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 8, 2020

Contents

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 [1] 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 [1], the source code and the structural and dependency graphs created by Structure101 (figure 3.1).

3.1 Requirements

According to the documentation [1] of the Pyne project, the project has the following requirements:

1. **The program should be able to create a graph from Java source files using git.**
2. The output graph should be the same as that of Arcan.
3. The program should parse Java source files in such a way so it can find the different Java classes and packages.
4. The program needs to be able to use Git.
5. The program should provide a command line interface.

When inspecting the functionality of the program, one can see that all these requirements have been met. When inspecting the source code of the program and the documentation, one can even see how they implemented the requirements.

From the documentation it was made clear that most of the requirements were met using external dependencies. For example: Requirement 2 was met using Apache Tinkerpop[1] for graph creation, which is the same technology Arcan[2] uses.
Requirement 3 was met using Spoon[3], an extensive java source code parsing library. Lastly,

[1] `https://tinkerpop.apache.org/`
[2] `https://essere.disco.unimib.it/wiki/arcan/`
[3] `http://spoon.gforge.inria.fr/`

Requirement 4 was met using Jgit[4], a java git library.

In addition to that, using Structure101, we found that Requirement 5 is met using Apache Commons Cli[5] to provide a cli interface. Furthermore, Syncleus Ferma[6] was used as an extension to Tinkerpop. More information on this can be found in section 3.3.

3.2 Implementation

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.

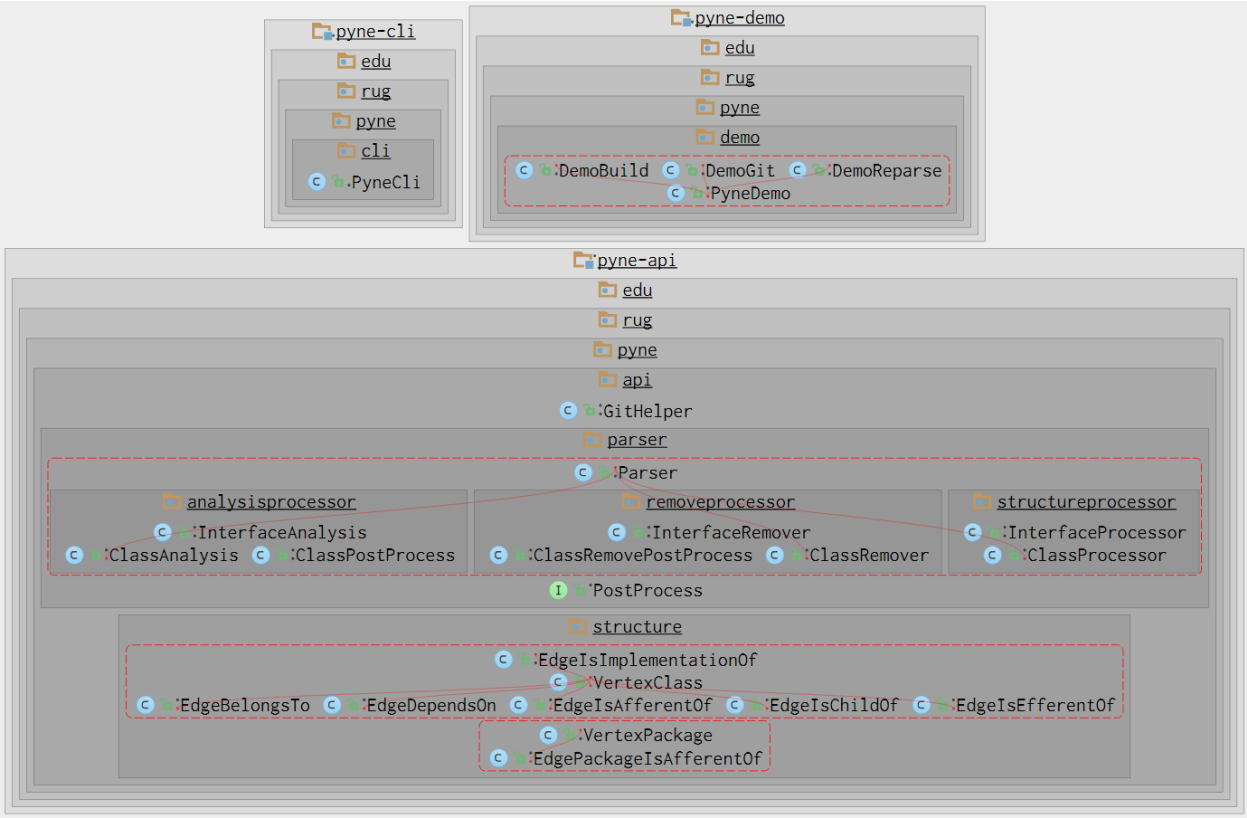

Figure 3.1: The full graph of the relations between the classes of Pyne, created by Structure101

explain figure 3.2

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

[4] https://www.eclipse.org/jgit/
[5] https://commons.apache.org/proper/commons-cli/
[6] https://github.com/Syncleus/Ferma

