**5/6/2023**

Shri – An AI Code Generator

Software Requirement Specification

**Krishnakant Dubey 3323**

Software Requirement Specification

Version 1.2

**Table of Contents:**

**Introduction**

* Purpose
* Scope
* Definitions, Acronyms, and Abbreviations
* References
* Overview

**General Description**

* Product Perspective
* Product Functions
* User Characteristics
* Hardware & Software Requirements
* Operating Environment
* Design and Implementation Constraints
* Assumptions and Dependencies
* Shri's SDLC Model

**Specific Requirements**

* External Interfaces
* Functional Requirements
* Design Constraints

**Non-Functional Requirements**

* Performance
* Security
* Availability
* Reliability
* Maintainability
* Portability
* Usability

**System Models**

* Use Case Diagram
* Data Flow Diagrams
* Class Diagram
* Sequence diagram

Introduction

Shri is an AI-powered code generator designed to transform the way developers approach programming. It utilizes advanced natural language processing and machine learning techniques to generate high-quality code that meets specific requirements. Initially, Shri provides code generation for Python only, with the possibility of supporting additional programming languages in the future.

The system offers a variety of functionalities such as code formatting, syntax highlighting, and code validation to improve the accuracy, flexibility, and ease of use of the generated code. By minimizing the effort required to write code, Shri makes programming more accessible to a wider audience, while reducing the likelihood of coding errors and improving code quality by adhering to coding standards.

Shri is a versatile tool that can be used in a variety of programming tasks, such as web development, software engineering, and data analysis. It is designed to be user-friendly and requires no prior knowledge of programming languages, enabling developers to increase their productivity and streamline their workflow. Overall, Shri represents a significant advancement in AI-powered code generation and has the potential to revolutionize the way developers approach programming in Python and potentially other programming languages in the future.

**Purpose**

The purpose of Shri is to revolutionize the way developers approach programming by providing an innovative and efficient solution for code generation. Shri aims to make coding faster and easier by leveraging advanced natural language processing and machine learning techniques to generate code automatically based on user inputs. The ultimate goal of Shri is to minimize the effort required to write code and make programming accessible to a wider audience.

This document serves as a comprehensive guide for the development team, stakeholders, and users to understand the functional and non-functional requirements, external interface requirements, and system design of Shri. It is intended to provide a clear and detailed description of the software requirements for Shri and to ensure that the development team is aligned with the stakeholders' expectations.

**Scope**

The scope of Shri is broad and encompasses a wide range of programming tasks, including web development, software engineering, and data analysis. While Shri currently generates code exclusively for Python, it offers various functionalities, such as code formatting, syntax highlighting, and code validation, that are essential for efficient and accurate coding. Shri is designed to support other programming languages in the future, though the timeline for the addition of new languages is subject to change.

This document provides a comprehensive description of the software requirements for Shri, including the functional and non-functional requirements, assumptions and dependencies, external interface requirements, and system design. The functional requirements specify the system's behaviour and capabilities, while the non-functional requirements describe the system's performance, scalability, and reliability. The external interface requirements define how Shri interacts with other systems, and the system design specifies the architecture, data design, component design, and user interface design of the Shri system.

Overall, the scope of Shri is vast and ambitious, with the potential to transform the way developers approach coding. Shri aims to make coding faster, easier, and more accessible to a wider audience, ultimately improving code quality and reducing coding errors.

**Definitions, Acronyms, and Abbreviations**

**NLP**: Natural Language Processing

**IDE**: Integrated Development Environment

**API**: Application Programming Interface

**GUI**: Graphical User Interface

**ML**: Machine Learning

**DL**: Deep Learning

**References**

The following documents were used in the development of this SRS:

1. IEEE Recommended Practice for Software Requirements Specifications, IEEE Std 830-1998 (Revision of IEEE Std 830-1993)
2. Language Models are Few-Shot Learners

<https://arxiv.org/pdf/2005.14165.pdf>

1. Programming languages Datasets for ML

<https://metatext.io/datasets-list/programming-languages-language>

1. NLP Models for Writing Code: Program Synthesis <https://www.exxactcorp.com/blog/Deep-Learning/nlp-models-for-writing-code>
2. GAP-Gen: Guided Automatic Python Code Generation <https://www.researchgate.net/publication/358145080_GAP-Gen_Guided_Automatic_Python_Code_Generation>

**Overview**

This SRS document is organized into three main sections:

**System Requirements**: This section outlines the functional and non-functional requirements, assumptions and dependencies, and external interface requirements for Shri. These requirements include the system's ability to generate accurate code based on user inputs and adhere to coding standards.

**System Design**: This section provides a detailed description of the architectural design, data design, component design, and user interface design for Shri. The system design should enable the user to generate code quickly and easily, with features such as autocomplete and real-time code suggestions.

