SOFTWARE REQUIREMENT SPECIFICATION

FOR PYTHON TO C++ TRANSLATOR

Submitted by:

1. Abhishek Kumar (140101003)
2. Mehul Sawarkar (140101040)
3. Nikhil Yadala (140101083)
4. Venkatesh Nagaraju (120101078)

Dept. Of Computer Science and Engineering,

Indian Institute of Technology Guwahati(IIT GUWAHATI).

19Th February, 2016.

Contents:

1.0. Introduction

1.1. Purpose

This Document is intended to present a detailed description of Python to C++ translator.

It explains the objective, interfaces, different sub parts of project as well as constraints under

which it is expected to perform.This document is intended for both developers and as well as other

adversaries included course TAs and Instructor.

1.2 Scope of Project

This project will be a Python to C++ translator capable of converting following features of Python language to good quality optimized C++ code:

a)Object Oriented Programming support

a.1)Classes

a.1.1)Classes can be declared with only methods and

variables.Other than "__init__" no other standard python methods are supported though same effect can be achieved by simple methods.

a.1.2)Dynamic object allocation is not supported.

a.1.3)Python specific design pattern implementations are not

guaranteed to work if they don't comply with above specifications.

a.2)Differentiate between Public and Private methods

a.2.1) Though C++ does differentiate between public /private and protected members in python all members are Public by default as it expects maturity from Programmers instead of Compilers.So this translator will convert all members to public programmer is responsible

for any side-effects.

a.3)Constructors and Destructors

a.3.1)Python "__init__" is translated to constructors.This translator relies on default destructors of C++ language.

a.4)Inheritance

a.4.1)Our Translator does not supports multiple inheritance though prrovided by C++ to avoid diamond ambiguity problem.

b)Make code strictly typed, inferring data types of python variables and return types of functions is a challenging task

c)Keeping track of variables and their scope

d)Implement support for recursion

e)Input, Output support

1.3) Specifications:

External Interface specifications:

Translator takes a folder address as command line argument and produces exactly same directory structure with corresponding C++ supporting libraries.

Also translator needs to create a log file giving detailed description of how much code was translated in each file, which portions were left untouched etc.

It should also give performance analysis and specify bottleneck code.

Functional Specifications:

1)Translator should convert Python modules to C++ and should be capable of segregating out parts of code which can't be

converted.Also it should package supporting modules with it.

Performance Specifications:

Translator should be fast enough to translate about 1000 lines of code

per second.Translation can be done in concurrent manner to achieve performance.

1.4 Definitions

1.Translator

Translator is the product the SRS is all about i.e. translator refers to the system command line utility package that is converting a given reliable python code containing file to a optimized C++ code containing file.

2. User

User refers to the individual who is using the computer machine to give in input the python file and knows command line function to use the translator.

3. Command Line Interface This term refers to standard linux shell iterface of the Linux distribution which host is running like Fedora,Debian etc.

1.5 References

1. IEEE. *IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.* IEEE Computer Society, 1998.

(http://www.math.uaa.alaska.edu/~afkjm/cs401/IEEE830.pdf)

2. Lecture in Class.

3. Wikipedia.

(https://en.wikipedia.org/wiki/Software_requirements_specification)

1.6 Overview

The remainder of this document includes two chapters.The second one provides an overview of the translator functionality and translator's interaction with other systems. This chapter also introduces different

types of stakeholders and their interaction with the system. Further, the chapter also mentions the system constraints and assumptions about the product.

The third chapter provides the requirements specification in detailed terms and a description of the translator interfaces with user. Different specification techniques are used in order to specify the requirements more precisely. Both sections of the document describe the same translator in its entirety, but are intended for different audiences and thus use different language.

2. Overall description

This section will give an overview of the whole system. The system will be explained in its context to show how the system translates and what from the line of code gets translated. Basic functionality of the translator will be introduced and the stakeholders who can benefit from this translator will be described in this section. At last, the constraints and assumptions for the system will be presented.

2.1 Product perspective

The Python to C++ converter comes as a package in command line utility. The translator basically takes in a file containing python code and then reads the code on a line to line basis understanding the indentation of the python code, the classes made, the functions called in the code and the variables that are used along with various other things. Its also looks for the scope of the variables declared and libraries that the python code includes.

Once it is done with analyzing the input it then starts to translate the code C++.

It notes the extent of the loops, conditional statements, functions called by the the indentation in the python code. It looks for the variables and its extent in Python and correspondingly translates the variables in C++. After checking the functions used in python file the translator itself looks for the libraries it should

include to support those functions other than those which were specifically predeclared in python.

Finally the translator also check for the comments present in the python file and then includes those comment, too, in the output C++ file.

The translator is a command line utility and hence the use of translator is done in a command line envoirnment i.e. the translator is very easy to use by just typing a command than using any heavy user interface, only

a line of command and user is presented with a C++ translated file. The time taken by translator is the time to convert the python to C++ code only which makes it easy, simple and comfortable to use.

Since all the given code in python cannot be very certainly converted to C++ and hence the translator smartly segregates the code that can be translated to

C++ and the code that cannot be translated. Only after doing this the translator translates the code to C++ and leaves the segregated inconvertible Python code as it is. Again the translator keeps a 'log' file that shows both the part of code translated and the code that was left in python. So the user can decide if the code is compilable and understand to what extent the code was translated to

C++.

2.2 Product Functions

With the help of this product, any user who can code using python, which is easy to learn and similar to English, convert the code to C++ safely. The translator will take in python code containing file as an input which it reads and then translates into a correct and optimized C++ code file that can be compiled separately provided that the original given python code is reliable, too.

