CodeNect: Visual Programming Software for Learning Fundamentals of Programming

An Undergraduate Thesis
Submitted to the Faculty of the
College of Engineering and Information Technology
Cavite State University
Indang, Cavite

In partial fulfilment
of the requirements for the degree
Bachelor of Science in Information Technology

Lim-it, Brandon B.,
Punay, Jaykel O.
November 2020

CodeNect: Visual Programming Software for Learning Fundamentals of Programming

Lim-it, Brandon B.,
Punay, Jaykel O.

An undergraduate thesis outline submitted to the faculty of the Department of Information Technology, College of Engineering and Information Technology, Cavite State University, Indang, Cavite, in partial fulfillment of the requirements for the degree of Bachelor of Science in Information Technology. Prepared under the supervision of Ms. Gladys G. Perey.

INTRODUCTION

Background of the Study

As the innovation in technology is continuously making its progress in improving the quality of life. With this attribute of digital technology comes the essential need for programming skill as core profiency. The competition in the field that developers and programmers alike strive for is becoming harder to get into due to the standards and requirements. Programming is a discipline as it is designed for communicating and instructing to a computer (Prahofer, Hurnaus, Wirth, and Mossenbock, 2007). Without the proper knowledge and understanding in programming in its fundamental level and depth, one will find it difficult to adapt to the constantly shifting world of computer. Since programming is a skill wherein it focuses in the connection of logics rather than memorizing information as that of in other domain, the curve in starting to learn it is steep and complex compared to actually applying it in real works and mastering it. One of the requirements for a programmer and developer is to have expertise in technical skills that include multiple programming languages (Tsai, Yang, and Chang, 2015).

The goal of producing globally competitive individuals in the digital field highlights the importance of programming education. Learning and mastering a programming language is not an easy task, but in general all share common ideas and concepts that go back to the machine hardware layer. Learning by heart these core concepts and fundamental knowledge of programming will help the programmer to easily learn and master any existing or new programming languange through the reiteration of principles that all programming languages are built and modeled upon. For the design and decisiton that go behind the creation of new programming languages are reevaluation of existing studies, syntax, semantics, and inspired by widely used and accepted languages (Chang

Boon Lee, 2011). The preparation of a developer for the industry and career takes place through education. In the current education system of the country, the subject of programming is introduced in the college and senior highschool level. For students who are new to programming, the topics can be a daunting and intimidating task. Failure in familiarization and application in the early academic years and progressing to the next period wherein advance subjects are covered bereaves the overall learning of the student.

The fundamentals concepts of programming are essential basics that are necessary for programmers to master. Concepts such as syntax, semantics, variables, data types, data structures, logic, conditionals, loops, algorithm, and memory are key to easily understanding and getting better at programming. Starting from simple display of message into the screen, performing basic arithmetics, processing user inputs, and interaction through bare terminal programs are foundations and stepping stones before going through advance topics such as Object-Oriented Programming (OOP), Graphical User Interface (GUI), and website, game, and software development. With this important subjects learnt during the preliminary year in education, the complexities and difficulties of advance concepts are reduced.

The improvement of the learning of a student without prolonging the time alloted in each academic year will need to focus on enhancing the properties of the softwares that are both utilized as teaching and learning tools by the instructor and the student. An application that is implemented using modern tools and industry standard design and functionality that focuses in simplicity, readability, and learning experience. Modern technology incrases the rate of knowledge acquisition and absorption through its usage and implementation in education (Raja and Nagasubramani, 2018). A program that is accessible anywhere and anytime by the user whether in a computer laboratory at school or in home. The advancement in technology greatly contributes to education as it enables convenience in communication and presentation of knowledge and information almost instantaneously (Anggrawan, Ibrahim, M., and Satria, 2016). A software that prioritize functionality over bloated features to ensure that the user is not overloaded with information in the screen that is unnecessary. Users perceive numerous features in a product to be useful and engaging but such can result in fatigue (Thompson, Hamilton, and Rust, 2005).

The integration of a visual nodes and graphs as the interface in programming instead of the traditional text-based language is simpler, more appealing to use, and more intuitive to get started with. Visual programming saves time in specifying the code textually and manually input. It enables learning through interaction and manipulation of graphical elements which can provide more feedback through the use of color, size, and icons. The abstract and high-level concepts are represented visually that can facilitate learners by variable observation, logic flow tracing, and debugging skills (Tsai et al., 2015). This visual programming tool can also export an equivalent source code to other programming languages that are in the curriculum such as C/C++, Java, and Python. The tandem of visual code and textual code allows programmers to write programs while learning fundamental concepts as seeing the code in both formats. This allows to ease the transition from introductory visual programming skill to real text-based programming language. (Alam and Bush, 2016)

Aside from the necessary tools and functionalities oriented towards learning purposes, the software is also designed to and packaged with coding exercises and problems which range from beginner, intermediate, to advance difficulties. Each is carefully selected and designed towards showing and comparing various solutions that are working in the context that they meet the requirements and output and are technically correct, but not all will meet the standard when it comes to better performance, lower memory usage, lower power consumption, and faster processing time which are the some of the advantages in skills acquired when learning completely the fundamentals of programming. This approach in problem solving will allow learners to develop their logical and critical thinking through the application of the theory of variation, wherein some aspects that are critical must vary while other aspects stay constant (Cheng, 2016). This is effective in the domain of programming as even a slight change in data amounts to a change in effect and output.

Statement of the Problem

Learning a discipline such as programming that is constantly emerging with updates and changes is vastly difficult without the proper knowledge and deep understanding of the core concepts and fundamentals. Most individuals, students, and even graduates, with multiple years of experience still do not understand the foundational concepts of programming and find it hard to keep up with technology. (See Figure 3)

Education in programming requires the assistance of technology itself through softwares in improving the quality of learning. The traditional method of pure lecture is nowadays complimented with the help of softwares. But most tools are not beginner-friendly and are cluttered with features that present a steep learning curve in familiarity and mastery that diminish the learning experience. (See Figure 4)

The problem of using existing softwares for programming and in computer laboratory as teaching and practice tool is that the majority of the editors are text-based which are not recommended for people who are just getting started in the field of programming. (See Figure 5)

Objectives of the Study

The general objective of the study is to develop a Visual Programming Software that will help in learning the fundamentals of programming.

Specifically, this study seeks to answer the following questions:

1. Identify the concepts learners find difficult to understand through conducted survey.
2. Analyze the problems through a Ishikawa/Fishbone Diagram.
3. Design the system using the Use Case Diagrams.
4. Develop the software with the following main features:
    (a) Visual Nodes Module handles the core elements and building blocks in the software for writing logic and code.
    (b) Filesystem Module handles the creation, modification, reading, and deletion of files.

(c) Input and Output Module is the interface for user actions such as mouse events and key events and what is displayed to the screen for the user.

(d) Debug Module lints the visual code for errors and warnings before the running the code. It also captures errors and warnings during runtime and report it to the user.

(e) Simulation Module for compiling and running the visual program to a command line program.

(f) Transpiler Module that will convert the visual logic to other programming languages such as C/C++, Java, Python, Javascript, Lua, and more.

(g) Assessment Module that will evaluate the knowledge and learnings of the users by providing basic and common coding exercises. Exercises can be imported, shared, and distributed through simple package files.

5. Test and evaluate the usability, functionality, and acceptability of the software.

Significance of the Study

The result of the study will be of great benefits to the following:

Students and Learners of Programming Course. The software will help in the education and improvement in the knowledge, skills, understanding, and expertise of the students and learners about programming. Thus, allowing them to compete and increasing the opportunities for their careers.

Teachers and Instructors of Programming Courses. The result will aid in the work of sharing and teaching of knowledge and lessons in programming of the educators. Relieving workload, stress, and maximizing their lessons each class time.

Tech Companies. The result will produce individuals who are capable and skilled in programming efficient, safe, and high-quality softwares and tech products. Beneficial to the industry and field of technology.

Future Researchers. This study would serve as a guide and reference in the field of software development and education.

Conceptual Framework

The conceptual framework (1) represents the relationship and flow of the concept of developing a visual programming software for learning the fundamentals of programming.

It shows the order of actions required by the study to achieve the desired output.

The inputs has the following requiments for the development of the study. Knowledge requiments include software development, user-interface design, user experience, transpilation, and pedagogical knowledge. Software requiments include Linux 5.4 - Manjaro as operating system and distribution, Vim as text and code editor, terminal, Kha and zui library for graphics, and the Haxe programming language. Hardware requiments are machine with atleast Intel Core 2 Duo at 1.4 GHz, 2 GB RAM, and 80 GB HDD storage.

The process to be followed and used for the development of the software is the V model. The process model involves the requirements of gathering necessary data and information from respondents through conducted survey. High-level design of the implementation of the requirements of its technical usage for system design. Design of the relationships and dependencies of the modules, creation of diagrams, and selection of technology to be used. Preparation of the design and test method for each module. Followed by the coding of the modules and the software. The evalation is based on the ISO/IEC/IEEE 29119-4.2015, specifically the experience-based test design technique, to assure the functionality, efficiency, usability, portability, and reliability of the software.

The impact expected from the study is the availability of a software designed for learning programming and its fundamentals for learners, students, and educators. Another impact is the improvements in the understanding, skills, and academic performances of the students in the course of programming.

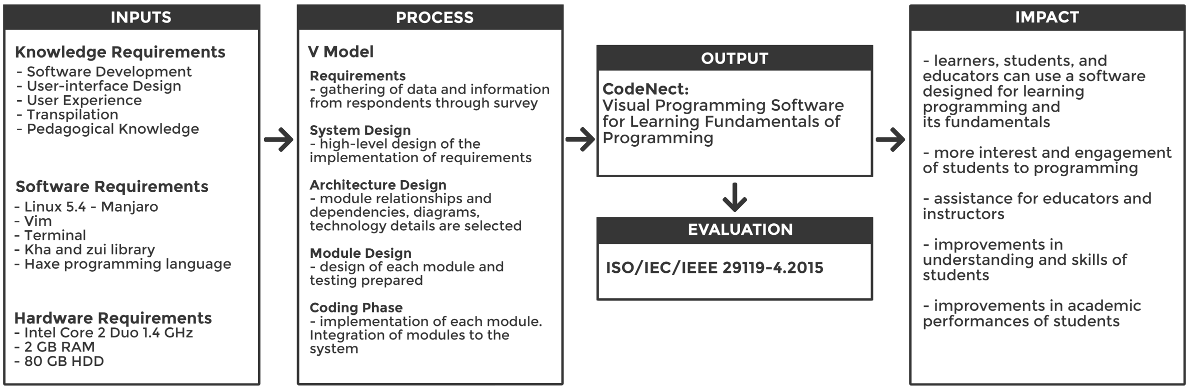

INPUTS
Knowledge Requirements
- Software Development
- User-interface Design
- User Experience
- Transpilation
- Pedagogical Knowledge
Software Requirements
- Linux 5.4 - Manjaro
- Vim
- Terminal
- Kha and zui library
- Haxe programming language
Hardware Requirements
- Intel Core 2 Duo 1.4 GHz
- 2 GB RAM
- 80 GB HDD
PROCESS
V Model
Requirements
- gathering of data and information from respondents through survey
System Design
- high-level design of the implementation of requirements
Architecture Design
- module relationships and dependencies, diagrams, technology details are selected
Module Design
- design of each module and testing prepared
Coding Phase
- implementation of each module. Integration of modules to the system
OUTPUT
CodeNect:
Visual Programming Software for Learning Fundamentals of Programming
EVALUATION
ISO/IEC/IEEE 29119-4.2015
IMPACT
- learners, students, and educators can use a software designed for learning programming and its fundamentals
- more interest and engagement of students to programming
- assistance for educators and instructors
- improvements in understanding and skills of students
- improvements in academic performances of students

Figure 1: Conceptual Framework of proposed Visual Programming Software for Learning Fundamentals of Programming

Time and Place of the Study

The study started from the approval of the title on the month of February of year 2020. The development is expected to be finished on the month of March of year 2021. The necessary data were gathered through survey and research at the Cavite State University and other information were researched and obtained from the Internet.

Scope and Limitations of the Study

The study focuses on the development of a Visual Programming Software for learning the fundamentals of programming. The software will prioritize simple and basic functionalities over numerous features for the purpose of learning and education.