**Appendices**: This section includes a glossary of terms, use cases, data flow diagrams, and sequence diagrams to aid in understanding the system's functionality and design.

General Description

Shri is an innovative AI-powered code generator designed to simplify and accelerate the coding process for developers. It uses advanced natural language processing (NLP) and machine learning (ML) techniques to automatically generate code based on user inputs. Shri's main objective is to reduce the effort required to write code and make programming more accessible to a wider audience.

While it currently supports Python as the primary language, it has the potential to expand to other languages in the future. It also provides various features, such as code formatting, syntax highlighting, and code validation. Shri is an excellent tool for developers who want to boost their productivity and streamline their workflow.

The system works by taking user inputs, such as the desired function or algorithm, and analyzing them using machine learning algorithms. The system then generates code that meets the user's requirements and can be easily integrated into their project.

Shri offers several benefits over traditional code generators, including increased accuracy, flexibility, and ease of use. It also decreases the likelihood of coding errors and improves code quality by producing code that is more consistent and adheres to coding standards.

Shri is a versatile tool that can be used in a variety of programming tasks, such as web development, software engineering, and data analysis. The system is designed to be user-friendly and requires no prior knowledge of programming languages. Shri represents a significant step forward in the field of AI-powered code generation and has the potential to revolutionize the way developers approach programming.

**Product Perspective**

Shri is a standalone software product that operates independently of any other systems. However, it can be easily accessed through a web server or API, allowing developers to generate code in Python. In the starting phases, Shri is only capable of generating Python code, but it may add support for other programming languages in the future.

Shri is designed to be compatible with a wide range of operating systems, including Windows, Mac OS, and Linux, ensuring that it is accessible to a diverse range of users. While Shri won't be available as an IDE plugin or extension initially, developers can still use Shri's code generation capabilities by accessing it through its web server or API.

**Product Functions**

**Code generation**: Shri will generate code for programming languages, including Python, Java, and C++. But in initial stage it will generate code in Python only. The system analyzes user inputs and requirements to generate the code automatically.

**Automated analysis**: Shri uses advanced natural language processing and machine learning techniques to analyze user inputs and determine the code requirements. This minimizes the effort required to write code and makes programming more accessible to a wider audience.

**Code formatting and syntax highlighting**: Shri offers code formatting and syntax highlighting functionalities that help to improve code readability and consistency. This feature is especially useful for developers who want to maintain a consistent coding style across different projects.

**Code validation**: Shri also includes code validation functionalities that help to identify and correct coding errors. This feature reduces the likelihood of errors in the generated code and ensures that the code adheres to coding standards.

**User-friendly interface**: Shri has a user-friendly interface that is easy to use and navigate. The interface is designed to be intuitive and requires no prior knowledge of programming languages.

**Integration with existing software development environments**: As Shri will initially be available only on web servers for public use as well as with the facility of API, it won't be possible to integrate it into existing software development environments such as IDEs and text editors. However, developers can still use Shri's code generation capabilities by making API calls to the Shri web servers.

**Flexibility**: Shri is a flexible tool that can be used in a variety of programming tasks, such as web development, software engineering, and data analysis. The system is designed to adapt to the user's needs and preferences and can be customized accordingly.

**Code optimization**: Shri can optimize the generated code by reducing redundancies, improving performance, and reducing the code's memory footprint. This feature is especially useful for developers who want to improve the efficiency and speed of their code.

**User Characteristics**

Shri may also be useful for students learning programming languages or individuals who need to generate code quickly and efficiently. The system should be able to provide a learning platform for beginners and help them learn programming concepts with ease.

Users of Shri may have different levels of familiarity with software development environments, so the user interface should be intuitive and easy to navigate. The system should provide clear and concise instructions to help users generate code without any errors.

The system should be designed to accommodate users with varying levels of technical expertise, from beginner to expert programmers. Shri should offer an advanced mode for experienced programmers to enable them to fine-tune the code generation process.

Users may have different requirements or preferences when it comes to programming languages, so the system should be flexible enough to support multiple languages and allow for customization. Shri should provide support for popular programming languages and offer a mechanism to add support for additional languages as needed.

**Hardware & Software Requirements**

**Hardware Requirements**:

* Processor: Intel Core i5 or higher
* RAM: 8 GB or higher
* Storage: 500 MB of free disk space

**Software Requirements**:

* Operating System: Windows 10, macOS 10.12 or later, or Ubuntu 18.04 LTS or later
* Web Browser: Google Chrome, Mozilla Firefox, Apple Safari, or Microsoft Edge
* Python: version 3.6 or higher

It's important to note that the software requirements may vary depending on the specific features and functionalities of Shri that are being used. Additionally, certain features may require additional software or libraries to be installed on the user's system.

