FLOW2CODE - FROM HAND-DRAWN FLOWCHART TO CODE EXECUTION

A Thesis

by

JORGE IVAN HERRERA CAMARA

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee, Tracy Hammond
Committee Members, Thomas Ioerger
Zenon Medina-Cetina

Head of Department, Dilma Da Silva

May 2017

Major Subject: Computer Science

ABSTRACT

Flowcharts play an important role when learning to program by conveying algorithms graphically and making them easy to read and understand. When learning how to code with flowcharts and transitioning between the two, people often use computer based software to design and execute the algorithm conveyed by the flowchart. This requires the users to learn how to use the computer-based software first, which often leads to a steep learning curve.

We claim that the learning curve can be decremented by using off-line sketch recognition and computer vision algorithms on a mobile device. This can be done by drawing the flowchart on a piece of paper and using a mobile device with a camera to capture an image of the flowchart. Flow2Code is a code flowchart recognizer that allows the users to code simple scripts on a piece of paper by drawing flowcharts. This approach attempts to be more intuitive since the user does not need to learn how to use a system to design the flowchart. Only a pencil, a notebook with white pages, and a mobile device are needed to achieve the same result.

The main contribution of this thesis is to provide a more intuitive and easy-to-use tool for people to translate flowcharts into code and then execute the code.

DEDICATION

To the few people that had always believed in me: my parents, brothers and friends.

Thanks for all the unconditional support. Dios bo'otik! (God bless you)

ACKNOWLEDGMENTS

I would like to express my gratitude to all the people who helped over this past two years developing my thesis project. Special mention to my advisor and committee chair Dr. Tracy Hammond for providing advice and all the tools I needed to complete my degree. Her constant enthusiasm, mentoring and support helped me reach this goal. I would like to also thank my committee members Dr. Zenon Medina-Cetina and Dr. Thomas Ioerger for sharing their knowledge and giving advice. Finally, I would like to thank all the members of the Sketch Recognition Lab at Texas A&M University for giving comments and suggestions during the weekly meetings.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supervised by a thesis committee consisting of Professors Tracy Hammond, Thomas Ioerger of the Department of Computer Science and professor Zenon Medina-Cetina of the Department of Engineering. All work for the thesis (or) dissertation was completed by the student, under the advisement of Tracy Hammond of the Department of Computer Science.

Funding sources

Graduate study was supported by a fellowship from two Mexican institutions CANIETI and CONACYT.

NOMENCLATURE

TAMU	Texas A&M University
B/CS	Bryan and College Station
OGAPS	Office of Graduate and Professional Studies at Texas A&M University
IRB	Institutional Review Board
SVM	Support Vector Machine
HMM	Hidden Markov Model
UML	Unified Modeling Language
IDE	Integrated Development Environment
GUI	Graphical User Interface
SUS	System Usability Scale
QUIS	Questionnaire For User Interaction Satisfaction
CSUQ	Computer System Usability Questionnaire
AAS	Axis Aligned Score

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Most engineering programs today require students to be able to do programming at some level as several engineering courses require programming skills [1]. Learning how to code and develop algorithm design skills can be a difficult process. Problems encountered when dealing with introductory programming topics for the first time include: Too much focus on syntax, not enough emphasis on problem-solving, and lack of support for experiencing program execution [2]. New techniques for introductory programming courses have been developed to overcome those problems; one of the most popular is the use of flowchart diagrams to give structure and spatial positioning to code instructions, otherwise called flowchart-based programming. A major benefit of using the flowchart model is that it greatly reduces syntactic complexity, allowing students to focus on solving the problem instead of finding missing semicolons [3].

Novice programmers can find a few instructional approaches and tools which aim to assist them to cope with their various problems. Nonetheless, introductory programming continues to be a considerable obstacle for the unfamiliar novice trying to learn the subject at school, college or university [4].

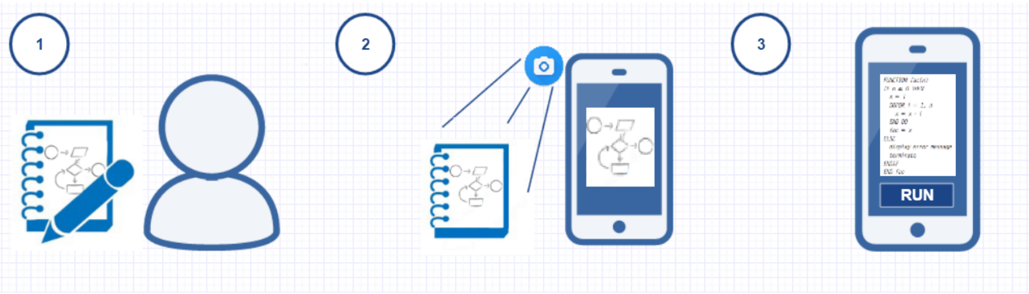
Researchers over the years have proposed and developed PC and tablet based systems that interpret visual languages [5], include mathematical expressions [6], chemical diagrams [7], digital circuits [8], mechanical systems [9] and UML class diagrams [10].

Many of these automated software tools have emerged to assist the users in particular tasks. However, most systems require that the users learn how to interact with it. Making the design process more restricted has resulted in the drifting away from the benefits of using flowchart diagrams to assist the learning process of algorithm fundamentals. Instead of designing the flowchart using UI controls on a software or a tablet computer, sketch recognition and computer vision algorithms can recognize hand-sketched diagrams to make the system more intuitive and consequently reduce the learning curve. The action of sketching can be considered a tool of thought that enables the mind to capture the things which are in flux and iteratively refine them [11]. Aside from allowing designers to communicate their ideas better, sketching also allows the learners improve their general academic achievement and problem-solving thinking [12].

Studies have shown that people prefer to use paper and pen during the early design stage[13]. This combined with the overall benefits of sketching serve as the major motivation to make an offline [14] flowchart recognizer that runs on a mobile platform and uses computer vision algorithms to extract the strokes from a picture and sketch recognition techniques to identify shapes.

Flow2code is an ongoing project in Sketch Recognition Lab. It consists of a mobile-based application which can be used as a tool for students that design algorithms using flowcharts. The application receives a picture either from the camera or the external memory of the device and using computer vision and sketch recognition extract and classify the content of it to translate it into pseudo-code. The overall high-level design of the application is shown in Figure 1.1. One important feature of the application is that it allows the pseudo-code execution of the recognized flowchart by internally translating the pseudo-code to javascript and executing the script to retrieve an output. This way the users can get some feedback about the flowchart and also keep a list of runnable algorithms available from the convenience of their pocket. This thesis describes a useful and handy mobile tool for anyone that is new to the world of coding.

The remaining chapters discuss the motivation, approach, methodology and contributions of this thesis more elaborately. Chapter 2 discusses the relevant work in the related fields as well as a high-level review of the works created in the area of flowchart based systems, image processing techniques, and sketch recognition. Chapter 3 outlines the goals of this work. Chapter 4 presents details about the system design, including the pre-processing of hand drawn diagrams, the algorithms used, and and sketch recognition techniques that helped to recognize shapes. Chapter 5 provides details about the evaluation of the system developed, the results, and a discussion of those results. Chapter 6 summarizes the conclusions reached for the overall project. Chapter 7 concludes with a discussion of future work.



```
FUNCTION fac(n)
IF n ≥ 0 THEN
  x = 1
  DOFOR i = 1, n
    x = x · i
  END DO
  fac = x
ELSE
  display error message
  terminate
ENDIF
END fac
```

Figure 1.1: The image presents the overall high level design of Flow2Code mobile application

2. LITERATURE REVIEW

Before explaining the details of Flow2code this section will review the most important research contributions that influenced the design of Flow2code, including previous flowchart based systems, sketch recognition techniques and algorithms, and other important topics that were taken into consideration while designing and developing this project.

2.1 Related research studies and frameworks

Past research attempts to recognize hand drawn flow charts have been used sketch recognition techniques that are either off-line or on-line. The offline approach is meant to be a passive technology, the sketching process and design is not assisted or intervened by any device. Further processing of scanned images or photos is required to actually perform the recognition. On the other hand, an on-line approach will require an electronic device to assist the sketching process. This approach has been most recently explored thanks to the increasing advances in finger or pen-based mobile devices like smartphones and tablets.

Both approaches come with advantages and disadvantages. The offline approach comes with the advantage of requiring no overhead in terms of learning how to interact with the sketch, as the user simply uses a pen and paper as they would normally. A disadvantages is the lack of temporal data in an offline sketch, as the computer only sees the finished sketch and not the process the user went through while drawing the sketch. This includes precise ordering, location, and possibly pressure of each of the points laid by the pen, which allows for a number of features that can be used for computation. When using an electronic device for drawing (online recognition) those features are computed and saved during the actual process of drawing. Other hindrances include the noise on the picture itself which can be mistaken for sketch data, producing difficulties for the recognition system and can lead to false positives on the during recognition and image processing. Additionally, offline systems can be time-consuming and computationally intensive.

An on-line approach are usually more accurate than an offline since the sketching process is assisted by an electronic device [15]. Users can get immediate feedback as to how the system is recognizing the image and adjust their drawing style to ensure accurate recognition. Several important features can be extracted from the drawing on online recognition systems such as: pressure, time-stamps, tilt, and others [16]. It also removes any confusion that may come from non-sketched artifacts on the page. There main disadvantage to online systems is that they require a pen-based device to provide for high-definition drawing, and the overhead on learning how to sketch or draw on this device requires more initial effort from the user.

More research has been done using on-line sketch recognition than off-line sketch recognition, mainly due to the difficulty of recreating the temporal data [17, 18, 19] that is easily available on online recognition systems.

Offline recognition systems do exist, however. Notowidigdo and Miller [20] described a project called User-Directed Sketch Interpretation (UDSI) that allowed to recognized paper based sketched diagrams and translating them into a graphical interface for editing as seen in Figure 2.1. The system combined low-level recognizers with domain-specific heuristic filters and greedy algorithm that eliminated incompatible interpretations. This project takes a bitmap image [21] as input and converts it into geometric primitives that are later presented to the user on a GUI on a PC software that allows the correction of false positives. The system recognizes: text regions (not actual text), rectangles, diamond circles and arrowheads. The authors followed a four-step approach involving: image smoothing, segmentation, text/shape recognition, and generation of alternatives.

It is important to notice that the overall purpose of UDSI is to provide an alternative tool for creating flowcharts other than Microsoft PowerPoint. While the study showed that overall the time it took to design a flowchart using UDSI was a little more than using

PowerPoint, most users appreciated a better alignment on their flowchart. This paper is important since it demonstrates a successful offline flowchart sketch recognition system, and even though the purpose of the project is different than the one this thesis will describe it contains a few similarities when it comes to methodology.

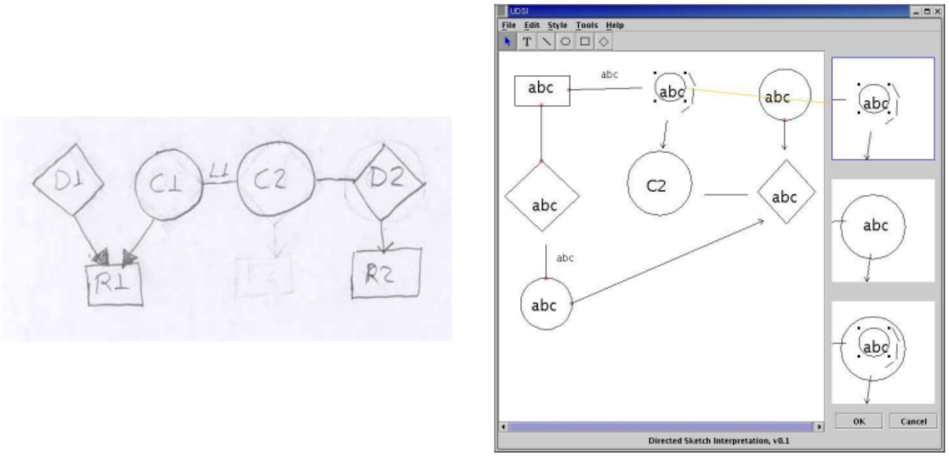

D1
C1
L1
C2
D2
R1
R2
UDSI
File Edit Style Tools Help
abc
C2
OK
Cancel
Directed Sketch Interpretation, v0.1

Figure 2.1: User Directed Sketch Interpretation input image and the software editing GUI

Sahoo and Singh [22] described a framework for both on-line and off-line sketch recognition using a global heuristic algorithm. The framework consisted of three different blocks: A set of domain classes, input refining block and recognition engine block. One noticeable aspect of this work was that it included a fuzzy logic [23] set membership as a possibility distribution to improve recognition. The main contribution of this project is the use of their own heuristic while using the A* search algorithm [24] to close shapes. Once the user inserts an input image, they apply a stroke and Gaussian filter and employ a segmentation algorithm. After this fist step, they employ an A* search algorithm to close all the strokes found using their own heuristic that gives the best result if a shape is found. One important aspect to mention is that this paper does not refer to a system itself but more to a framework that can be used as blueprint for a more robust recognition system.

Peterson and Stahovich [25] worked on grouping strokes into shapes in hand drawn diagrams describing a two-step algorithm that first classifies individual strokes according to the type of object they belong. The system then groups the strokes with similar classifications into clusters [26] representing individual objects. This approach is defined by two steps: (1) Classify single strokes into two or more different classes, and (2) Clustering strokes of the same class into individual objects.

Using stroke/shape features that describe both topological and curvature related properties like the degree which the stroke forms a closed path, self-intersections for topological properties and bounding box size, curvature, squared curvature for curvature the authors are able to do the single stroke classification. Using a threshold grouping classifier that uses decision Trees [27] like AdaBoost [28] is how the process of joining strokes is done by categorizing pairs of strokes as: Don't Join, Near Join, and Far Join. This approach proved to be accurate enough to be used, however it requires examples of each class to be able to train the single stroke classifier.

A recent research project that involves off-line sketch recognition was done by Wu, Wang, Zhang and Rui [29]. They presented a three-stage cascade framework for off-line sketch parsing. The authors used an existing selection recognition approach [30] with smaller candidate groups than usual thanks to a novel concept. The major contribution was the introduction to the idea of this novel concept "sharpness estimation" to reduce the number of candidate groups that are selected for the selection recognition. The idea behind their algorithm is not to jump immediately to the group recognition or classification, but to include a previous step that detects a small number of stroke groups that represent good shapes in a fast way using sharpness estimation. One limitation is that the shapes need to be well defined to use the sharpness estimation approach, so if a quick drawing is made, and the shapes are not well closed or defined the algorithm will give false positives.

Another method for recognizing high level shapes uses geometrical sketch recognition techniques to create a natural sketch recognition environment for UML diagrams [10] (and generalized in [31, 32, 33, 34, 35, 36, 37, 38, 39]) as seen in Figure 2.2. Hammond's system is based on a multi-layer recognition framework that recognized multi-stroke objects by their geometrical properties allowing users to draw freely as they would on paper. This work is really meaningful when it comes to how to use geometrical properties to recognize shapes rather than requiring the user draw objects in a pre-defined manner. The system described by Hammond falls into the category of on-line sketch recognition systems since the sketching is assisted by a PC and a mouse while allowing editing at any time. The stages of the multiple-layer recognition system described are: 1) Pre-processing, 2) Selection, 3)Recognition, 4) Identification. The non-modal dual approach is intuitive since users can see what has been recognized from their sketching and make adjustments in real time. The system not only recognizes sketched UML diagrams but also converts it to Rational Rose and also automatically generates java code based on the drawn input.

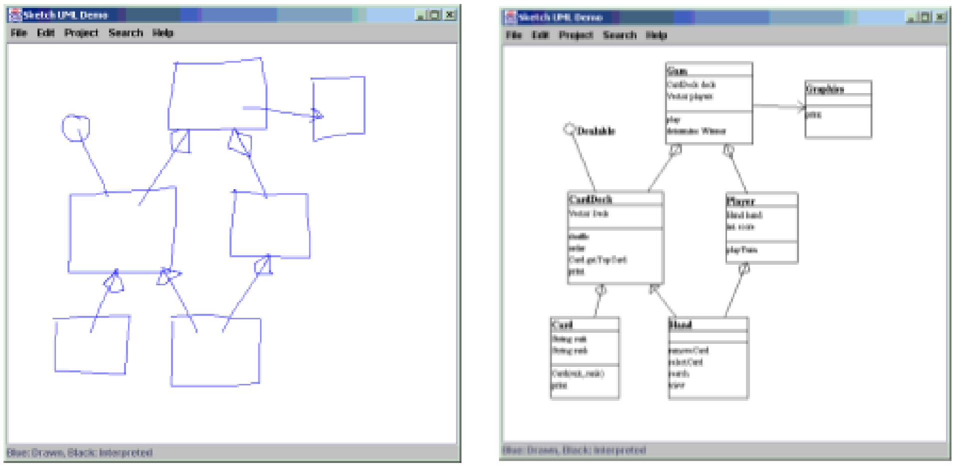


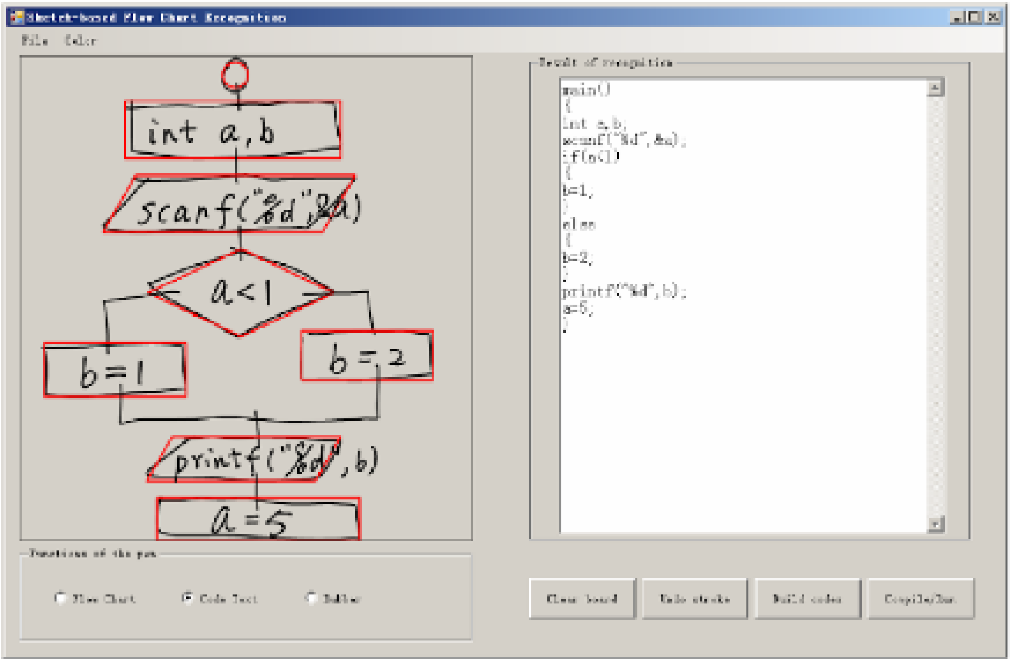
Figure 2.2: This image shows an example of TAHUTI and how it recognizes a drawn class diagram

One project that presents strong similarities with the work presented in this thesis was done by Zheming Yuan [40]. The authors defined a pen-based flowchart recognition system for programming teaching see figure 2.3. Zheming uses an on-line approach using a pen and a tablet for input. Using on-line sketch features like curvature, speed, angle, and others he is able to do feature extraction [41, 42]. After that he uses a hybrid SVM-HMM algorithm to aid the sketch recognition by learning temporal patterns [43]. This project is the most similar to the project described in this thesis with the major difference that their system uses an on-line approach combined with HMM. The use of an HMM for shape recognition purposes requires training to be able to produce acceptable results, but the major advantage of using it is that it can allow partial recognition as the user draws the flowchart. Another key difference is that the system described translates the flowchart into C syntax code but does not allow for the execution of the user's code to see the output. This prevents the user from getting some feedback about the accuracy of their flowchart diagram to be able to fix their mistakes if there was any.

2.2 Standalone flowchart-based programming systems

Since the flowchart-based programming approach proved to be an effective technique for programming introduction a few available software solutions have emerged. Most of these solutions are Windows-based systems that are designed using a point-and-click [44] building block approach and do not include any sketching capabilities (neither on-line or off-line). These systems rely on GUI controls to provide the users a way to create flowchart diagrams and then translate them into code as you can see in Figure 2.4. Many of these systems do not include any code execution capabilities so there is no way the user can verify if the user's flowchart will produce the desired result.

Hooshyar [4] created a survey of standalone (non-sketching) MS windows systems while evaluating their strengths and weaknesses. Here are some of those options available:



```
main()
{
int a,b;
scanf("%d",&a);
if(a<1)
{
b=1;
}
else
{
b=2;
}
printf("%d",b);
a=5;
}
```

Figure 2.3: This shows the pen-based programming teaching system by Zheming Yuan, the left panel is the drawing canvas and the right one the C code translation

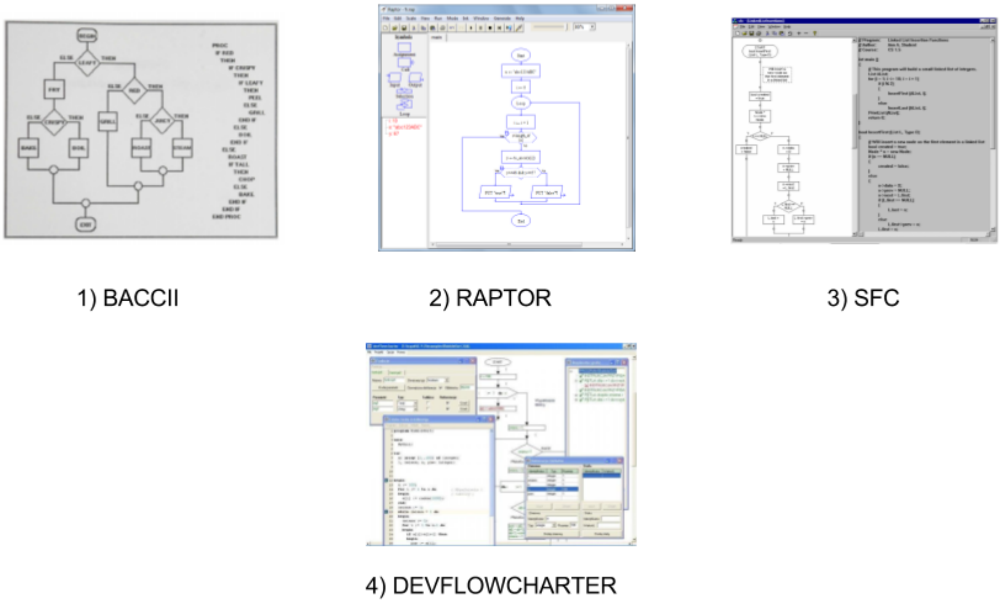
BACCII is one of the oldest systems that provide flowchart-to-code capabilities to novice programmers in order to help them to express their programs. The system mainly focuses on algorithm development rather than syntactical correctness [45]. Since the system is one of the oldest approaches, it is designed to construct the flowchart using modal windows per structure which might not be the best option available nowadays. The system translates a constructed flowchart into either Pascal, C++, or FORTRAN. The system uses a non-standard flowchart notation making it not transferable to a program design methodology and therefore requiring extra learning. The system does not provide any way to execute the generated code so the users would not be able to validate their work and put in practice their tracing and testing skills.