The software is designed with seven core modules: Visual Nodes Module, Filesystem Module, Input and Output Module, Debug Module, Simulation Module, Transpiler Module, and Assessment Module.

Visual Nodes Module

Nodes are graphical elements that serve as the building blocks of the software. Nodes can be used as a variable, logic, and conditionals. The properties of the node are internal randomly generated UUID (Universally Unique Identifier), position, size, and type. The fields of the node that are visible to the users, which can be modified, are name, and value(s). Each node has input socket(s) and output pins which are used for the flow of logic and redirection of data. The visibility of the sockets and pins of a node is dependent on its type. For example, nodes that are constant variables will only enable the the output socket as it is read-only, while regular variable nodes allow for both sockets. Nodes are connected to one another through the use of wires. This relation of nodes is called the node graph. The flow of logic is easily determined using the wires with directional arrows signifying the direction of the logic.

Filesystem Module

This module serves as the interface between the software and the user's machine for handling files. The module have four main functionalities, creation, modification, reading of files, and backup. One feature of the software that benefits from this module is the importation of exercises from package format file which allows for more learning materials that greatly increases the possible usage and expandability of the software.

When a user starts a new project, a template project structure with base files are created by the module and is saved into the user's machine. The modules assures that the project is stored with proper permission and in safe location. Modification such as addition or deletion is also handled by the module. The reading of files and project functionality takes into consideration the validity and safeness of the file and handles if the file is corrupted. The backup functionality regularly makes a backup of file in case of emergency such as program crash or user-side accident.

Input and Output Module

This module captures user input events such as key press, mouse movement, mouse click, and so on. The module is responsible for processing and responding events and performing actions based on the event. This ensures that the interaction between the user and the software provides rich experience in terms of usability and learning.

This module handles output to the user. File, displays, views, and screens are examples of the possible types of output. The module manages everything that is rendered to the screen for the user to see such as the elements, the visual graph, a combination of the visual nodes connected to each other, the assessment files or reports, and the simulation view.

Debug Module

This module will linter and give feedback and indication to the user whenever there is an attempt to perform an action that is faulty in logic. For example, the red color means error or danger while the yellow color means warning. The color based feedback and highlight is used in combination with useful messages giving more detailed information regarding the fault. These are placed accordingly to the source of the fault, whether in the node or in the wire. Runtime errors or warnings during the simulation stage is propagated to the user with detailed information and explanations about the probable cause of error and displays tips in debugging and fixing the program.

Simulation Module

The process of simulation involves the compiling, building, and running the visual code is executed by this module. The compilation stage involves going from the main node which is the entry point of the program foloweed by the importing of libraries and packages (depending on the target language). After that is the declaration of variables and methods and will continue to parse and convert the visual code to its equivalent source code.

The compilation stage is followed by the building, also known as linking stage. During this stage, the compiled source code is linked with the necessary libraries required by the target programming language. Examples of this are Dynamic Linking Libraries (.dll) and Shared Objects (.so) files.

After the compilation and building stage is the simulation or execution stage. This executes the program and run it with additional features enabled to allow a more dynamic and intuitive interaction between the user and the program.

Transpiler Module

This module transpiles the visual code made by the user into source code in target programming language. This module tests that the transpiled source code compiles and runs correctly as well. This allows for learners to see and compare their work into other programming languages which is part of the education curriculum. This serves as helper for their transition from learning the fundamentals of programming into its application towards programming languages that are more robust, high quality, and industry-standard.

Assessment Module

The functionality of providing exercises designed for the learning of topics and concepts in programming and evaluation of the results are handled by the Assessment Module. The possible types of exercise range from output-based program to writing an algorithm that has memory and time limitation and complexity. The module can store the evaluated performances or grades of the user for further analyzation and can provide basic reports such as determining what concepts do most students find it difficult to learn within a set of time.

The software is limited to simulating text-based or command/terminal prompts as the priority is learning the fundamentals of programming. The software does not also compete as an Integrated Development Environment (IDE). The software does not provide networking functionalities such as connection to the internet to send or fetch data. There is also no access level or account management for the user.

Definition of Terms

Algorithm is a set of instructions designed to perform a specific task.

Bug is the term used in computers for the unwanted behaviors that happen during the runtime of the program. This can be caused by faulty logic and cases that are not handled during development.

Building is also called as Linking. It is the process after compilation wherein the compiled source code is linked along the libraries and other dependencies and bundles it to make an executable or binary.

Compilation is the process of translating human-readable source code into machine code which the computer can understand.

Conditionals are statements or expressions in programming that evaluate to true or false. This is used for controlling the flow of program execution. The basic example of this is the if-statement.

Data is information stored and processed by a computer. It can be in the form of text, numbers, images, documents, audio, and more.

Data Structure is the organization and storage of data that allows for efficient access and modification. It is a collection, relationships, functions, and operations that can be applied to data. Examples of this are arrays, list, and dictionary.

Data Types are attributes of data which allow the compiler to determine the usage and purpose of the data provided by the programmer. Examples of this are integer, boolean, character, double, and float.

Debugging is the processing of finding and resolving of errors and bugs in source code or computer programs.

GUI or Graphical User Interface is an interface that includes graphical elements that are usually interactive through the use of mouse and pointer such as windows, icons, and buttons.

Loops are statements, expressions, or structures in programming that repeat a sequence of instructions until the specified condition is met. Examples of this are while-loop and for-loop.

Memory stores data in digital format. In programming, it is used to store states and data which can be accessed, modified, or read by the program.

Module is a component in software or program that contains one or more routines. It is used in separation of functionalities into independent structures which are used in conjuction with one another to make up a larger system.

Nodes are graphical elements that contain minimum data and fields which can be connected to another to make up a node graph.

OOP or Object-Oriented Programming is a paradigm based on the concepts of classes and objects. Data are modeled through a class which have properties and methods, and are instantiated as objects.

Pedagogy is the science, art, or profession of teaching. It is the application of knowledge and skill in imparting or sharing one's knowledge or skill to others.

Programming is the implementation of logic for the purpose of peforming computer operations for achieving a task. It is a process wherein programmers input instructions for the computer to execute.

Programming Language is a human-readable language comprising a set of instructions that produce various kinds of output based. It is implemented in various ways such as difference with syntax and semantics. Examples of this are C/C++, Java, Python, Lua, and Javascript to name a few.

Runtime Error is an error that occured during the running of the program. This may be caused by error in logic, accessing a memory that is invalida, performing operations on invalid or incomparable values, and more.

Semantics is the determination and study of a programming language based on the evaluation of its syntax and tokens. Each programming language has different semantics, though some are identical.

Software is a program or collection of data and instructions that operates the computer in various functionalities and tasks.

Source Code is the set of codes written using a programming language.

Syntax in programming languages is the set of rules that defines and identifies the combinations of symbols, also called tokens. This set provides the structure on how the compiler or interpreter will process the source code.

Technology is the application of knowledge in a particular area or field.

Text-based Programming Language is the general and common type of programming language which allows programmers to write program code in text form. Examples of this are C, Java, and Python.

Terminal is an interface that accepts input or command from the user and send it to the computer to perform a certain task. It also displays text output. Typically, terminals only require keyboard hardware for functionality.

Transpilation is the process of converting a source code of a programming language to another programming language.

User-Interface is the layer that the user controls and interacts with that allows communication between the software or application and the hardware layer.

User-Interface Design is a set of principles that adhere to a standard of implemeting design to the user-interface of a software or an application.

User Experience provides aspects of the interaction experience of the end-user to the user-interface. Aspects such as usability, accessibility, and usefulness. It is also called UX.

Variable in programming is used for storing a data or value to the memory. Additional attributes can be added to a variable such as data type or its access method.

Visual Programming Language is a type of programming language that lets programmers manipulate and control logic through graphical elements instead of typing in text mode. Examples of this are Scratch, Alice, and Stagecast Creator.

REVIEW OF RELATED LITERATURE

This chapter discusses the collected literature and studies that contribute to attain the objectives of this study.

Context Diagram

Context diagram is a simple diagram that shows the source systems contributing data to a system as well as the major user constituents and downstream information systems that it supports. This diagram is so simple that it makes it perfect for agile requirement management. This diagram also called “Level 0” data flows diagrams because if one were to put arrows on the connections between sources and targets, the diagram could serve as the cover sheet of a data flow diagram packet that many analysts prepare for traditionally managed projects. This diagram greatly reduce project risk because they are easy for a team’s business partners to understand. (Hughes, 2016)

Data Flow Diagram

A data flow diagram illustrates the processes, data stores, and external entities in a business or other system and the connecting data flows. It is a graphical representation of the flow of data through information system. DFD was first proposed by Larry Constantine, the original developer of structured design in 1970s. It is a primary artifact and is required to be created for every system in a structured approach. It provides a different abstraction level that is useful in system designing because of its hierarchical structure. It shows data flow from external into the system and shows how the data moved from one process to another. There are four symbols for a data flow diagram: 1.) Squares or Ovals which represent external entities. It can be a person or a group of people outside the control of the system being modeled. It shows where information comes from and where it goes. 2.) Circles or Rounded Rectangles shapes represent processes within the system. They show a part of the system that transforms inputs to outputs. The name of the process in the symbols usually explains what the process does so that it is generally used with the verb-object phase. 3.) Arrows represents how the data flows. It can be electronic data or physical items or both. The name of the arrows represents the meaning of the packet that flows along. It also shows direction to indicate whether data or items are moving out or into a process. The last symbol is 4.) Open-Ended Rectangles which represents data stores, including both electronic stores and physical stores. Data stores might be used for accumulating data for a long or short period of times. (Aleryani, 2016)

Difficulty in Learning Programming

Different studies prove that many students has a poor learning in programming in the midst of their programming education. Through observation they found out that a lot of students are unable to read and write code effectively. There are few teachers who claimed that their students are able to meet the standards of programming by graduation however it was admitted that many programming graduates are still unable to program. (Carter and Jenkins, 1999)

An average student does not make much progress in an introductory programming course (Robins, Rountree, and Rountree, 2003). Most of the students struggled to get past in learning language features and never had a chance to learn higher programming skills and problem-solving strategies in programming (Linn and Dalbey, 1985). Several working groups have looked into the skill levels of the student at the end of CS1 courses in the past decades. These frequent studies has been beneficial as they prove the mismatch between programming education and the actual programming skill gained (McCracken et al., 2001). Programming teachers believe that one must learn how to read code before learning how to write a code. However, programming students give more attention to reading compared to writing codes. Some says that writing a code is much easier than reading (Lister et al., 2009)

A study that measured students' ability to trace through a given program's execution to follow-up McCraken's investigation. Multiple-choice questionnaire was given to CS1 graduating students around the world. The questions required the students to predict the values of variables at given points of execution and to complete short programs by inserting a line of code chosen from several given alternatives. The result was disappointing across the board as it shows that many students are unable to trace (Lister, 2004).

Another study found that novices were unable to mentally trace interactions within the system they were themselves designing (Adelson and Soloway, 1985). Another study reports that an inability to "trace code linearly" as a major theme of novice difficulties (Kaczmarczyk, Petrick, East, and Herman, 2010). The analyses of quiz questions indicate that many students fail to understand statement sequencing to the extent that they cannot grasp a simple three-line swap of variable values (Corney, Lister, and Teague, 2011).

End-User Programming Approaches

The types of programmer range from professional, novice, and end-user. Professional programmers are whole main work is to develop or maintain a code base. Novice programmers can be thought of as professional programmers under training. End-user programmers are those that program but programming is not their main function or career. Another case for comparing the types of programmers are their interest and knowledge when it comes to programming itself. Professional and novice programmers has more in-depth understanding about the processes involved in programming and are capable of programming using traditional semantic and text-based code whereas end-user programmers do not. The following are the various approaches to programming for end-users.

Preferences Programming

This is provided by applications to allow the users to modify the behaviors and visual appearances of the application itself. These are predefined options in the form of checkbox, radio button, or dropdown menu that the user can interact with to suit their preferences.