Furthermore, it's recommended that users have a stable internet connection to use Shri, as it requires internet connectivity to access the web-based interface and utilize its features.

Overall, these hardware and software requirements ensure that Shri runs smoothly and efficiently, providing users with a seamless and hassle-free coding experience.

**Operating Environment**

* The software should be compatible with popular operating systems such as Windows, macOS, and Linux.
* The system should be able to access the internet to retrieve necessary data and updates for the software.
* The system should have a minimum screen resolution of 1024x768 for optimal display of the user interface.
* Shri may require installation of additional software libraries or dependencies, which should be clearly documented for the user.
* The system should be able to handle large amounts of data for training and model development, which may require additional storage space or resources.
* The system should be able to run without significant performance degradation on a variety of hardware configurations, from low-end laptops to high-performance desktops.

**Design and Implementation Constraints**

* The system must be compatible with various operating systems such as Windows, Mac OS, and Linux.
* The system must have a well-defined API that can be used by other systems to integrate with Shri.
* The system should be designed to handle errors and exceptions gracefully, with appropriate error messages provided to the user.
* The system should be designed to allow for easy maintenance and updates, with version control and documentation provided for changes made to the codebase.
* The system should be designed to be modular and extensible, allowing for future enhancements and additions to be made to the system.
* The system should comply with industry standards and best practices for software development, such as using secure coding practices and adhering to coding standards.

**Assumptions and Dependencies**

**Assumptions**:

* Shri assumes that the user has basic knowledge of programming concepts and syntax.
* Shri assumes that the user has access to a reliable internet connection to ensure proper functionality.
* Shri assumes that the user has access to a computer or mobile device that meets the minimum system requirements.
* Shri assumes that the user will provide accurate and valid inputs for the system to generate code.

**Dependencies**:

* Shri is dependent on the availability and reliability of the machine learning models and algorithms used to generate code.
* Shri is dependent on the availability and compatibility of the programming languages and software dependencies required for code generation.
* Shri is dependent on the availability and stability of any third-party tools or services used in the development and operation of the system.
* Shri is dependent on the security measures implemented by the hosting environment to ensure the security and privacy of user data.

**Shri's SDLC Model**

Shri's development will follow an Agile SDLC (Software Development Life Cycle) model, specifically the Scrum framework. As a solo developer, it is essential to have a flexible development process that allows for quick and efficient changes. Scrum's incremental and iterative approach aligns with Shri's development requirements, where small features and updates can be added in sprints.

The Scrum framework comprises several essential components, such as sprints, backlog, and weekly meetings, to ensure the project's success. The sprints will typically last for two to four weeks, during which specific tasks will be accomplished. The backlog will serve as a prioritized list of tasks, features, and improvements, which will be completed during the sprints. Weekly meetings will provide an opportunity to track the progress, discuss issues, and plan the next steps.

In addition to the Scrum framework, other Agile practices will be incorporated into the development process. Continuous integration, frequent testing, and regular code reviews will be used to ensure that the code is of high quality and any issues are identified and addressed promptly. This approach will enable Shri to be developed efficiently, with quick turnaround times for new features and updates, while ensuring a high-quality end product.

Specific Requirements

The Specific Requirements section of the SRS document outlines the detailed specifications for Shri. It includes the requirements for code generation, analysis of user inputs, and user interface design. The section covers the functionality of the system, including the ability to generate code for Python programming language. It also details the automated analysis of user inputs to determine code requirements, as well as the code formatting, syntax highlighting, and validation functions. Additionally, any special or unique requirements, such as support for specific programming languages or compatibility with certain software development environments, are included in this section. The specific requirements serve as a guideline for the development team to ensure that the system is developed to meet the desired functionality and quality standards.

**External Interfaces**

Shri has several external interfaces that are critical to its functionality and usability. These include:

**User Interface**: Shri provides a user-friendly graphical user interface (GUI) that allows users to easily interact with the system. The GUI includes menus, toolbars, and dialogs that enable users to input their requirements, select the programming language(if languages are added in future) and other preferences, and view the generated code.

**APIs**: Shri provides APIs that allow other software systems to interact with it. These APIs can be used to integrate Shri with other software development environments, such as IDEs or text editors. The APIs also enable users to automate the code generation process by integrating Shri with other tools or software systems.

**Command-line Interface**: Shri includes a command-line interface (CLI) that allows users to interact with the system through a terminal or command prompt. The CLI provides a convenient way for users to generate code without having to use the GUI.

**File Input and Output**: Shri won’t support file input and output features in the initial releases. However, this functionality may be added in the future to allow users to input their code requirements from existing files or output the generated code to a file for later use.