Raptor is an (non-sketching) MS Windows based tool originally expanded for the United State Air force Academy as instruction and computing course. This system focuses mainly on improving the problem-solving skill of novices as well as help to avoid syntax errors [46]. The system translates to Ada, C++, C#, and Java. While a robust solution, the system does not allow code viewing and execution. An important feature of Raptor is that it includes the widely popular object-oriented paradigm. Raptor is currently used in 17 countries around the world, according to its developers make it one of the most popular flowchart-based programming systems. The system was also used by the U.S. Military Academy on an IT course focused mainly on algorithm design [47].

SFC editor or Structured Flowchart Editor is an MS Windows based system developed at the Sonoma State University in California [48]. The overall design of the system stands out than the rest of the other options when allowing side by side comparison between flowchart and code. The system translates the flowchart to C++ and can be exported to a text file for further execution on an IDE. The system takes advantage from the dual view approach that allows users create the flowchart on one side while in the other the pseudocode is being processed simultaneously. The text cannot be modified directly from the code view. It does not provide execution capabilities or any other feedback.

DevFlowCharter is an open-source (non-sketchint) MS Windows based system. The system translates flowcharts to Pascal code [49] and supports variable assignments, decisions, loops but does not handle arrays. An external environment is needed to execute the code. This project is rather small and non-academic compared to the previously defined option.

Many of the systems mentioned provide a reliable way to learn how to program and show that value of flowcharts in design, however they are unfortunately outdated since most of the popular programming languages today are different, and those systems are not longer supported by its developers. In recent years there have been many different flowchart-based programming systems designed for novice programmers that are more up to date and overall in line with current programming languages and paradigms as seen in



1) BACCII

2) RAPTOR

3) SFC

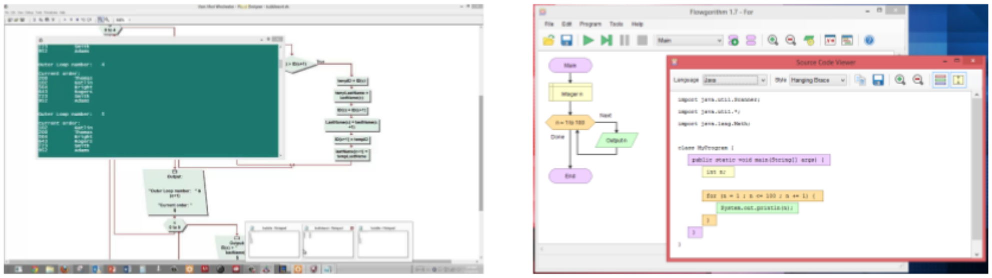
4) DEVFLOWCHARTER

Figure 2.4: This image shows the GUI of four of the most popular and older systems that used the flowchart-to-code approach

Figure 2.5.

Visual Logic [50] is a paid-only (non-sketching) graphical authoring tool that allows students to write and execute programs using flowcharts. The system is only supported for Windows PCs. The system does not provide a way to see the generated code from the flowchart and only provide a visualization of the result on a CMD terminal. The system is currently supported by its developers and provide a good approach for flowchart design beginners without dealing with source code.

Flowgorithm is an open-source (non-sketching) MS Windows software that provides the tools for flowchart design and edition and the subsequent translation of the structure to many languages such as C#, C++, Delphi/Pascal, Java, JavaScript, Lua, Perl, Python, QBasic, Ruby, and others [51]. The system is continually being maintained by its developers. A limitation is that it only allows one input and an output flow and as all the other it is only available on Windows.



```
import java.util.Scanner;
import java.util.*;
import java.lang.Math;

class MyProgram {
    public static void main(String[] args) {
        int n;

        for (n = 1 ; n <= 100 ; n += 1) {
            System.out.println(n);
        }
    }
}
```

1) VISUAL LOGIC

2) FLOWGORITHM

Figure 2.5: This image shows the GUI of two of the most recent and updated systems that used the flowchart-to-code approach

3. GOALS

Combining a hand-drawn flowchart recognition with the portability of mobile devices could give the beginning programmer a powerful tool to verify the correctness of their algorithm design. The idea is to provide an intuitive application that requires little to no initial training. This allows the users to focus on the algorithm itself and not worrying about details like syntax. Using image processing together with sketch recognition on a mobile device would create a useful solution that allows the user to "code" on paper. The goal of the project is to measure both the correctness of the tool itself together with the usability metrics and overall feedback gathered from users. This can be subdivided into following research questions which the evaluation will try to answer:

1. What is the recognition accuracy obtained using this approach?
2. How usable is the application according to the user's feedback?
3. Will the users see any benefit in using this application?
4. What were the major limitations and experience breakdowns while using the application?

4. METHODOLOGY

This chapter discusses the methodology steps that Flow2code uses to go from a picture of a flowchart to a pseudo-code and its execution. This section is divided into three parts: 1) the overall process flow to explain the design choices of the system, 2) the system architecture to explain how the system was technically build and how the software and its architecture was designed, 3) the interface design to provide an example story case of how the system is used as well as the UI design choices that were taken.

4.1 Overall process flow

There are three processing steps in: pre-processing, recognition and post-processing, as seen in Figure 4.1. The pre-processing step will take an image as an input and will employ computer vision/image processing algorithms to remove as much noise as possible from the picture to improve further recognition. The second step is recognition, this step uses computer vision algorithms to identify strokes from the previously cleaned picture and store the recognized strokes. After that this step takes as an input a set of identified strokes and using sketch recognition metrics and techniques will group them into shapes using a top-down followed by a bottom-up approach. The third and last step is post-processing. This step has two different objectives, first to allow the user to manually edit the recognized shapes and text, and second generate pseudo-code as a graph by using the nearest neighbor approach.

4.1.1 Pre-processing

Since the processing and recognition of the flowchart are set to be done on the mobile device, the first step is to re-size the picture to a standard width and height so it can handle the processing more easily. In order to reduce the processing time of the pre-

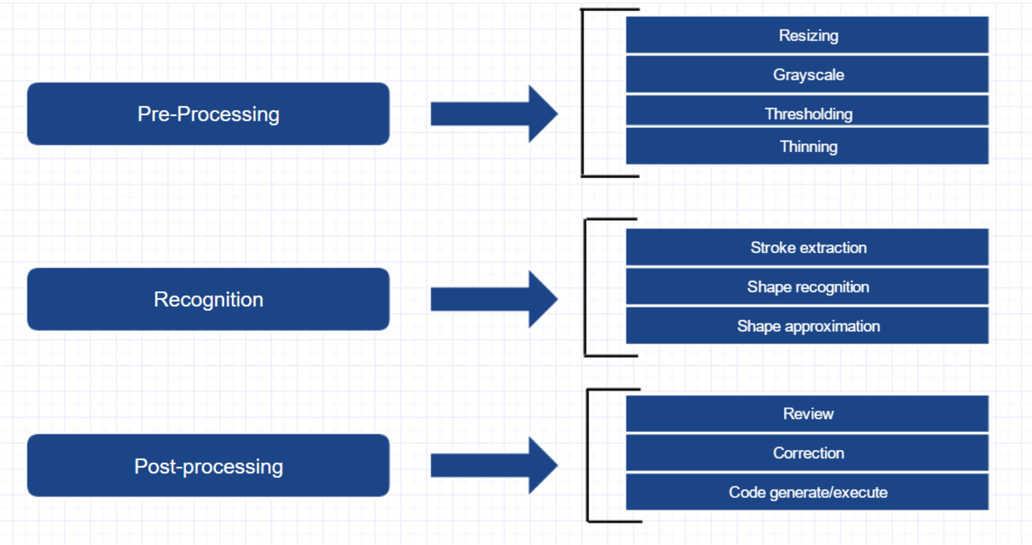


Figure 4.1: The image shows the process flow of the data. From the picture pre-processing to the code generation and execution

processing steps, the design of the system includes a bridge to be able to execute C++ code on the mobile Application (Android) that also uses the OpenCV library [52]. The OpenCV library has been used in many computer vision and image processing related projects since its release[53, 54, 55] and it's proven to be quite effective. The resizing step is primarily to decrease the overall processing time that it takes to extract the strokes from the picture.

The image re-sizing should be upscale or downscale according to the current image dimensions and should be somewhere near 600 pixels for height and 400 pixels for weight. Once the image is re-sized the system converts it to grayscale [56]. Since the system takes in a photo of the hand-drawn flowchart, this can lead to very noisy images because of the uneven brightness. One way to avoid this problem would be to get a reading of the image using a scanner since it would remove all the noise caused by shadows and overall brightness issues. However, that would deviate from the overall purpose of the system, to be a quick and easy to use tool for beginner programmers since it would require more effort by scanning the image. So, the system allows to either take a picture or use a picture from the smartphone internal memory. Now, to alleviate the brightness/shadow problem the system first applies first a binary threshold to convert the input image into black or white [57]. Then, the system applies an Adaptive Gaussian Threshold to remove most of the noise that is always present because of the picture environment [58]. The adaptive Gaussian threshold is used to deal with photos taken with a camera when the overall properties are not known [59].

In other words, since the brightness/light source of the image is most likely to be uneven depending on which part of the photo is in focus (and if you used flash or not); there is likely to be shadows on the picture. The adaptive Gaussian threshold takes those items into consideration and removes most of the background noise generated by those problems.

The system should have by now a black and white binary picture with most of the external and pepper-and-salt [60] noise reduced. However, in order for the stroke extraction to work, the system needs strokes of only one-pixel width. Thus, the next step is to use a thinning algorithm, which is a morphological operation [61] that is used to remove selected foreground pixels from binary images and it is mainly used for the purpose of skeletonization. Thinning is commonly used to tidy up the output for edge detectors by reducing all lines to a single pixel thickness as seen in Figure 4.2. It is normally applied to binary images and produces another binary image as output [58].

There are a few thinning algorithms available such as Tamura [62], Guo-Hall [63], Hilditch [64] Zhang-Suen [65], Jang&Chin [66], Arcelli-Baja [67], O'Gorman [68] just to mention a few. Unfortunately, there is no best all purpose thinning solution available and all of the current thinning algorithms have drawbacks such as low performance, loose topology output or not being able to keep the topology of certain curves or shapes. Having

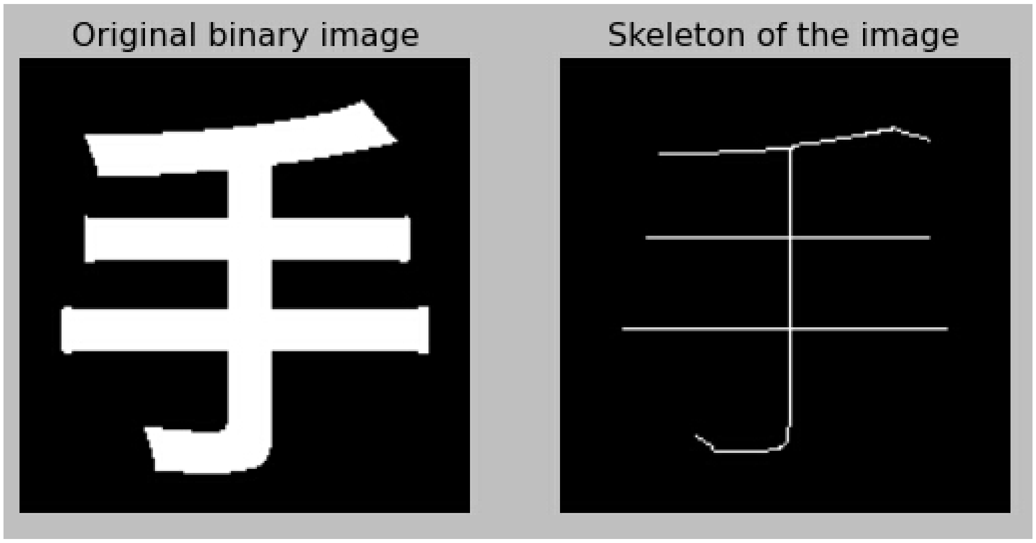


Figure 4.2: The picture shows an example of the a bitmap image before and after applying a thinning algorithm.

tried the options previously mentioned, the final system uses the Zhang-Suen algorithm due to its fast processing time and overall reliable skeletonization results [69].

The system uses a modified version of the Zhang-Suen Algorithm because it does not generate disturbances on the topology of the image as the other morphological thinning approaches do [65]. The original version of the algorithm generates a good approximation of the skeleton of the shape keeping the overall digital topology [70] of the entire image.

The Zhang-Suen algorithm is a two-step algorithm that operates on all black pixels P1 that can have eight neighbors [P9,P2,P3 | P8, **P1**, P4 | P7, P6, P5]. The boundary pixels of the image cannot have the full eight neighbors. To have a better idea of what the Zhang-Suen algorithm what follows is a simplified version for explanation of the two steps and iteration process needed to be done.

Declare

- Define A(P1) = The number of transitions from white to black, (0 -> 1) in the sequence P2, P3, P4, P5, P6, P7, P8, P9, P2. (Note the extra P2 at the end because it is circular).

- Define B(P1) = The number of black pixel neighbors of P1. (=sum(P2...p9)).

Step 1 All pixels are tested and pixels satisfying all the following conditions are just noted at this stage.

1. The pixel is black and has eight neighbors

2. 2 < = B(P1) < = 6

3. A(P1) = 1

4. At least one of P2 and P4 and P6 is white

5. At least one of P4 and P6 and P8 is white

After iterating over the image and collecting all the pixels satisfying all the step 1 condition satisfying pixels are set to white.

Step 2 All pixels are again tested and pixels satisfying all the following conditions are just noted at this stage.

1. The pixel is black as has eight neighbors

2. 2 < = B(P1) < =6

3. A(P1) = 1

4. At least one of P2 and P4 and P8 is white

5. At least one of P2 and P6 and P8 is white

After iterating over the image and collecting all the pixels satisfying all the step 2 conditions; all these conditions satisfying pixels are again set to white.

If any pixels were set in this round of either step 1 or step 2 then all the steps are repeated until no image pixels are so changed.

The Zhang-Suen Algorithm has a disadvantage since it does not ensure that the overall output would be one-pixel width particularly noticeable in curves. The solution to this is create a small modification to the algorithm to overcome that problem by applying a custom filter that discards the pixel that is redundant and breaks the one-pixel rule. An easy way to implement the filter is to use Rajan and Hammond's method to look for every pixel window neighborhood that is considered a *point of ambiguity* [71, 72] if one of the neighbor pixels leads to nothing. If that is the case that pixel is removed to ensure that the system preserves the overall image topology.

Figure 4.3 shows how the pre-processing step is acting on the original image until the system reaches the final thinned output.

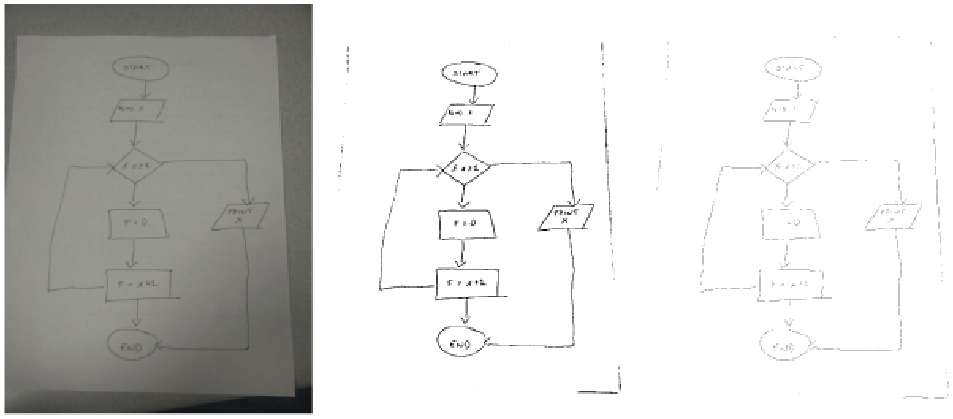


Figure 4.3: The first picture shows the original image, the second the image after the grayscale and thresholding steps. Finally the the last one after the thinning algorithm.

4.1.1.1 Stroke extraction

Before shape recognition, the system first performs stroke extraction.

After the thinning algorithm is applied the system can start with one of the most important parts of the framework, the stroke extraction. The algorithm takes as input a matrix of pixels that represents the entire image. The matrix is filled with 0 (if pixel is off/white) and 1 (if pixel is on/black). The matrix needs to be iterated through in an organized way. A nested loop would be ideal to iterate from Top-Left to Bottom-Right of the entire matrix and start looking for black pixels in every iteration. If a black pixel (1) is found then the system needs to call a recursive [73] algorithm called *extract stroke*. The *extract stroke* algorithm will add the point to a new list of points and then the pixel will be changed to 0. The system then calculates the neighboring 8-pixel window of the current pixel being iterating on. There are three different steps that can be followed now according to the neighbors of the current pixel.

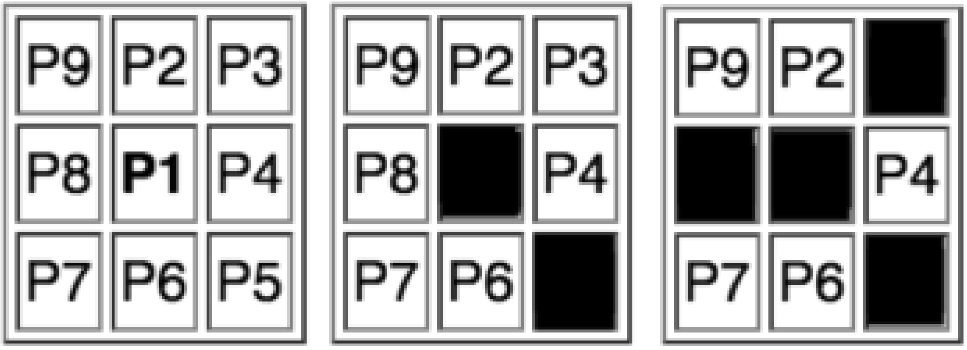


Figure 4.4: The first picture shows the neighboring window. The current point is P1. The second image shows the case where you only have one option to go to (from P1 to P5). Finally, the last image shows the option where you have more than one option to go to (P1 is considered a *point of ambiguity*

If the current pixel only has one neighbor left, then the system again calls the *extract stroke* algorithm.

If the current pixel has more than one pixel as neighbor then the point is considered a *point of ambiguity* (see Figure 4.4). If there is a *point of ambiguity* and the current list of points size equals to one, that pixel is added to a different list called ”ignored point”. Then the pixel is again to the matrix of points, and the current pixel of points is discarded. If the size of the list of points is greater than one then there exists more than one different path to follow. To determine which path should be followed, an algorithm ’get linear option’ calculates the angles between the current pixel and its non-zero neighbors and returns the one that preserves the least amount of change. The system then recursively calls stroke extraction with the more linear point.

The main reason for ignoring the points is to try to find the start or end point of a stroke and use that as a starting point, rather than a random point in the middle of a curve that may show up early during the iteration of the entire pixel matrix. For example, Figure 4.5 shows a curve in which the first pixel that the algorithm will verify would be the one marked in yellow. Rather than start the processing with that point, the system instead identifies that the point is a *point of ambiguity*, and also is able to identify that is a part of a curve and the middle of a stroke. By ignoring that point in this instance, the algorithm starts adding the points only when finding the start or end of a stroke, which is helpful in the next phases. The algorithm deals with the ignored points after it is finished iterating the matrix.

If the current pixel does not have more pixels to go to (dead end) then the algorithm saves the list of points and considers that list as a stroke. The algorithm then exits the recursive function and returns back to the matrix iteration. At this point the algorithm declares that it has found a stroke that consists of a list of points.

After the system finishes iterating the entire pixel matrix the system iterates through the ignored point list. This is because there is a closed shape then all of the strokes would

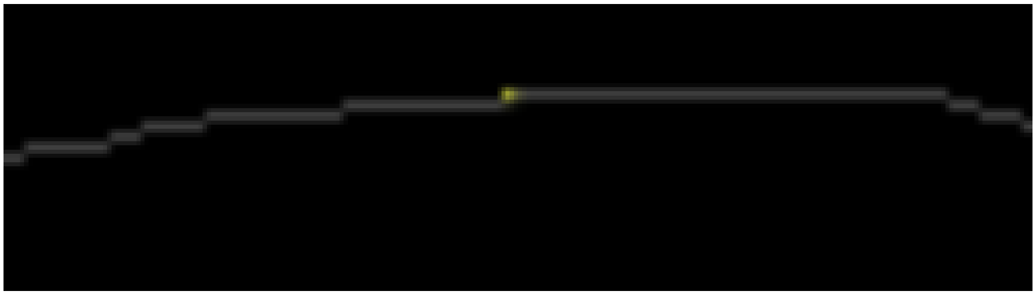


Figure 4.5: The image shows how when iterating from top-left to bottom-right a pixel that is not start or end of stroke is selected first.

have been considered as ignored points and since there is no start or end point the stroke would never have been added. By iterating through this list of points using a similar stroke extraction method this closed shape as a stroke is added, only this time the algorithm would not ignore any point if the list of points equals to one and there is a *point of ambiguity*.

4.1.2 Stroke processing: features, corners, and metrics

After stroke approximation is complete, the system calculates a number of features and metrics on those strokes and also performs corner finding to break the strokes down into polylines.

4.1.2.1 Features and metrics

The bottom-up approach uses traditional shape recognition techniques to classify the current strokes into two categories: *Recognized shapes* and *Unrecognized Strokes*. Metrics that describe important features from strokes are used to create rules for identifying shapes from those strokes. Rubine defined a set of thirteen incrementally computable in constant time per input point features[74, 75]. These features can be used to create a classifier that is able to categorize strokes into shapes.