Programming by Demonstration

This programming approach uses a system for recording user inputs for future playback. This allows users to work in a general way to program the system what to accomplish by showing the actual actions. This approach is tightly rule-based to enforce the smooth replaying of actions. (Harrison, 2004)

Spreadsheet Programming

This approach focuses on the requirements of mathematical knowledge and skills in building formuals and models in the form of functions. Since this approach is visual-based as the spreadsheets constantly indicate the result of calculations for errors. (Abraham, Burnett, and Erwig, 2009)

Script Programming

This approach uses scripting languages as opposed to full programming language. A scripting language can be a subset of a programming language or embedded language. These languages are tiny and are generally designed to be used by people whose main domain is not programming. This application can range from game design, music generator, video effects, and prototyping. (Ousterhout, 1998)

Gantt Chart

Gantt chart is a classic tool in project management. It is one of the most known and widely used planning and management tool in projects in different domains. The principles for the development and design of Gannt chart are time-focused, objective, deterministic, analytic, accountable, and sequential. (Geraldi and Lechter, 2012)

Time-focus as projects have a target time for the either the completion or progress milestone. Each task should be well coordinated in time and work as a crucial part in project management.

Objective as projects must have ground in reality for the objectives to be met in a realistic and feasible manner.

Deterministic as each task should be properly defined, studied, and defined. This ensures that there should be no uncertainty in the objective and method of the tasks.

Analytical as projects are the sum of different and subset of tasks. A project must be analyzed very well and divided properly into smaller tasks. This should take into consideration the execution and scope of each task.

Accountable as a project gets divided into smaller parts, a project is also divided and assigned to different person. Each person should be accountable for the progress and completion of the task assigned.

Sequential as in the management of a project, there is always a sequence or order required for further tasks to be started by waiting for the completion of other tasks it depend upon. This sequence fit as a timeline analogy.

Gulf of Execution and Evaluation

This book features a study regarding the usage of things. The learning phase that occurs during usage has two gulfs, the gulf of execution wherein the user figures out and attempts how it works, and the gulf of evaluation wherein the user observes and comprehends the results of usage. This understanding in the part of the user is applicable in designing a system where the goal is to assist and improve learning. The gulf presents cases wherein users failing to use a simple object results in blaming one's self and users failing to learn a complex object results in forfeit in further learning. In reality, the fault is not solely on the user, but from the designer and the design of the object.

The study recommends that the design of the system should bridge the gulf by developing it to be accessible and understandable relative to the expectations of the users either through concise information or feedback per step on the behaviors. (Norman, 2013)

Haxe

Haxe is a high-level, Turing-complete, and packed with features programming language. It is modernly designed and implemented that there are times it feels native Java, sometimes JavaScript, and sometimes Python. The Haxe framework is suitable for complex projects that can target desktop, mobile, web, and the cloud.

The unique feature of Haxe is its cross-language compilation, also called transpilation. This language can target whatever platforms the target language is capable of. It can run natively if targetted to C/C++, it can run in the web if targetted to JavaScript, it can run in the mobule if targetted to Java, and more.

Haxe is also a statically-typed language which allows for the safest code to be written, analyzed, and checked during compilation to catch minor issues. This also allows for IDE and toolings support across a variety of softwares. (Coates, 2018)

Haxe being open-source allows for a population of contributors, testers, and users who actively continue to improve the language along with popular libraries (written in other programming languages) to be available for the Haxe ecosystem. A large repository of packages and libraries that complement the standard library can be easily found and integrated using the Haxe Library Manager (haxelib). To prove that Haxe can be used in the industry and in complex and big projects, Haxe showcases successful big games, softwares, tools, and websites. (Cannasse, 2020)

Ishikawa Diagram

Ishikawa diagram is also called the fishbone diagram and cause-and-effect diagram in that it is a graphical technique in the shape of a fish skeleton used to numerous causes of a phenomenon. It is commonly used to identify and analyze causes and its complex relation to each other that contribute to the specific problem.

A finding of a study about the use of Ishikawa diagram as an appropriate theoretical framework for representing visually and analyzing technology of complex factors of major improvements and innovations over the course of history and time. This graphical representation tool presents a simple and clear the order and relation of the causes and roots of a problem addressed by the change in technology. (Coccia, 2017)

Six Learning Barriers in End-User Programming Systems

The researchers identified the following aspects prone to false assumptions that include fundamental and basic concepts in programming as barriers to learning programming. These barriers closely related to the concept of interfaces of a programming environment such as the constructs of the language itself and the availability of libraries, features, and syntax that can be used to achieve desired procedures. (Ko, Myers, and Aung, 2004)

Design barriers are internal difficulties of a problem in programming. Solutions to problems which are difficult to visualize affect the learning experience and may lead to false assumptions and confusions.

Selection barriers occur when learners know what to do but are unable to identify which of the available tools and features in the programming interface is to correct.

Coordination barriers are difficulties in using libraries provided by the programming environment in compliment with another. Learner may know how to solve individual and simple tasks but fails to combine the approaches to solve complex problems.

Use barriers are inherent to users new to the environment. The unfamiliarity to the interface hinders its usage due to the lack of information and guide.

Understanding barriers involve the obscurity of the processes the programming interface do that are hidden to the users. This occurs when learners fail to evaluate and undestand the behavior of the program relative to their expectations.

Information barriers are difficulties in obtaining information about the internal workings of the interface. This occurs when the environment provides no method for the users to test their hypothesis regarding the behaviors of the environment.

The barriers explicitly defined are closely related to each other. The effect of having difficulties in overcoming a barrier affects the learning of another barrier.

Software Visualization

Software visualization tools have different types for different use cases. The following are broad classifications of the visualization tools: program visualization, algorithm visualization, and visual programming.

Program visualization is used to determine the runtime behavior of a program and visualize it for the user to see and inspect the information. This is commonly used for debugging programs such as showing of virtual memory and CPU usage.

Algorithm visualization is used to visually show the each step in the process of running an algorithm. An example of this is sorting algorithm wherein elements that represent a value to be sorted are in every iteration selected, compared, and sorted. This is to show a high level of abstraction for learning and understanding the concept of an algorithm.

Visual Programming is similar to program visualization but is distinct enough to be set as a different classification. Compared to other visualization type, visual programming allows interaction with the visualization rather than the visualization can be interacted. This is used as a mean to program visually as opposed to visually see the program.

Teaching and learning programming through visualization is a pedagogically sound approach. The nature of a program is that the code is static but during runtime it is dynamic. The dynamic aspect is difficult to learn at first especially for novice programmers as they need to form a mental model of the processes involves based on logic and set of theories. (Sorva, 2012)

User Interface and User Experience

A user interface (UI) refers to a system and a user interacting with each other through commands or techniques to operate the system, input data, and use the contents. This ranges from systems such as computers, mobile devices, games, to application programs and content usage. On the other hand, user experience (UX) refers to the overall experience of the user. This includes the perception, reaction and behavior that the user may feel and think in direct or indirect usage of the system, product, content or services. It is a concept that is widely applied not only in software and hardware development but also in services, products, processes, society and culture. Both UI and UX is an interface through which a person can interact with a system or application in a computer and communication environment, which is classified into a software and hardware interface. Software interface is represented by the user interface while hardware interface is categorized into a plug or an interface card connecting the computer and its peripheral devices. UX's has four key axes which are needs, expectations, attributes and capabilities. Hence, it identifies the problem with the need of the users, applies motivation and manage the expectations of the users. (Joo, 2017)

Visual Learning

Information is retained more in memory through visual formats. Visual information can be presented in various formats such as images, diagrams, graphs, video, and simulations. This approach in learning helps the instructors to convey their lesson better and clearer while the students develop visual thinking skills. This skill is the comprehension of association of data such as concepts, theories, and ideas into graphical elements like imagery and diagram. (Raiyn, 2020)

Visual learning can be improved more through the addition of interaction using visual interactive tools. This is beneficial in many domains that require logical thinking and skill such as programming. Interaction and visualization at the initial level of coding motivates the interests and engagement of learners even at the young age. This approach has been very effective for the Scratch programming environment as they adapted to adding visualization and media content creation to programming activities which are trends in the culture of youth. Learning through exploration and sharing to peers, this motivated young people to focus less on direct instruction that other programming languages provide. (Maloney, Resnick, Rusk, Silverman, and Eastmond, 2010)

Having a physical design, blueprint, or a diagram that serves as guide for the product or machine to be made has been the traditional method for manufacturing complex and expensive things. The same principle applies in programming. Programmers manually input code from their brain which can be called as mental model to task the computer into doing a complex routine. But this is a challenge for many reasons such as other people does not inherently have the same mental model regarding the solution and structure. Another reason is that the level of familiarity and expertise to a particular tool or environment used is not the same for all programmers. So intead of from one's mental model to code, it should suffice to create and visualize the model itself before jumping into directly generating the code. This allows for coordination between multiple programmers as they have the same guide for the solution and concept. This also applies for novice in programming to learn that visualization before coding is a discipline one must come to understand and put into practice. This application of visuals into learning and execution could be of great benefit to reduce the complexity, effort, and time consumption. (Ottosson and Zaslavskyi, 2019)

Visual Programming Language

Conveniently, explaining what a program does leads to the usage of graphical representation of the control flow, connections, shapes, and more elements. This could also be applied for programming and learning it. Visual programming languages enable users to achieve the same concept. (D., 2015)

Aside from programming logic, visual programming languages are also used in a wide variety of applications and has corresponding types. Some of these are the following:

The drag-and-drop type of visual programming language uses blocks as elements that can fit into other blocks for composition, similar to a jigsaw puzzle piece. A study that compared drag-and-drop visual programming to text-based programming concluded that respondents were confident in their knowledge and skill in performing simple and basic command with visual language programming, but found it harder to express what they want in drag-and-drop for complex problems. This suffice to using visual programming as first steps in basic programming before proceeding to complex concepts (DiSalvo, 2014).

Flowchart-inspired type visual programming languages provide basic and limited capabilities for programming. The common usage for this is evaluation of the conditionals and flow of the program. It mainly uses arrows and boxes with simple value. Examples of this are Flowgorithm, Raptor, and WebML.

Dataflow type visual programming language commonly used in professional applications moreso for designers than programmers. With this format, there is a wide selection of available capabilites as each block represents a function or procedure which can store and output multiple values through lines or wires. Examples of this are Unreal Blueprint and CryEngine Flow Graph.

The Finite-state Machines (FSM) type uses basic shapes and connections only. This is commonly used for animation and states to visualize the transition from one block to another. Example of this is NodeCanvas.

Behavior Trees type is similar to Finite-state Machines but is more complex and allows for multiple states to be triggered depending on the parameters that match the current state. This is mostly used for complex animation in the game industry. Examples of this are NodeCanvas and Craft.ai.

Event-based type of visual programming language is the simplest and most akin to the traditional text-based programming languages. The simplest illustration to define this is to write a programming code in text form and then assign a graphical element or picture of each keyword. For example, the picture for the keyword ”for” will be a looping arrow. Examples of this are Construct, IFTTT, and Kodu.

Visual Programming Software

Visual programming is commonly built-in or provided by industry-size softwares for allowing non-programmers to also perform complex logic and controls without the need to learn traditional text-based programming languages. This is widely popular in the game development field as desginers and artists can create visual effects through the use of visual programming language.

The development of softwares that support visual programming for novice programmers such as Scratch and Snap! results in learners learning to code without the need for grammer correctness as needed in traditional text-based programming languages (Bau, Gray, Kelleher, Sheldon, and Turbak, 2017). But Scrach and Snap! has their own programming languages which the visual programming side exports to. This increases the effort and time required to transition into learning popular programming languages like C, Python, and Java. There are environments wich allow the visual code to output in C language but it can not execute, one needs to copy and paste the output to another editor to run it. (Abe, Fukawa, and Tanaka, 2019)

For Java, there exist numerous visual programming softwares such as Symantec Cafe, Visual J++, and Visual Age for Java. The core feature of the softwares is to enable end-users to manipulate elements of the interface in their natural graphical representation. The softwares allow for editing the packages, classes, methods, and variables of the available elements. But the following require basic understanding and or experience already with programming, particularly in Java, to make full use of the softwares. (Prokhorov and Kosarev, 1999)

AgentSheets is one of the early pioneers of the concept of visual programming. This visual programming softwares use block-based type of graphical elements similar to a jigsaw puzzle piece. The elements can be dragged and dropped onto another to create composition for the logic of the program. Since most of the elements are visually represented, this allows for comprehension as a block or group of blocks can explain itself in terms of its purpose, control, and logic. Another key concept of visual programming expressed in AgentSheets is easy sharing of blocks with one another instead of looking at plain texts. (Repenning, 2017)

Visual Programming vs. Text-based Programming

Programming is seen to be as the career of the future as technology continues to scale up and improve. For this reason many countries are already promoting and implementing programming subjects to their curriculum in primary education (Williams, Alafghani, Daley, Gregory, and Rydzewski, 2015). Most of these formal subjects use a visual programming language for teaching instead of the traditional text-based programming language.