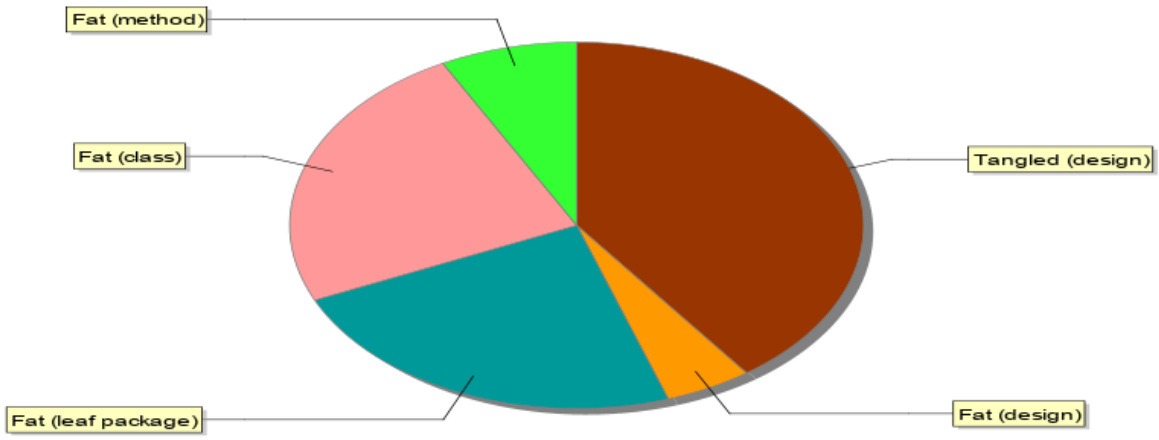

Figure 3.2: The XS diagram of the structure of Pyne, created by Structure101

3.2.1 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.

3.2.2 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 will 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.

3.2.3 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, as can be seen in figure 3.3.

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

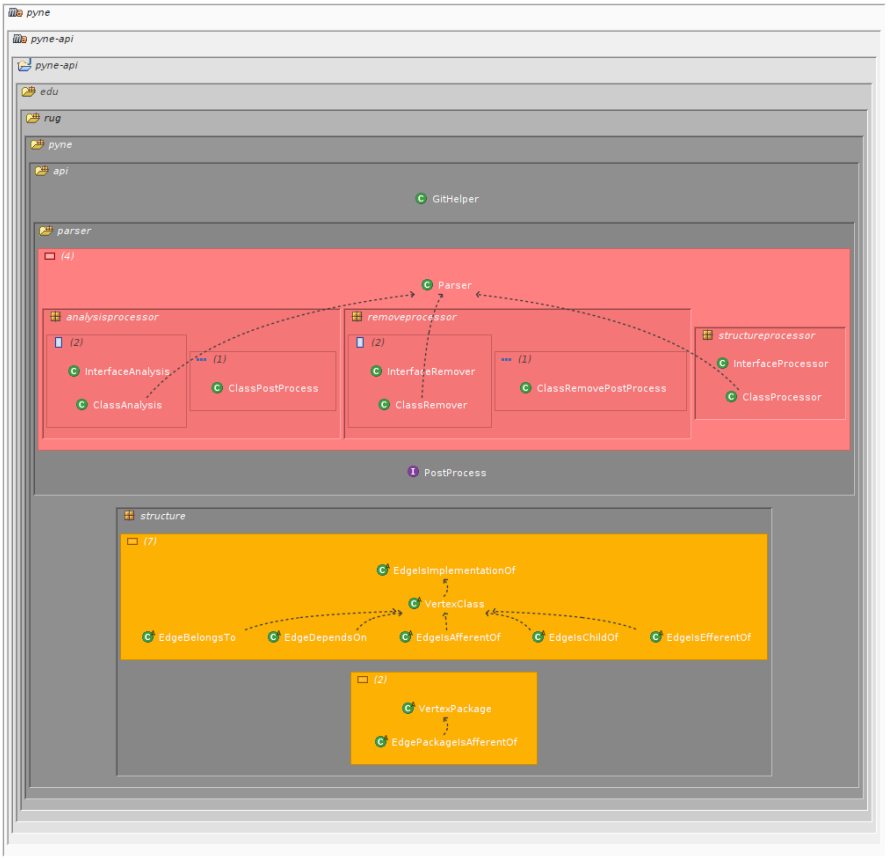

Figure 3.3: The graph of the relations between the classes of the pyne-api package created by Structure101

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.

