Natural Language Programming Language Functional Specification

# Table of Contents

[Table of Contents](#_lhbnp1b2i4yu)

[Introduction](#_ek82s5hekb6r)

[Overview](#_dw7pqckoktrl)

[Business Context](#_7bqqtdvxph11)

[Glossary](#_y2hz1f3jadkp)

[General Description](#_xjnapx8gcki3)

[Product / System Functions](#_fdp3l2pqkj3i)

[User Characteristics and Objectives](#_v07r7w6p5500)

[Operational Scenarios](#_lhp4hdafha2f)

[Constraints](#_e77y9fcdwi3n)

[Functional Requirements](#_wq758tvx9ban)

[System Architecture](#_xug8vkemvddi)

[High-Level Design](#_d6ri5r3wy37m)

[Preliminary Schedule](#_ufd9bfmszb6y)

[GANTT Chart](#_msi8wvsl1r5v)

[PERT Chart](#_emg816t66cou)

[Appendices](#_wocnl5j7z9az)

# Introduction

## Overview

For our third-year project, we plan on creating an interpretive programming language with the idea that it is easy to learn for non-programmers to create their programs. As well as creating and testing the programming language to ensure it is functional we would like to test the programming language with people who are not familiar with programming to see if our language succeeds in allowing them a greater understanding of code and the logic that is needed for future languages.

The goals of our project are to

* Create a programming language
* Create an interpreter for the language
* Test the language to see if it is easier to learn when compared to other languages e.g. python, java, javascript, C

## Business Context

This project does not have heavy business involvement but once the project is completed the possibility of our programming language being used in schools to teach young children about code. Our code could be used as an alternative to the scratch programming language where the code is easy to understand and simple to put together but similar to more standard programming languages where the user has to write the code that they want to implement. Once our project is complete we will have it available on GitHub for people to download and use for their own projects.

## Glossary

* **Lexer** - the process of taking a sequence of characters and turning them into tokens with assigned meanings.
* **Parser** - Taking the tokens created by the lexer and constructs the abstract syntax tree.
* **Abstract Syntax Tree (AST)** - A tree representation of the source code in normal language, that can be modified to change the final implementation of the source code.
* **Interpreter** - a program that executes instructions given to it by source code without it having to be compiled into a lower-level language.
* **Compiler** - a program that takes a source code file and translates it to a lower-level language for the computer to understand.

# General Description

## Product / System Functions

Our project consists of 3 main areas, Creating the language using Antlr, Creating the interpreter for the language, and testing the language with users who have not coded before. Our primary goal is to create the programming language and have it be able to create simple programs with more advanced features being pursued once more straightforward features have been completed. In order to achieve our primary goal we would focus on creating the language and its necessary interpreter with user testing being one of the last tasks that would be completed during the project duration.

Our objective for this project is that the language should be:

* Easy to learn
* Readable
* Easy to use

## User Characteristics and Objectives

Our intended users will be complete beginners to programming who have never coded before. The language will introduce them to the basics of imperative programming including variables, conditional statements, loops and procedures. It should be easy to learn and read and understand as it is intended to be an entry point to programming for beginners.

## Operational Scenarios

Users would use the language to achieve two main goals:

* Learn the basics of programming
* Create simple scripts for automation

If a user wanted to learn the language as a starting point before moving on to other languages they would be able to open up their preferred text editor and be able to create a script quickly and easily.

A user who has learnt our language and wishes to use it for automation in their day-to-day life can do so as they do not require high amounts of coding skill or practice in order to create a script. If the user wanted to achieve something our language's natural language and simple syntax would allow them to understand each step of their code and give them a better chance at completing the task and learning from it.