It would seem to be appropriate and more productive to teach text-based language as it is the standard and the type of programming language used for the development of softwares and applications in the industry and the real world. However, in considering the comparison of the engagement, interest, and actual learning of the students at the introductory level, visual programming language is more suited and ideal as results showed that the motivation of the subjects that use visual programming language improved (Tsukamoto et al., 2016).

Studies also show that using a visual programming language like Scratch to teach students programming and transitioning to real programming language (text-based) shows a marginal improvements to their understanding of computer science concepts (Armoni, Meerbaum-Salant, and Ben-Ari, 2015). Visual-based environment as compared to textual environment also displays positive results in terms of the interest and motivation of the learners to pursue programming (Saito, Washizaki, and Fukazawa, 2017)

Novice programmers take longer time to learn programming because of many constraints that are needed to be learnt first such as the syntax and semantics of a programming language. Common errors and difficulties in text-based programming languages include type corrections, misspellings, typographical errors, and grammatical or syntax errors. Another factor is that students whose native language is not English find it harder to type because the keywords in most programming languages are in English. Even the tools used, text editors or integrated development editors (IDE) are hard to operate and use. (Liu, Wu, and Dong, 2010)

METHODOLOGY

This chapter shows the materials and methods that the researchers will use for the development of CodeNect: Visual Programming Software for Learning Fundamentals of Programming.

Materials

For the development of CodeNect: Visual Programming Software for Learning Fundamentals of Programming, the researchers use the following specifications:

For the hardware requirements, the following materials are used in the development of the software: Laptop with 2 GB of RAM (Random Access Memory), processor of Intel Core 2 Duo (1.4 GHz), and storage of 80 GB HDD (Hard Disk Drive).

For the software requirements, the following materials are used in the development of the software: Linux 5.4 Kernel with Manjaro distrubution as Operating System, Terminal for running commands, Vim for text and code editor, Kha for the graphical, media, and control framework, zui for the base user-interface, and Haxe as programming language.

Methods

The researchers have decided to use the V-Model methodology of Software Development Life Cycle (SDLC) for the proposed software to be developed. The V-Model methodology is a linear development methodology that focuses and follows a strict and incremental steps of stages. The initial phases are generally focused on planning and designing the system, the next phases are focused on implementation and actual programming. After that, the model will go in upwards direction for testing and verification of the project. The development of the software follows the timeline (See Figure 8).

The V-Model figure shows the following stages:

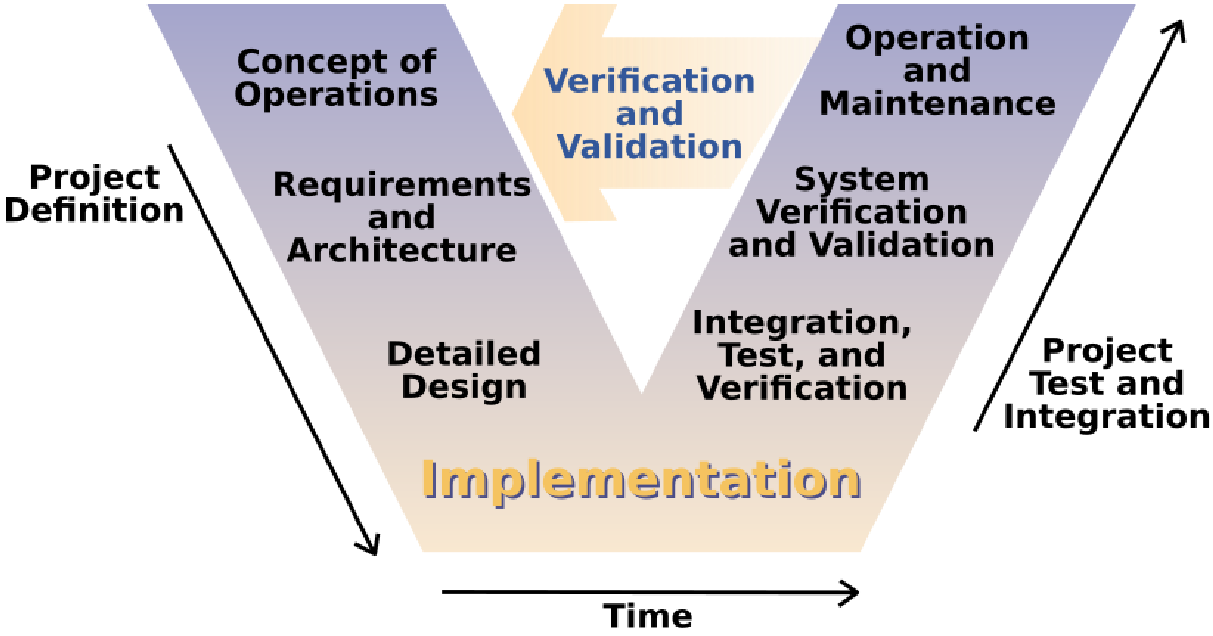

Concept of Operations
Verification and Validation
Operation and Maintenance
Project Definition
Requirements and Architecture
System Verification and Validation
Detailed Design
Integration, Test, and Verification
Project Test and Integration
Implementation
Time

Figure 2: V-Model Figure

Requirements. The first stage in the V-Model involves the gathering of data, analysis of the gathered data, and preparation of the system requirements in defining the scope and features of the project. This stage also involves the documentation of the requirements.

In this stage the researchers will conduct a survey to gather data from students, instructors, and learners in the field of technology and under the course with programming subjects such as Bachelor of Science in Information Technology, Bachelor of Science in Computer Science, and Bachelor of Science in Computer Engineering as respondents. The gathered data (See APPENDIX A) will be evaluated and assessed to determine the knowledge and understanding of the respondents in regards to the fundamentals of programming, experience and feedback on traditional text-based tools and softwares.

System Design. The documents created in the previous stage will be used to generate more specific and technical documents and designs about the software to be developed for this stage. The documents outline the components, modules, and high-level guidelines for business logic.

Architecture Design. The technical designs made from the System Design stage will be used to generate specifications with lower level of technical details about the software and its modules. The technology stack is also selected in this stage. During this stage, tests are also prepared for future use.

The researchers in this stage will design and develop specifications that will serve as the blueprint of the software. The libraries, packages, tools, and more are finalized and prepared for later use. (See APPENDIX C)

Module Design. In this stage, low-level designs are developed from the high-level designs generated from previous stages. This will include specifications regarding each individual module, component, interface, and so on. Unit tests are also prepared in this stage.

The researchers will identify and define the scopes and specific features of each module and how will each be integrated along the system to work with other modules and components to ensure that each module is decoupled and can be tested without dependency in other module.

Implementation and Coding. This stage the implementation through programming starts. It starts with coding the each module individually with unit tests along applied following the designs made in the initial stages of the development life cycle. Integration of the modules to the system is done afterward and is run through system tests.

The researchers will start to program each module, after each module pass the unit test, all will be coupled and integrated as a single system.

Testings. Tests are further applied to the system such as unit testing, integration testing, system testing, and acceptance testing. Passing all these tests will be considered as the verification and validation of the project.

The researchers will apply tests in the software developed. Further testings will be done and bugs, errors, and misbehaviors will be handled and fixed.

REFERENCES

Abe, K., Fukawa, Y., & Tanaka, T. (2019). Prototype of visual programming environment for c language novice programmer.

Abraham, R., Burnett, M., & Erwig, M. (2009). Spreadsheet programming.

Adelson, B., & Soloway, E. (1985). The role of domain expenence in software design.

Alam, A., & Bush, V. (2016). Haskeu: An editor to support visual and textual programming in tandem.

Aleryani, A. (2016). Comparative study between data flow diagram and use case diagram.

Anggrawan, A., Ibrahim, N., M., S., & Satria, C. (2016). Influence of blended learning on learning result of algorithm and programming.

Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From scratch to “real” programming.

Bau, D., Gray, J., Kelleher, C., Sheldon, J., & Turbak, F. (2017). Learnable programming.

Cannasse, N. (2020). Haxe - the cross-platform toolkit. Retrieved from https://haxe.org

Carter, J., & Jenkins, T. (1999). Gender and programming: What’s going on?

Chang Boon Lee, P. (2011). Toward intuitive programming languages.

Cheng, E. (2016). Learning through the variation theory: A case study.

Coates, P. (2018). Haxe — the coolest language you’ve never heard of - dzone integration. Retrieved from https://dzone.com/articles/haxe-the-coolest-language-youve-never-heard-of

Coccia, M. (2017). The fishbone diagram to identify, systematize and analyze the sources of general purpose technologies.

Corney, M., Lister, R., & Teague, D. (2011). Early relational reasoning and the novice programmer: Swapping as the ”hello world” of relational reasoning.

D., R. (2015). The maturity of visual programming. Retrieved from https://www.craft.ai/blog/the-maturity-of-visual-programming

DiSalvo, B. (2014). Graphical qualities of educational technology: Using drag-and-drop and text-based programs for introductory computer science.

Geraldi, J., & Lechter, T. (2012). Gantt charts revisited.

Harrison, W. (2004). From the editor: The dangers of end-user programming.

Hughes, R. (2016). Artifacts for the enterprise requirements value chain.

Joo, H. (2017). A study on understanding of ui and ux, and understanding of design according to user interface change.

Kaczmarczyk, L. C., Petrick, E. R., East, J. P., & Herman, G. L. (2010). Identifying student misconceptions of programming.

Ko, A. J., Myers, B. A., & Aung, H. H. (2004). Six learning barriers in end-user programming systems.

Linn, M. C., & Dalbey, J. (1985). Cognitive consequences of programming instruction: Instruction, access, and ability.

Lister, R. (2004). Teaching java first: Experiments with a pigs-early pedagogy.

Lister, R., Clear, T., Simon, Bouvier, D. J., Carter, P., Eckerdal, A., . . . Robbins, P. e. a. (2009). Naturally occurring data as research instrument.

Liu, Y., Wu, L., & Dong, X. (2010). Research on controls-based visual programming.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The scratch programming language and environment.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., . . . Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year cs students.

Norman, D. (2013). *The design of everyday things*.

Ottosson, S., & Zaslavskyi, V. (2019). Visualize what to be coded before programming.

Ousterhout, J. (1998). Scripting: Higher level programming for the 21st century.

Prahofer, H., Hurnaus, D., Wirth, C., & Mossenbock, H. (2007). The domain-specific language monaco and its visual interactive programming environment.

Prokhorov, V., & Kosarev, V. (1999). Environment pij for visual programming in java.

Raiyn, J. (2020). The role of visual learning in improving students' high-order thinking skills.

Raja, R., & Nagasubramani, P. C. (2018). Impact of modern technology in education.

Repenning, A. (2017). Moving beyond syntax: Lessons from 20 years of blocks programming in agentsheets.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion.

Saito, D., Washizaki, H., & Fukazawa, Y. (2017). Analysis of the learning effects between text-based and visual-based beginner programming environments.

Sorva, J. (2012). Visual program simulation in introductory programming education.

Thompson, D. V., Hamilton, R. W., & Rust, R. T. (2005). Feature fatigue: When product capabilities become too much of a good thing.

Tsai, C.-Y., Yang, Y.-F., & Chang, C.-K. (2015). Cognitive load comparison of traditional and distributed pair programming on visual programming language.

