



university of  
groningen

faculty of science  
and engineering

# Dependency Graph Creator Engine

Software Maintenance and Evolution

*<https://github.com/richardswesterhof/pyne>*

## **Authors:**

Job Heersink (s3364321)  
Richard Westerhof (s3479692)

## **Supervisors:**

Mohamed Soliman  
Filipe Capela

University of Groningen  
The Netherlands  
December 1, 2020



# CONTENTS

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>User Guide</b>	<b>3</b>
<b>3</b>	<b>Design</b>	<b>4</b>
3.1	Pyne-demo . . . . .	5
3.2	Pyne-cli . . . . .	5
3.3	Pyne-api . . . . .	5
<b>4</b>	<b>Comparing Pyne to Structure101</b>	<b>7</b>
4.1	Running Pyne on Tajo . . . . .	7
4.2	Running Structure101 on Tajo . . . . .	7
<b>A</b>	<b>Troubleshooting</b>	<b>9</b>
<b>B</b>	<b>Change Log</b>	<b>10</b>

# CHAPTER 1

## INTRODUCTION

Software maintenance has always been of great concern to software engineers. An application is almost never perfect on deployment and needs regular bug-fixes and enhancements to the system. To help with the software maintenance of an application, several different techniques have been developed.

One of these techniques is a dependency graph. A dependency graph is a directed graph that represents dependencies between objects of some application domain, where the nodes represent packages or classes of a software system and the edges the relation between them.

In this document we will compare an incomplete existing dependency graph creator called `pyne` with another widely used dependency graph creator engine called `structure101`, which we will consider as complete. In addition to that we will find the missing functionality in `pyne`, which `structure101` does offer, and implement this functionality in `pyne`.

The document is structured as follows: Chapter 2 will show how `pyne` should be installed and used. Chapter 3 will discuss the design of `pyne` itself. In Chapter 4 we will apply `Pyne` and `structure101` to a large java application called Apache Tajo and finally in Chapter 5 and 6 we will discuss and conclude the project.

# CHAPTER 2

## USER GUIDE

### Installing

To install Pyne, Java JDK 11+ and Maven is required. The installation itself is rather straightforward: Clone or download our fork of the Pyne repository:

```
git clone https://github.com/richardswesterhof/pyne.git
```

Install Pyne's dependencies and build an executable jar by running the following command inside Pyne's root directory:

```
mvn install
```

### Executing

After the application has been built, one can use this application via CLI. This can be done by running the following command from the root directory:

```
java -jar  
  ↪ <path-to-jar>/pyne-cli-1.0-SNAPSHOT-jar-with-dependencies.jar  
  ↪ <github-url>
```

Where the <path-to-jar> is the path to the built jar file (by default this is in <pyne\_root>/pyne-cli/target/) and <github-url> is the url of the project you want to create a dependency graph for. For example, this would be the command to create a yearly dependency graph for the commits of Apache Tajo from 2015 to November 2020:

```
java -jar  
  ↪ pyne-cli/target/pyne-cli-1.0-SNAPSHOT-jar-with-dependencies.jar  
  ↪ https://github.com/apache/tajo -s "2015-01-01" -e  
  ↪ "2020-11-21" -p "YEAR"
```

# CHAPTER 3

## DESIGN

Here we will go into more detail about the design of pyne itself. The project has been analysed using the existing documentation, the source code and the dependency graph created by structure101 (figure 3.1).

This application is divided into 3 different packages: pyne-demo, pyne-cli and pyne-api. The relations between these packages and their classes can be seen in figure 3.1.

