**PIP**

**By Augustine, Arif, Sanjog, Masinde**

**Åbo Akademi University**

**Abstract**

Pip is the your friend when doing any python project. It is availble in all platfroms, i.e windows, linux and mac. It is used in installing different python packages. In this report we will give an overview of this open source project. We will view different stakeholders, the models of the project and we will conclude the way we see it from our findings.

**Introduction**

pip is a recursive acronym that stands for Pip installs Packages. It is a package management system for installing and managing Python software packages. It is an open source software by the Python Packaging Authority (PyPA) initiated and led by Donald Stufft and was initially released on 4th April, 2011. The most recent stable release was on 6th November, 2016.

pip is the defacto package manager for software packages developed with Python and ships with most Python distributions. As such its user base is almost as large as the entire Python community of developers. It is notable for the ease of its command-line interface, which simplifies the installation and uninstallation of Python software packages.

**Stakeholders**

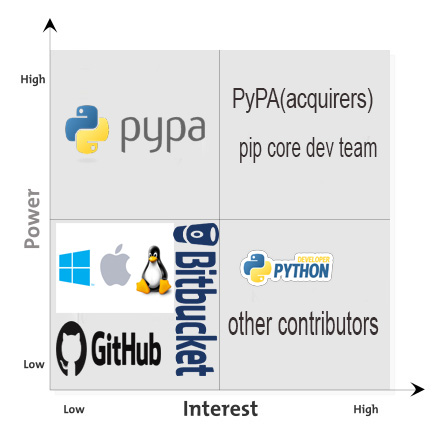
Stakeholders are people who have direct impact with the product/project of the organization.

**Over view**

The following bullets explain in brief different stakeholders who are involved in pip.

* **Acquirers**: The Python Packaging Authority is the acquirer of pip.
* **Assessors:** The Python Packaging Authority is responsible for assessing and approving contributions from pip contributors to the source code.
* **Communicators**: The communicators maintain communication with other stakeholders. They analyze and make documentation of the system. The Python Packaging Authority (PyPA) is the communicators of pip.
* **Developers**: PyPA developers and other contributors from Python Developers community.
* **Maintainers**: pip is based on core developers from PyPA and other contributors from Python Developers community. When the system is operational the PyPA takes care of it.
* **Production Engineers:** Production engineers are the core developers.
* **Suppliers:** PIP works on different operating system such as Unix/Linux, macOS, and Windows. These stakeholders supply appropriate software and hardware in which the system runs.
* **Support Staff:** The Python Packaging Authority (PyPA) is a working groupthat supports many of the relevant projects in Python packaging. Besides this support staff for pip also consist of github and bitbucket staff and many other volunteers also.
* **System Administrators:** Python organisation is responsible for the system administration.
* **Testers:** The pip development team is responsible for testing the software to ensure that it meets the requirements.
* **Users:** Python developers