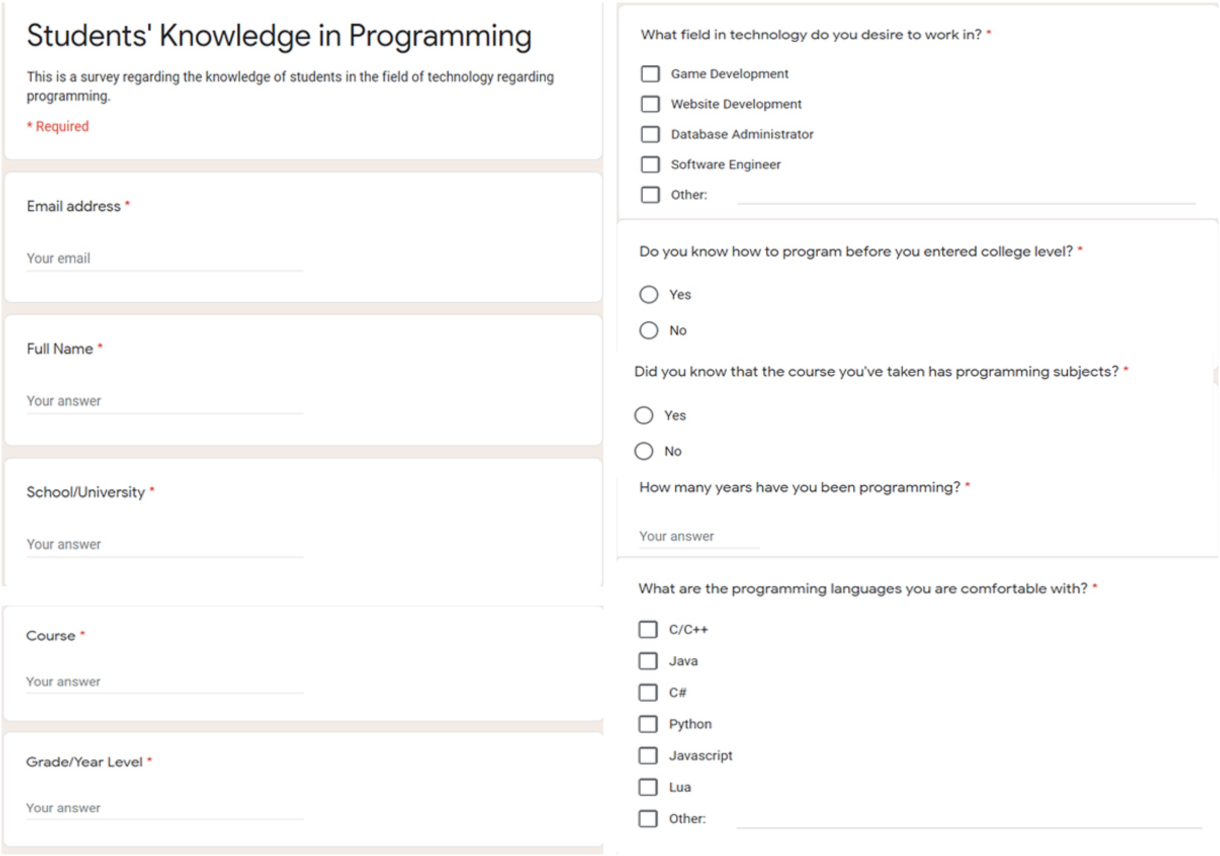
Tsukamoto, H., Takemura, Y., Oomori, Y., Ikeda, I., Nagumo, H., Monden, A., & Matsumoto, K.-i. (2016). Textual vs. visual programming languages in programming education for primary schoolchildren.

Williams, C., Alafghani, E., Daley, A., Gregory, K., & Rydzewski, M. (2015). Teaching programming concepts to elementary students.

APPENDICES

APPENDIX A

Form A



Students' Knowledge in Programming

This is a survey regarding the knowledge of students in the field of technology regarding programming.

* Required

Email address *

Your email

Full Name *

Your answer

School/University *

Your answer

Course *

Your answer

Grade/Year Level *

Your answer

What field in technology do you desire to work in? *

□ Game Development

□ Website Development

□ Database Administrator

□ Software Engineer

□ Other: ____________

Do you know how to program before you entered college level? *

○ Yes

○ No

Did you know that the course you've taken has programming subjects? *

○ Yes

○ No

How many years have you been programming? *

Your answer

What are the programming languages you are comfortable with? *

□ C/C++

□ Java

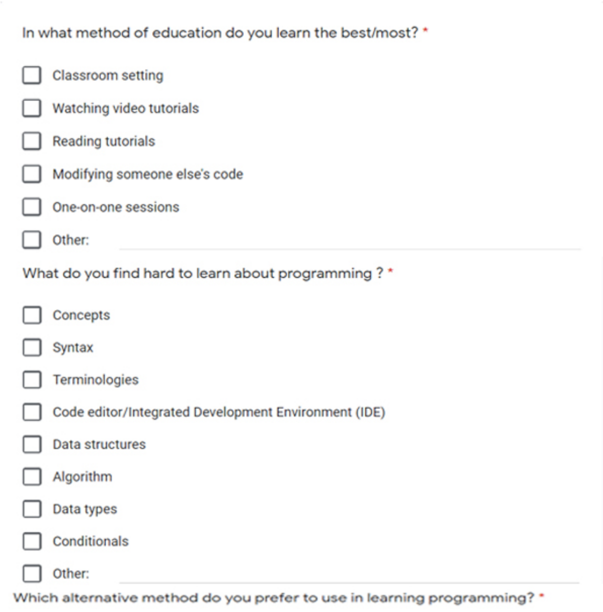
□ C#

□ Python

□ Javascript

□ Lua

□ Other: ____________



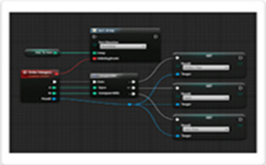
In what method of education do you learn the best/most? *

☐ Classroom setting
☐ Watching video tutorials
☐ Reading tutorials
☐ Modifying someone else's code
☐ One-on-one sessions
☐ Other: ______

What do you find hard to learn about programming ? *

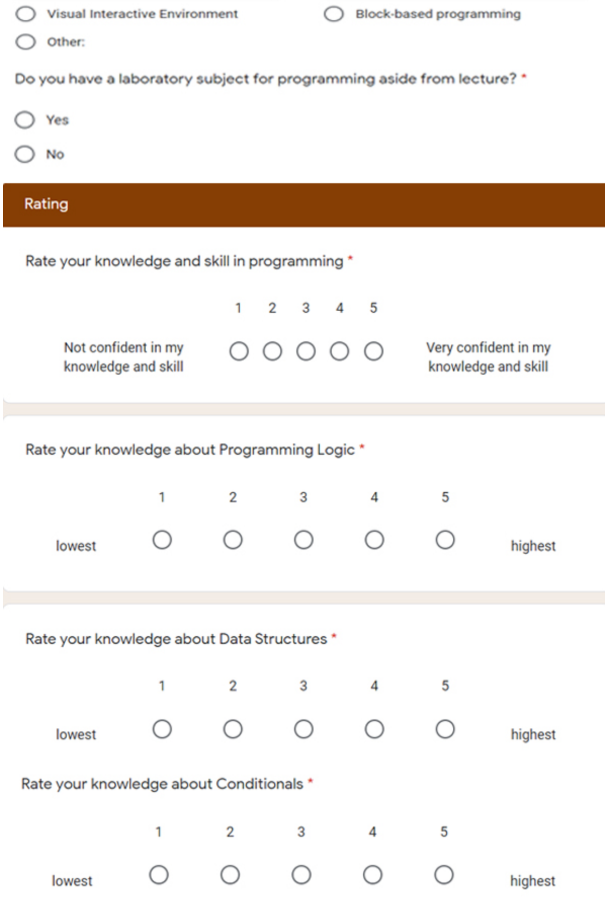
☐ Concepts
☐ Syntax
☐ Terminologies
☐ Code editor/Integrated Development Environment (IDE)
☐ Data structures
☐ Algorithm
☐ Data types
☐ Conditionals
☐ Other: ______

Which alternative method do you prefer to use in learning programming? *



○ Visual Interactive Environment

○ Block-based programming



○ Other:

Do you have a laboratory subject for programming aside from lecture? *

○ Yes
○ No

Rating

Rate your knowledge and skill in programming *

	1	2	3	4	5	
Not confident in my knowledge and skill	○	○	○	○	○	Very confident in my knowledge and skill

Rate your knowledge about Programming Logic *

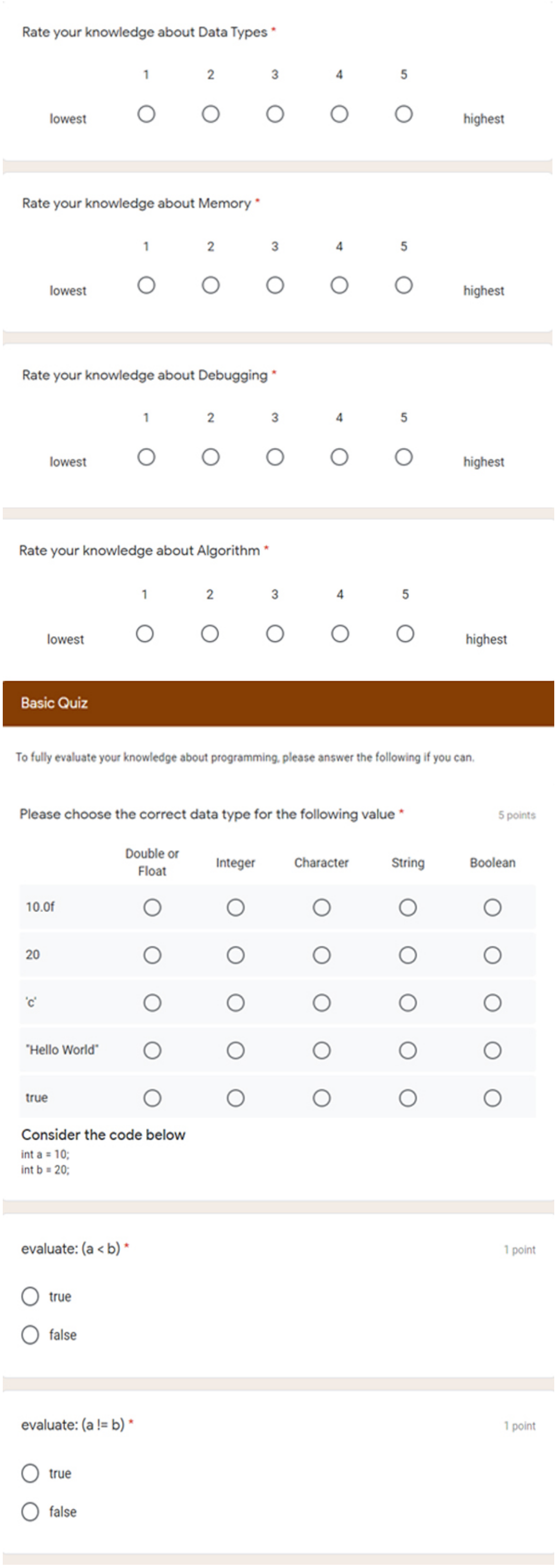
	1	2	3	4	5	
lowest	○	○	○	○	○	highest

Rate your knowledge about Data Structures *

	1	2	3	4	5	
lowest	○	○	○	○	○	highest

Rate your knowledge about Conditionals *

	1	2	3	4	5	
lowest	○	○	○	○	○	highest



Rate your knowledge about Data Types *

	1	2	3	4	5	
lowest	○	○	○	○	○	highest

Rate your knowledge about Memory *

	1	2	3	4	5	
lowest	○	○	○	○	○	highest

Rate your knowledge about Debugging *

	1	2	3	4	5	
lowest	○	○	○	○	○	highest

Rate your knowledge about Algorithm *

	1	2	3	4	5	
lowest	○	○	○	○	○	highest

Basic Quiz

To fully evaluate your knowledge about programming, please answer the following if you can.