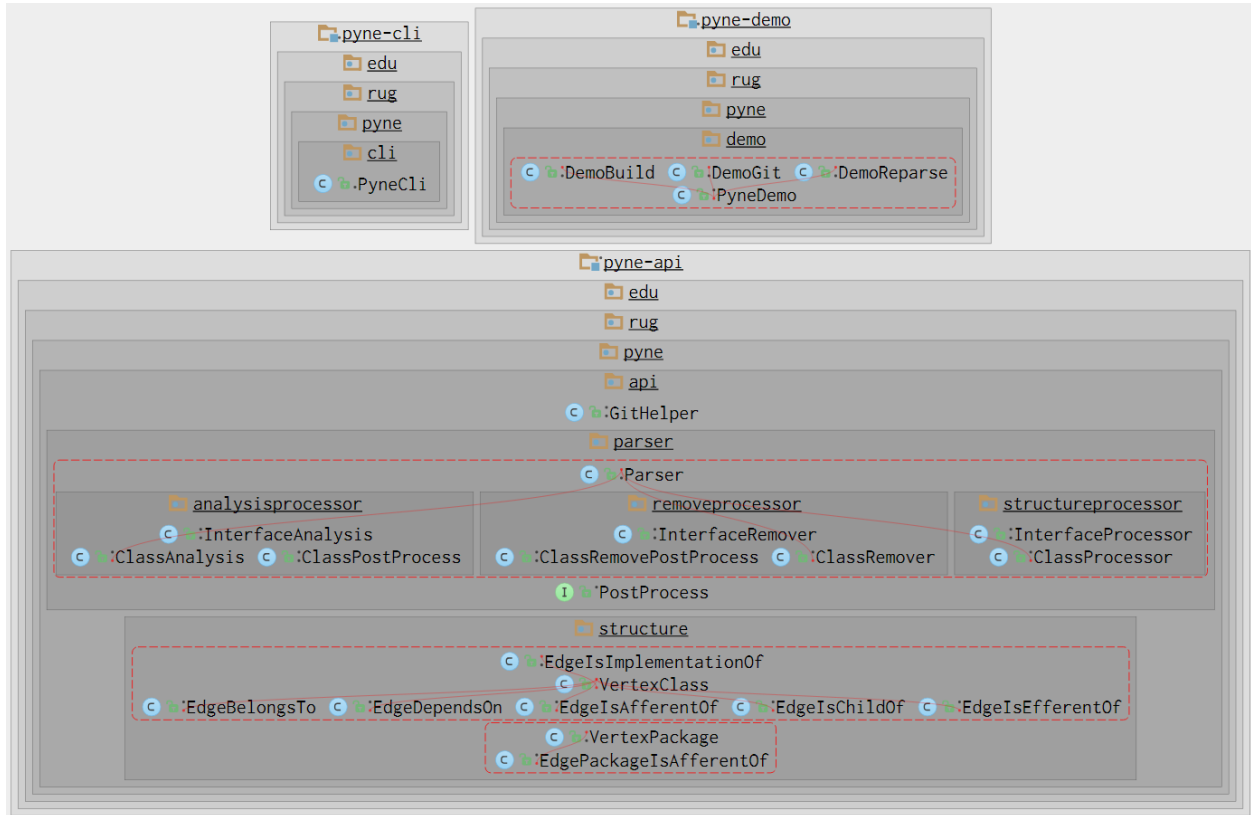


Figure 3.1: The full graph of the relations between the classes of pyne created by structure101

The purpose of these 3 packages will be briefly explained here. `pyne-api` will be explained in more detail, since this is the main component of the system.

## 3.1 PYNE-DEMO

This package acts as a testing ground for `pyne`. Most of the data in this package has been hard-coded to work only with a specific repository that was probably present on the creators local machine. Since we do not know which repository they used for testing and it is not present within the project, This package will not be able to work and we will consider this package deprecated.

## 3.2 PYNE-CLI

This package acts as a layer of communication between the user and the `Pyne-api` via a CLI (Command Line Interface). Given a git repository as a program argument, it'll automatically start communicating with the `pyne-api` and create a dependency graph in `.graphml` format for each commit. A number of options can be given to the `pyne-cli` program. The first required argument is of course the repository itself, but one can also specify the starting and end date to parse from, the interval between each commit and an input and output directory.

## 3.3 PYNE-API

This package is the heart of the program and is the one that is actually responsible for the dependency graph creation. This package is again subdivided into 2 packages: `parser` and `structure`, and contains one class: `GitHelper`.

The `GitHelper` is a part of the API responsible for git related functionality like cloning a repository in a temporary location, parsing a commit or processing a commit. Most of the functionality of the `GitHelper` depends on the `parser` class to create the dependency graph.

The `structure` package does not contain any functionality, instead it provides abstract classes to build a dependency graph from. Special edges have been created for different kinds of relations: `BelongsTo`, `DependsOn`, `AfferentOf`, `ChildOf`, `EfferentOf`, `ImplementationOf` and `PackageisAfferentOf`. Special vertexes have also been created to represent either a package or a class.

The `parser` package again consist of three sub-packages: `analysisProcessor`, `removeProcessor`, and `structureProcessor`. It also contains the `Parser` class and the `PostProcess` interface.

All communication with this package goes through the `parser` class. This class is used to keep track of the files that have changed in the selected commit, and it holds lists of all

types of processor classes, each of which are responsible for analysing the code in a different way.

The parser class will then in turn communicate with the analysis processor package, remove processor package and structure processor package to create a dependency graph. These packages provide the following functionality: The remove processor package will take care of removing nodes or edges from the graph if classes or relations have been removed or changed. The structure processor package takes care of adding nodes to the dependency graph and the analysis processor package analyses the different classes and packages and their relations to each other.