A function called *generate metrics* receives a stroke as input and generates a set of metrics that will be used further to categorize the given stroke[74, 76]. The metrics that are being calculated are the following:

- *f1. cosine of the initial angle of the stroke*:
- *f2. sine of the initial angle of the stroke*:
- *f3. Length of the bounding box diagonal*:
- *f4. Angle of the bounding box diagonal*:
- *f5. Distance between the first and last point*:
- *f6. Cosine of the angle between the first and last point*:
- *f7. Sine of the angle between the first and last point*:
- *f8. Total stroke length*:
- *f9. Total angle traversed*:
- *f10. Sum of the absolute value of the angle at each point*:
- *f11. Sum of the squared value of the angle at each mouse point*:
- *f12. Maximum speed of the gesture*:
- *f13. Duration of the gesture*:

These features are calculated given only a set of points as shown in Figure 4.6

Feature numbers 12 and 13 are only applicable when dealing with on-line sketch recognition since speed and duration time is not determinable using a picture, leaving 11 available features. However when dealing with flowchart shapes, the differences between the basic used shapes (rectangle, parallelogram, diamond, circle) are actually quite noticeable allowing the use of fewer features. The final system implementation only uses 5 of the 11

$$f_1 = \cos\alpha = (x_2 - x_0)/\sqrt{(x_2 - x_0)^2 + (y_2 - y_0)^2}$$

$$f_2 = \sin\alpha = (y_2 - y_0)/\sqrt{(x_2 - x_0)^2 + (y_2 - y_0)^2}$$

$$f_3 = \sqrt{(x_{max} - x_{min})^2 + (y_{max} - y_{min})^2}$$

$$f_4 = \arctan\frac{y_{max} - y_{min}}{x_{max} - x_{min}}$$

$$f_5 = \sqrt{(x_{P-1} - x_0)^2 + (y_{P-1} - y_0)^2}$$

$$f_6 = \cos\beta = (x_{P-1} - x_0)/f_5$$

$$f_7 = \sin\beta = (y_{P-1} - y_0)/f_5$$

Let $\Delta x_p = x_{p+1} - x_p \qquad \Delta y_p = y_{p+1} - y_p$

$$f_8 = \sum_{p=0}^{P-2} \sqrt{\Delta x_p^2 + \Delta y_p^2}$$

Let $\theta_p = \arctan\dfrac{\Delta x_p \Delta y_{p-1} - \Delta x_{p-1} \Delta y_p}{\Delta x_p \Delta x_{p-1} + \Delta y_p \Delta y_{p-1}}$

$$f_9 = \sum_{p=1}^{P-2} \theta_p$$

$$f_{10} = \sum_{p=1}^{P-2} |\theta_p|$$

$$f_{11} = \sum_{p=1}^{P-2} \theta_p^2$$

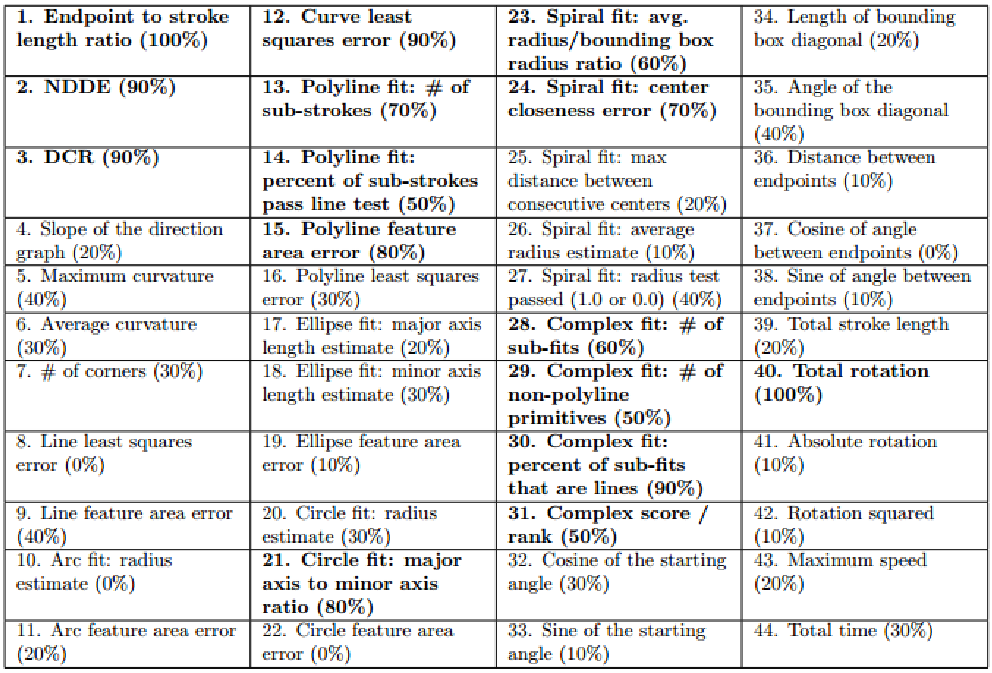
Let $\Delta t_p = t_{p+1} - t_p$

$$f_{12} = \max_{p=0}^{P-2} \frac{\Delta x_p^2 + \Delta y_p^2}{\Delta t_p^2}$$

$$f_{13} = t_{P-1} - t_0$$

Figure 4.6: List of the formulas to calculate each Rubine feature

usable Rubine features: Start point, Endpoint, Stroke length, Distance between start and end point, Total absolute rotation. In addition to Rubine features, the system incorporates other features from Paulson and Hammond [75, 38, 36]. These features are geometric based and are critical to differentiating between shapes since most of them differ greatly geometrically speaking. Figure 4.7 shows the list of the geometric features.



1. Endpoint to stroke length ratio (100%)	**12. Curve least squares error (90%)**	**23. Spiral fit: avg. radius/bounding box radius ratio (60%)**	34. Length of bounding box diagonal (20%)
2. NDDE (90%)	**13. Polyline fit: # of sub-strokes (70%)**	**24. Spiral fit: center closeness error (70%)**	35. Angle of the bounding box diagonal (40%)
3. DCR (90%)	**14. Polyline fit: percent of sub-strokes pass line test (50%)**	25. Spiral fit: max distance between consecutive centers (20%)	36. Distance between endpoints (10%)
4. Slope of the direction graph (20%)	**15. Polyline feature area error (80%)**	26. Spiral fit: average radius estimate (10%)	37. Cosine of angle between endpoints (0%)
5. Maximum curvature (40%)	16. Polyline least squares error (30%)	27. Spiral fit: radius test passed (1.0 or 0.0) (40%)	38. Sine of angle between endpoints (10%)
6. Average curvature (30%)	17. Ellipse fit: major axis length estimate (20%)	**28. Complex fit: # of sub-fits (60%)**	39. Total stroke length (20%)
7. # of corners (30%)	18. Ellipse fit: minor axis length estimate (30%)	**29. Complex fit: # of non-polyline primitives (50%)**	**40. Total rotation (100%)**
8. Line least squares error (0%)	19. Ellipse feature area error (10%)	**30. Complex fit: percent of sub-fits that are lines (90%)**	41. Absolute rotation (10%)
9. Line feature area error (40%)	20. Circle fit: radius estimate (30%)	**31. Complex score / rank (50%)**	42. Rotation squared (10%)
10. Arc fit: radius estimate (0%)	**21. Circle fit: major axis to minor axis ratio (80%)**	32. Cosine of the starting angle (30%)	43. Maximum speed (20%)
11. Arc feature area error (20%)	22. Circle feature area error (0%)	33. Sine of the starting angle (10%)	44. Total time (30%)

Figure 4.7: A list of the 44 geometric based features from Paulson and Hammond *ShortStraw*

Out of the 44 metrics, the systems final algorithm uses the following features: number of corners, total stroke length, total rotation, absolute rotation, and number of sub-strokes.

4.1.2.2 Novel feature: Axis aligned score

As part of this thesis, a new geometrical feature was defined called *Axis Aligned Score (AAS)* that was particularly helpful in differentiating between diamonds and other quadrilaterals. AAS first obtains the corners of the stroke. If the corners are not consecutive then they need to be reordered to be in consecutive order, either clockwise or anticlockwise (as in [77]. Once the corners are ordered, the feature's algorithm calculates both diagonals between opposite corners. Then the midpoint between the diagonals is calculated, as well as the midpoint of those two diagonal midpoints. This provides an approximated midpoint of the entire quadrilateral. The feature's algorithm then instructs to calculate the maximum and minimum values of X and Y of the whole shape and create four new corners adding or subtracting the maximum and minimum X, Y values from the approximated shape midpoint. Finally, the AAS algorithm instructs to calculate the difference between the closest shape corner and the axis new generated corner and add the difference value to a variable. The idea behind this metric is that ideally, the diamond will have its corners aligned perfectly with the X and Y axis so the AAS would be equals or really close to 0 as shown in Figure 4.8. The algorithm sets a threshold of 80 for diamonds to give some margin of error to the user. The other quadrilaterals scored above 100 or even more; AAS ended up being a quite useful feature.

4.1.2.3 Corner finding

Short-Straw [78] was used to identify the number of corners on a given stroke to be used later in the recognition process. *Short-Straw* is an accurate poly-line corner finder that is simple to understand and implement (compared to more complex corner finding measures [79, 80, 81]), while still achieving a high all-or-nothing accuracy measure [78]. The overall idea of *Short-Straw* is simple: given a stroke, imaging that a small short straw that travels along the entire stroke point by point or in this case pixel by pixel as in Figure 4.9.

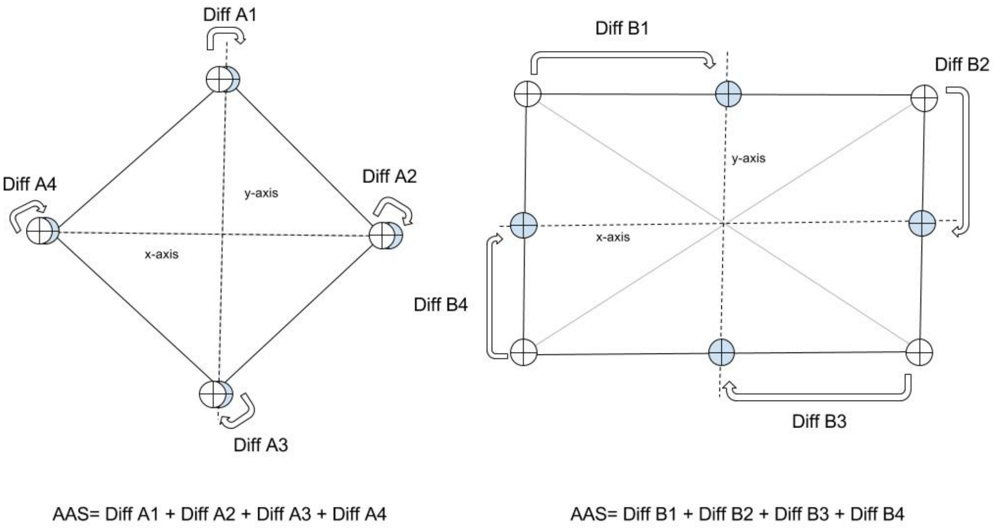

Diff A1
Diff A4
y-axis
Diff A2
x-axis
Diff A3
AAS= Diff A1 + Diff A2 + Diff A3 + Diff A4
Diff B1
Diff B2
y-axis
x-axis
Diff B4
Diff B3
AAS= Diff B1 + Diff B2 + Diff B3 + Diff B4

Figure 4.8: Example of the aligned axis score.

The Short-Straw algorithm calculates the angle that is generated between the current point in the stroke with the ends of the short straw to determine when there is a possible corner (close to 90 degrees). The complexity of this algorithm [82] is O(N), however since a photo contains a high number of pixels (points), the window was increased to jump every 3 pixels, reducing the complexity to O(N/3). This might not always find the best pixel that represents a given corner, but is is close enough for the purposes of our system.

At this point, the algorithm has the following components of a stroke:.

- *Start point*: First point of the stroke point list.
- *End point*: Last point of the stroke point list.
- *Stroke Length*: Number of points in points list.

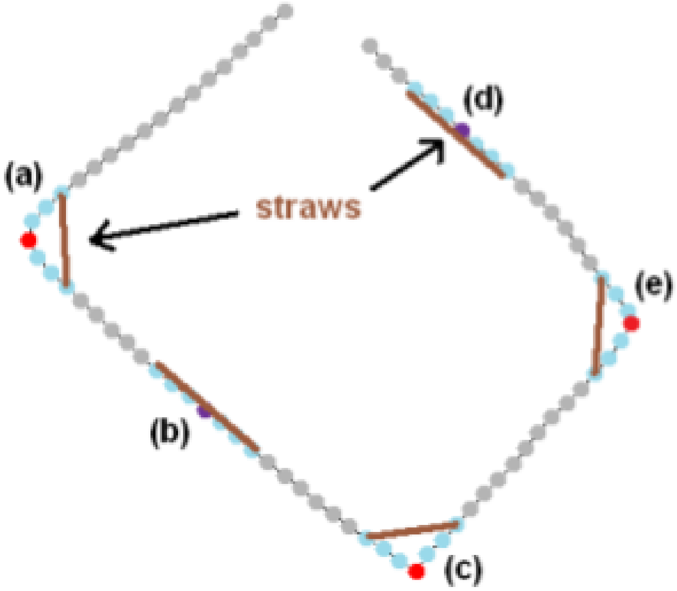


Figure 4.9: An example of the poly-line corner finder *Short-Straw*

- *Total Rotation*: Overall change in direction from one point to another (actual value - so a wavy line could have a small or even 0 rotation value)

- *Total Absolute Rotation Change*: Overall change in direction from one point to another. (values are absolute - so a wavy line would necessarily have a large rotation value).

- *Corners*: Best approximation to a point (using shortstraw) that have a nearly 90-degree angle with his neighbors.

- *Number of sub-strokes*: Number of lines found in a stroke after corners detected.

- *Distance between Start-End point*: Euclidean distance between the Start point and the Endpoint.

- *# of corners*: Number of corners of the stroke.
- *Axis aligned score (AAS)*: Measure of how aligned are the corners of the stroke (shape) with the X and Y axis (only work for quadrilaterals).

Once the above metrics are calculated, the system uses a set of geometry rules to categorize the strokes into complete shapes or unrecognized strokes if the rules are not met. The system detects: Circles/Ellipses, rectangles, diamonds, parallelograms and arrows as shown in Figure 4.10.

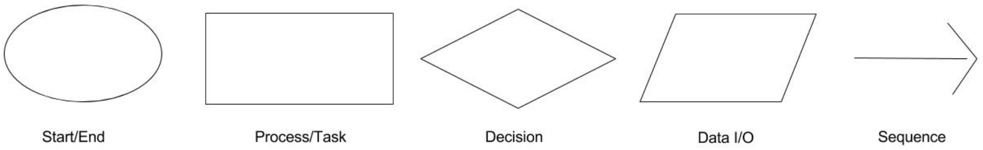


Figure 4.10: Flowchart shapes and their respective operations

The algorithm then breaks down all the recognized strokes by its corners and store the lines on a list for future processing.

4.1.3 Shape recognition

Once strokes are extracted and processed, the algorithm performs shape recognition and attempts to differentiate between rectangles, diamonds, parallelograms, ellipses and arrows. Shape recognition [83] attempts to classify the strokes that have been saved so far as either complete shapes or unrecognized strokes. First, the system uses a bottom-up approach to classify the strokes as complete or unrecognized strokes and then uses a top-down approach to try to approximate the shapes from the unrecognized strokes using a k-nearest neighbor approach. Finally, a tree of shapes and arrows is generated from the recognized shapes and arrows

The system uses a novel algorithm called *closedShapeRecognizer* that takes a stroke as an input and return a category as output. This category can be rectangle, diamond, parallelogram, ellipse or unrecognized. The arrow recognition step is performed after the closed shape recognition is made.

The set of features calculated was pruned as described in the stroke processing section (Section 4.1.2) so that this part of the recognition process could be simple, fast, efficient, and effective in recognizing between the aforementioned shapes. Shape recognition is performed by ruling out or pruning the shapes depending on the features they have.

4.1.3.1 Overall ordering of shape recognition

The first rule is to see if it matches the feature thresholds for an ellipse, it is worth mentioned that also a circle would work as a terminal or initial shape.

If the ellipse rule is not met, then the algorithm proceeds to check if the stroke features meet the rules for a rectangle.

Then if the rectangle rule is not met, the algorithm proceeds to check if it can be either a diamond or a parallelogram. These two shapes share metrics to a degree, however, the angles within the shapes help to differentiate between the two.

If none of the above rules are met then the stroke is marked as an unrecognized stroke and the algorithm ends.

4.1.3.2 Ellipse recognition

Several thresholds must be within certain values for a shape to be defined as an ellipse:

- *DistanceBetweenStartToEnd* must be less than 20px. A margin must be allowed because the users almost never fully close the ellipses as preliminary tests indicated.

- the *number of corners* must be less than 3. The reason why any corners are allowed is because the corner algorithms sometimes give one or two false positives because of the angles of the extremes of the ellipse.

- *AbsoluteRotationalChange* must be more than 40 The major feature that differentiates between the ellipse and the other shapes is the *AbsoluteRotationalChange*. This Rubine feature measures the sharpness of the stroke, if the number is high it means that is smoother and it usually contains curves instead of lines.

Note that the *RotationalChange* (without the absolute value) would be the same for ellipses, rectangles, diamonds, or essentially any closed shape, which is why the absolute value is used instead. Since the *AbsoluteRotationalChange* is significantly different between the ellipses and the rest of the shapes, ellipses are checked first.

4.1.3.3 Quadrilateral recognition

After ruling out the possibility of a set of strokes are not an ellipse, the system then checks if the shape is a quadrilateral shape, or in this case if is either a diamond, a parallelogram or rectangle. The following thresholds must hold for a shape to be a quadilateral.

- the stroke has more than 2 and less than 5 corners

- the *DistanceBetweenStartToEnd* is less than 20

- the *AbsoluteRotationalChange* is more than 20

In the case the stroke has exactly 3 corners, there is one additional step. This is most likely to be because the start and the end of the stroke do not join together and there is a gap between the endpoints, and thus ShortStraw would not show it to be a corner. If there are only 3 corners, the system attempts to approximate a fourth corner by calculating the midpoint between the start and the end point of the stroke and adding it as a corner. Since the distance between the start and the endpoint need to be less than 20 then the midpoint between them as corner would not change the topology of the quadrilateral.

Now the system has identified that the stroke is a quadrilateral but the system still has to differentiate between a diamond, a parallelogram or a rectangle.

4.1.3.4 Diamond recognition

To classify amongst the three quadilaterals, the system takes as an input the four corners (x,y coordinates) of the quadrilateral. The corners are checked to make sure that they are listed in a clockwise orientation. This helps to ensure that the set of corners one-third and second-fourth are opposite. The Axis Aligned Score, described above, calculates the X and Y differences between the set of corners and the closest axis aligned generated corners to provide a measurement of how aligned are the corners to either the X or Y axis.

1. If the Alix Aligned Score is less than 80 pixels the stroke is labeled as a *diamond*.

4.1.3.5 Rectangle & parallelogram recognition

If the shape is determined to be a quadrilateral, but not an ellipse, the system then attempts to see if the quadrilateral is either rectangle, parallelogram or an unrecognized stroke.

The systems then calculates the two diagonal lengths by calculating the distance between the first-third and second-fourth corners. The diagonal difference between the two of them should be the about the same; if so, the shape is labeled as a *rectangle*. If one of the diagonals is bigger then the other to a certain threshold then the stroke is labeled as a *parallelogram*.

If none of the above rules are met then the stroke is labeled as unrecognized.

The methodology used to distinguish between quadrilaterals is merely geometrical and is aided by a few thresholds. This is possible cause only a few shapes are to be recognized. If the system will include more and more shapes, then these thresholds and feature combination will most likely need to change or increase.

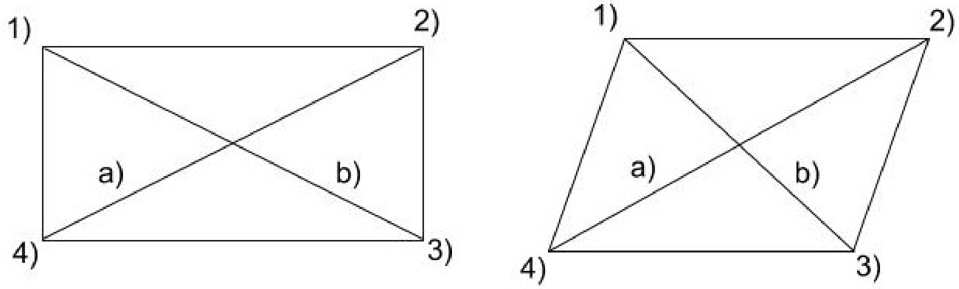


Figure 4.11: RECTANGLE - The system calculates diagonals a) and b) to see if they are similar enough, PARALLELOGRAM - The system calculates diagonals a) and b) to see if a) is bigger than b) given a threshold

4.1.3.6 Arrow recognition step

The arrow recognition step is performed after all the shapes have been recognized. This ensures that the remaining unrecognized strokes could only be one of two: the shaft or the head of the arrow. The system currently only recognizes the two stroke arrows where the shaft and the arrow head are drawn separately. To recognize arrows, the system runs a recursive algorithm with the unrecognized strokes available and checks if the strokes are shafts or arrows and adding them together using a euclidean distance threshold as a rule. This joins all the strokes that compose an arrow and also identify the orientation of the head and the start and end of the shaft. That will be crucial when dealing with generating code from the existing recognized shapes.

4.1.3.7 Top down approach

At this point, the system has two sets, the recognized shapes and the individual unrecognized strokes. This top-down approach will run a recursive algorithm to try to approximate shapes by using a nearest neighbor approach [84]. The recursive depth is limited to four to prevent any stack overflow problems. The algorithm runs as follows, it first iterates through the unrecognized strokes and find the closest stroke available giving a certain threshold. If the stroke is close enough the algorithm joins the strokes and tests if the shape recognition rules are met, if not then the algorithm tries the same on the next closest stroke to our new partial-composed stroke. This is done recursively, increasing a recursion counter to limit the recursion to four nested ones since all of our shapes have four sides. The system marks all strokes that are being joined to avoid repetition in the recursion. If the partially composed stroke passes any of the above shape recognition rules then the recursion is complete and the composed shape is added to the recognized shapes set. Then the individual strokes of the composed stroke are pruned from the list of unrecognized strokes to avoid repetition.

4.1.4 Post-processing

The post-processing step reviews the results of the recognition step and allows for correction of the text recognized on the flowchart. It also allows for the option of generating and execute the code if possible. These steps are grouped into a single category because they occur after the shape and text recognition steps.

The review step allows the user to visualize what was and wasn't recognized from the drawing. If not all shapes and arrows are recognized then the code generation will not work as well, and the user can go back the drawing and make changes where his/her drawing wasn't recognized. This review step is optional so the user can choose to go to the correction module even if not all shapes and arrows are recognized.

Next, is the correction step allows to make sure that the text was recognized properly. This step is very important to allow for the system to convert the recognized input to code, and if something is not recognized properly the output algorithm will not be able to execute even though the diagram was correctly designed. Another reason the post-processing step is necessary is because the handwritten text recognition libraries are also known as OCR (Optical character recognition) [85] available today do not provide a great accuracy output. The system currently using Tesseract [86] an open source [87] library for Object Character Recognition, the library was released under the Apache License Version 2.0 [88], and development has been sponsored by Google since 2006. Tesseract is considered one of the most accurate open source OCR engines currently available and has been used in many projects since its release [89, 90, 91].

After the post-processing step, the application uses a recursive function to go through the shape tree to generate the pseudo-code from the previously recognized shapes. The user is then given the option to execute that pseudo-code.