**Database Integration**: Shri can integrate with databases to store and retrieve user preferences and settings. This enables users to save their preferences and reuse them in future code generation tasks.

The external interfaces of Shri are designed to make it easy for users to interact with the system and integrate it with other software development tools and systems.

**Functional Requirements**

**Code generation:** Shri must be able to generate high-quality, efficient, and error-free code for multiple programming languages (in future updates), including but not limited to Python, Java, and C++. The generated code must be well-documented, easy to understand, and conform to standard coding conventions and practices.

**Automated analysis of user inputs:** Shri must be able to analyze user inputs, such as natural language descriptions, pseudocode, or incomplete code snippets, to determine the user's intent and generate the corresponding code. The system must use machine learning algorithms and natural language processing techniques to parse and understand the user's input accurately and efficiently.

**Code formatting, syntax highlighting, and validation**: Shri must be able to format the generated code in a readable and consistent manner, highlighting important syntax elements such as keywords, variables, and functions. The system must also be able to validate the code for syntax errors, logic errors, and other common programming mistakes, and provide informative error messages and suggestions for corrections.

**User interface design:** Shri must have a user-friendly and intuitive interface that allows users to input their requirements and preferences easily, visualize and modify the generated code, and access additional features and tools. The system must support multiple modes of interaction, including command-line interface, graphical user interface, and API.

**Version control:** Shri must support version control features, such as branching, merging, and committing code changes, to facilitate collaboration and code management. The system must integrate with popular version control systems, such as Git, and provide visual and textual feedback on changes, conflicts, and updates.

**Debugging and testing:** Shri must support debugging and testing features, such as step-by-step execution, breakpoints, and unit testing, to help developers diagnose and fix errors in the generated code. The system must provide informative error messages and detailed feedback on the code's performance, memory usage, and other metrics, to help users optimize the code.

**Customization and extensibility:** Shri must be customizable and extensible, allowing users to modify and add their own code templates, libraries, and configurations. The system must provide a flexible and modular architecture, supporting multiple plugins, modules, and APIs, and allowing users to integrate their own tools and technologies.

**Design Constraints**

Design constraints are the limitations or restrictions imposed on the design and development of Shri. These constraints include technical considerations, such as the compatibility of the system with specific operating systems or software development environments, hardware requirements, and software dependencies. These constraints impact the overall design of the system, as well as its performance and functionality.

One of the design constraints for Shri is compatibility with different operating systems, such as Windows, macOS, and Linux. The system should be designed to work seamlessly on each of these platforms, with consideration for differences in hardware and software configurations. Shri should also be compatible with popular software development environments, such as Visual Studio Code, Sublime Text, and Eclipse, to ensure ease of integration into existing development workflows.

Another important design constraint is memory and processing power requirements. Shri should be designed to run efficiently on a wide range of computer systems, with minimal impact on system resources. This includes optimization of algorithms and data structures, as well as careful management of memory usage.

Other design constraints may include compliance with industry standards or regulations, such as the General Data Protection Regulation (GDPR) or the Health Insurance Portability and Accountability Act (HIPAA). Shri should be designed to ensure compliance with these standards, with appropriate measures in place for data security and privacy.

Overall, the design constraints for Shri should be carefully considered and documented to ensure that the system meets the needs of its users and can be developed and maintained effectively over time.

Non-Functional Requirements

Non-functional requirements are a set of characteristics that describe how well a software system performs its functions. These requirements do not necessarily relate to the specific features of the system but rather to its performance, security, availability, reliability, maintainability, portability, and usability. For Shri, these non-functional requirements are critical for ensuring that the system functions effectively and efficiently.

**Performance**

Shri must be able to generate code quickly and efficiently, and it should have minimal processing and response times. The system should be designed to handle a large number of user requests simultaneously, without sacrificing performance. Shri should also be able to run on different types of hardware and operating systems, ensuring that it can meet the performance requirements of users with varying technical specifications.

**Security**

The security of user data is critical for Shri, and the system should be designed to protect user information from unauthorized access, manipulation, or disclosure. The system should have appropriate access controls, encryption mechanisms, and authentication protocols to ensure that only authorized users can access the system. Additionally, the system should be designed to minimize the risk of data loss or corruption, and it should have robust backup and recovery mechanisms in place.

**Availability**

Shri should be available for use 24/7, and the system should be designed to minimize downtime and ensure maximum uptime. The system should have a fault-tolerant design, with redundant servers and failover mechanisms in place to ensure uninterrupted service in the event of a hardware or software failure. Additionally, the system should have robust monitoring and alerting mechanisms to notify the support team in the event of an issue.

**Reliability**

Shri must be reliable and consistent in its performance, and it should be able to handle user requests without errors or crashes. The system should be designed with fault tolerance and error handling mechanisms to ensure that it can recover from errors or unexpected events without causing data loss or corruption. Additionally, the system should have a well-defined testing and validation process to ensure that it meets the reliability requirements of users.