## Constraints

* Must be able to communicate with standard io
* Needs to be easily runnable with a single command
* Must run on a Linux environment
* The interpreter needs to run fast enough to provide feedback to the user in a reasonable time

# Functional Requirements

1. **Requirement:** The language shall support variables, if statements, while loops, for loops, functions and procedures

**Criticality:** Highly Essential as these are the basics of most imperative programming languages

**Technical issues:** Difficulty in finding a syntax for procedures/functions that are more “natural”

1. **Requirement:**  The interpreter shall provide feedback on syntax errors when they occur

**Criticality:** High priority but not necessary

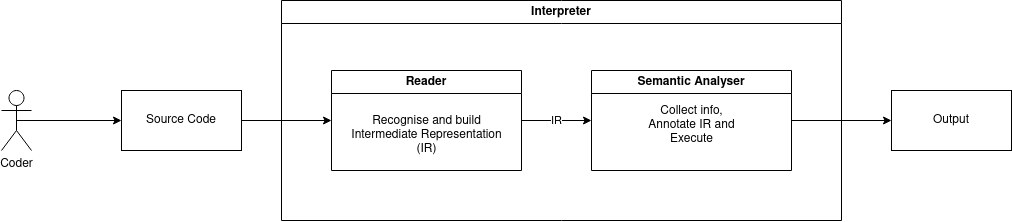
**Technical Issues:** Need to define where the error happened in the source code so the user can easily fix it

1. **Requirement:** The interpreter shall start running the program within 10 seconds of being launched

**Criticality:** Low. Would be nice for a short development feedback loop

**Technical Issues:** Requires an efficient implementation of the interpreter

# System Architecture



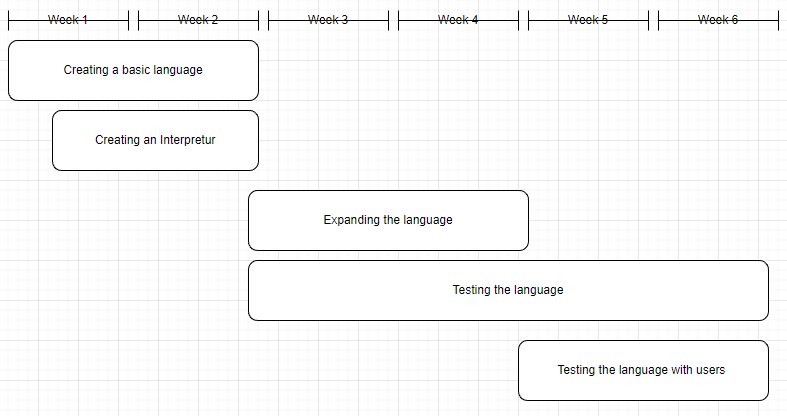
# High-Level Design

The coder will start by creating their source code file using the language, the language will then be turned from its plain text state into a series of tokens by the lexer which will then be used by the parser to create the abstract syntax tree (AST). an AST is a tree representation of the source file that is used to create and manipulate the code. The AST provides us with enough information about the structure of the code that we can choose what outcome we want based on how we manipulate the code. Now that we have the code made we can now run the code and using an interpreter the code will be brought down to a lower language for the computer to understand the request and execute them.

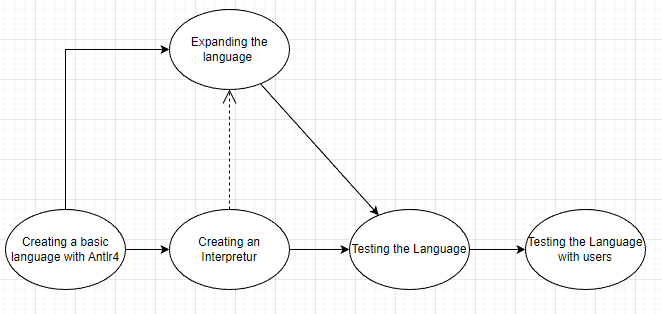
# Preliminary Schedule

For the first weeks of the project, our primary focus would be on creating the language via the lexer, parser and AST. This would take up a majority of the time in the project and we plan on that being the case. Once the language has reached a point where simple programs can be written by users and executed without a problem arising from the language itself we will move on to testing the language with users who have limited experience with computer programming, as they are our target demographic for the language, and begin testing to see if the language succeeds in its attempt to make it easier for inexperienced programmers to begin writing programs. We hope to have at least one week where we can test, analyse, and report on our findings.

### GANTT Chart



### PERT Chart



# Appendices

Parr, T. (2009) *Language implementation patterns create your own domain-specific and general programming languages*. 1st edition. Raleigh, North Carolina: The Pragmatic Bookshelf.