To execute the psuedo-code, the system quickly formats the pseudo-code to javascript notations[92] and then uses a javascript execution library for android called Rhino [93] (developed by Mozilla Foundation[94]) to run the code and get the result. The Rhino API has been used by many projects [95, 96, 97] since it release and its proven to be very efficient when compiling and executing javascript code.

4.2 System design

4.2.1 Architecture design

This section describes the overall architecture of the framework and the sketching design choices that were made to built Flow2Code. Figure 4.12 shows a UML [98] component diagram [99]. Each component is represented with a box; the biggest blue box is the whole Flow2Code application.

The system has four other components which are: ProjectManager, Pre-Processing, SketchRecognition, and Post-Processing. ProjectManager takes care of the user input and allows a user to Create, View, Update, Delete and Share a new project. The create module follows the data flow process begins as described above going from the pre-processing steps, followed by the recognition, and finishing with the review, correction, and finally code generate/execute.

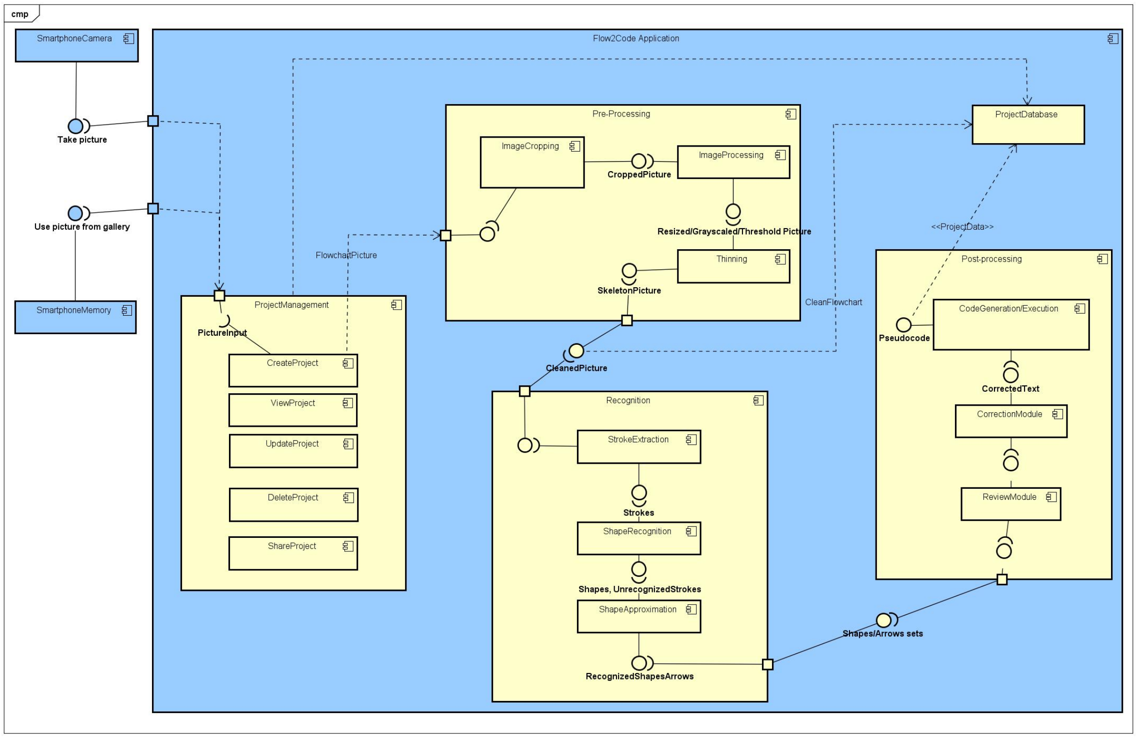


Figure 4.12: Flow2Code UML component diagram

The previous diagram attempts to make the high-level design of the architecture of the system and the overall data flow understandable and easy to follow.

4.2.2 Sketching design choices

When designing Flow2Code many design wise choices had to be made to be able to provide a complete, easy-to-use tool for beginner programmers. Flowchart-based programming had been around a while, and there are already some systems that use this approach as mentioned in the prior work chapter. The set of rules and constraints were chosen to improve accuracy when translating the flowchart into code. Ideally, the system would take as input a free hand diagram with little to no constraints when drawing. However, this imposes a big threat to the overall recognition since the noise or overlapping strokes will most certainly decrease the accuracy.

The first design decision was the set of shapes to include to serve as building blocks for the flowchart. The shapes that are most accessible to new users are the four basic flowchart shapes because it doesn't impose a great amount of knowledge to learn how to use them: Circles or Ellipses for Start/End indicators, Rectangles for processing steps, Parallelograms for Input/Output data, Diamonds for decision making, and finally Arrows for sequence.

To allow the user to draw the flowchart as intuitive as possible the user can draw shapes that are not fully closed in the corners. In early stages of testing, many users do not close the shapes if they are drawing as quickly as possible. Figure 4.13 shows an example of a few different ways a shape can be drawn by the user and still be recognized by the system.

It is still recommended that the user closes the shapes for a faster and more accurate recognition.

For arrow recognition, there are also a number of ways in which the user can draw them, closed-empty head, closed-filled head, separated-open head as shown in Figure 4.14. Although Flow2Code can only recognize the last one at this time. The two-stroke separated-open head arrow is fast to draw, a common method for drawing them, and does not suffer

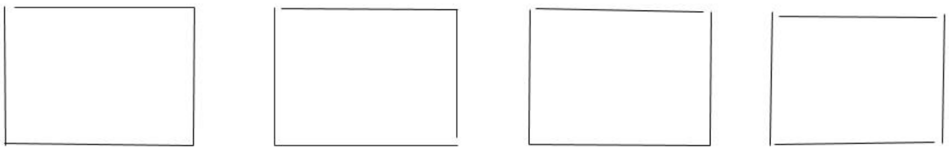


Figure 4.13: A few non-closed ways in which a rectangle can be drawn

drastic changes in the topology of its form when it goes through the thinning algorithm.

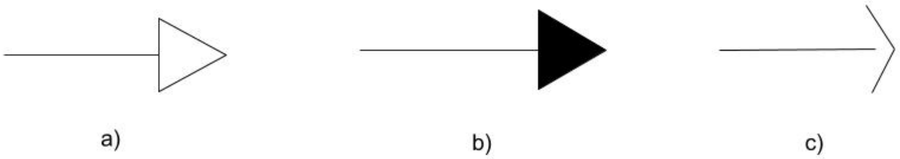


Figure 4.14: A few different ways in which a rectangle can be drawn. Flow2Code will only recognize c).

The arrow consists of two parts the shaft and the head. Although Flow2Code can only recognize one time of arrow head, it can recognize several types of arrow shafts. The shaft of the arrow can be drawn using multiple lines to allow the user more flexibility when designing the algorithm, as seen in Figure 4.15

Flow2Code implements an offline stroke separation algorithm [71]. However, this approach has difficulties separating collided strokes in some circumstances. Users that separated their objects had much higher accuracy. Examples of arrows that have better

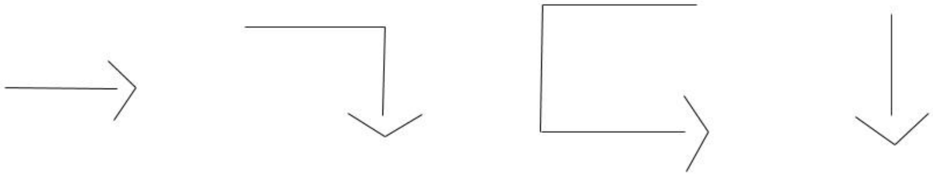


Figure 4.15: Some of the different arrows with different shafts that are recognized by flow2code.

and worse accuracy are shown in Figure 4.16. Shapes should not collide with the arrows have better accuracy. Likewise, text that does not collide with the outer shape also has better accuracy. Users that adopt this approach the system will work faster for users since it would not need to run the collision algorithm, and provides for better results. Also, it does not impose a big constraint to the user and the flowchart ends up showing cleaner to the sight.

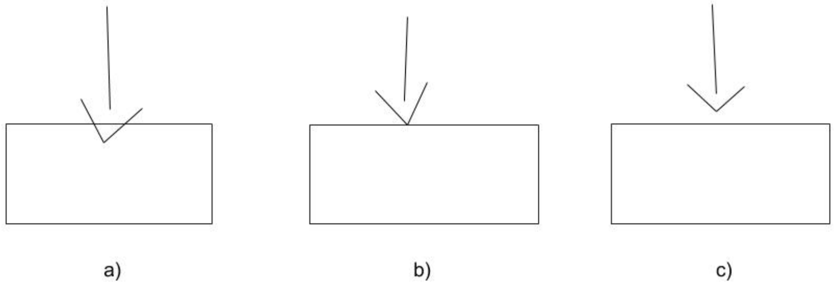


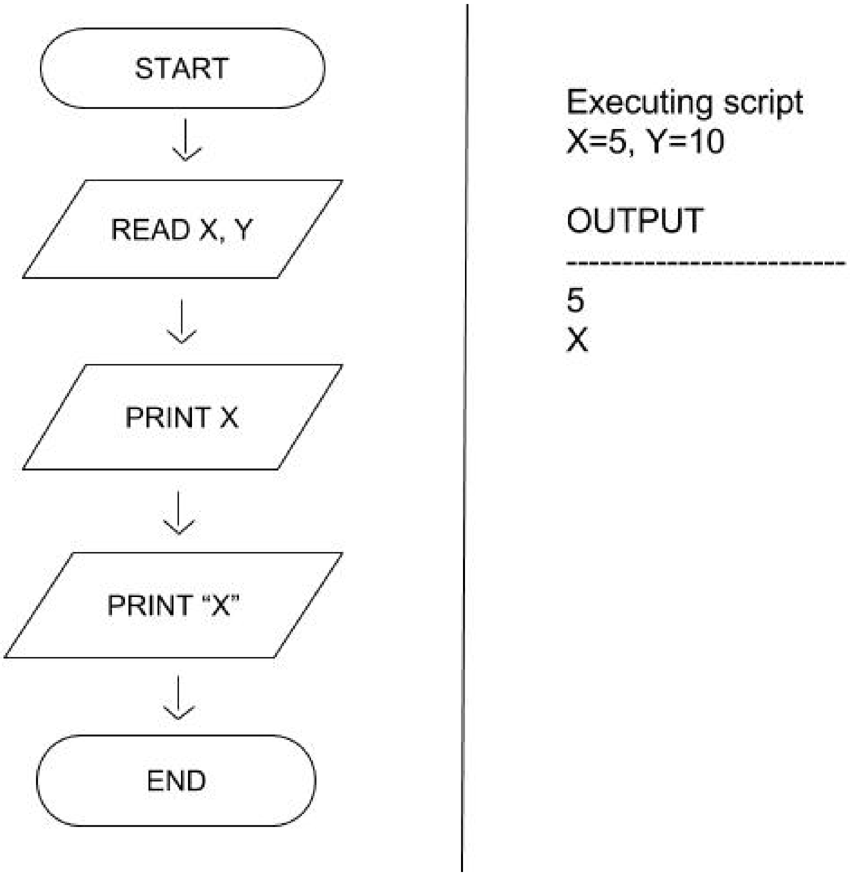
Figure 4.16: Example of the recommended way to draw sequence arrows and shapes; Flow2Code will all ways have more accuracy when avoiding collision like in the option C.

In order to improve the usability of the code execution, there are defined functions that can be used on the pseudo-code. The READ and PRINT function. The READ function must be used once at the beginning of the algorithm to ask for variable initializations. The PRINT function will print out whatever is passed through it adding a break-line at the end for formatting. The READ function can take one or more parameters separated by a comma. The user can use as many PRINT tags as he desires, all of them will be added to the output console when executed. If the user adds the PRINT tag followed by a variable name it will print the variable value. If the user chooses to use the " characters then the string value of what is inside them will be displayed. As shown in Figure 4.17 the user first reads two variables and then chooses to print the value of X followed by the string X. This way the user will have a way to keep track of the processing steps and be able to assess if the algorithm is well defined if the output is the desired one. These functions were defined in javascript and the overall implementation is not visible to the user to avoid any confusion.

Finally, the last constraint that the user needs to be aware is that the flowchart design does not need to include the TRUE/FALSE or YES/NO text near the conditional arrows. The design allows for if the arrow goes down it is immediately recognized as TRUE/YES, and if it goes left or right it is considered FALSE/NO as shown in Figure 4.18

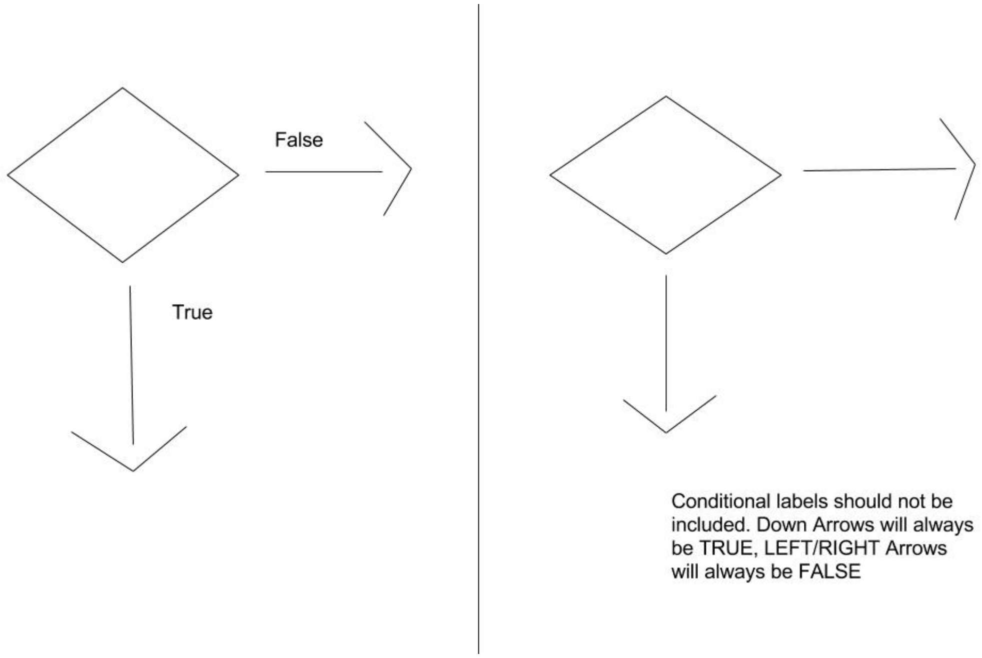
4.3 User interface design

The application was designed and implemented using the Google Material Design guidelines [100] and the Android API 5.0 codenamed Lollipop [101]. The GUI consists of three different screens: (1)Flowchart Project Management, (2) Editing/Correction flowchart, (3) Camera module and (4)Presentation and code editing/execution. The application is now in beta testing, and we are currently working to improve the recognition rates as much as possible.



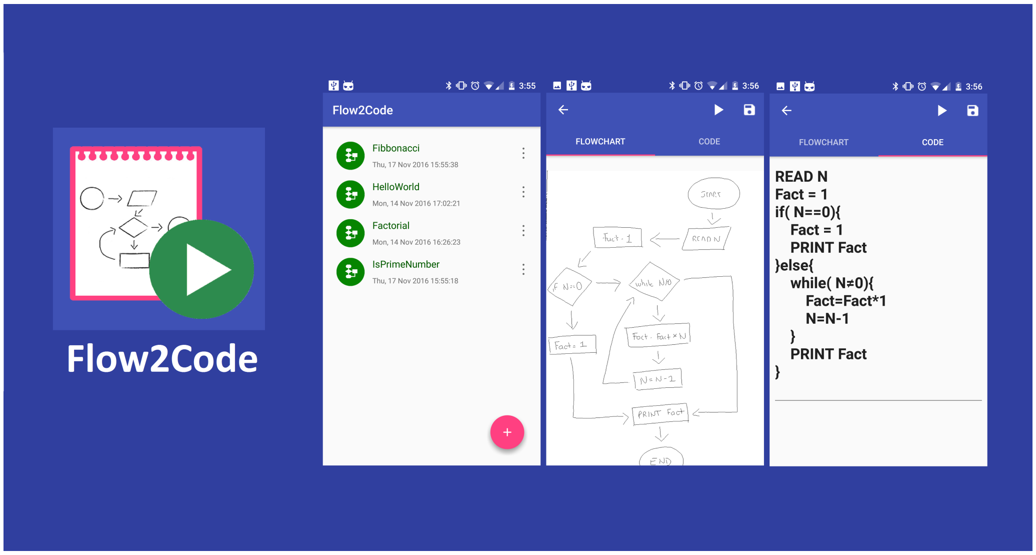
```
Executing script
X=5, Y=10

OUTPUT
-------------------------
5
X
```

Figure 4.17: Example of the the READ and PRINT tags added to Flow2Code



Conditional labels should not be included. Down Arrows will always be TRUE, LEFT/RIGHT Arrows will always be FALSE

Figure 4.18: Example of how conditional labels should be drawn excluding the labels).



Fibbonacci	Thu, 17 Nov 2016 15:55:38
HelloWorld	Mon, 14 Nov 2016 17:02:21
Factorial	Mon, 14 Nov 2016 16:26:23
IsPrimeNumber	Thu, 17 Nov 2016 15:55:18

```
READ N
Fact = 1
if( N==0){
   Fact = 1
   PRINT Fact
}else{
   while( N≠0){
      Fact=Fact*1
      N=N-1
   }
   PRINT Fact
}
```

Figure 4.19: This image present the main modules of the mobile application. The first one from left to right enable the management of projects, followed by the recognized flowchart and then the code that was generated from it. The step of taking a picture with the flowchart was not included since the camera itself is outside the scope of the application.

4.3.1 Flowchart project management

This module provides the users with capabilities to manage their flowchart projects. It is the main UI screen that the user will see. From this screen, the user is capable of adding a new project if s/he needs to either by providing an image from the external or internal memory of by choosing to take a picture with the device. This module also allows the user to view and edit existing projects, delete them if they are no longer needed and even sharing capabilities to export the project to any cloud service the user have installed in their device such as Mail, Google Drive, Dropbox, etc. The different screens are shown in Figure 4.20.

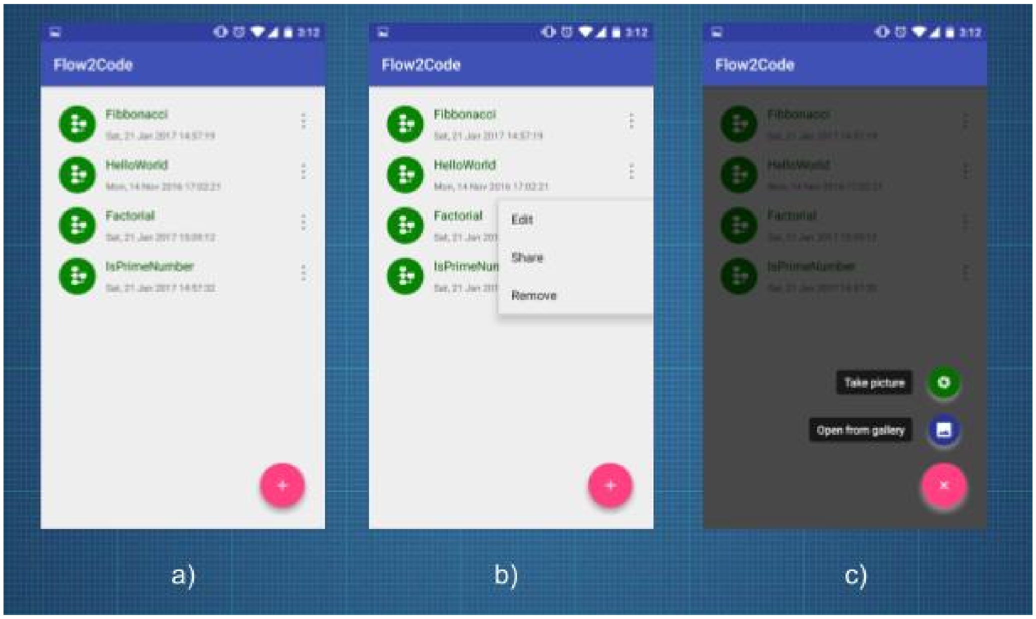


Figure 4.20: This image shows the different GUI screens that the management module uses. a) The image shows the different previously saved project listed, a button for adding new ones and an options button in each of the item lists, b) The image shows different options per project that can be performed: Edit, Share and Remove. c) The image shows the input options when adding a new project: Take picture or Open from gallery.

4.3.2 Camera module

This is the first intermediary module between the actions of adding a new flowchart by taking a picture and the view/execute module. This module will launch a camera application that the user has installed on their device and will use the picture taken as an input for the flowchart recognition to begin. After a picture is taken the user will be required to make any adjustments or cropping to the image to be able to remove the external noise that is usually present at the borders of the picture. Once the user accepts the crop boundaries of the image, the system will begin the overall process for recognition and will be presented with the correction module next. Figure 4.21 shows the camera modules.

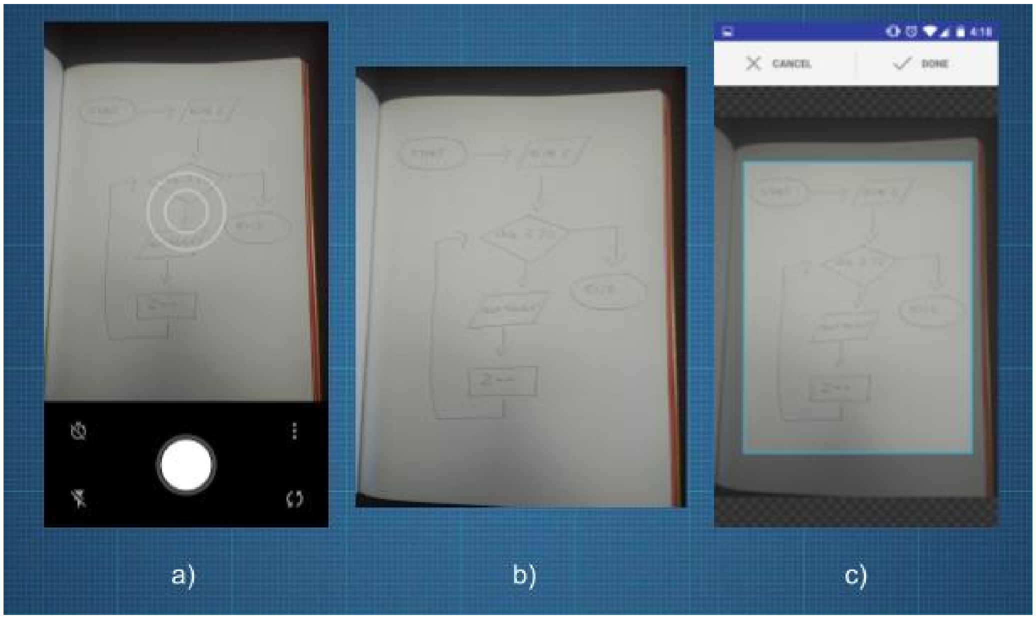
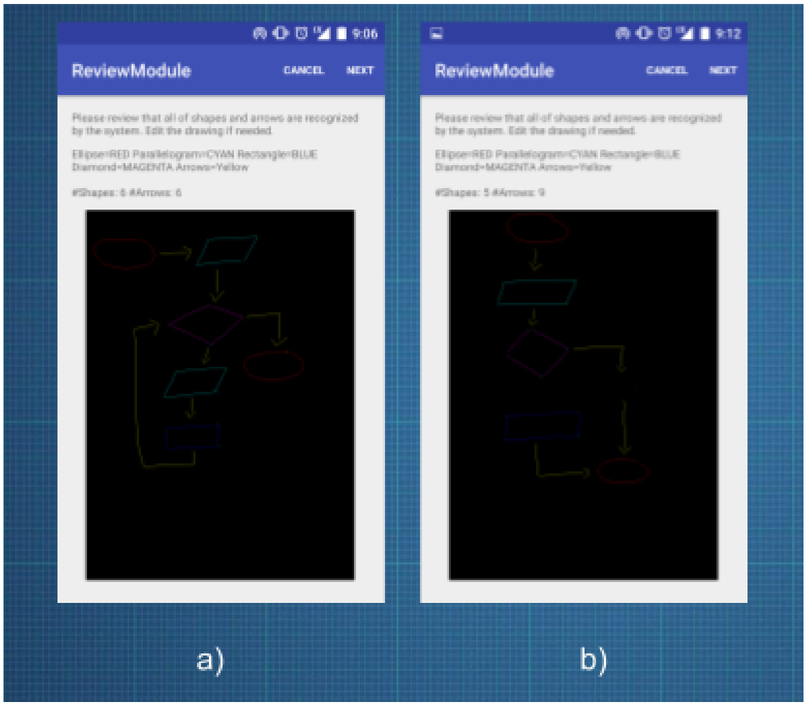


Figure 4.21: This image shows the different GUI screens that the camera module uses. a) The camera takes a picture of a previously drawn flowchart, b) This is the picture taken with the camera. c) The user can crop the image after taking or selecting a picture. The blue boundaries will crop the image as desired.