**Maintainability**

The maintainability of Shri is essential to ensure that the system remains usable and functional over time. The system should have clear documentation and modular design to facilitate code maintenance and modification. Additionally, the system should be designed with scalability in mind, ensuring that it can handle increased demand as the user base grows. Finally, the system should be portable and compatible with different operating systems and software development environments, making it easier to maintain and update.

**Portability**

Shri should be portable and compatible with different hardware and operating systems, ensuring that it can meet the needs of users with different technical specifications. The system should be designed to run on different platforms without requiring significant modifications, and it should have clear documentation to facilitate installation and configuration on different systems.

**Usability**

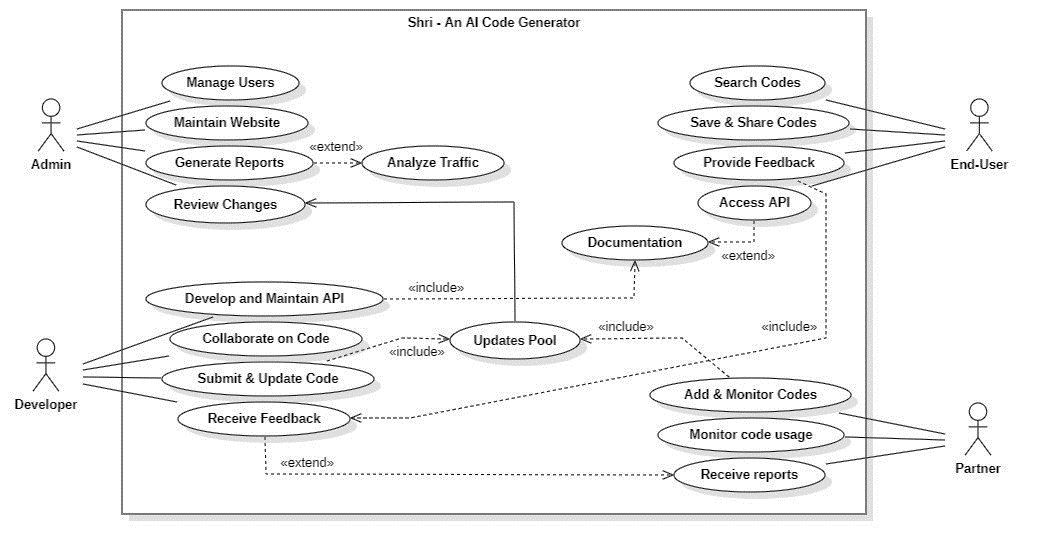
Shri should be user-friendly and accessible, requiring no prior knowledge of programming languages. The system should have an intuitive user interface that is easy to navigate and understand, with clear instructions and guidance for users. Additionally, the system should have well-defined error messages and feedback mechanisms to help users troubleshoot any issues they encounter.

System Models

System models can be used to help visualize the overall architecture and structure of the system, as well as how data flows through it. For Shri, system models could include a high-level diagram that shows the different modules or components of the system, such as the code generation module, user interface module, and machine learning module.

System models can also help identify potential areas of improvement or optimization in the system. System models can be useful for communicating the design and structure of the system to stakeholders, including developers, users, and management. They can also aid in testing and verification, by allowing testers to visualize how data flows through the system and identify potential points of failure.

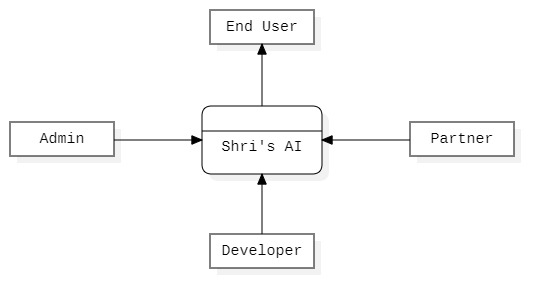
**Use Case Diagram**

A use case diagram is a visual representation of the system's functional requirements, showing the various use cases and actors involved in the system. In the context of Shri, a use case diagram could show the different use cases of the system, such as code generation, analysis of user inputs, and user interface design, along with the different actors involved, such as the system administrator, software developers, and end-users.