Please choose the correct data type for the following value * 5 points

	Double or Float	Integer	Character	String	Boolean
10.0f	○	○	○	○	○
20	○	○	○	○	○
'c'	○	○	○	○	○
"Hello World"	○	○	○	○	○
true	○	○	○	○	○

Consider the code below

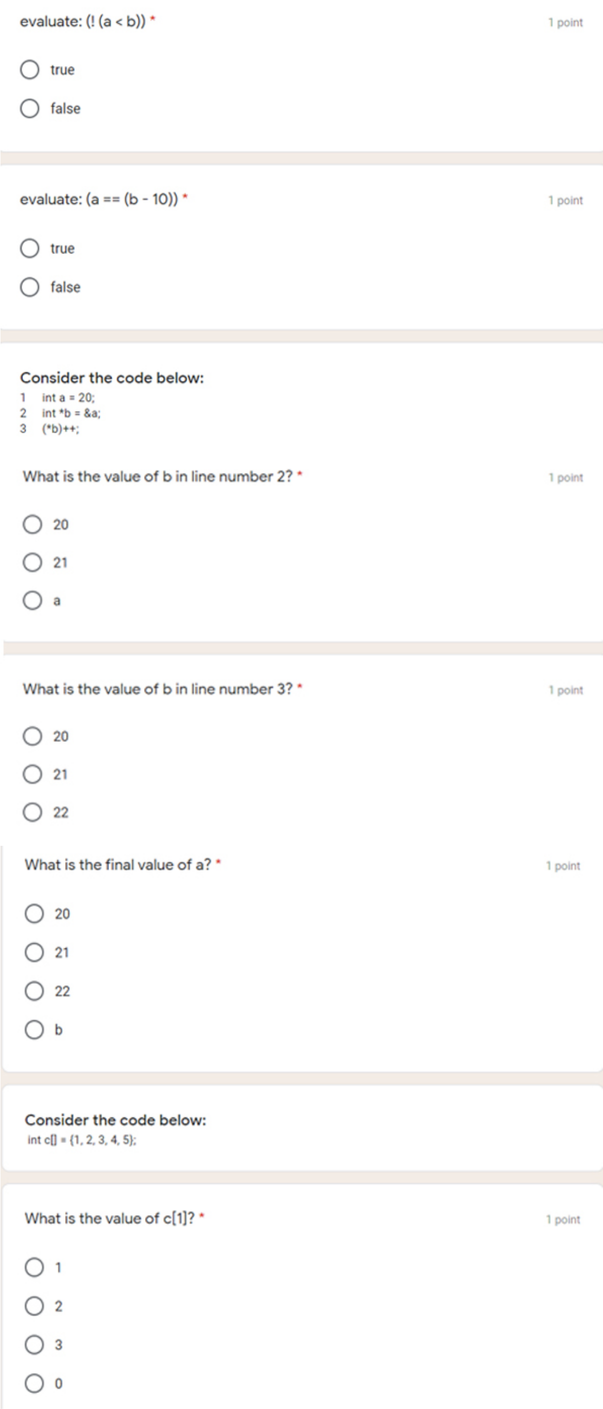
```
int a = 10;
int b = 20;
```

evaluate: (a < b) * 1 point

○ true
○ false

evaluate: (a != b) * 1 point

○ true
○ false



evaluate: (! (a < b)) *

1 point

○ true

○ false

evaluate: (a == (b - 10)) *

1 point

○ true

○ false

Consider the code below:

```
1   int a = 20;
2   int *b = &a;
3   (*b)++;
```

What is the value of b in line number 2? *

1 point

○ 20

○ 21

○ a

What is the value of b in line number 3? *

1 point

○ 20

○ 21

○ 22

What is the final value of a? *

1 point

○ 20

○ 21

○ 22

○ b

Consider the code below:

```
int c[] = {1, 2, 3, 4, 5};
```

What is the value of c[1]? *

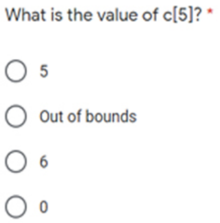
1 point

○ 1

○ 2

○ 3

○ 0



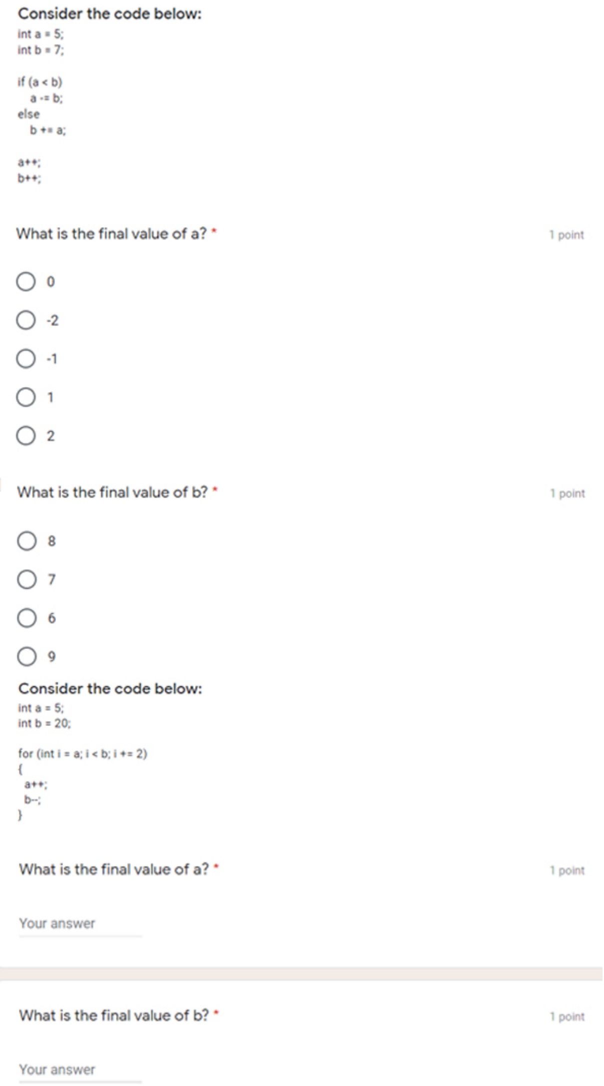
What is the value of c[5]? *

○ 5

○ Out of bounds

○ 6

○ 0



Consider the code below:

```
int a = 5;
int b = 7;

if (a < b)
   a -= b;
else
   b += a;

a++;
b++;
```

What is the final value of a? *

1 point

○ 0

○ -2

○ -1

○ 1

○ 2

What is the final value of b? *

1 point

○ 8

○ 7

○ 6

○ 9

Consider the code below:

```
int a = 5;
int b = 20;

for (int i = a; i < b; i += 2)
{
  a++;
  b--;
}
```

What is the final value of a? *

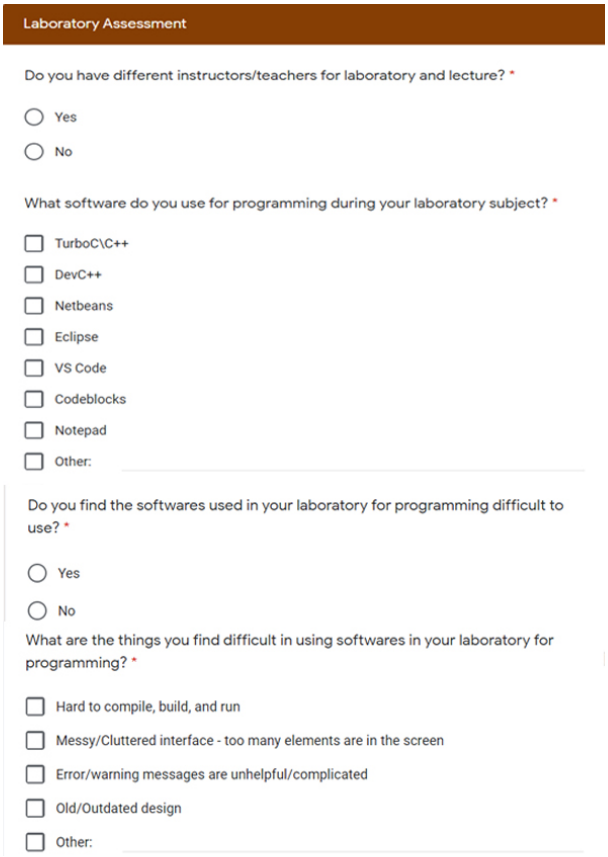
1 point

Your answer

What is the final value of b? *

1 point

Your answer



Laboratory Assessment

Do you have different instructors/teachers for laboratory and lecture? *

○ Yes

○ No

What software do you use for programming during your laboratory subject? *

□ TurboC\C++

□ DevC++

□ Netbeans

□ Eclipse

□ VS Code

□ Codeblocks

□ Notepad

□ Other: __________

Do you find the softwares used in your laboratory for programming difficult to use? *

○ Yes

○ No

What are the things you find difficult in using softwares in your laboratory for programming? *

□ Hard to compile, build, and run

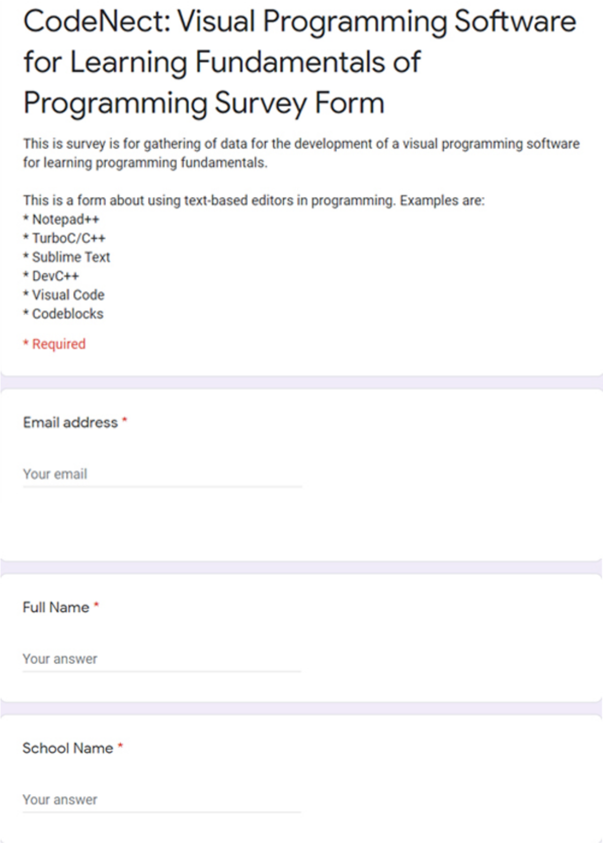
□ Messy/Cluttered interface - too many elements are in the screen

□ Error/warning messages are unhelpful/complicated

□ Old/Outdated design

□ Other: __________

Form B



CodeNect: Visual Programming Software for Learning Fundamentals of Programming Survey Form

This is survey is for gathering of data for the development of a visual programming software for learning programming fundamentals.

This is a form about using text-based editors in programming. Examples are:
* Notepad++
* TurboC/C++
* Sublime Text
* DevC++
* Visual Code
* Codeblocks