4.3.3 Review module

This is the second intermediary module between the actions of adding a new flowchart and the view/execute module. This module was added to give a preview of what was recognized by the system. Using color coded strokes the user can easily see not only what was recognized but also if the shapes were recognized correctly. This way the user can see for themselves if they have to go back to the drawing and make edits or continue to the next step. There is no processing done in this module other than the color-coded flowchart generation and that is the reason why this step was not included in the system process described previously.



Please review that all of shapes and arrows are recognized by the system. Edit the drawing if needed.

Ellipse=RED Parallelogram=CYAN Rectangle=BLUE Diamond=MAGENTA Arrows=Yellow

#Shapes: 6 #Arrows: 6

Please review that all of shapes and arrows are recognized by the system. Edit the drawing if needed.

Ellipse=RED Parallelogram=CYAN Rectangle=BLUE Diamond=MAGENTA Arrows=Yellow

#Shapes: 5 #Arrows: 9

Figure 4.22: This image shows the different GUI screen that the review module uses. a)Shows an example of a flowchart with all its shapes and arrows recognized correctly, b) Shows an example of a flowchart with one missing (not recognized) shape and arrow

4.3.4 Correction module

This is the third intermediary module between the actions of adding a new flowchart diagram and the view/execute module. This allows the user to manually edit all the text that is being recognized by the system. This module was developed to guarantee a 100% accuracy when dealing with hand-drawn text recognition since the OCR libraries in existence are not very accurate. The user can navigate by swiping left or right and manually edit the recognized text. This module is important to the system due to the nature of the project since it needs a 100% accuracy to be able to execute the code extracted from the flowchart. Figure 4.23 shows the correction module screens.

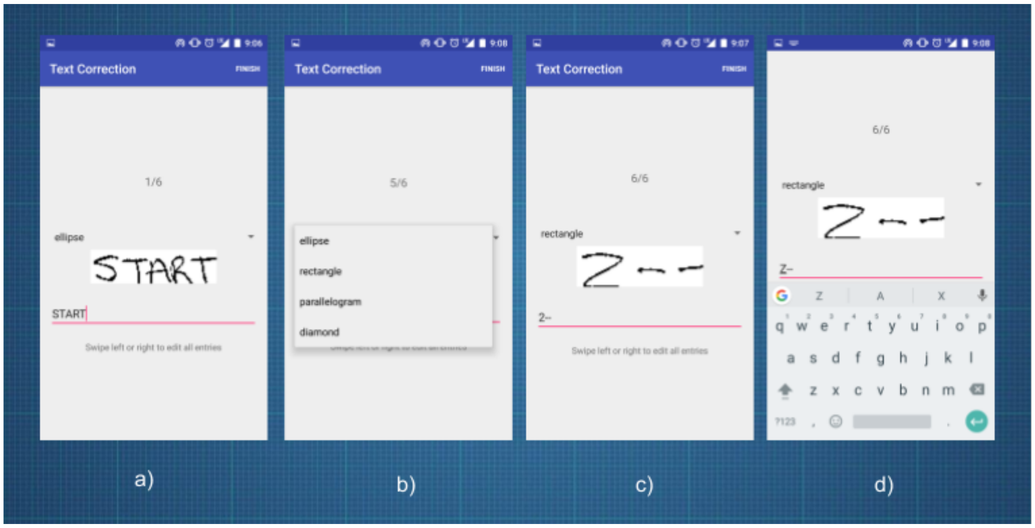
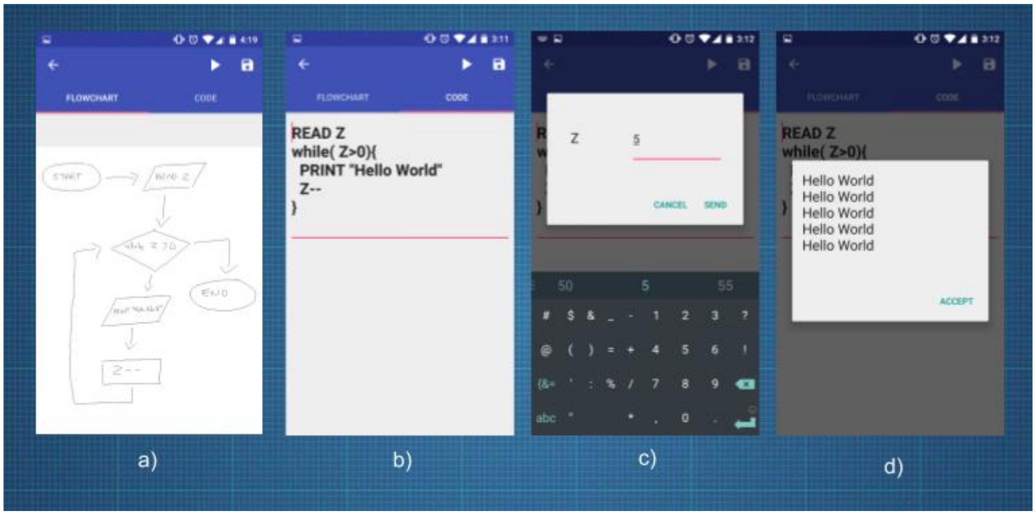


Figure 4.23: This image shows the different GUI screens that the correction module uses. a) Shows an example of a 100% recognized text string. b) Shows an example of a wrong recognition, the system recognized the number 2 instead of the letter Z. c) Shows how the user can edit the text in case errors in text recognition occur.

4.3.5 View/execute module

This is the last module that the user will interact with, first, it shows the user with a cleaned black and white version of their flowchart and also a translated pseudo-code from the flowchart itself. The pseudo-code is only for presentation only and it is not bound to any programming language, although javascript notation is fully integrated. The user can choose to manually edit the code if it was not recognized appropriately and will be given the option of executing the code from the same module. If the user chooses to execute the module, the system will first verify if there is any input variable needed for the algorithm to run. If there is at least a variable needed the system will prompt a dialog box asking for the input after it is set the system will execute the code with the input provided and will show the result in a different dialog. If there is no input variable needed for its execution the system will skip the input variable dialog and will proceed to show the output console dialog. The system can choose to save the current project or to exit without saving. If the user decides to save the project they system will ask for a name to be given to the project, after that the system will show the main module the flowchart project management and the project will be listed there for future view or edit. Figure 4.24 shows the correction module.



```
READ Z
while( Z>0){
  PRINT "Hello World"
  Z--
}
```

Z 5

```
Hello World
Hello World
Hello World
Hello World
Hello World
```

Figure 4.24: This image shows the different GUI screens that the view/execute module uses. a) Shows the cleaned flowchart after all the processing and recognition, b) shows the pseudo-code on a side tab, it allows also manual editing, c) shows how the system detects and ask for an input variable, in this example the variable Z, d) shows the output of the script once executed with the input variable Z=5.

5. EVALUATION

5.1 Evaluation plan

In order to fully evaluate the usability of the system, an evaluation was performed gaining both quantitative and qualitative feedback from users with a goal to measure both the recognition accuracy of the application and the usability acceptance from the users. Two different studies were implemented. The first one was a complete study involving both qualitative and quantitative results with the researcher as facilitator and able to answer doubts from the users.

The first study was be done by 20 subjects and only one requirement was asked when selecting them: The user must learn how to model algorithms using flowcharts. Which since most of the subjects were computer science students from graduate school all of them already had flowchart background. The subjects recruitment was done via email using the Texas A&M bulk mail and also by asking around campus. The study consisted of four sections: 1) Purpose of the study, 2) Pre-questionnaire, 3) Experiment and 4) Usability Survey, Interview, and comments from the user.

The user study was conducted in the Sketch Recognition Lab at Texas A&M University. The set up consisted of a single desk, chair, pieces of white paper, pencil, eraser, pencil sharpener, and an Android smartphone with the Flow2Code application installed and the drawing guidelines as you can see in Figure 5.1. All of it was provided by the researcher.

The second user study was performed in a classroom for an Introductory Computer Science course at TAMU (CSCE 206) at ETB 2005 by 45 Undergraduate Students. This study was meant to be more quick and unsupervised study. This meant that the researcher will not be able to answers from the users. The idea of this second study is to recreate

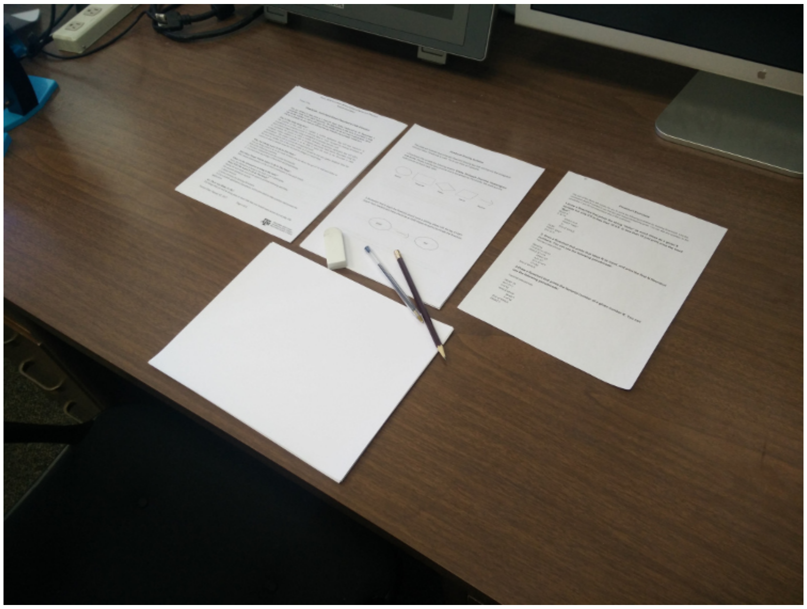


Figure 5.1: The set up environment for the user studies

a worst case scenario in which a user would be performing a flowchart drawing without any guidance. This will allow us to identify strengths and weaknesses of the system. No qualitative data was collected from this study. Only one drawing of a flowchart with minimum instructions. The sketches were collected for further analysis.

We will focus on the first and complete user study done by 20 TAMU students since it will give us both qualitative and quantitative data. We will discuss later the results obtained from both the complete study and the smaller one.

5.2 Complete user study

5.2.1 Purpose of the study

The first part of the evaluation plan was to explain the user about the overall purpose of the application, explain the idea behind it, as well as explaining the further steps of the study and the amount of time that it will take. After that, the user was asked to sign a consent form according to the TAMU Institutional Revision Board (IRB) [102] requirements. The researcher then answered all the questions that the subject had before starting the next step.

5.2.2 Pre-questionnaire

The researcher handed a pre-questionnaire to the user to gather information and data about demographics and previous experience with the flowchart based systems for programming as you can see in appendix A. The purpose of this part is to measure not only demographic information but to also be able to know a little more about the subject's background before the experiment take place. Google Forms was used as a tool to speed up the questionnaire process. Again, the researcher answered all the questions that the subject had before starting the next step.

5.2.3 Experiment

The experiment consisted of two steps: 1) Drawing exercises, 2) Flow2code use. Before the experiments take place the user was handed a sheet of drawing guidelines that describe briefly how to use the application, as well as drawing tips to get better recognition results. The drawing guidelines are not meant to be a detailed manual of the application. The drawing guidelines can be seen in appendix B.

Once the subject read the drawing guidelines the user was handed the exercise sheet, five blank sheets of paper, a pencil and an eraser. The subject was allowed to keep the drawing guidelines during the drawing step. Only three sheets of paper needed to be used, one per exercise. If the subject needed more, the researcher handed them. You can see the exercises sheet on appendix C.

The drawing step consisted of three different flowchart related exercises in which the subject needed to draw the algorithm using a flowchart. You can see in picture 5.2 a subject that is drawing of the flowcharts from the exercises.

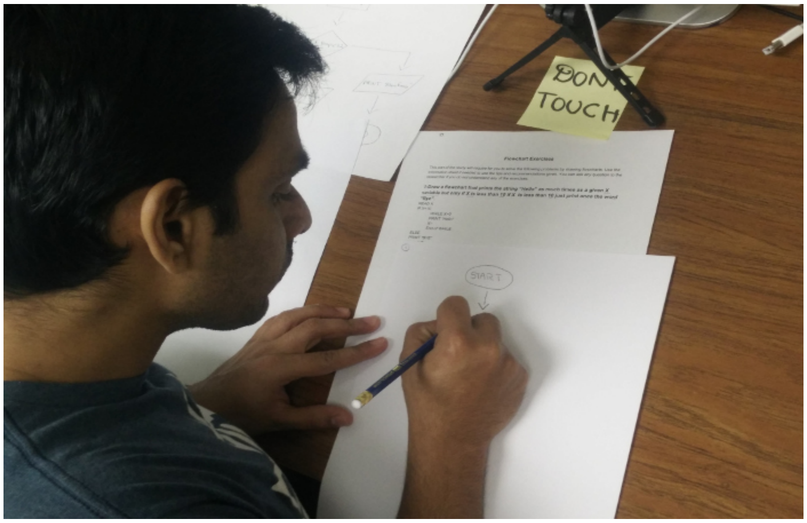


Figure 5.2: A subject drawing a flowchart to solve one of the exercises

It is important to mention that in this step the researcher played the role of observer and only spoke when the user did not understand any part of the exercise to avoid any type of coercion.

Once the subject finished drawing the exercises, the next step of the experiment took place. The subject was handed with an Android smartphone with the specifications seen in table 5.1:

Table 5.1: Hardware specifications of the android smartphone used in the study

Hardware	**Specifications**
CPU	Qualcomm Snapdragon 801 processor with 2.5GHz Quad-core CPU'S
RAM	3 GB LP-DDR3 1866MHz
Display	5.5 inch JDI 1080p Full HD (1920X1080), 401PPI
Back camera	13 Megapixel - Sony Exmor IMX 214, f/2.0 aperture, Dual LED flash
Android	Lollipop version 5.1.1

All users used a smartphone that was supplied to them for the study so that privacy about their data was not a concern and also to ensure they had a device for the study. We also could then ensure that the users had the app on their screen.

Once the user had the smartphone with the application on the screen it is important to mention again that the researchers did not explain how to use the application. Also, the smartphone had no application running in the background other than Flow2Code.

Next, the subject was asked to create a new project for the third drawing (E03-Factorial) and see if the system is able to execute the code that was recognized from the drawing as you can see in picture 5.3. The researcher provided only guidance if the subject was lost during the use of the application. Once the user finished adding and saving all three of the new projects the smartphone was taken away as well as the smartphone.

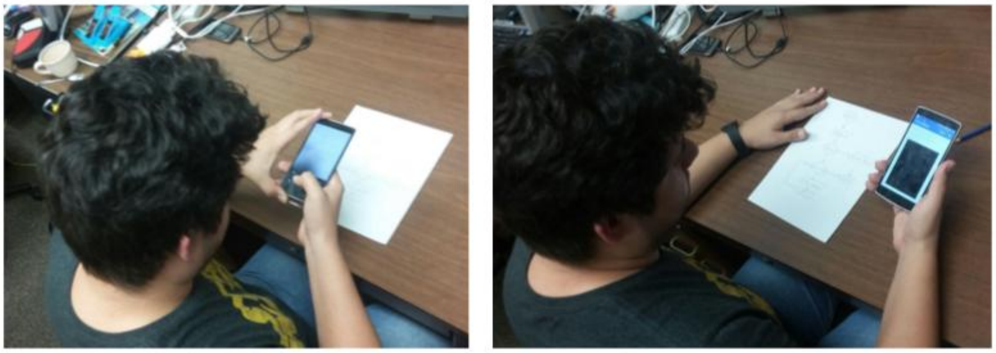


Figure 5.3: A subject taking a picture of his handrawn flowchart and then comparing the recognized shapes preview

5.2.4 Post-survey, interview and comments

For the last part of the evaluation, the researcher handed a Likert scale survey to the user. The scale is called SUS (System Usability Scale)[103] and measures the usability of a software or application as you can see in appendix D. After the scale was completed the researcher and the subject engaged on a semi-structured interview[104] and the subject was asked a few questions regarding the overall application. All the answers were written down in a notebook by the researcher. The researcher asked finally any open suggestions or comments that the user had at the end and also recorded the same in a notebook. You can see the semi-structured interview script on appendix E. After this step, the researcher thank the subject and the user study was completed.

5.3 Results

This section will show the results of the user studies in detail providing all the information that were gathered during the user studies. First of all, we will describe the user background and demographics, followed by showing the quantitative evaluation of the prediction accuracy results of the system. Finally, we will show quantitative information gathered from the System Usability Scale, and the semi-structured interview.

5.3.1 Subject demographics

The user studies took place at Texas A&M Sketch Recognition Lab. Data about the 20 subjects was collected before starting the user study. As you can appreciate in picture 5.4 most of the subjects were male, between 19 and 30 years old.

All except two subjects were not in the Engineering and Computer Science department. However, they mentioned knowing flowcharts and programming basics fundamentals and that's the reason they were allowed to take the user study. Most of the students were in Grad School (80%) and the rest were Undergrads(20%) as you can see in Figure 5.6.

Two questions were asked in the pre-questionnaire to the users about if they were familiar with flowchart design and programming. The first question was: Have you taken any courses that involve learning to code using flowcharts at any point in your life? This question was answered by a simple yes or no as you can see in figure 5.7. The reason for this question is because even though flowchart-based programming is a widely used

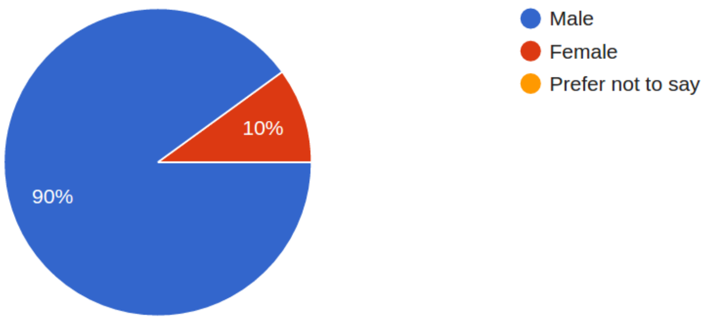


Figure 5.4: The graph show the gender distribution of the subjects from the user study

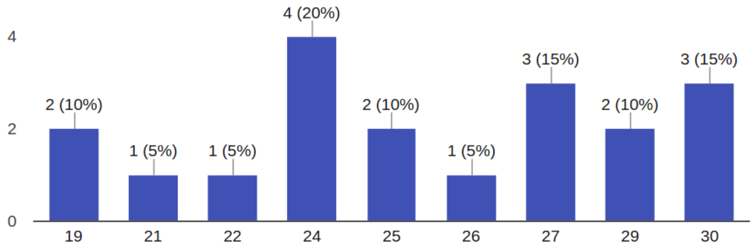


Figure 5.5: The graph show the age distribution of the subjects from the user study

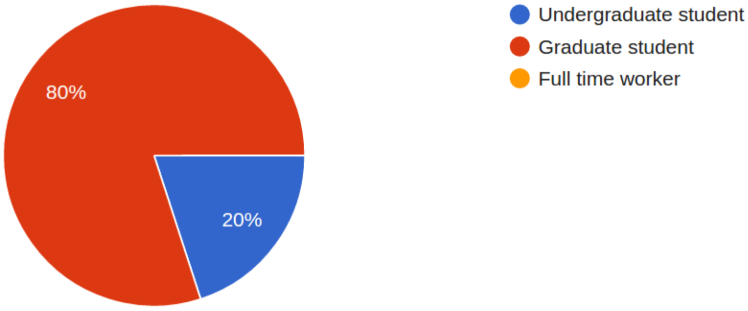


Figure 5.6: The graph show the occupation distribution of the subjects from the user study

technique to get started with programming some institutions do not use them and begin with code lessons since day one. 80% of the users had used flowcharts as a tool for coding prior to the study and 20% did not as shown in Figure 5.8.

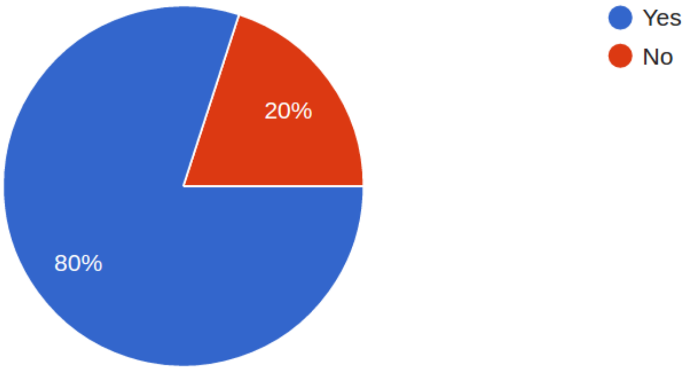


Figure 5.7: The graph show the distribution of the subjects who agreed to having taken a course where flowchart design was necessary

For the users that answered positively to the previous question, they were asked if they used a software as a tool for drawing or even just designing flowcharts. 60% of the subject indeed used a software and the other 40% did not. After that, the users were asked to give the name of the software tool if they remembered. Only 7 subjects remembered the tool they used being: MS Word, MS PowerPoint, OmniGraffle, and Origami the only answers given.