**Data Flow Diagrams**

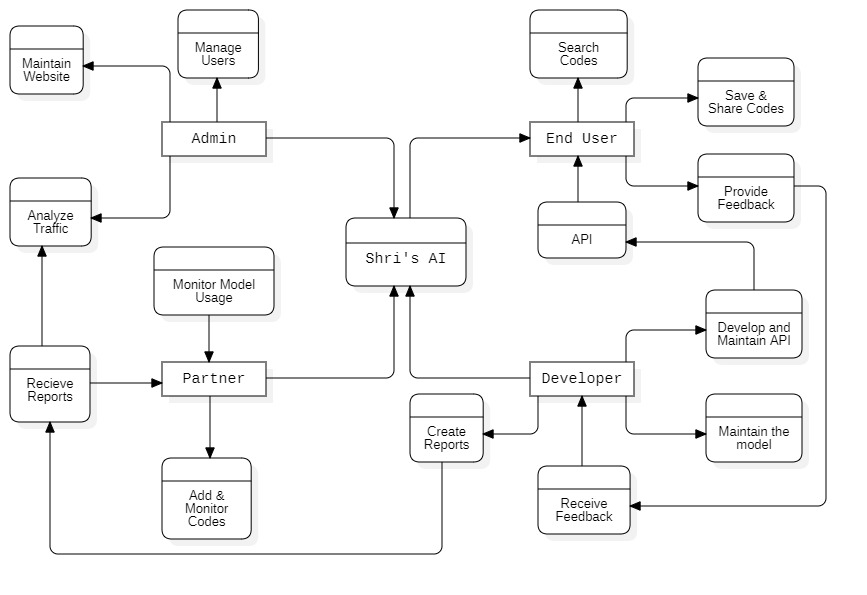
Data flow diagrams are used to show how data flows through a system. In the context of Shri, a data flow diagram could be used to show how user inputs are processed by the system, and how the system generates code outputs. The diagram could be divided into different levels, such as level 0, level 1, and level 2, each providing more detail on the system's data flow.

**DFD Level 0**: This is the highest level of abstraction in a DFD, which represents the entire system as a single process or entity. At this level, inputs and outputs are shown as simple arrows, and the focus is on the overall flow of data through the system.



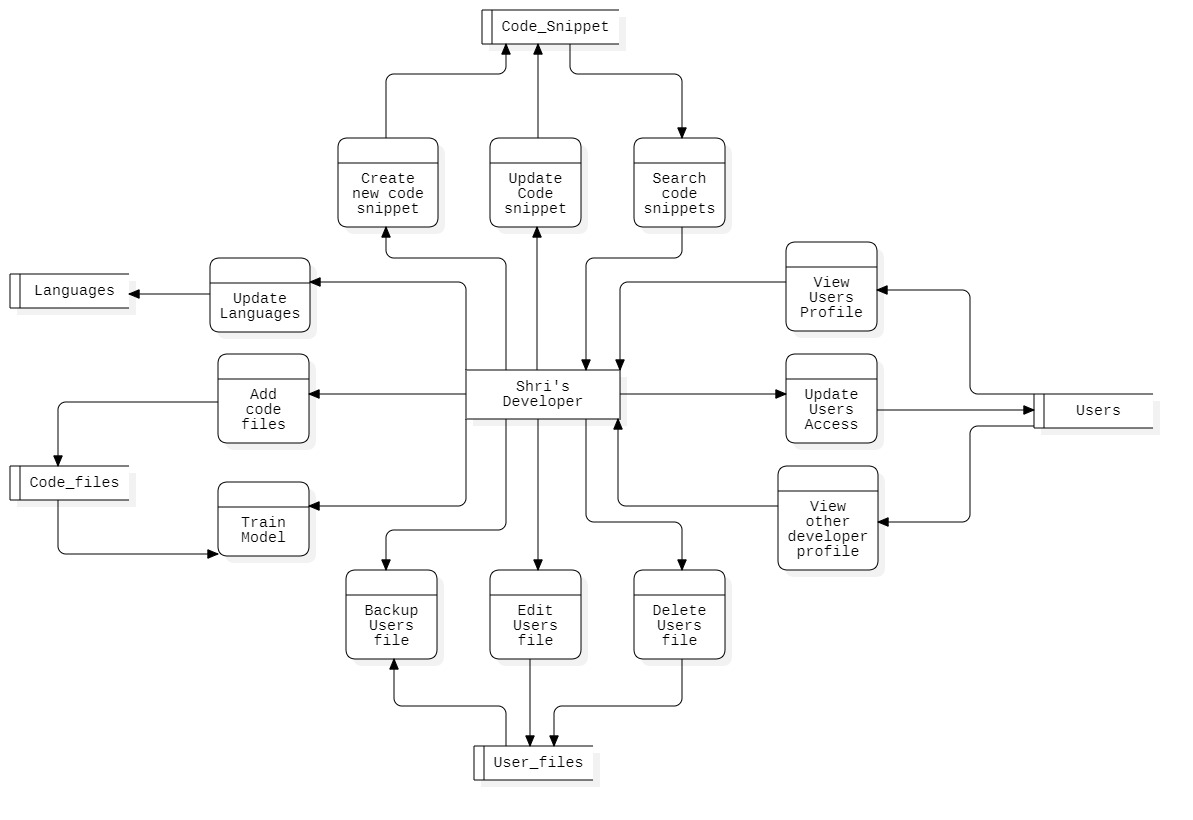
A level 0 DFD could be used to show the overall flow of data through the system, with inputs coming from the user and outputs being generated by the code generation module.