**Stakeholders Power/Interest Grid**

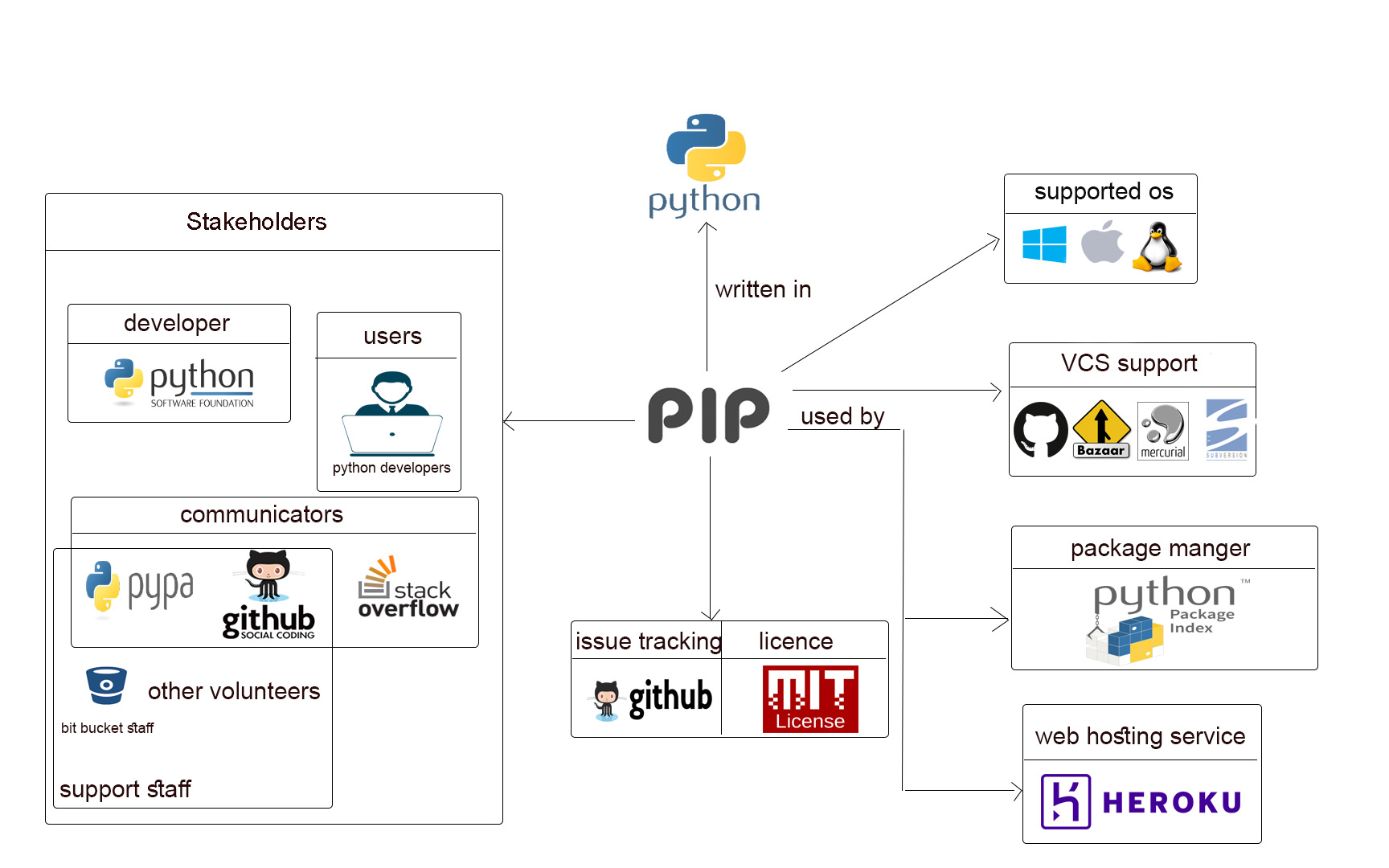
****

**Fig 1. Power Interest Grid**

Figure 1 above visualizes the level of interest and power of the different stakeholders in the pip project. From the figure, PyPA at the top right corner of the grid, as the acquirer and represented by Donald Stufft, has the highest interest and power in the project. The success of the project is of utmost importance to it and it has the final say on important decisions concerning the project. At the top left corner of the grid is PyPA again, this time as the accessor. In this role, it has very little or no interest in the project but has significant power regarding decision making in the direction of the project. At the bottom right corner of the grid are the developers and maintainers, which include PyPA core developers, led by Donald Stufft, and other contributors from the Python developers community, and the users, which include Python developers. These do not have much say in the decision making concerning the project but they have relatively high interest in the success of the project. Lastly, at the bottom left corner of the grid are the communicators, testers, support staff, etc. Although these play significant role in the project’s life cycle as stakeholders, they have relatively low power and interest in the project.

**The Context Viewpoint**

The Context viewpoint describes the relationships, communication interfaces and channels, interdependencies between a system and its environment, which includes users, developers, maintainers, other systems, data input source, etc. Its concern is to describe the nature of the contents of the system’s environment (context), how the system relates with the different players and systems within its context, the nature of the interfaces between the system and its environment, what the system needs from its dependencies and what it does and does not do (scope).



**Fig 2. Context Model**

The fig 2 above is a pictorial overview of pip’s context. Pip is written in Python and has support for the three major operating system platforms, Windows, Mac and Linux. It depends on github for issue tracking and version control and is licensed under the MIT license. Pip installs python packages from a repository of software for the Python programming language called the Python Package Index (PyPI). It was developed, and is being maintained by PyPa core developers and contributors from the Python developers community. PyPa is responsible for moderating and supervising inputs from contributors. As a package manager for the Python programming language, pip is used by Python developers to install and manage Python packages in their Python-based projects.

**The Development Viewpoint**

The development viewpoint refers to a description of the architecture that supports a system’s development process. It is concerned with module structure and organization, module dependencies, standardization of design and testing, and codeline organization and consistency for technical integrity. This directly addresses the concerns of the stakeholders within the development and testing groups.

Pip is basically a command line based utility software for installing, uninstalling, and managing software packages developed with the Python programming language. Pip was originally developed to be self-contained, implementing all the functionalities it needed to eliminate the need for dependencies and its attendant challenges and problems. However, with version 1.5, pip introduced the idea of dependencies replacing some codes implemented internally with reusable libraries from PyPi in order to leverage the benefits of reusing libraries other than re-inventing the wheel.