3.3 External dependencies

The main external dependencies used by the project are:

- **Apache Tinkerpop**
  A flexible graphic framework. This library was chosen since Arcan, the software this project is suppose to mimic, also uses this framework.
- **Spoon**
  A java source code parsing library. This library was chosen so that a java parser did not need to be build from scratch[1].
- **Jgit**
  A java library that provides git functionality. This library was chosen so that the program is able to collect the source files using git.
- **Apache Commons CLI**
  A java library that provides CLI parsing and some basic CLI functionality. This library was used to not have to write a CLI parser from scratch.
- **log4j**
  A logging library. Used to log actions in the program.
- **Ferma**
  An extension to Apache Tinkerpop. Used to help building the graph.

Most of these dependencies were noted in the report [1], like Apacke Tinkerpop, Spoon and Jgit. Others, like Apache Commons Cli, log4j and Ferma, were found using Structure101.

In figure 3.5 and figure 3.4 you will find the graphs depicting the relation between the packages of Pyne and the dependencies. Note that the used external dependencies have not been shown for pyne-demo. This is because pyne-demo does not provide any graph creation functionality, only a ”demo” and it uses approximately 20 extra external dependencies that are not used in the rest of the project. It would therefore be unreadable and not really relevant to the workings of Pyne itself.

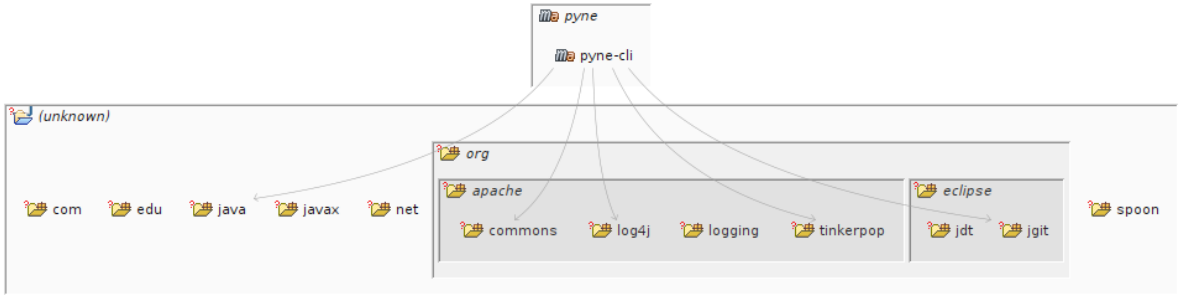

Figure 3.4: The graph of the external dependencies of pyne-cli, created by Structure101

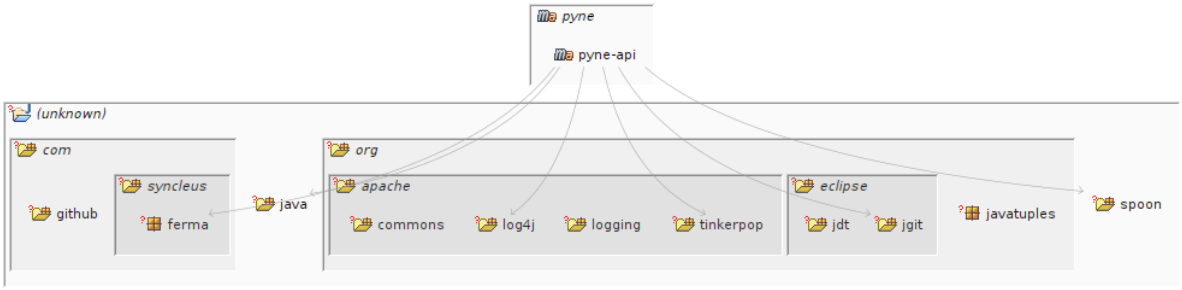

Figure 3.5: The graph of the external dependencies of pyne-api, created by Structure101

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`). 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[1], 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. We decided to use yEd Graph Editor[2] for this purpose. This has the advantage of being able to automatically rearrange the graph. Using this feature, we obtained figure 4.1.