**DFD Level 1:** In a level 1 DFD, the system is broken down into major sub-processes or modules, with each module represented by a separate bubble. The inputs and outputs for each module are shown as arrows, and the focus is on the interaction between the modules and the flow of data between them.



A level 1 DFD could break down the system into major modules such as input processing, code generation, and output formatting, with inputs and outputs for each module shown as arrows.

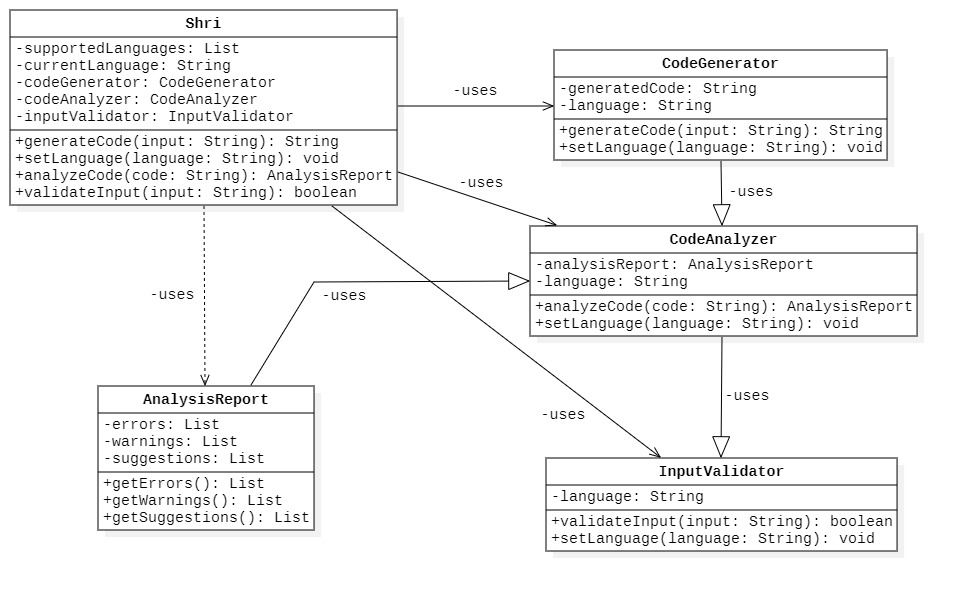
**DFD Level 2:** At level 2 DFD, each module from the level 1 DFD is further broken down into sub-processes, which are represented as separate bubbles. The inputs and outputs for each sub-process are shown as arrows, and the focus is on the detailed flow of data within each module.