## CHAPTER 4

# COMPARING PYNE TO STRUCTURE101

To compare Pyne to Structure101, we will test them both on a large existing system. For this we have selected Apache Tajo ([www.tajo.apache.org](http://www.tajo.apache.org)). Note that in the commands we show, we renamed the generated pyne-cli jar to simply `pyne-cli.jar` for brevity.

### 4.1 RUNNING PYNE ON TAJO

We will start by running Pyne on Apache Tajo. The command we used to get a graph for the latest commit of Tajo is as follows:

```
java -jar pyne-cli.jar https://github.com/apache/tajo -s  
  ↳ "2020-05-10" -e "2020-05-12" -p "DAY"
```

This creates a `.graphml` file<sup>1</sup>, but unfortunately this does have some issues. See Appendix A. This is the only format Pyne can currently export, and opening a `.graphml` does require users to install a specialized program for this purpose. The nice thing about Pyne though, is that it can generate this graph without the need to compile the target system first, which in the case of Tajo, proved to be a difficult task on its own as we will see in the next section.

### 4.2 RUNNING STRUCTURE101 ON TAJO

Next we will run Structure101 on Apache Tajo. Since Structure101 uses a GUI instead of a CLI, we followed the steps in the GUI to create a Structure101 project. For ease of use, we have provided this Structure101 project in our github repository under `structure101/tajo_structure101.java.hsp`. As mentioned before, the downside of Structure101 is the need to compile the target system before it will allow you to import it. However, by being able to use a maven pom file, Structure101 does have the advantage of being

---

<sup>1</sup>available on our github repository, under `graphs/tajo_dependencies_pyne.graphml`



able to find dependencies to external packages instead of only internal packages, like Pyne does.

Structure101 has more options for exporting as well, allowing users to export to `.png`, `.jpeg`, and "Graph as XML", which sounds a lot like `.graphml`, but they are not the same, and in fact the program<sup>2</sup> we used to open the `.graphml` files did not display this file correctly.

---

<sup>2</sup><https://www.yworks.com/products/yed>

# APPENDIX A

## TROUBLESHOOTING

Here we present several problems encountered and how we tried to solve them.

1. **Bug in graph creator:**

When I open a graph in a graph editor like draw.io or yEd graph editor, only one square appears. But the file is 2.25mb. Turns out that all the squares are stacked upon each-other, but the relation between the squares is still there (which is visible from the neighbours tab in yEd graph editor).

It seems that the squares itself do not contain any data on the classes they represent. They do not have any value assigned to them. Furthermore, opening any graph will give a warning about the 'linesOfCode' property being of type long, which apparently isn't supported. I sincerely hope that an integer would be enough to count the amount of lines of code anyway, so we'll change that.

2. **Default cli values:**

The default options for cli are not the best choice. Namely the period option is set to days by default, which will create a separate graph for each day. The start date is set to 5 periods (so 5 days on default) from the end date (which is the current date by default). It is relatively rare for a git repository to have commits from the past 5 days. We shall therefore change the default values.

3. **Error handling:**

In a few cases, where something goes wrong or no graph is created, no error is thrown and the user does not know why no graph has been made. For that reason more error logging statements need to be added.

4. **Java Version:**

Pyne requires java version 11, however to compile Tajo version 1.8 is needed. This leads to confusing errors if you try to compile Tajo for yourself after upgrading to java 11 to compile Pyne.

# APPENDIX B

## CHANGE LOG

ID	Author	Student Number	Email Address
JH	Job Heersink	s3364321	j.g.heersink@student.rug.nl
RW	Richard Westerhof	s3479692	r.s.westerhof.2@student.rug.nl

Table B.1: The authors that contributed to this document

Ver.	ID	Date	Revision
0.1	JH	21-11-20	Create User guide.
	JH	21-11-20	Create Design section.
	JH	21-11-20	Create problem (troubleshooting) section.
	RW	22-11-20	Added classes and dependency graph to design
	RW	22-11-20	Revised User guide
	RW	22-11-20	Revised problem section

Ver.	ID	Date	Revision
0.2	JH	1-12-20	Reformatted the document.
	JH	1-12-20	Added frontpage.
	JH	1-12-20	Added introduction.
	JH	1-12-20	Rewrote design section.
	JH	1-12-20	Moved and renamed troubleshooting section to appendix.
	RW	1-12-20	Create chapter 4.
	RW	1-12-20	Add "java version" item to Troubleshooting appendix.