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

[1] available on our github repository, under `graphs/tajo_dependencies_pyne.grahpml`

[2] `https://www.yworks.com/products/yed`

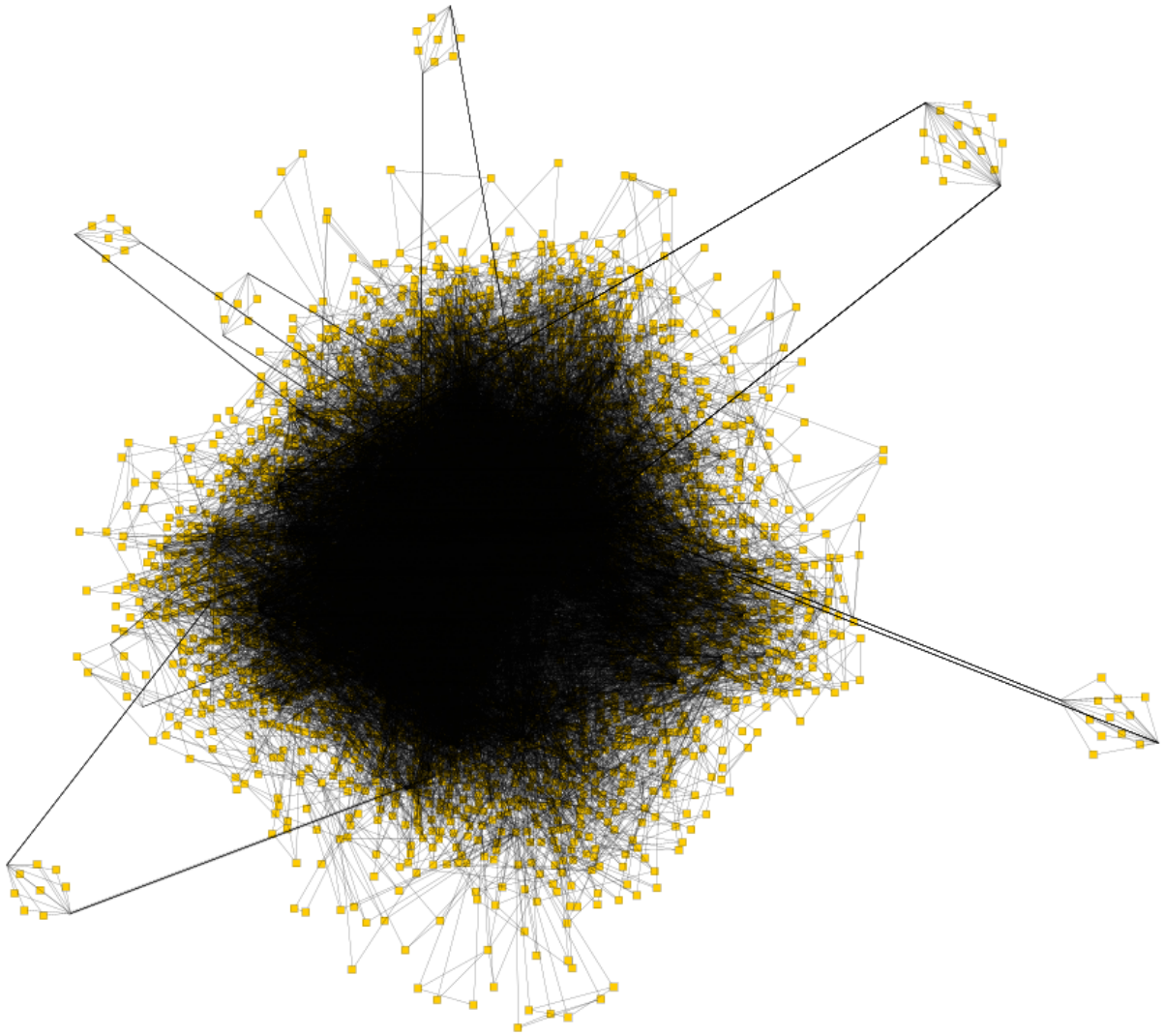

Figure 4.1: The automatically rearranged graph of Tajo, created by Pyne and rearranged by yEd

`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. This was difficult since we had first switched to java 11 to compile and run Pyne, but Tajo requires java 8. This was however not indicated clearly, so it took some time to figure out.