* Required

Email address *

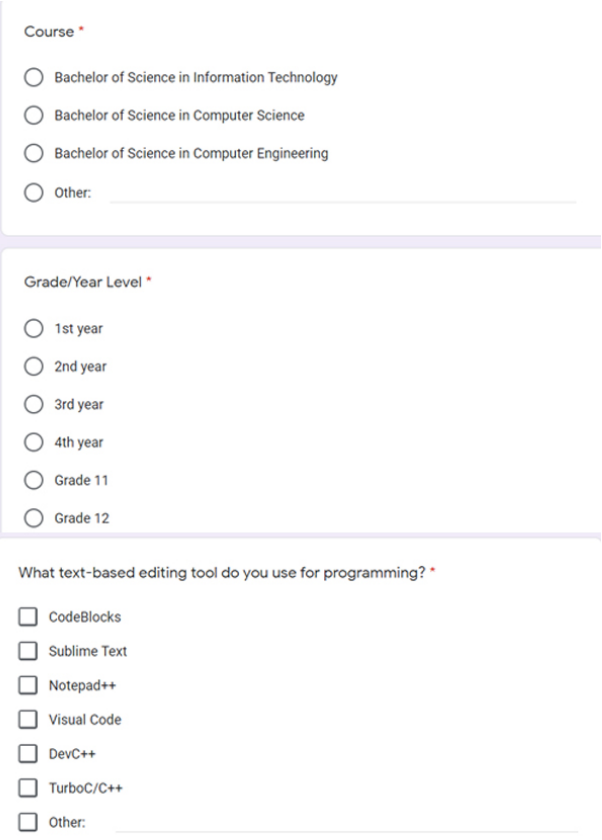
Your email

Full Name *

Your answer

School Name *

Your answer



Course *

○ Bachelor of Science in Information Technology

○ Bachelor of Science in Computer Science

○ Bachelor of Science in Computer Engineering

○ Other: __________

Grade/Year Level *

○ 1st year

○ 2nd year

○ 3rd year

○ 4th year

○ Grade 11

○ Grade 12

What text-based editing tool do you use for programming? *

□ CodeBlocks

□ Sublime Text

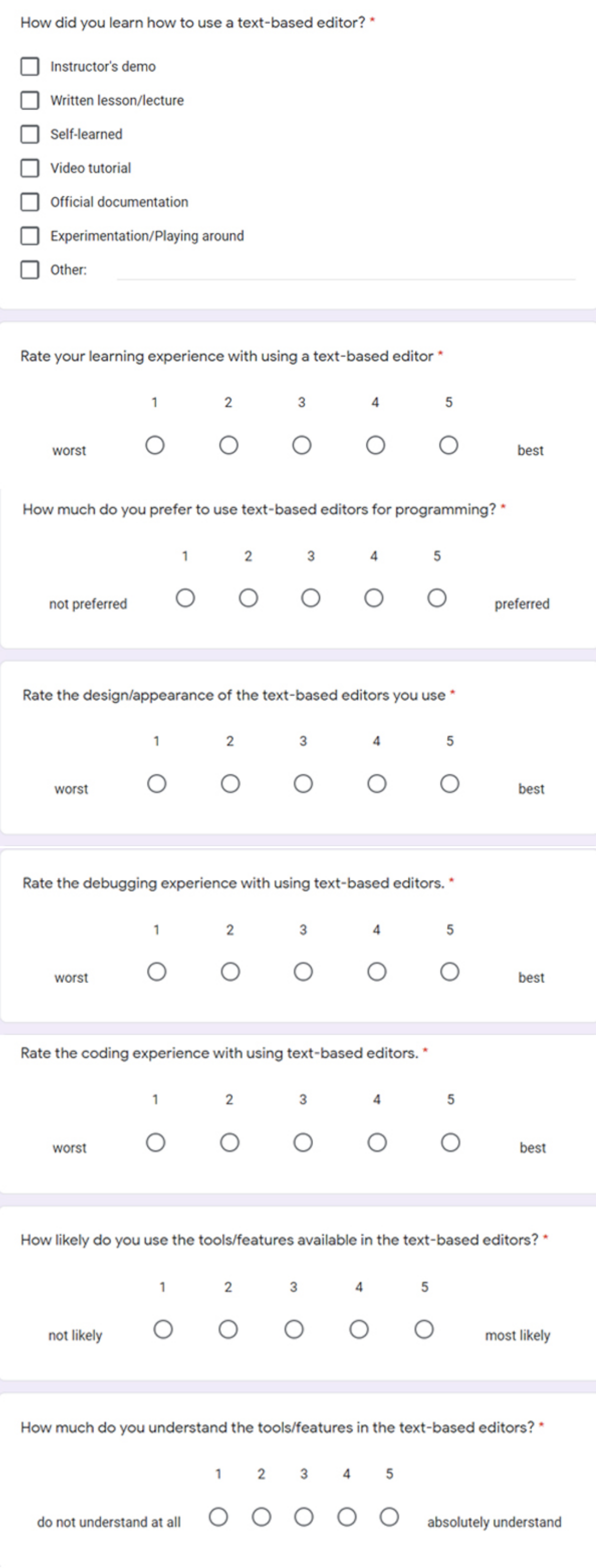
□ Notepad++

□ Visual Code

□ DevC++

□ TurboC/C++

□ Other: __________



How did you learn how to use a text-based editor? *

□ Instructor's demo

□ Written lesson/lecture

□ Self-learned

□ Video tutorial

□ Official documentation

□ Experimentation/Playing around

□ Other: ____________

Rate your learning experience with using a text-based editor *

	1	2	3	4	5	
worst	○	○	○	○	○	best

How much do you prefer to use text-based editors for programming? *

	1	2	3	4	5	
not preferred	○	○	○	○	○	preferred

Rate the design/appearance of the text-based editors you use *

	1	2	3	4	5	
worst	○	○	○	○	○	best

Rate the debugging experience with using text-based editors. *

	1	2	3	4	5	
worst	○	○	○	○	○	best

Rate the coding experience with using text-based editors. *

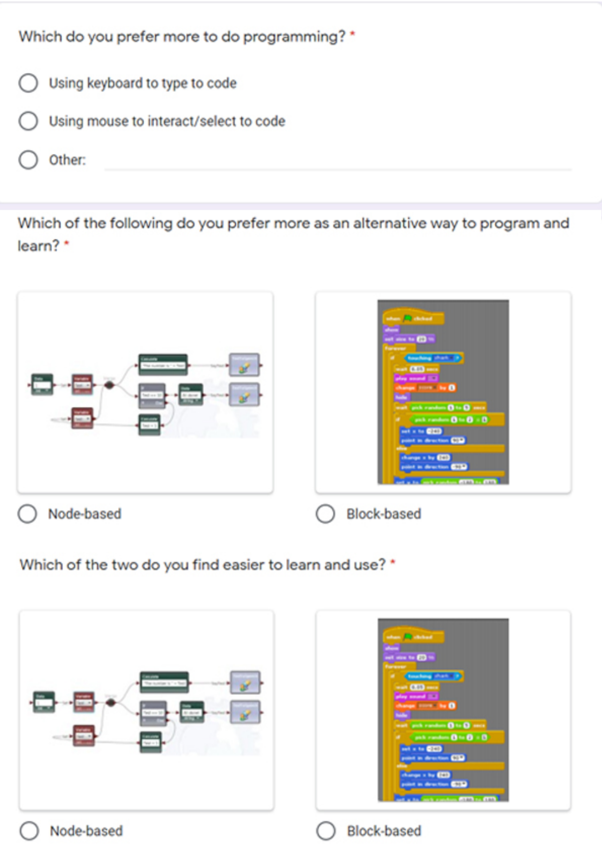
	1	2	3	4	5	
worst	○	○	○	○	○	best

How likely do you use the tools/features available in the text-based editors? *

	1	2	3	4	5	
not likely	○	○	○	○	○	most likely

How much do you understand the tools/features in the text-based editors? *

	1	2	3	4	5	
do not understand at all	○	○	○	○	○	absolutely understand



Which do you prefer more to do programming? *

○ Using keyboard to type to code

○ Using mouse to interact/select to code

○ Other: ____________

Which of the following do you prefer more as an alternative way to program and learn? *