Finally, the survey asked them to rate both their flowchart design skills and their programming skills on a scale from 1 to 5. Figures 5.9 and 5.10 show that 50% of the subjects marked 3 as the answer for the flowchart design skills and 35% of them marked 4 for their programming skills. The reason behind this set of questions was to get some information

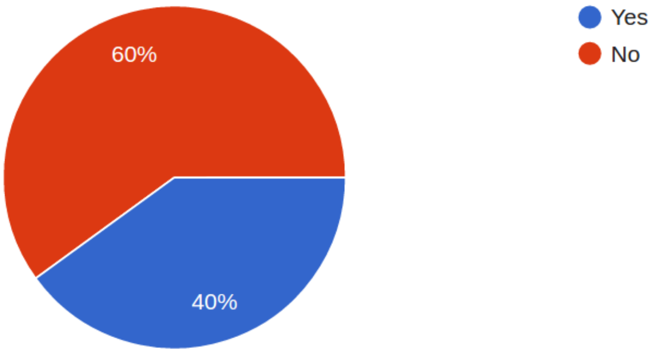


Figure 5.8: The graph show the distribution of the subjects who agreed to have used a software tool for software design in the past

about how confident they are around flowcharts and code (programming).

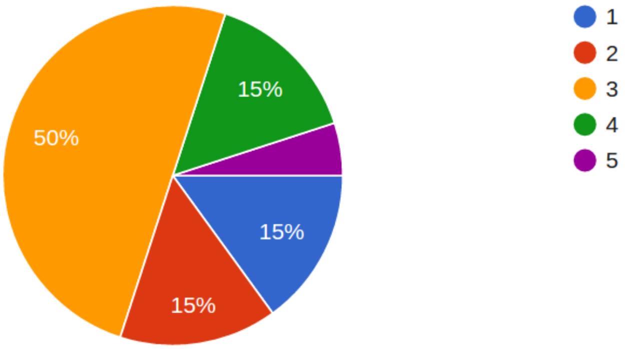


Figure 5.9: The graph show the distribution of the flowchart design skills that the subjects gave themselves, 1 being minimum and 5 maximum

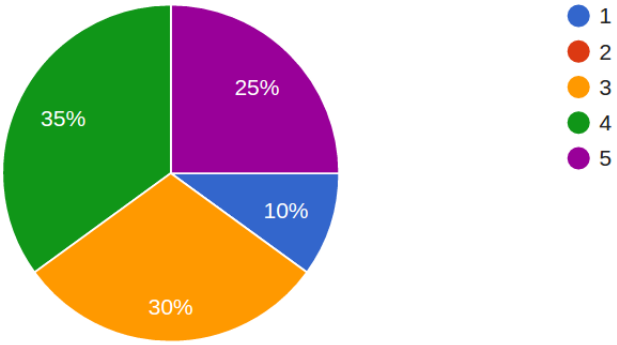


Figure 5.10: The graph show the distribution of the programming skills that the subjects gave themselves, 1 being minimum and 5 maximum

5.3.2 Predictive analysis

In order to measure the accuracy of the recognition of the system, we will provide first the results of each of the exercises followed by the overall results. Confusion matrices portray the performance of the recognition algorithm, while evaluation metrics to discuss the goodness of the recognition algorithm.

As mentioned previously three different exercises were handed to the subjects to be solved by drawing a flowchart:

- E01/Hello
- E02/Fibonacci
- E03/Factorial

You can see an example of the E01 flowchart in appendix F, E02 in appendix G and E03 in appendix H. Table 5.2 shows the recognition results of the first exercise E01/Hello in a confusion matrix. In all experiments this was the first approach the subjects had with the exercises, followed by E01, and then E03. We included the classification "none" to include which shapes were not recognized at all by the system, Table 5.2 shows the missed categorization of shapes was not common and most of them present on parallelogram and rectangle. Also, there were 12 arrows, 3 ellipses, and 1 parallelogram that were not recognized at all (classified/predicted as none).

Table 5.2: Confusion Matrix of the first exercise (E01/Hello)

E01		PREDICTED						
		Ellipses	Parallelogram	Rectangle	Diamond	Arrow	None	
A	Ellipses	37	0	0	0	0	3	40
C	Parallelogram	1	51	5	0	0	1	58
T	Rectangle	0	0	22	0	0	0	22
U	Diamond	1	1	0	38	0	0	40
A	Arrow	0	0	0	0	168	12	180
L	None	0	0	0	0	0	0	0
		39	52	27	38	168	16	

For the second exercise E02/Fibonnaci, Table 5.3 shows almost the same results as the previous one, although in this case, two ellipses are recognized as diamonds or parallelograms. Also, 14 arrows were not recognized at all, as well as 2 ellipses, 1 parallelogram, and 1 diamond.

Table 5.3: Confusion Matrix of the first exercise (E02/Fibonacci)

E01		PREDICTED						
		Ellipses	Parallelogram	Rectangle	Diamond	Arrow	None	
A	Ellipses	35	0	2	1	0	2	40
C	Parallelogram	1	35	2	1	0	1	40
T	Rectangle	0	1	77	0	0	2	80
U	Diamond	0	1	1	17	0	1	20
A	Arrow	0	0	0	0	167	14	181
L	None	0	0	0	0	0	0	0
		36	37	82	19	167	20	

Finally for the third and last exercise E03/Factorial we can appreciate a drop on the non recognized arrows (only 8) as well as only 2 ellipses and 2 diamonds. You can see the confusion matrix for the third exercise in table 5.4.

Table 5.4: Confusion Matrix of the first exercise (E03/Factorial)

E01		**PREDICTED**						
		Ellipses	Parallelogram	Rectangle	Diamond	Arrow	None	
A	Ellipses	38	0	0	0	0	2	40
C	Parallelogram	0	39	1	0	0	0	40
T	Rectangle	0	0	78	0	0	2	80
U	Diamond	0	1	0	19	0	0	20
A	Arrow	0	0	0	0	172	8	180
L	None	0	0	0	0	0	0	0
		38	40	79	19	172	12	

Since the number of shapes needed per exercise does not change greatly between exercises them we added up the previous three confusion matrix into the one as you can see in table 5.5 which will give us the overall recognition results and will be useful to compute recognition metrics.

Table 5.5: Confusion Matrix of all the exercises together

ALL		**PREDICTED**						
		Ellipses	Parallelogram	Rectangle	Diamond	Arrow	None	
A	Ellipses	110	0	2	1	0	7	120
C	Parallelogram	2	125	8	1	0	2	138
T	Rectangle	0	1	177	0	0	4	182
U	Diamond	1	3	1	74	0	1	80
A	Arrow	0	0	0	0	507	34	541
L	None	0	0	0	0	0	0	0
		113	129	188	76	507	48	

Predictive analysis metrics from the aggregated confusion matrix give more meaning to the results by computing a 2-class confusion matrix for all of the shapes available except the "none" class since that one is not an actually predicted shape.

The evaluation metrics [105] used to describe the recognition of the algorithm per shape are:

- **Accuracy** - Measure of how often is the classifier correct,
- **True Positive Rate (TPR, sensitivity, recall)** - Measures the proportion of positives that are correctly identified as such.
- **False Positive Rate (FPR, fallout)** - Measure the proportion of positives that are incorrectly identified as such.
- **True Negative Rate (TNR, specificity)** - Measures the proportion of negatives that are correctly identified as such.
- **False Negative Rate (FNR, miss rate)** - Measure the proportion of positives which yield negative test outcomes with the test
- **Precision (Predictive value** - Measure for classifier exactness, low precision can indicate a large number of False Positives.
- **f-score (f-measure)** - Another measure for accuracy, it can be interpreted as the weighted average of the precision and True positive rate.

Figure 5.6 shows the evaluation metrics for shape analysis. The accuracy of the ellipses is considerably high with 98% as well as the precision with 97%. The FPR and FNR are below 1% which indicates that there we rent many false positives or negatives overall. The F1 measure being a metric for overall goodness of the classifier is just below 95% which is considered good as well.

Table 5.6: Confusion Matrix and metrics of the shape Ellipse

ELLIPSE	**Predicted**			Accuracy	0.9878	FNR	0.0833
Actual		Ellipse	Not Ellipse	TPR	0.9166	Precision	0.9734
	Ellipse	110	10	FPR	0.0031	Fscore	0.9442
	Not Ellipse	3	951	TNR	0.9968		

The accuracy of the parallelogram is considerably high with 98% as well as the precision with 97% . The FPR and FNR are below 1% which indicates that there weren't many false positives or negatives overall. The F1 measure being a metric for overall goodness of the classifier is just below 95%. (See Figure 5.7.)

Table 5.7: Confusion Matrix and metrics of the shape Parallelogram

PRLGRAM	**Predicted**			Accuracy	0.9842	FNR	0.0942
Actual		Parallelogram	Not Pararellogram	TPR	0.9057	Precision	0.9689
	Parallelogram	125	13	FPR	0.0042	Fscore	0.9363
	Not Parallelogram	4	936	TNR	0.9957		

The accuracy of the rectangle is considerably high with 98% as well as the precision with 94%. The FPR and FNR are below 1% which indicates that there weren't many false positives or negatives overall. The F1 measure being a metric for overall goodness of the classifier is just below 95%. (See Figure 5.8.)

Table 5.8: Confusion Matrix and metrics of the shape Rectangle

RECTNGL	**Predicted**			Accuracy	0.9851	FNR	0.0274
Actual		Rectangle	Not Rectangle	TPR	0.9725	Precision	0.9414
	Rectangle	177	5	FPR	0.0122	Fscore	0.9567
	Not Rectangle	11	884	TNR	0.9877		

The accuracy of the diamonds is considerably high with 99% as well as the precision with 97%. The FPR and FNR are below 1% which indicates that there weren't many false positives or negatives overall. The F1 measure being a metric for overall goodness of the classifier is just below 95%. (See Figure 5.9.)

Table 5.9: Confusion Matrix and metrics of the shape Diamond

DIAMOND	Predicted		
Actual		Diamond	Not Diamond
	Diamond	74	6
	Not Diamond	2	987

Accuracy	0.9925	FNR	0.0750
TPR	0.9250	Precision	0.9736
FPR	0.0020	Fscore	0.9487
TNR	0.9979		

The accuracy of the arrows is considerably high with 96% as well as the precision with 97%. The FPR and FNR are below 1% which indicates that there weren't many false positives or negatives overall. The F1 measure being a metric for overall goodness of the classifier is just above 95%. (See Figure 5.10.)

Table 5.10: Confusion Matrix and metrics of the Arrows

ARROW	Predicted		
Actual		Arrow	Not Arrow
	Arrow	507	34
	Not Arrow	0	554

Accuracy	0.9689	FNR	0.0628
TPR	0.9371	Precision	1
FPR	0	Fscore	0.9675
TNR	1		

Overall the accuracy of the shapes was high, with only a few missed shapes or wrong classifications. Even though the recognition rates were high it is possible that if user follows the drawing guidelines perfectly the recognition accuracy should be near 100%.

It is important to mention that all of the previous results were done on the user's first drawing. In other words, even though there is an intermediary review step in which the user can go back to the drawing to re-draw parts of the flowchart that were not recognized, those re-recognition results were not taken into account in this experiment since we were interested in the first drawing to test the algorithm recognition accuracy.

Now, as shown previously, the recognition accuracy for all the shapes is considerably high and that can be because of the enforced drawing guidelines that the users need to keep in mind while drawing. This means that a more valuable metric to measure this system recognition accuracy would be to measure the All-or-Nothing accuracy per drawing. All-or-Nothing accuracy is defined as the number of flowcharts in which all shapes (i.e., nothing was wrongly recognized in the entire sketch) were recognized properly on the first try over the total number of flowcharts. This metric reflects whether or not the user would have to go back to make any edits. This metric is important because ideally, the user wouldn't have to go back to the drawing to make edits. Table 5.11 shows the results for the first exercise; only 4 out of the 20 were recognized entirely on the first try, yielding an accuracy of 0.2 or 20%. For the second exercise, 5 out of 20 were recognized entirely, or .25 or 25%. For the last exercise 10 out of 20 were recognized entirely, or 50%. These results align with the feedback the user gave at the end of the experiment. Most of them felt lost when drawing the first flowchart, either because they didn't remember how to do it or because they had to keep track of the drawing guidelines. As the exercises continue the subjects mentioned feeling more confident when drawing them and that can be seen in the table below.

Table 5.11: All-or-nothing accuracy of each one of the three exercises

	All-or-nothing	
E01/Hello	4/20	0.2
E02/Fibonacci	5/20	0.25
E03/Factorial	10/20	0.5
		avg=.3166

Our smaller user study also draw some interesting results. It is worth to reiterate that this smaller user study was done to recreate a bad scenario in which the users do not have the instructor to clarify any doubts or answer questions. Just minimum instructions were provided and a simple exercises in which the users needed to draw a flowchart given a simple algorithm description.

All the sketches were collected and a confusion matrix was put together from the recognition results. The aggregated confusion matrix of all the 45 students from the smaller user study are shown in table 5.12.

Table 5.12: Confusion Matrix of unsupervised user study

UNSP		**PRE DICTED**						
		Ellipses	Parallelogram	Rectangle	Diamond	Arrow	None	
A	Ellipses	53	0	5	4	13	10	85
C	Parallelogram	1	49	6	2	5	5	68
T	Rectangle	0	0	74	2	4	3	83
U	Diamond	1	0	2	33	4	2	42
A	Arrow	0	0	0	0	216	64	280
L	None	0	0	0	2	97	0	99
		55	49	87	43	339	84	

The results for the ellipse were mostly similar than the complete study. Table 5.13 shows both the accuracy and precision stayed above 95%. However the F-score was lowered from 95% to 75%.

Table 5.13: Confusion Matrix and metrics of the shape Ellipse for the unsupervised study

ELLIPSE		**Predicted**	
		Ellipse	Not Ellipse
Actual	Ellipse	53	32
	Not Ellipse	2	604

Accuracy	0.9507
TPR	0.6235
FPR	0.0033
TNR	0.9966

FNR	0.3764
Precision	0.9636
Fscore	0.7571

The accuracy of the parallelogram is still considerably high with 97% as well as the precision with 100%. The FPR is low which indicates that there weren't many false positives overall. Although the FNR increased considerably. The F1 measure being a metric for overall goodness of the classifier dropped to 75%.

Table 5.14: Confusion Matrix and metrics of the shape Parallelogram for the unsupervised study

PRLGRAM	**Predicted**		
Actual		Parallelogram	Not Pararellogram
	Parallelogram	49	19
	Not Parallelogram	0	608

Accuracy	0.9718	FNR	0.2794
TPR	0.7205	Precision	1
FPR	0	Fscore	0.8376
TNR	1		

The accuracy of the rectangle is consistently high with 96% as well as the precision with 85% although less than previous study. The FPR and FNR are below 2% which indicates that there weren't many false positives or negatives overall. The F1 measure being a metric for overall goodness of the classifier dropped to 85% but is still considered a good rate.

Table 5.15: Confusion Matrix and metrics of the shape Rectangle for the unsupervised study

RECTNGL	**Predicted**		
Actual		Rectangle	Not Rectangle
	Rectangle	74	9
	Not Rectangle	13	583

Accuracy	0.9675	FNR	0.1084
TPR	0.8915	Precision	0.8505
FPR	0.0218	Fscore	0.8705
TNR	0.9781		

The accuracy of the diamonds remains considerably high with 97%. However the precision dropped to 76%. The FPR remains low but the FNR increased. The F1 measure being a metric for overall goodness of the classifier dropped to 77%.

Table 5.16: Confusion Matrix and metrics of the shape Diamond for the unsupervised study

DIAMOND		**Predicted**		Accuracy	0.9718	FNR	0.2142
Actual		Diamond	Not Diamond	TPR	0.7857	Precision	0.7674
	Diamond	33	9	FPR	0.0157	Fscore	0.7764
	Not Diamond	10	624	TNR	0.9842		

The diamonds evaluation metrics were the most affected during this user study. The accuracy of the diamonds dropped to 78% and the precision to 63%. Both FNR and FPR increased to 22% approximately. And the goodness of the classifier being the F-measure dropped to 69%.

Table 5.17: Confusion Matrix and metrics of the Arrows for the unsupervised study

ARROW		**Predicted**		Accuracy	0.7784	FNR	0.2285
Actual		Arrow	Not Arrow	TPR	0.7714	Precision	.6371
	Arrow	216	64	FPR	0.2180	Fscore	0.6978
	Not Arrow	123	441	TNR	0.7819		

Since the unsupervised user study was meant to be helpful to identify strengths and weaknesses on unsupervised flowcharts, we calculated a set of percentages that will be helpful to give a clearer idea why the evaluation metrics dropped overall. Since the drawing guidelines were omitted in the exercise the users incur in many different drawing characteristics that hindrance the recognition of our classifier. Here is a list of the percentages of these characteristics found on the 45 flowchart sketches on the unsupervised study.

The characteristics identified next we be used to understand why the evaluation metrics dropped.

- **All-or-nothing** - Percentage of how many flowcharts were completly recognized and were also correct,

- **LogicCorrect** - Percentage of how many subjects had the algorithm logic correctly .
- **LogicIncorrect** - Percentage of how many subjects had the algorithm logic incorrectly.
- **ClosedShapes** - Percentage of completely closed shapes or with just one corner unclosed .
- **Non-ClosedShapes** -Percentage of two or more corners unclosed.
- **Scribbles** - Percentage of how many sketches contained scribbles on them.
- **ArrowPartOverlap** - Percentage of arrows that had the shaft nad the head of the arrow joined.
- **ShapeArrowOverlap** - Percentage of collisions between arrows and shapes found on the sketch.
- **ShapeTextOverlap** - Percentage of collisions between text and shapes.
- **Overtracing** - Percentage of how many shapes and arrows were drawn with overlaping strokes.
- **Pencil** - Percentage of sketches done with a regular pencil.
- **MPencil** - Percentage of sketches done with a mechanical pencil.
- **MPencil** - Percentage of sketches done with a pen.
- **Erased** - Percentage of sketches that contained erasures.
- **ClearText** - Percentage of clearly recognizable text in shapes.

- **AmbiguousText** - Percentage of ambiguous text in shapes.
- **WrongShapeUsed** - Percentage of times in which subjects had used a unappropiated shaes to perform an action.

Table 5.18: Characteristics found on the unsupervised study

	Percentage%	Significance
All-or-nothing	15%	Higher is better
Correct	47%	Higher is better
Incorrect	53%	Lower is better
Closed Shapes	79%	-
Non-Closed Shapes	21%	-
Scribbles	26%	Lower is better
ArrowPartOverlap	86%	Lower is better
ShapeOverlap	22%	Lower is better
ShapeTextOverlap	6%	Lower is better
Overtracing	16%	Lower is better
Pencil	74%	-
MPencil	8%	-
Pen	18%	-
Erased	1.2	Average
ClearText	84%	Higher is better
AmbiguousText	16%	Lower is better
WrongShapeUsed	6%	Lower is better

As we can see many of these characteristics could of been voided if the complete set of drawing guidelines were provided. Or if the researcher answered all the doubles an questions. The ArrowPartOverlap were clearly high. However even though the guidelines ask for a separated shaft an arrow to use the system then the accuracy results were not much different. ShapeArrow overlap also caused recognition rate to drop as well as overtracing of a shape or arrow.

5.3.3 Qualitative analysis

To measure usability we did a post-experiment survey called System Usability Scale or SUS [103]. SUS was developed by John Brooke in 1986 and have proven to be more reliable than other questionnaires such as QUIS [106] and CSUQ [107] according to Tullis and Stetson [108]. Tullis and Stetson also proved that even a small sample of users (8-12) can be enough to give a good assessment of how people see your system or product.

SUS is technology independent and has been tested on hardware, consumer software, websites, cell-phones, IVRs and even the yellow pages. The questionnaire consists of 10 different questions about the usability of the system. To answer the questions the user needs to fill a Likert [109] type scale which range from 1-5 (Strongly disagree - Strongly agree). The questionnaire structure is organized in a way that for some questions (odd numbers) the maximum score is good and for the rest of them, the maximum is bad. This is because the user can just mark strongly agree on all of the questions, so that measure will require that the user reads the question before answering. It is important to mention that the overall SUS scores are not mean to be as a percentage measure of usability. Even though the scores range from 0 to 100, they just indicate the maximum score that the users gave to the system.

The average SUS score was 84.25 as shown in Table 5.19 which is usually seen as good. According to literature a score between 80 and 90 is considered to be an “excellent” measure for usability [110].

However, this is no perfect metric to evaluate the usability of the system. We were also interested in the feedback that the subjects might have after using the system. As we mentioned earlier we engaged in a semi-structured interview at the end of the experiment to collect this valuable feedback from the users.

We engaged in Grounded Theory[111] using the open coding strategy[112]. We began

Table 5.19: SUS table

SUS								S	U	B	J	E	C	T	S						
		S01	S02	S03	S04	S05	S06	S07	S08	S09	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20
Q	Q1	2	2	3	4	2	3	3	4	4	3	3	1	3	4	3	3	3	3	3	4
U	Q2	4	3	4	3	3	3	4	4	3	4	4	3	4	4	3	3	4	3	3	4
E	Q3	3	3	3	3	3	4	2	4	3	3	4	3	3	4	4	3	3	2	3	4
S	Q4	3	2	3	4	3	4	4	3	4	4	3	4	4	4	3	3	4	4	4	4
T	Q5	3	3	4	4	4	3	4	4	3	4	4	3	4	4	4	3	3	2	0	4
I	Q6	2	3	4	4	2	4	4	4	4	4	4	4	4	4	4	3	4	3	3	4
O	Q7	4	3	4	3	4	4	4	4	4	4	3	3	4	4	4	3	3	3	3	4
N	Q8	3	1	3	4	2	4	3	3	4	4	4	3	4	4	4	3	4	3	3	4
S	Q9	2	3	3	4	3	3	4	4	4	4	3	3	4	4	3	2	3	2	3	4
	Q10	4	2	4	4	4	4	4	4	3	3	3	4	3	3	4	3	3	3	0	4
		30	25	35	37	30	36	36	38	36	37	35	31	37	39	36	29	34	28	25	40
x2.5		75	62.5	87.5	92.5	75	90	90	95	90	92.5	87.5	77.5	92.5	97.5	90	72.5	85	70	62.5	100
																				avg	84.25

by transcribing all the responses and observation about the subjects and then identifying categories between those. Those categories were then merged into other categories until we came up with the following table 5.20. As you can appreciate we ended up with 9 different categories. We included also a few of the keywords and phrases from the responses so that we can have a clearer understanding of the user feedback.

Table 5.20 shows that the majority of the subjects liked or saw the potential of using the application. Not all of the users gave their opinion or were articulated enough but the following highlights a few positive and negative responses:

One of the subjects was a computer science graduate and was teaching a computer science course for undergraduates. Not only the SUS score he gave was really high but also talked about how this would be a great tool for their students to use and saw the idea as good overall.