However, by being able to use a maven pom file, Structure101 does have the advantage of being 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 we used to open the `.graphml` files did not display this file correctly.

Figure 4.2 shows what an exported image from Structure101 looks like.

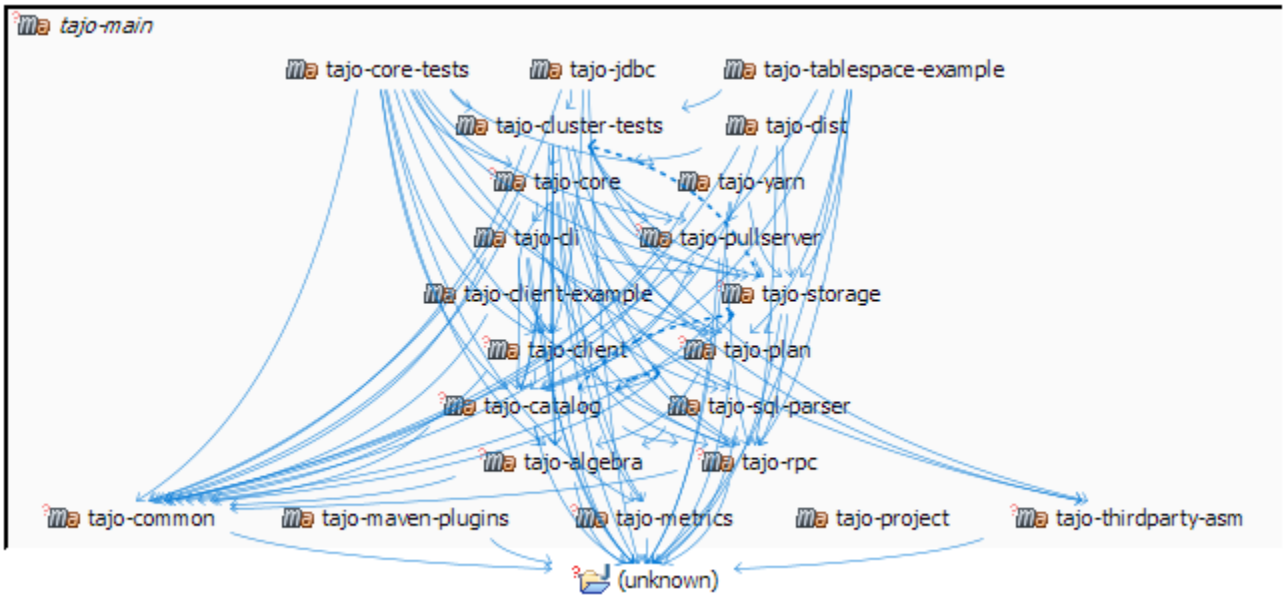

Figure 4.2: The graph of Tajo, created by Structure101

4.3 Comparing the feature sets

Feature	Pyne	Structure101
GUI	NO	YES
IDE plugin support	NO	YES[3]
Needs compiled source	NO	YES
Shows external dependencies	NO	YES
Open source	YES	NO
Freely available	YES	NO
Multiple OS support	YES	YES
Allows direct analysis of remote repositories	YES	NO
`.graphml` export	YES	NO
`.png` export	NO	YES
`.jpeg` export	NO	YES

Table 4.1: Comparing the features of Pyne and Structure101

[3] though this is quite a bit more limited than using the full program

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	21-11-2020	Create User guide.
	JH	21-11-2020	Create Design section.
	JH	21-11-2020	Create problem (troubleshooting) section.
	RW	22-11-2020	Added classes and dependency graph to design
	RW	22-11-2020	Revised User guide
	RW	22-11-2020	Revised problem section

Ver.	ID	Date	Revision
0.2	JH	01-12-2020	Reformatted the document.
	JH	01-12-2020	Added frontpage.
	JH	01-12-2020	Added introduction.
	JH	01-12-2020	Rewrote design section.
	JH	01-12-2020	Moved and renamed troubleshooting section to appendix.
	RW	01-12-2020	Create chapter 4.
	RW	01-12-2020	Add "java version" item to Troubleshooting appendix.

Ver.	ID	Date	Revision
0.3	RW	03-12-2020	Incorporate TA feedback in document.
	RW	07-12-2020	Improve internal structure of document and improve consistency.
	RW	08-12-2020	Add XS diagram and graphs of Tajo created by both Structure101 and Pyne.

References

[1] Patrick Beuks. *Building a dependency graph from Java source code files*. 2019.