The libraries included in the C++ depends totally on the functions given by

user. The way in which objects are treated in Python remains the same in C++ code too. The C++ file can be used by the user independently. The translator also provides alog file that shows the part of code converted to C++ along with the inconvertible Python code kept as it was.

Precisely, the translator is very simple command line package to translate a Python code to C++ code.

2.3.1) User Characteristics

We expect the user to be well versed with python language and should be able to write a reliable and logically correct python code. The user

should also be Linux literate and should be able to use simple software; i.e. clicking on proper buttons and writing proper commands in command line; in order to use this translator.

The translator is basically aimed at the researchers or professors who are used to python language but cannot use C++ to make it faster. Using this translator saves the time for those professors and researchers as they no longer need to learn C++. Added to it is the simplicity of the translator and easy to use nature that increases the utility for researchers.

Since python is easy to learn, even children can use this utility by learning to code in Python and knowing just a single and simple command after getting the package installed. Also if there are a number of files with thousands of lines of codes in Python it is difficult to convert it into C++ manually and still keep it optimized. Hence in this case the users can use the translator because of it simplicity and ease.

2.3.2) User interface

The interface would be very simple, so the user can easily convert the code and produce the c++ bindings automatically by just running the source with the python code as input.

If the project is made open source, then the user may have to build the package with cmake .

Depending on the functions used, the user may have to download/link some additional libraries of

C++. Nevertheless, all the basic required libraries are already linked into the package.

As this software is aimed to run on Linux machine, the user should be comfortable with basic shell commands, to download required libraries at

times.

2.4) Constraints:

1)Python is a loosley typed language and has no constraints over size of Integers,etc. Whereas C++ is very strict in this regard,thus we need to declare everything as most liberal or biggest scope type available,this compromises with space and efficiency of converted program though it remains reltively faster.

2)There are many NLP and other third-party libraries like SciKit etc.Code using such third party libraries can't be converted by Translator unless those libraries are available for C++ also with same interface.

3)While conversion code bottlenecks may arise because of Python and C++ belonging to different family of languages and having pretty different coding constructs.

3) Requirements:

3.1) Software Requirements:

1)This translator assumes given Python code input to be correct and also needs a Python 2.7+ version to link C++ and Python code properly.

2)Because of large number of external Python Libraries this translator may not be able to translate projects relying on third-party libraries.

3)Code generated using this Translator will need the supporting libraries shipped with the package to compile.

4)This package is intended to work on Linux platform, user needs to shift to a Linux platform to use it.

3.2) Hardware Requirements:

1)Users need a multi-core processor to get benefit of fast translation features though they can be controlled with flag(For details refer to user manual).

2)Hardware supporting Linux is required.

3)Program thread must have access to reasonable amount (depending on overall size of code to be translated) of memory heap to store global dependencies, and call graph to figure out data type of variables and functions.

Note:-

This software can be most efficient when run on Linux machine because we use some Linux system API's for some of the operations that the translator deals with. It is better. So the system requirements are not

heavy as the software does not require any GUI as of now, But we plan to bring out the software extensible, portable so that GUI may be introduced in the near future.

3.3)Implementation Details:

1)Translator will be designed in C++. Core conversion code will be written using C++ STL(Standard Template Library) and some parts in Scala and Golang which require final synchronization of different sections.

2)Inbuilt data structures in Python like Dictionary will be converted to C++ STL containers and wherever suitable new such template classes will be implemented.They will be globally accessible and client can have only one copy of supporting libraries exported to a system path(specific for Linux environment).

3) Supporting libraries will be provided as Python library counterpart pre-implemented using C++ STL.Developers are encouraged to make supporting libraries as standalone as possible.

3.4) Software Testing Details:

1)”Google C++ Testing Framework” will be used for unit testing of various parts of Translation including but not restricted to following list:

*)Functionalization

*)Breaking of class

*)Recursive code conversion

*)I/O channels

*)Arithmetic overflow errors

2)For comprehensive final testing we will write our own shell scripts to

run full fledged Python programs and then compare output provided by C++ code generated.

3)If input Python code uses third-party Python libraries which are not yet available for C++, program logs should clearly cite it and we'll be using awk to check for such comparisons on log text files.

4) SOFTWARE CHARACTERISTICS

4.1) Reliability:

1)Translator should be able to identify the code it can no translate completely due to functional deficiency, python specific constructs or third-party libraries and exclude such portions of code but it should be made sure that it does not gives garbage result instead reports such anomalies and out of scope codes.

2)Translator will be thoroughly tested in later stages of development using Google Test Google's C++ unit testing framework(for details see Testing section).

3)Supporting libraries shipped with this package are not guaranteed to work for other purposes.
For example:

Pre-Converted Python string functions are not expected to work robustly with other codes and supporting libraries should not be used independently as they might be designed with certain assumptions relevant only to this Translator.

Thus supporting libraries are intended to be used only internally by projects.

4.2) Maintainability:

This project is planned to be released as an open source project.So once the first iteration is completed, it is expected to worked upon by many contributors for maintenance, bug fixing and enriching available library support.Project needs to be well documented to modular for such a vast community to work upon.

4.3) Documentation:

We'll be using “***doxygen***” for generating documentation for C++ source code of the Tranlator.Developers are supposed to follow standards as per

doxygen Manual.

4.4) Portability

The end user product is fully portable and can be used on any Linux based machine(i.e Linux of any flavor) , and the software can be very well compiled with any underlying linux based OS like Ubuntu, Gnome, Fedora,Mac- OS etc