Two students were also considered to be "beginners" in terms of programming, one being an undergrad and the other starting grad school but having a different major than Computer Science. Their feedback was mostly appreciated since they are the target of the whole system. They mentioned seeing potential when using the application to clean their own flowcharts and that the code execution was a major perk. One of them was really

Table 5.20: Open coding final categorization

Categories	Keywords/Phrases from Interview
Usability	Straightforward, great, I knew what every function was, easy to use, not complex to use at all.
User Experience	Great, similar to apps I've used before, learn it quickly, cool to try it out, recognition was good, enjoyed using it.
Difficulties	Hard time recognizing text, miss-recognized arrow, shapes need to be too aligned, arrow guidelines were annoying.
Educational	Good as educational tool, great for understanding loops, great to verify flowchart correctness, I see the potential.
Tool	Cool, useful, mobile, I would download it, great idea, visual approach tool, handy, practical.
Usefulness	Good idea for beginners, I would use this if I were an undergrad, I already know how to code so I wouldn't use this.
Suggestions	Beautification, help section in app needed, color coded editor, better label placement, different pencils.
Drawing	Confident drawing once the guidelines were learned, didn't really remember how to draw flowcharts, simple drawing directions.
Tedious	Guidelines were annoying at the beginning, type again the text is a pain, retype text is boring.

eager to use the system and mentioned that it is way easier to learn how to code using that approach since it is rather visual, and since they only needed a pencil, a white sheet and a smartphone the whole idea seemed practical.

On the negative side, some students even though they were familiarized with the flowchart concept they never took the flowchart-based approach when they were in undergrad. And were reluctant in seeing the actual potential of the system and the whole approach as not good and that the students should "jump right into coding" as they did. One thing that was made clear when the post interview took place is that international students were familiarized with the flowchart-based programming concept while American students were not.

Most of the subjects liked the idea of being able to code on paper and thought it was an interesting and useful idea for beginner programmers.

5.3.4 Discussion

The motivation behind this project was as mentioned in previous chapters to provide a reliable and usable tool for beginner programmers to translate flowchart drawings into code and be able to execute it to verify its correctness. In the previous chapters, we described the literature review, the process of the system, and its evaluation.

As described previously the recognition results of the system were good overall. The recognition accuracy of the shapes was above 90% and the feedback gathered and usability survey showed a good acceptance from the users overall. The weaknesses of the system can be lowered by implementing the drawing guidelines as appreciated in the unsupervised user study.

Originally the system was thought to allow free hand sketch recognition and let the user draw without restrictions, but in unit testing, before the user studies, we noticed that it would have been complicated and would of yield a low accuracy results. The guidelines provided helped overcome that issue, and even though they add an extra layer of complexity to the learning curve of the system we could see how the users didn't really spend a lot of time to learn them. The all-or-nothing accuracy was a clear indicator of this factor. The all-or-nothing accuracy improved from 20% in the first drawing to 25% on the second drawing and finally 50% in the third and last drawing.

The idea of being able to use paper and pencil to being able to develop and algorithm and verify its correctness by allowing its execution was one of the major advantages of using the system. However, after the user studies were done it was noticeable why most of the current research is leading towards online recognition techniques.

When using offline recognition techniques, the system relies heavily on the user's ability to draw and introduce variables that are not present in online recognition which leads usually to not so good recognition results. However, the advantage of using pen and pencil is still there and even though the whole process might be more troublesome we strongly believe that was worth exploring according to the user's feedback.

6. CONCLUSIONS AND FUTURE WORK

In this thesis, we designed and develop a system that takes a hand-drawn flowchart as input and translates it into code for its immediate execution that can be used to help beginners in programming to understand coding and algorithm design. Programming nowadays is being taught in the majority of engineering programs around the world and it's seen as a basic skill. Flowchart-based programming is a teaching approach widely used and takes the advantage of drawing visual diagrams to understand algorithm basic structures and operations. The correctness of a flowchart is only currently verified by desktop based systems and need that the users learn how to design the flowcharts on their system.

We designed and developed a novel flowchart-based programming application that helps the students to verify the correctness of a hand-drawn flowchart by converting the flowchart into code and allow its execution. From the data collected from 20 students, we perform a predictive analysis to verify the accuracy and overall correctness of the recognition. We also did a usability questionnaire and engaged in a post-interview to have a better understanding of the user experience. The results for the shape recognition were overall very good having an accuracy of above 95% and an average all-or-nothing accuracy of 31%. Few of the shapes were not recognized at all or miss-classified as a different shape although after the further analysis was not a surprise and the reasons were mostly because of either badly drawn shapes, overlapping or even too much brightness on the input image. The SUS usability questionnaire showed promising results scoring "excellent" according to literature and the user experience of the users were mostly regarded as great with no significant experience breakdown or usability hindrances.

In this section, we will also mention the work that can be done in the future to improve either the usability or the recognition rates of the system. The following future work was gathered mostly from the feedback of the users and the identifiable problems that the system had when the users interact with it.

One of the major hindrances of the system was the difficulty for recognizing text from inside the shapes. This was because at this time there is no out of the box best solution for handwriting recognition. There is the possibility though of improving over time the accuracy of the text recognition by using either a Support Vector Machine or a Neural Network [113]. However, this will add another layer of complexity to the learning curve of the system because the user would have to be involved in training the text recognizer. Since this is intended for beginner programmers we do not expect that the users will engage a huge amount of time using the system. That's why adding the need of training a handwritten recognizer contradicts the purpose of the system. However, adding this functionality and make it optional for the user would probably be the best choice.

Some of the feedback gathered from the users were more related to usability and user interface. A few of the users mentioned that the code editor should have colored keywords like on a desktop Integrated Development Environment (IDE). Highlighting keywords such as if, while, numbers/text will provide visual cues to the user and will also be useful for them to familiarize with the code so that when they use an IDE they can see the same colored mapped keywords.

A hindrance that was detected during the unit testing of the project and the user studies was that the exposure and brightness of the pictures sometimes led to non-recognized shapes. For example, if the user took a picture during the night, they will be prone to use a flash to get a clearer image however if the smartphone is too close from the actual drawing the brightness of the flash can be too intense and remove parts of the shapes or arrows. The dynamic thresholding we are using to delete the noise from the image then can be too aggressive if the exposure and brightness are too intense. A way to overcome this would be to develop our own camera module that takes into consideration the current light and keep the user away from the flash. Knowing the exposure and brightness level of the image could be useful also to adjust the dynamic threshold parameters resulting in a cleaner and more accurate recognition at the end of the process.

Feedback for flowchart correctness from the system is something that is not done explicitly by the system. So far the system provides a visual feedback [114] at the review module so that the user can see if what was or wasn't recognized. And also if the code at the code edit module is wrong and the user tries to execute it a javascript error message will prompt. However, that does not take into consideration of what would happen if the user draws an incorrect flowchart. It would be interesting to add the feedback functionality so that the user can realize that his/her drawing was not drawn correctly and why. This way the system would become more of a tool them to learn and not only a tool for the users to verify if their flowcharts are correct or not.

Another improvement that can be done to the system is to make most if not all of the thresholds used dynamic. Thresholds like, start to end distance of a stroke, stroke length, smoothness and others were used and adjusted to give the users some flexibility when the recognition takes place. This means that the user does not have to draw perfect shapes or arrows for the system to recognize them. However, over time the system could adjust this threshold depending on the user drawings so that it improves the recognition overall.

Another feature that was suggested was to include a beautification flowchart module like the one used in [115, 116, 117]. This module is already under development and will basically take the graph that represents the flowchart and draw it again on a digital canvas so that the alignment, shapes, and text are now perfectly even and properly distributed in the output flowchart and not only cleaned like it is currently.

Drawing prediction is something that might be useful as well to improve the user experience. For example, if the system does not recognize the first arrow it does not know where to go next and the code generation cannot go forward. This would be an example of how the prediction can improve the user experience. The system could ask the user if there should be an arrow between two shapes since there are none and the algorithm does not know what to do instead of asking for the user to redraw the missing arrow and take another picture.

Finally, we could make the whole system dual when it comes to online and offline recognition. By allowing the modification of the flowchart via drawing directly on the device. Erasing noise, correcting shapes, adding missed arrows electronically with one finger or with the use of a stylus directly on the smartphone of a tablet might be a good approach worth to look into. This could also enable the full online recognition if the flowchart and see the offline approach as more of an input option.

REFERENCES

[1] M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B.-D. Kolikant, C. Laxer, L. Thomas, I. Utting, and T. Wilusz, “A multi-national, multi-institutional study of assessment of programming skills of first-year cs students,” in *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, ITiCSE-WGR ’01, (New York, NY, USA), pp. 125–180, ACM, 2001.

[2] T. Crews and U. Ziegler, “The flowchart interpreter for introductory programming courses,” in *Frontiers in Education Conference, 1998. FIE ’98. 28th Annual*, vol. 1, pp. 307–312 vol.1, Nov 1998.

[3] K. Powers, P. Gross, S. Cooper, M. McNally, K. J. Goldman, V. Proulx, and M. Carlisle, “Tools for teaching introductory programming: What works?,” in *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE ’06, (New York, NY, USA), pp. 560–561, ACM, 2006.

[4] D. Hooshyar, R. B. Ahmad, M. H. N. M. Nasir, S. Shamshirband, and S.-J. Horng, “Flowchart-based programming environments for improving comprehension and problem-solving skill of novice programmers: A survey,” *Int. J. Adv. Intell. Paradigms*, vol. 7, pp. 24–56, July 2015.

[5] T. Sezgin, “Overview of recent work in pen-centric computing: Vision and research summary,” in *Workshop on Pen-Centric Computing Research*, 2007.

[6] J. Joseph, J. LaViola Jr, and C. Robert, “Mathpad2: A system for the creation and exploration of mathematical sketches,” *ACM Transactions on Graphics*, vol. 23, no. 3, pp. 432–440, 2004.

[7] T. Y. Ouyang and R. Davis, “Recognition of hand drawn chemical diagrams,” in *Proceedings of the 22Nd National Conference on Artificial Intelligence - Volume 1*, AAAI’07, pp. 846–851, AAAI Press, 2007.

[8] J. Lo, C. Torres, I. Yang, J. O’Leary, D. Kaufman, W. Li, M. Dontcheva, and E. Paulos, “Aesthetic electronics: Designing, sketching, and fabricating circuits through digital exploration,” in *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, UIST ’16, (New York, NY, USA), pp. 665–676, ACM, 2016.

[9] L. B. Kara, L. Gennari, and T. F. Stahovich, “A sketch-based tool for analyzing vibratory mechanical systems,” *Journal of Mechanical Design*, vol. 130, no. 10, p. 101101, 2008.

[10] T. Hammond and R. Davis, “Tahuti: A geometrical sketch recognition system for uml class diagrams,” in *Technical Report SS-02-08: Papers from the 2002 Association for the Advancement of Artificial Intelligence (AAAI) Spring Symposium on Sketch Understanding*, (Menlo Park, CA), AAAI, 7 2002. 8 pages.

[11] B. Buxton, *Sketching User Experiences: Getting the Design Right and the Right Design*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

[12] D. Roam, *The Back of the Napkin (Expanded Edition): Solving Problems and Selling Ideas with Pictures*. Penguin Publishing Group, 2009.

[13] P. J. Farrugia, J. C. Borg, K. P. Camilleri, and S. Christopher, “Experiments with a cameraphone-aided design (cpad) system,” *ICED 05: 15th International Conference on Engineering Design: Engineering Design and the Global Economy*, pp. [737]–[750], 2005. Peer reviewed.

[14] C. Y. Suen, M. Berthod, and S. Mori, “Automatic Recognition of Handprinted Characters - The State of the Art,” *Proceedings of the IEEE*, vol. 68(4), pp. 469–484, Apr. 1980.

[15] R. Plamondon and S. N. Srihari, “Online and off-line handwriting recognition: a comprehensive survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, pp. 63–84, Jan 2000.

[16] B. D. Eoff and T. Hammond, “Who dotted that ’i’?: Context free user differentiation through pressure and tilt pen data,” in *Proceedings of Graphics Interface 2009*, GI ’09, (Toronto, Ont., Canada, Canada), pp. 149–156, Canadian Information Processing Society, 2009.

[17] Y. Kato and M. Yasuhara, “Recovery of drawing order from single-stroke handwriting images,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, pp. 938–949, Sept. 2000.

[18] Y. Qiao and M. Yasuhara, “Recovering dynamic information from static handwritten images,” in *Ninth International Workshop on Frontiers in Handwriting Recognition*, pp. 118–123, Oct 2004.

[19] D. S. Doermann and A. Rosenfeld, “Recovery of temporal information from static images of handwriting,” in *Proceedings 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 162–168, Jun 1992.

[20] M. Notowidigdo and R. C. Miller, “Off-line sketch interpretation,” in *AAAI Fall Symposium. Menlo Park, CA*, pp. 120–126, AAAI Press, 2004.

[21] J. D. Foley, *Computer graphics: principles and practice*. Addison-Wesley Systems Programming Series, 1997.

[22] G. Sahoo and B. K. Singh, “A new approach to sketch recognition using heuristic,” *International Journal of Computer Science and Network Security*, pp. 102–108, 2008.

[23] V. NovÃąk, J. Mockor, and I. Perfilieva, *Mathematical principles of fuzzy logic*. Kluwer international series in engineering and computing science, Boston, MA: Kluwer, 1999.

[24] D. Delling, P. Sanders, D. Schultes, and D. Wagner, “Algorithmics of large and complex networks,” ch. Engineering Route Planning Algorithms, pp. 117–139, Berlin, Heidelberg: Springer-Verlag, 2009.

[25] E. J. Peterson, T. F. Stahovich, E. Doi, and C. Alvarado, “Grouping strokes into shapes in hand-drawn diagrams.,” in *AAAI* (M. Fox and D. Poole, eds.), AAAI Press, 2010.

[26] C. C. Aggarwal and C. K. Reddy, *Data Clustering: Algorithms and Applications*. Chapman & Hall/CRC, 1st ed., 2013.

[27] C. Reviews, *Finite Math and Applied Calculus: Mathematics, Mathematics*. Cram101, 2016.

[28] R. E. Schapire, “A brief introduction to boosting,” in *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI’99, (San Francisco, CA, USA), pp. 1401–1406, Morgan Kaufmann Publishers Inc., 1999.

[29] J. Wu, C. Wang, L. Zhang, and Y. Rui, “Offline sketch parsing via shapeness estimation.,” in *IJCAI* (Q. Yang and M. Wooldridge, eds.), pp. 1200–1207, AAAI Press, 2015.

[30] M. Bresler, D. Prù&#154;a, and V. Hlavác, “Modeling flowchart structure recognition as a max-sum problem,” in *Proceedings of the 2013 12th International Confer-*

ence on Document Analysis and Recognition, ICDAR '13, (Washington, DC, USA), pp. 1215–1219, IEEE Computer Society, 2013.

[31] T. Hammond and R. Davis, "Ladder: A language to describe drawing, display, and editing in sketch recognition," in *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, IJCAI'03, (San Francisco, CA, USA), pp. 461–467, Morgan Kaufmann Publishers Inc., 2003.

[32] T. Hammond and R. Davis, "Shady: A shape description debugger for use in sketch recognition," in *AAAI Fall Symposium on Making Pen-Based Interaction Intelligent and Natural (AAAI)*, (Arlington, VA), AAAI, 10 2004. 7 pages.

[33] T. Hammond and R. Davis, "Automatically transforming symbolic shape descriptions for use in sketch recognition," in *Proceedings of the 19th National Conference on Artifical Intelligence*, AAAI'04, pp. 450–456, AAAI Press, 2004.

[34] T. Hammond and R. Davis, "Ladder, a sketching language for user interface developers," *Computers & Graphics*, vol. 29, no. 4, pp. 518–532, 2005.

[35] T. Hammond and R. Davis, "Interactive learning of structural shape descriptions from automatically generated near-miss examples," in *Proceedings of the 11th International Conference on Intelligent User Interfaces*, IUI '06, (New York, NY, USA), pp. 210–217, ACM, 2006.

[36] B. Paulson and T. Hammond, "A system for recognizing and beautifying low-level sketch shapes using ndde and dcr," in *ACM Symposium on User Interface Software and Technology (UIST)*, (Newport Rhode Island), ACM, 10 2007. 2 pages.

[37] T. A. Hammond, *Ladder: A Perceptually-based Language to Simplify Sketch Recognition User Interface Development*. PhD thesis, Cambridge, MA, USA, 2007. AAI0818371.

[38] B. Paulson and T. Hammond, “Paleosketch: Accurate primitive sketch recognition and beautification,” in *Proceedings of the 13th International Conference on Intelligent User Interfaces*, IUI ’08, (New York, NY, USA), pp. 1–10, ACM, 2008.

[39] T. Hammond and R. Davis, “Creating the perception-based ladder sketch recognition language,” in *Proceedings of the 8th ACM Conference on Designing Interactive Systems*, DIS ’10, (New York, NY, USA), pp. 141–150, ACM, 2010.

[40] Z. Yuan, H. Pan, and L. Zhang, *A Novel Pen-Based Flowchart Recognition System for Programming Teaching*, pp. 55–64. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[41] B. D. Eoff and T. Hammond, “Who dotted that ’i’?: Context free user differentiation through pressure and tilt pen data,” in *Proceedings of Graphics Interface 2009*, GI ’09, (Toronto, Ont., Canada, Canada), pp. 149–156, Canadian Information Processing Society, 2009.

[42] B. Paulson, P. Rajan, P. Davalos, R. Gutierrez-Osuna, and T. Hammond, “What!?! no rubine features?: Using geometric-based features to produce normalized confidence values for sketch recognition,” in *HCC Workshop: Sketch Tools for Diagramming (VL/HCC)*, (Herrsching am Ammersee, Germany), pp. 57–63, VL/HCC, 9 2008.

[43] T. M. Sezgin and R. Davis, “Hmm-based efficient sketch recognition,” in *Proceedings of the 10th International Conference on Intelligent User Interfaces*, IUI ’05, (New York, NY, USA), pp. 281–283, ACM, 2005.

[44] A. Sears and J. Jacko, *Human-Computer Interaction: Designing for Diverse Users and Domains*. Human Factors and Ergonomics, CRC Press, 2009.

[45] B. A. Calloni and D. J. Bagert, “Iconic programming in baccii vs. textual programming: Which is a better learning environment?,” *SIGCSE Bull.*, vol. 26, pp. 188–192, Mar. 1994.

[46] M. C. Carlisle, T. A. Wilson, J. W. Humphries, and S. M. Hadfield, “Raptor: Introducing programming to non-majors with flowcharts,” *J. Comput. Sci. Coll.*, vol. 19, pp. 52–60, Apr. 2004.

[47] J. C. Giordano and M. Carlisle, “Toward a more effective visualization tool to teach novice programmers,” in *Proceedings of the 7th Conference on Information Technology Education*, SIGITE ’06, (New York, NY, USA), pp. 115–122, ACM, 2006.

[48] T. Watts, “The sfc editor a graphical tool for algorithm development,” *J. Comput. Sci. Coll.*, vol. 20, pp. 73–85, Dec. 2004.

[49] “Sourceforge devflowcharter.” `https://sourceforge.net/projects/devflowcharter/`. Accessed: 2016-11-23.

[50] “Visual Logic 2.2.10.” `http://www.visuallogic.org/`. Accessed: 2016-11-23.

[51] “Flowgorithm roberto atzori.” `http://flowgorithm.org/index.htm`. Accessed: 2016-11-23.

[52] “OpenCV | open source computer vision.” `http://opencv.org/`. Accessed: 2016-11-23.

[53] F. Calderon, J. Flores, and A. Garnica-Carrillo, “A fast algorithm for binary segmentation using color information,” in *2015 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC)*, pp. 1–7, Nov 2015.

[54] W. F. Abaya, J. Basa, M. Sy, A. C. Abad, and E. P. Dadios, “Low cost smart security camera with night vision capability using raspberry pi and opencv,” in *2014*

International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM), pp. 1–6, Nov 2014.

[55] T. Dai, Y. Dou, H. Tian, and Z. Huang, "The study of classifier detection time based on opencv," in *2012 Fifth International Symposium on Computational Intelligence and Design*, vol. 2, pp. 466–469, Oct 2012.

[56] S. Johnson, *Stephen Johnson on Digital Photography*. O'Reilly Media, Inc., 2006.

[57] G. Stockman and L. G. Shapiro, *Computer Vision*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1st ed., 2001.

[58] R. C. Gonzalez and R. E. Woods, *Digital Image Processing (3rd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006.

[59] M. Sezgin and B. Sankur, "Survey over image thresholding techniques and quantitative performance evaluation.," *J. Electronic Imaging*, vol. 13, no. 1, pp. 146–168, 2004.

[60] A. C. Bovik, *Handbook of Image and Video Processing (Communications, Networking and Multimedia)*. Orlando, FL, USA: Academic Press, Inc., 2005.

[61] N. Efford, *Digital image processing: a practical introduction using Java*. 2000. Includes CD-ROM.

[62] H. Tamura, "A comparison of line thinning algorithms from digital geometry viewpoint," in *ICPR*, pp. 715–719, 1978.

[63] Z. Guo and R. W. Hall, "Parallel thinning with two-subiteration algorithms," *Commun. ACM*, vol. 32, pp. 359–373, Mar. 1989.

[64] C. J. Hilditch, "Linear skeletons from square cupboards," *Machine Intelligence*, vol. 4, pp. 403–420, 1969.

[65] T. Y. Zhang and C. Y. Suen, "A fast parallel algorithm for thinning digital patterns.," *Commun. ACM*, vol. 27, no. 3, pp. 236–239, 1984.

[66] B.-K. Jang and R. T. Chin, "Analysis of thinning algorithms using mathematical morphology," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-12, pp. 541–551, June 1990.

[67] C. Arcelli, "Pattern thinning by contour tracing," *Computer Graphics Image Processing*, vol. 17, pp. 130–144, Oct. 1981.

[68] L. O'Gorman, "K x K thinning," *CVGIP: Image Understanding*, vol. 51, pp. 195–215, Aug. 1990.

[69] A. R. Widiarti, "Comparing hilditch, rosenfeld, zhang-suen,and nagendraprasad -wang-gupta thinning," *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, vol. 5, no. 6, pp. 563 – 567, 2011.

[70] A. McAndrew, *A Computational Introduction to Digital Image Processing, Second Edition*. CRC Press, 2015.

[71] P. Rajan and T. Hammond, "From Paper to Machine: Extracting Strokes from Images for use in Sketch Recognition," in *Eurographics Workshop on Sketch-Based Interfaces and Modeling* (C. Alvarado and M.-P. Cani, eds.), The Eurographics Association, 2008.

[72] P. Rajan, P. Taele, and T. Hammond, "Evaluation of paper-pen based sketching interface.," in *Proceedings of the 16th International Conference on Distributed Multimedia Systems (DMS)*, pp. 321–326, 2010.