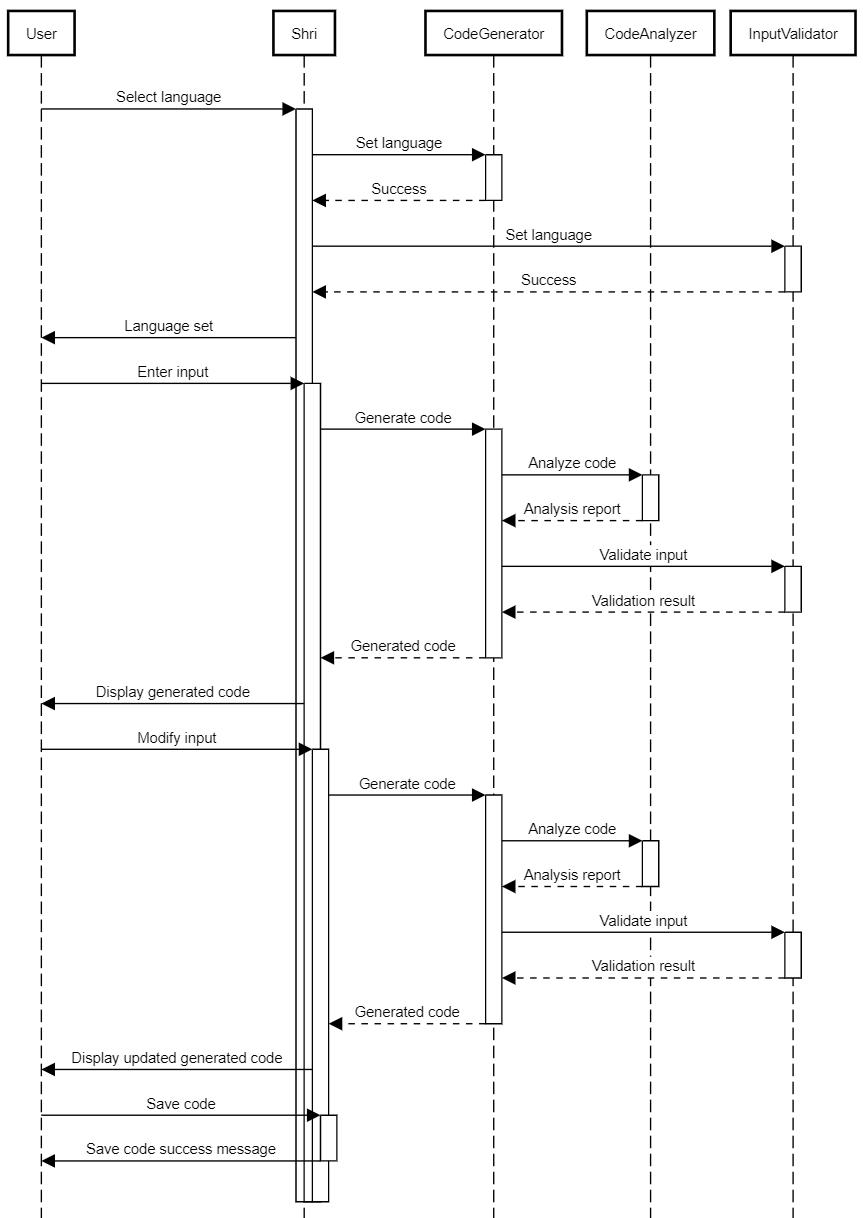
A level 2 DFD could further break down each module into sub-processes, such as lexical analysis, syntax analysis, and code optimization, and show the detailed flow of data within each sub-process.

**Class Diagram**

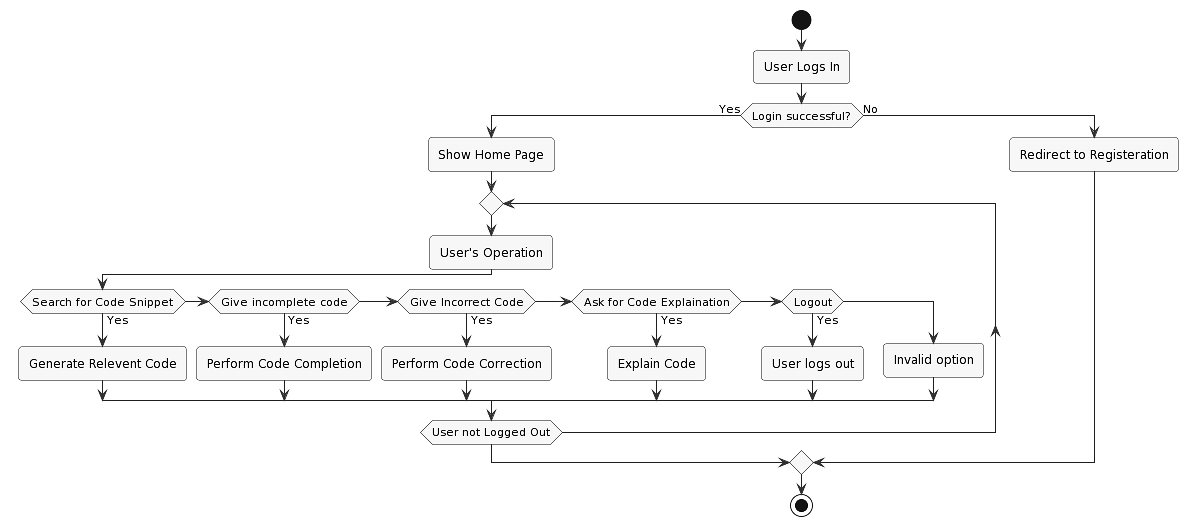
A class diagram is a visual representation of the system's class structure, showing the different classes, attributes, and methods involved in the system. In the context of Shri, a class diagram could show the different classes involved in the system, such as the user interface, code generator, and data analysis modules, along with their attributes and methods.



**Sequence Diagram**

A sequence diagram is used to show the interaction between different objects or components in a system over time. In the context of Shri, a sequence diagram could be used to show how user inputs are processed by the system, and how the system generates code outputs. The diagram could show the different objects or components involved in the system, such as the user interface, code generator, and data analysis modules, and how they interact with each other.

**Activity Diagram**

An activity diagram is used to represent the flow of activities or actions within a system or process. In the context of Shri, an activity diagram can illustrate the steps involved in the code generation process and the interactions between various components. It provides a visual representation of how the system progresses from one activity to another.