simplifying and standardizing the documentation process, Doxygen enhances collaboration and maintenance across diverse programming languages and project scales.
In Practice: Mermaid

https://mermaid.js.org

Mermaid
Diagramming and charting tool
JavaScript based diagramming and charting tool that renders Markdown-inspired text definitions to create and modify diagrams dynamically.

- Easy to use!
  Easily create and render detailed diagrams and charts with the Mermaid Live Editor.

- Integrations available!
  Use Mermaid with your favorite applications, check out the integrations list.

- Award winning!
  2019 JavaScript Open Source Award winner for "The Most Exciting Use of Technology".

- Mermaid + Mermaid Chart
  Mermaid Chart is a major supporter of the Mermaid project.
In Practice: Mermaid

GitHub Discussion – same for any GitHub-flavored Markdown product

Scope, architecture, and dependency structure #23

Unanswered  rafmudaf asked this question in Ideas

Write Preview

```
/* mermaid
classDiagram
windID "-- loader
windID "-- Plant
windID "-- Turbine
class loader{
  +dict (load)
  +validate()
}
class Plant{
  +common
  +energy_resource
  +site
  +Turbine
  +wind_energy_system
  +wind_farm
}
class Turbine{
  +ontology
```

Architecture

The proposed architecture below closely follows the scope stated above. Note that the top-level windID object in the diagram is intended to be a namespace that provides access to the objects beneath it.

The ontology definition and its YAML representation are defined in the Plant and Turbine modules via YAML schemas (the windID.proto and windID.turbine modules in the software library). The module for loading and validating the input files is available through windID.loader and it contains functionality similar to the existing load, load_yaml and validate_yaml.
In Practice: Mermaid

Sphinx-based documentation – add diagrams anywhere including in API docs
In Practice: pyreverse

In Practice: pyreverse

Static analysis of Python source code to generate package and class diagrams
In Practice: AppMap

https://appmap.io/product/appmap-in-the-code-editor
Summary

1. Read this book
2. Use these tools
3. Directly communicate your software design
Simplicity
3 elements used to create 3 spaces

Complexity created through excessive agglomeration
12 elements required to create 12 spaces

Complexity created through informed simplicity
3 elements combined to create 12 spaces

101 Things I Learned in Architecture School
Thank you!

rafael.mudafort@nrel.gov
@rafmudaf
rafmudaf.github.io/communicating-design