The approach to dependency in pip is different from the traditional way (that is via *install\_requires*). In pip, to avoid dependency-related issues, dependencies are bundled with the source. So pip’s source directory contains two main directories: **\_internal** and **\_vendor.**

**\_internal** : This contains pip’s internal API and helper functions. All the implementations of pip’s functionalities as a package manager are contained in this directory. It contains several directories and files one of which is the commands directory, which contains implementations of all pip’s commands.

**\_vendor** : This contains basically pip’s bundled dependencies (implementations of the libraries from PyPi on which pip depends). Each of these dependencies with its version is listed in the file vendor.txt also contained in this directory. This list is used for automatic dependency update.

INSTALL, UNINSTALL, SHOW, CHECK, FREEZE, DOWNLOAD, LIST, …

COMMANDS

UTILITY

…

VCS

REQ

OPS

UTILS

MODELS

PYPI LIBRARIES

VENDOR (DEPENDENCIES)

Figure 3 pip module structure

The figure 3 above is a simplified model of pip’s module structure describing the different layers of dependencies. Pip commands implementations are dependent on several internal utility modules and helper functions, which make use of third party libraries from PyPI. These commands also make use of the external libraries from PyPI directly.

The open source project pip is maintained by The Python Packaging Authority (PyPA), and is developed by Python core developer and other community hosted in GitHub. Any contributor can contribute in development of pip through GitHub.

In terms of software development Model-View-Controller (MVC) is a design pattern for software development and Python uses MVC architecture. The concept of MVC is, it split the full system into three interconnected segments, and that are a Model, View, and Controller. The Model part represents the domain, and domain is used for data storing and data retrieving. The View represents to the user and the Controller is used to make communication between the Model and The View.

However, to contribute in pip development the contributor needs to submit a Pull Request against the master branch in GitHub. The contributor must provide an explicit description about the tasks what he/she going to do. The task is tested locally first.

**Automated testing:** all kinds of pull requests and push to ‘master’ are tested in Travis, and Travis is based on .travis.yml file. Jenkins CI is used for certain versions of Python on centos and windows.

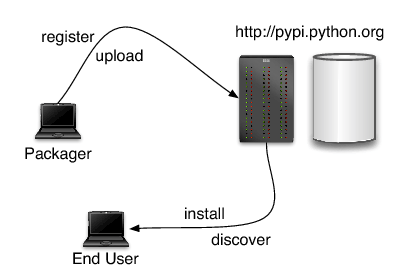
**Tests running:** to run the test an operating system needs to have subversion, git, bazaar, and mercurial. And Python requirements are a virtualenv, tox or pytest, scripttest, and mock.



**Fig. Local testing system**

**Pip usage**

The Python Package Index (PyPI) is a repository of software for the Python. It is the host of different Python packages and their dependencies. At present there are 119685 packages on PyPI. Donald Stufft is a core developer of pip, and PyPI is solely developed and maintained by him.



**Figure: Package management process in PyPI using pip**

Packager can register/upload their own work in PyPI using pip command and other users can browse existing projects. Source can be uploaded to an existing package. Pip uses PyPI as a default host for the packages and their dependencies.

Some important pip commands:

|  |  |
| --- | --- |
| Command | Action |
| Pip install | Install packages |
| Pip uninstall | Uninstall packages |
| Pip freeze | Output installed packages in requirements format |
| Pip list | List installed packages |
| Pip show | Show information about installed packages |
| Pip search | Search PyPI for packages |
| Pip zip | Zip individual packages |
| Pip unzip | Unzip individual packages |
| Pip bundle | Create pybundles |
| Pip help | Show help for commands |

**Quality Attribute Analysis**

Pip is known for its ease of use. Pip ships with most Python distributions so it requires zero installation time and effort. Its ease of use is mostly shown in the simplification of installation and uninstallation of Python software packages which can be done simply with the command ‘pip install’ <package name> and ‘pip uninstall <package name> respectively. It is also shown in the fact that pip can be used to update pip itself since it is a PyPI package like all other Python packages. The command ‘pip install –upgrade pip’ upgrades pip.

Another very useful feature of pip is the possibility of managing a list of several packages and corresponding version numbers through a ‘requirements’ file. With this feature, an entire group of packages can be efficiently created in multiple environments, physical or virtual. This can be achieved with the following command:

‘pip install –r requirements.txt’ where requirements.txt is a well formatted list of packages and their corresponding version numbers to be installed/managed.

**Architectural structures**

* Modules structures
* Component-and-connector structures
* Allocation structures

**Pip Modules**

The modules shows the units of implementation, or shows what is supposed to be implemented

Decomposition

class

Uses

Layered

**Component-and-connector**

Concurrency

Shared data

Client-Server

Process

**Allocation**

Work Assignment

Deployment

Implementation

**Conclusion**