○ Node-based

○ Block-based

Which of the two do you find easier to learn and use? *

○ Node-based

○ Block-based

APPENDIX B

Ishikawa Diagrams

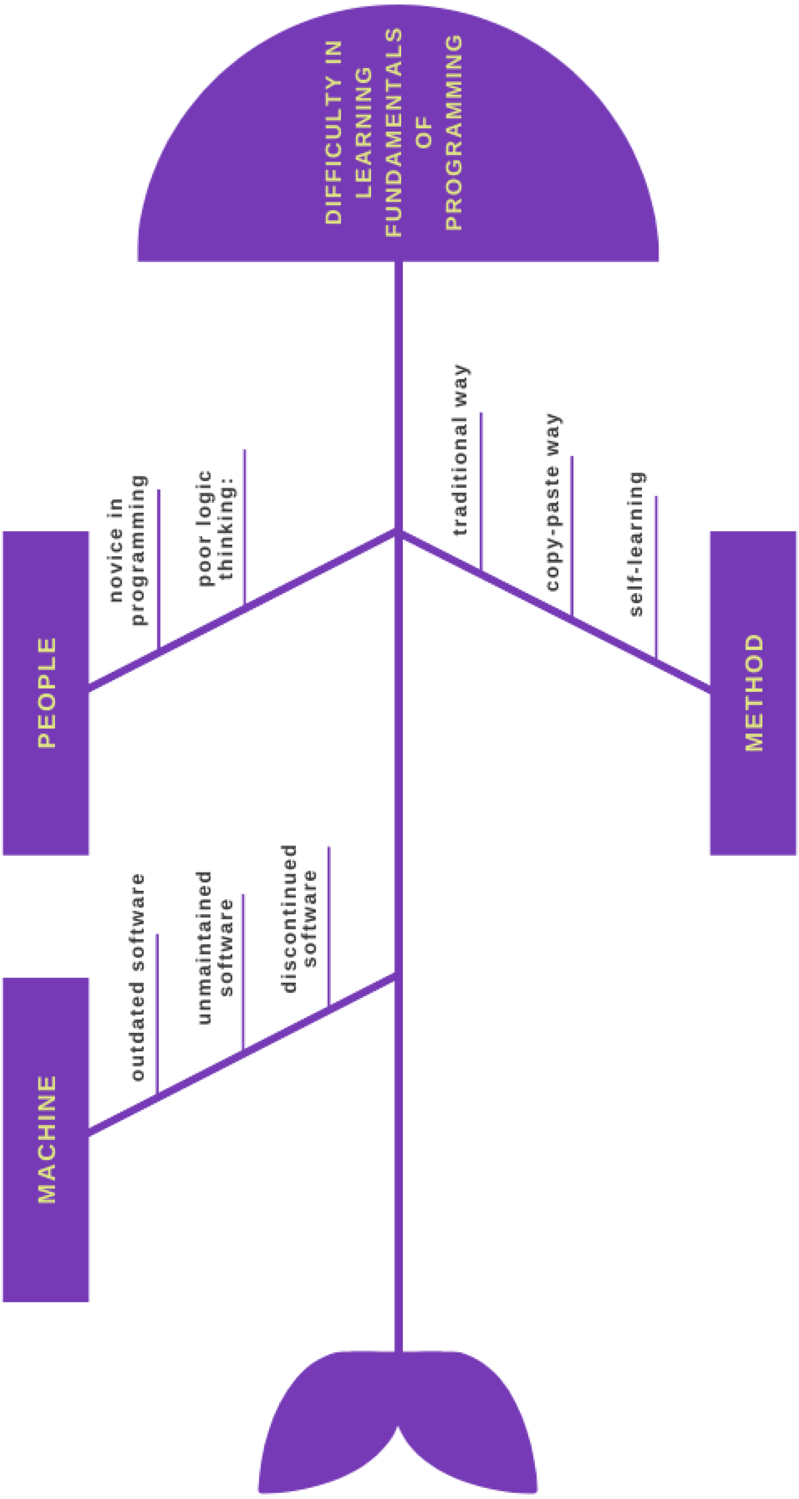


Figure 3: Difficulty in Learning Fundamentals of Programming

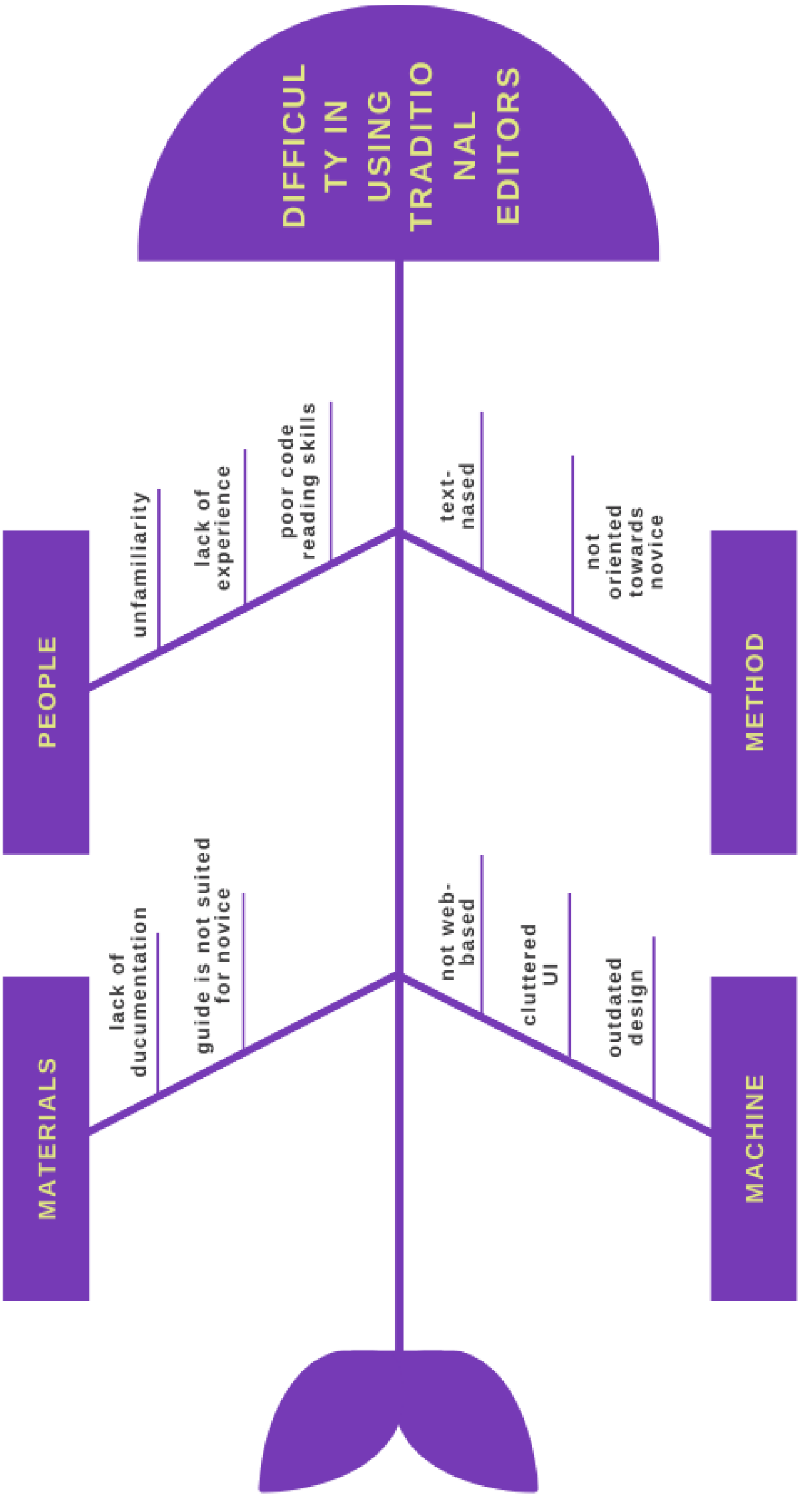


Figure 4: Difficulty in Using Traditional Editors

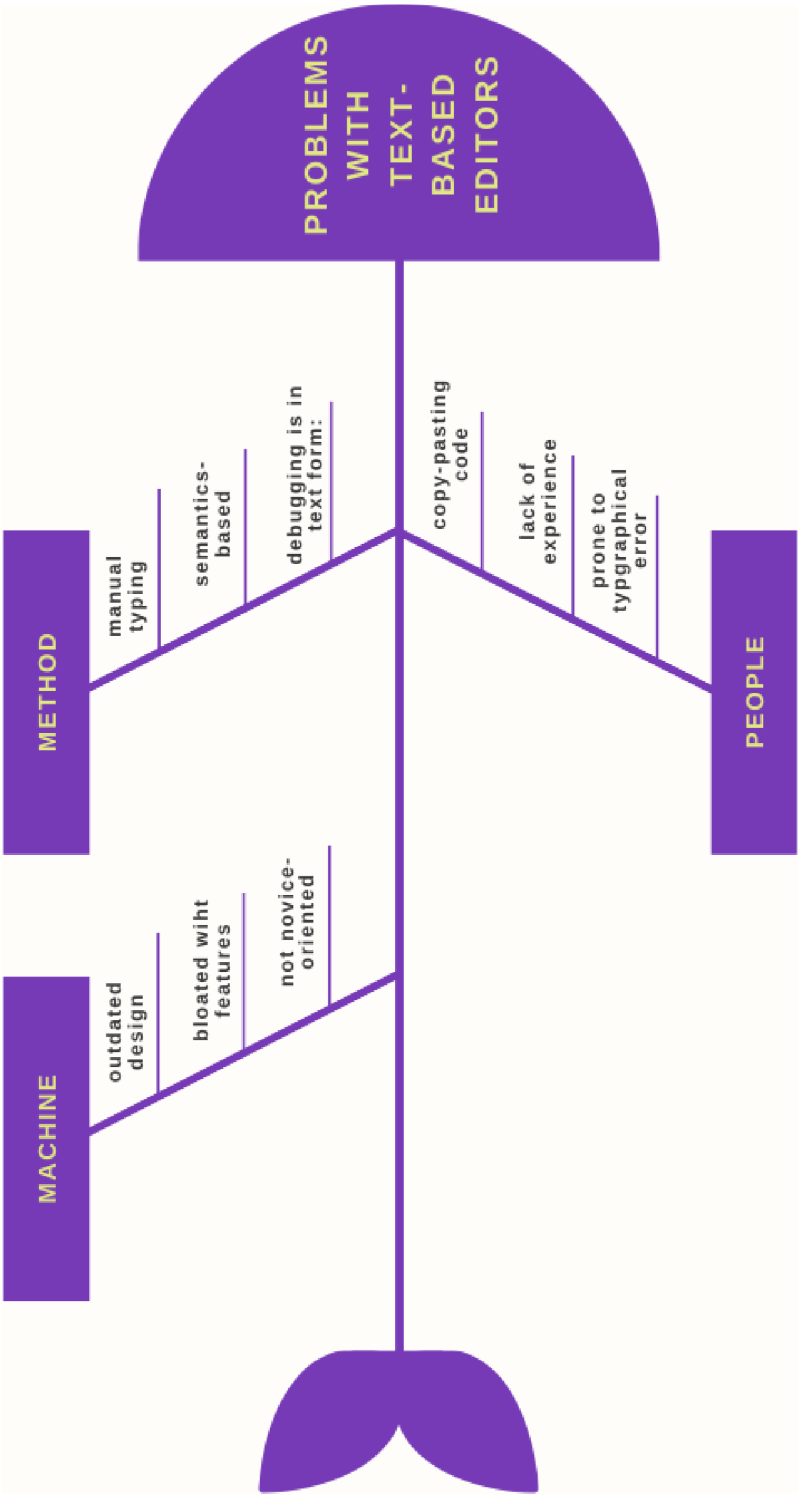


Figure 5: Problems with Text-based Editors

APPENDIX C

Context Diagram

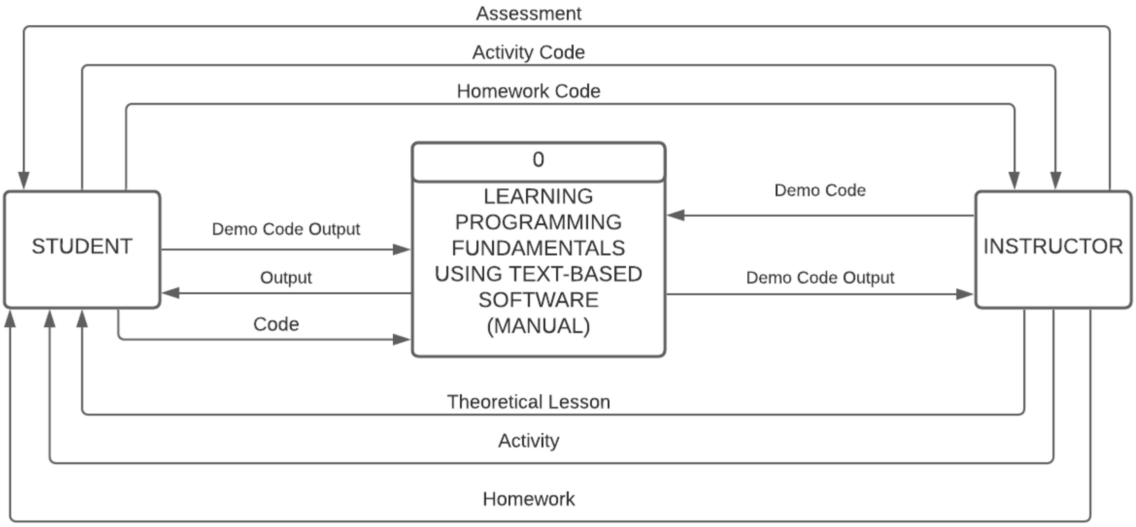


Figure 6: Context Diagram of Existing System

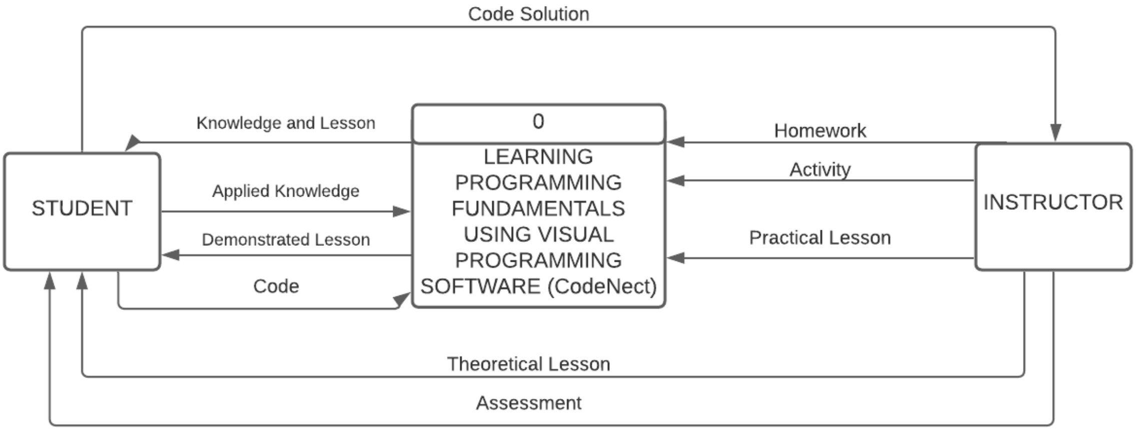
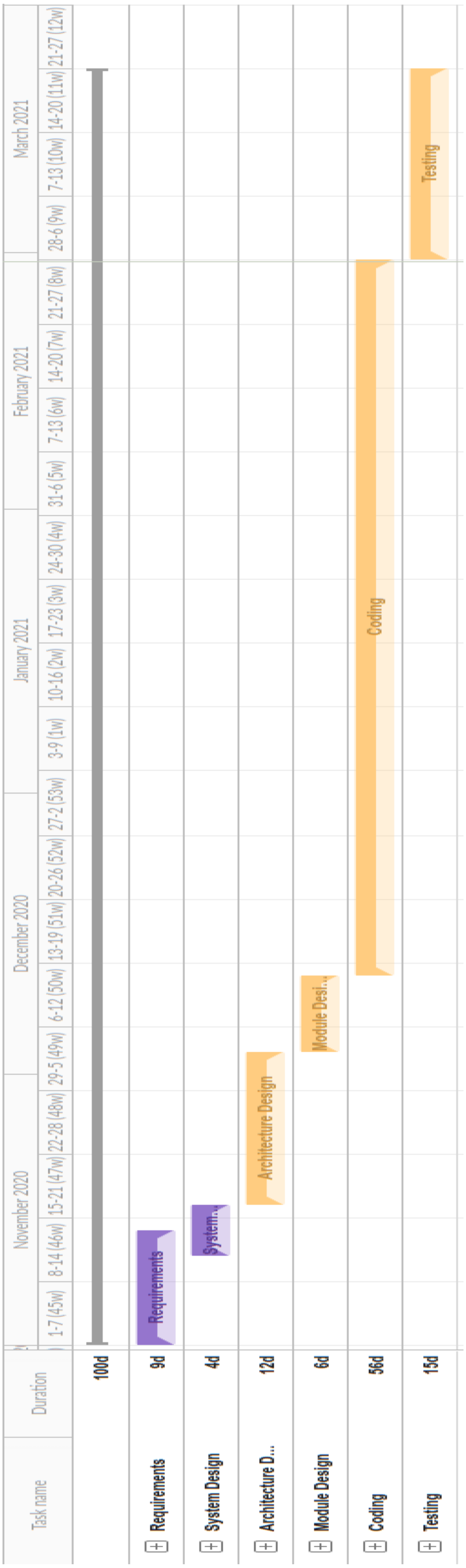


Figure 7: Context Diagram of Proposed System

APPENDIX D

Gantt Chart


Task name
Duration
November 2020
December 2020
January 2021
February 2021
March 2021
1-7 (45w)
8-14 (46w)
15-21 (47w)
22-28 (48w)
29-5 (49w)
6-12 (50w)
13-19 (51w)
20-26 (52w)
27-2 (53w)
3-9 (1w)
10-16 (2w)
17-23 (3w)
24-30 (4w)
31-6 (5w)
7-13 (6w)
14-20 (7w)
21-27 (8w)
28-6 (9w)
7-13 (10w)
14-20 (11w)
21-27 (12w)
100d
Requirements
9d
System Design
4d
Architecture D...
12d
Module Design
6d
Coding
56d
Testing
15d
Requirements
System...
Architecture Design
Module Desi...
Coding
Testing