[73] D. E. Knuth, *The Art of Computer Programming, Volume 3: (2Nd Ed.) Sorting and Searching*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1998.

[74] D. Rubine, "Specifying gestures by example," in *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '91, (New York, NY, USA), pp. 329–337, ACM, 1991.

[75] B. Paulson, P. Rajan, P. Davalos, R. Gutierrez-Osuna, and T. Hammond, "What!?! no rubine features?: Using geometric-based features to produce normalized confidence values for sketch recognition," in *HCC Workshop: Sketch Tools for Diagramming (VL/HCC)*, (Herrsching am Ammersee, Germany), p. 57âĂŤ63, VL/HCC, 9 2008.

[76] T. Hammond, *Sketch Recognition: Algorithms and Applications*. Cambridge University Press, 2018. draft from March 1, 2016, publication forthcoming.

[77] T. Hammond and B. Paulson, "Recognizing sketched multistroke primitives," *ACM Trans. Interact. Intell. Syst.*, vol. 1, pp. 4:1–4:34, Oct. 2011.

[78] A. Wolin, B. Eoff, and T. Hammond, "Shortstraw: A simple and effective corner finder for polylines," in *Proceedings of the Fifth Eurographics Conference on Sketch-Based Interfaces and Modeling*, SBM'08, (Aire-la-Ville, Switzerland, Switzerland), pp. 33–40, Eurographics Association, 2008.

[79] A. Wolin, M. Field, and T. Hammond, "Combining corners from multiple segmenters," in *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling*, SBIM '11, (New York, NY, USA), pp. 117–124, ACM, 2011.

[80] A. Wolin, B. Paulson, and T. Hammond, "Sort, merge, repeat: An algorithm for effectively finding corners in hand-sketched strokes," in *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, SBIM '09, (New York, NY, USA), pp. 93–99, ACM, 2009.

[81] A. Wolin, B. Paulson, and T. Hammond, “Eliminating false positives during corner finding by merging similar segments,” in *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, AAAI’08, pp. 1836–1837, AAAI Press, 2008.

[82] S. Arora and B. Barak, *Computational Complexity: A Modern Approach*. New York, NY, USA: Cambridge University Press, 1st ed., 2009.

[83] L. Wenyin, “On-line graphics recognition: State-of-the-art,” in *in GREC 2003: 5th IAPR International Workshop on Graphics Recognition, 2003*, pp. 291–304, Springer, 2003.

[84] T. Cover and P. Hart, “Nearest neighbor pattern classification,” *IEEE Trans. Inf. Theor.*, vol. 13, pp. 21–27, Sept. 2006.

[85] H. Bunke and P. Wang, *Handbook of Character Recognition and Document Image Analysis*. World Scientific, 1997.

[86] R. Smith, “An overview of the tesseract ocr engine,” in *Proc. Ninth Int. Conference on Document Analysis and Recognition (ICDAR)*, pp. 629–633, 2007.

[87] L. Rosen, *Open Source Licensing: Software Freedom and Intellectual Property Law*. Prentice Hall PTR, 2005.

[88] “Apache Software Foundation apache licence version 2.0.” `https://www.apache.org/licenses/LICENSE-2.0`.

[89] R. Neto and N. Fonseca, “Camera reading for blind people,” *Procedia Technology*, vol. 16, pp. 1200 – 1209, 2014.

[90] H. N. Do, M. T. Vo, B. Q. Vuong, H. T. Pham, A. H. Nguyen, and H. Q. Luong, “Automatic license plate recognition using mobile device,” in *2016 International*

Conference on Advanced Technologies for Communications (ATC), pp. 268–271, Oct 2016.

[91] K. C. Liu, C. H. Wu, S. Y. Tseng, and Y. T. Tsai, "Voice helper: A mobile assistive system for visually impaired persons," in *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, pp. 1400–1405, Oct 2015.

[92] "Javascript main website." `https://www.javascript.com/`. Accessed: 2016-11-23.

[93] "Rhino-Mozilla mdn." `https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino`. Accessed: 2016-11-23.

[94] "Mozilla Foundation." `https://www.mozilla.org/en-US/foundation/`. Accessed: 2016-11-23.

[95] M. Nakamura, Y. Fukuoka, H. Igaki, and K. i. Matsumoto, "Implementing multi-vendor home network system with vendor-neutral services and dynamic service binding," in *2008 IEEE International Conference on Services Computing*, vol. 2, pp. 275–282, July 2008.

[96] M. Lu, X. Sun, S. Wang, D. Lo, and Y. Duan, "Query expansion via wordnet for effective code search," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pp. 545–549, March 2015.

[97] M. Cova, C. Kruegel, and G. Vigna, "Detection and analysis of drive-by-download attacks and malicious javascript code," in *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, (New York, NY, USA), pp. 281–290, ACM, 2010.

[98] T. Pender, *UML Bible*. Bible, Wiley, 2003.

[99] O. M. Group, "OMG Unified Modeling Language (OMG UML), Infrastructure, V2.1.2," tech. rep., Nov. 2007.

[100] G. Inc., "Google material design." `https://material.google.com/`. Accessed: 2016-11-23.

[101] "Android API 5.0 codename lollipop." `https://developer.android.com/about/versions/android-5.0.html`. Accessed: 2016-11-23.

[102] "vpr.tamu.edu - Institutional Revision Board." Web, 2016.

[103] J. Brooke, "Sus: A quick and dirty usability scale," 1996.

[104] A. Galletta and W. Cross, *Mastering the Semi-Structured Interview and Beyond: From Research Design to Analysis and Publication*. Qualitative studies in psychology, NYU Press, 2013.

[105] C. Sammut and G. I. Webb, *Encyclopedia of Machine Learning*. Springer Publishing Company, Incorporated, 1st ed., 2011.

[106] J. P. Chin, V. A. Diehl, and K. L. Norman, "Development of an instrument measuring user satisfaction of the human-computer interface," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '88, (New York, NY, USA), pp. 213–218, ACM, 1988.

[107] J. R. Lewis, "Ibm computer usability satisfaction questionnaires: Psychometric evaluation and instructions for use," *Int. J. Hum.-Comput. Interact.*, vol. 7, pp. 57–78, Jan. 1995.

[108] T. S. Tullis and J. N. Stetson, "A comparison of questionnaires for assessing website usability.," in *Proceedings of UPA 2004*, June 2004.

[109] R. Likert, "A technique for the measurement of attitudes.," *Archives of Psychology*, vol. 22, no. 140, pp. 1–55, 1932.

[110] A. Bangor, P. Kortum, and J. Miller, "Determining what individual sus scores mean: Adding an adjective rating scale," *J. Usability Studies*, vol. 4, pp. 114–123, May 2009.

[111] A. L. Strauss and J. Corbin, *Basics of qualitative research : techniques and procedures for developing grounded theory*. Los Angeles, London, New Delhi: SAGE Publications, 2008.

[112] U. Flick, *An Introduction to Qualitative Research*. SAGE Publications, 2009.

[113] J. Feldman and R. Rojas, *Neural Networks: A Systematic Introduction*. Springer Berlin Heidelberg, 1996.

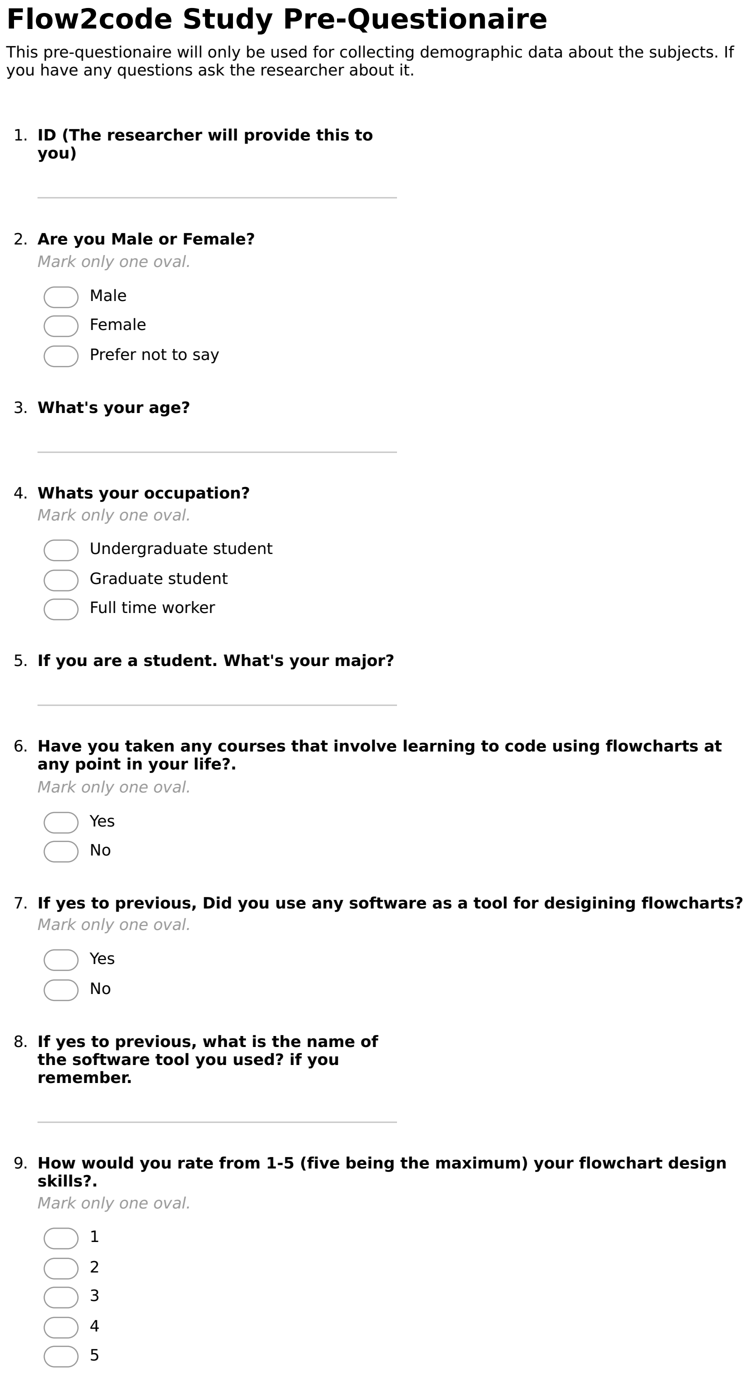
[114] *Interaction Design: Beyond Human-Computer Interaction, 2Nd Ed*. Wiley India Pvt. Limited, 2008.

[115] H. Miyao and R. Maruyama, "On-line handwritten flowchart recognition, beautification and editing system," in *2012 International Conference on Frontiers in Handwriting Recognition*, pp. 83–88, Sept 2012.

[116] M. Agrawal, A. Zotov, M. Ye, and S. Raghupathy, "Context aware on-line diagramming recognition," in *2010 12th International Conference on Frontiers in Handwriting Recognition*, pp. 682–687, Nov 2010.

[117] L. Julia and C. Faure, "Pattern recognition and beautification for a pen based interface," in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, vol. 1, pp. 58–63 vol.1, Aug 1995.

APPENDIX A



Flow2code Study Pre-Questionaire

This pre-questionaire will only be used for collecting demographic data about the subjects. If you have any questions ask the researcher about it.

1. **ID (The researcher will provide this to you)**

2. **Are you Male or Female?**
   Mark only one oval.
   ○ Male
   ○ Female
   ○ Prefer not to say

3. **What's your age?**

4. **Whats your occupation?**
   Mark only one oval.
   ○ Undergraduate student
   ○ Graduate student
   ○ Full time worker

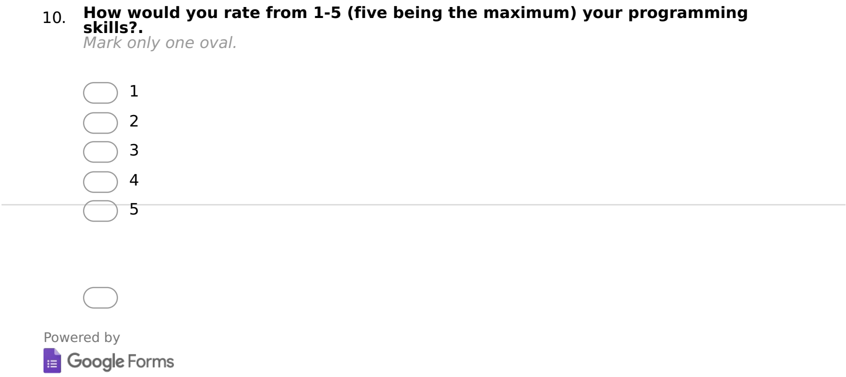
5. **If you are a student. What's your major?**

6. **Have you taken any courses that involve learning to code using flowcharts at any point in your life?.**
   Mark only one oval.
   ○ Yes
   ○ No

7. **If yes to previous, Did you use any software as a tool for desigining flowcharts?**
   Mark only one oval.
   ○ Yes
   ○ No

8. **If yes to previous, what is the name of the software tool you used? if you remember.**

9. **How would you rate from 1-5 (five being the maximum) your flowchart design skills?.**
   Mark only one oval.
   ○ 1
   ○ 2
   ○ 3
   ○ 4
   ○ 5



10. **How would you rate from 1-5 (five being the maximum) your programming skills?.**
Mark only one oval.

○ 1
○ 2
○ 3
○ 4
○ 5

○

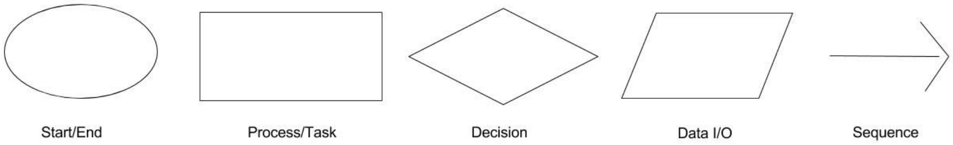
Powered by
Google Forms

APPENDIX B

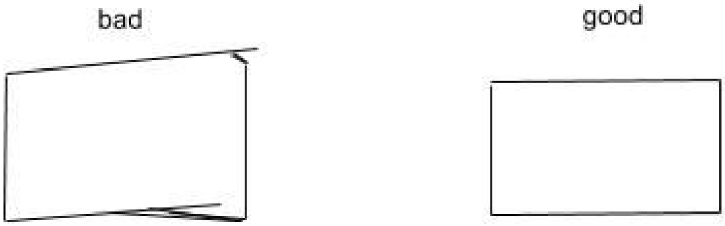
Flow2code Drawing Guideline

This page will illustrate recommended flowchart drawing tips that will improve the recognition accuracy when Flow2code is used. Try to use them all if possible.

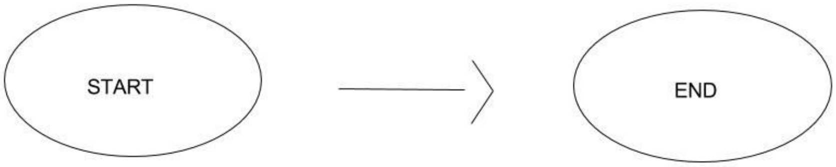
1-Flow2code only accepts the following shapes: **Ellipse, Rectangle, Diamond, Parallelogram and Arrow**. Please note the typo of arrow that is recognized by Flow2code, the shaft and the head should not touch each other.



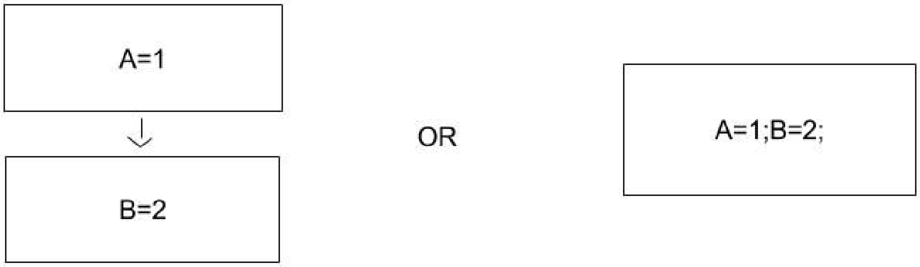
2-**Don't over trace** when drawing, you can use the eraser as much time as you want.



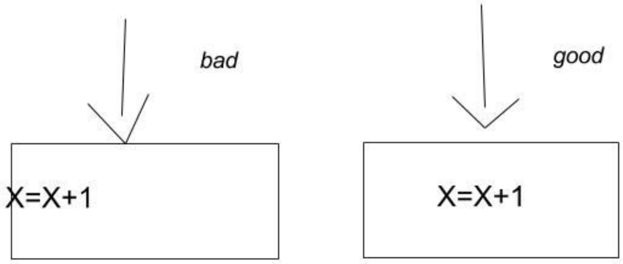
3-Remember that to begin the flowchart should have a **starting ellipse** with the tag START (Upper keys needed) and there should be an END **ellipse** as well to **terminate** the flowchart.



4-For processing steps (Rectangles), you don't need to add ";" at the end of each sentence if it is only one. If you are going to use more than one sentence on the same rectangle separate them using ";".

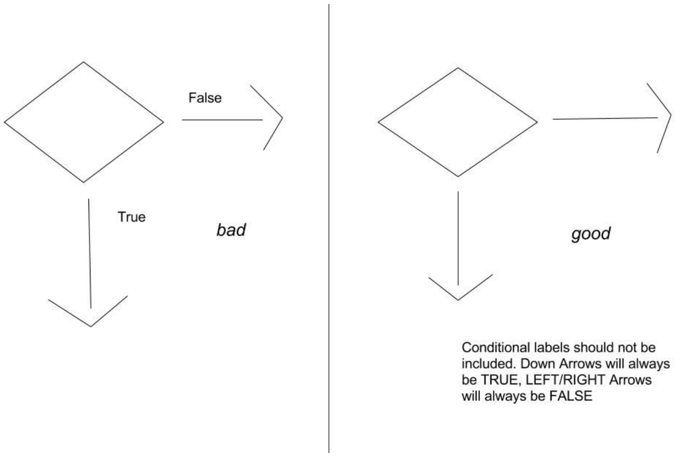


5-Flow2code recognition accuracy improve the more separated the shapes, arrows and text from inside the shape are from each other. It is not necessary to touch the shape with the arrow, the application will choose the closest shape to the arrow head orientation.



6-Remember to **write as clear as possible** inside the shapes to improve text recognition. You can use the pencil sharpener and eraser as much as you want.

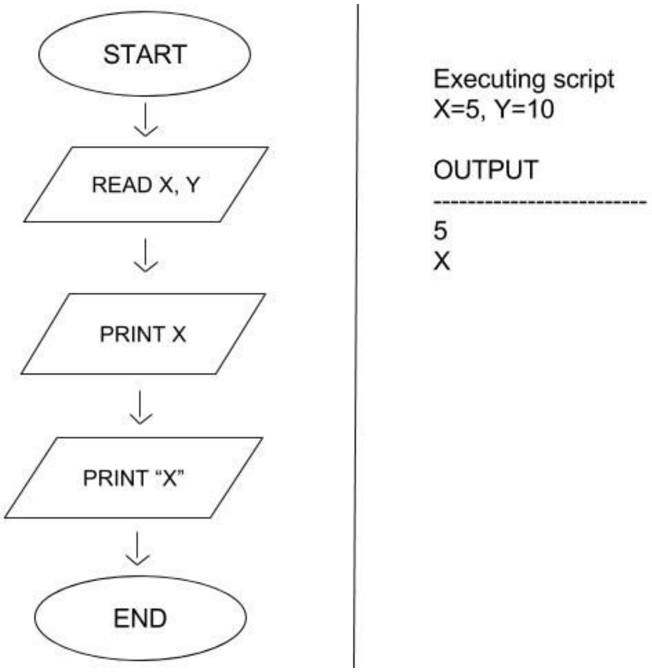
7-Flow2code do **NOT** uses FALSE/NO or TRUE/YES tags attached to decision arrows. Flow2code automatically detects the bottom arrow as TRUE/YES and the left or right arrow as FALSE/NO.



Conditional labels should not be included. Down Arrows will always be TRUE, LEFT/RIGHT Arrows will always be FALSE

8-Flow2code uses two special labels: **READ** and **PRINT** (Upper keys needed). Both should be declared inside **parallelograms** because they represent I/O of data.

The READ tag should be declared right after the START ellipse and can take either one variable or more. If more variables are needed, separate them with a comma. The PRINT function can receive either a single variable or a string using “variable”. The PRINT label can be used at any time between the START and END function.



```
Executing script
X=5, Y=10

OUTPUT
-------------------------
5
X
```

ID:

APPENDIX C

Flowchart Exercises

This part of the study will require for you to solve the following problems by drawing flowcharts. Use the Information sheet if needed to use the tips and recommendations given. You can ask any question to the researcher if you do not understand any of the exercises.

1-Draw a flowchart that prints the string "Hello" as much times as a given X variable but only if X is less than 10 if X is less than 10 just print once the word "Bye".

```
READ X
IF X<10
        WHILE X>0
        PRINT "Hello"
        X--
        End of WHILE
ELSE
PRINT "BYE"
End of IF
```

2- Draw a flowchart that prints that takes N as input, and print the first N fibonacci numbers. You can use the following pseudocode:

Fibonacci pseudocode

```
READ N
FT=0;ST=1;AX=0
WHILE ST<=N
        PRINT ST
        AX= ST
        ST= ST+FT
        FT=AX
End of WHILE
```

3-Draw a flowchart that prints the factorial number of a given number K. You can use the following pseudocode:

Factorial pseudocode

```
READ  W
M=1;F=1;
F=F*M
WHILE M!=W
        M=M+1
        F=F*M
End of WHILE
PRINT F
```

APPENDIX D

System Usability Scale

	Strongly disagree				Strongly agree
1. I think that I would like to use this system frequently	□ 1	□ 2	□ 3	□ 4	□ 5
2. I found the system unnecessarily complex	□ 1	□ 2	□ 3	□ 4	□ 5
3. I thought the system was easy to use	□ 1	□ 2	□ 3	□ 4	□ 5
4. I think that I would need the support of a technical person to be able to use this system	□ 1	□ 2	□ 3	□ 4	□ 5
5. I found the various functions in this system were well integrated	□ 1	□ 2	□ 3	□ 4	□ 5
6. I thought there was too much inconsistency in this system	□ 1	□ 2	□ 3	□ 4	□ 5
7. I would imagine that most people would learn to use this system very quickly	□ 1	□ 2	□ 3	□ 4	□ 5
8. I found the system very cumbersome to use	□ 1	□ 2	□ 3	□ 4	□ 5
9. I felt very confident using the system	□ 1	□ 2	□ 3	□ 4	□ 5
10. I needed to learn a lot of things before I could get going with this system	□ 1	□ 2	□ 3	□ 4	□ 5

APPENDIX E

Semi-Structured Interview Script

The following questions were asked to the subject after the experiment took place. Since it is a semi-structured interview, new questions arose during the interview process.

Do you found the application useful?

Do you see any advantages in using the application when learning how to code?

Did you found the system easy to use?

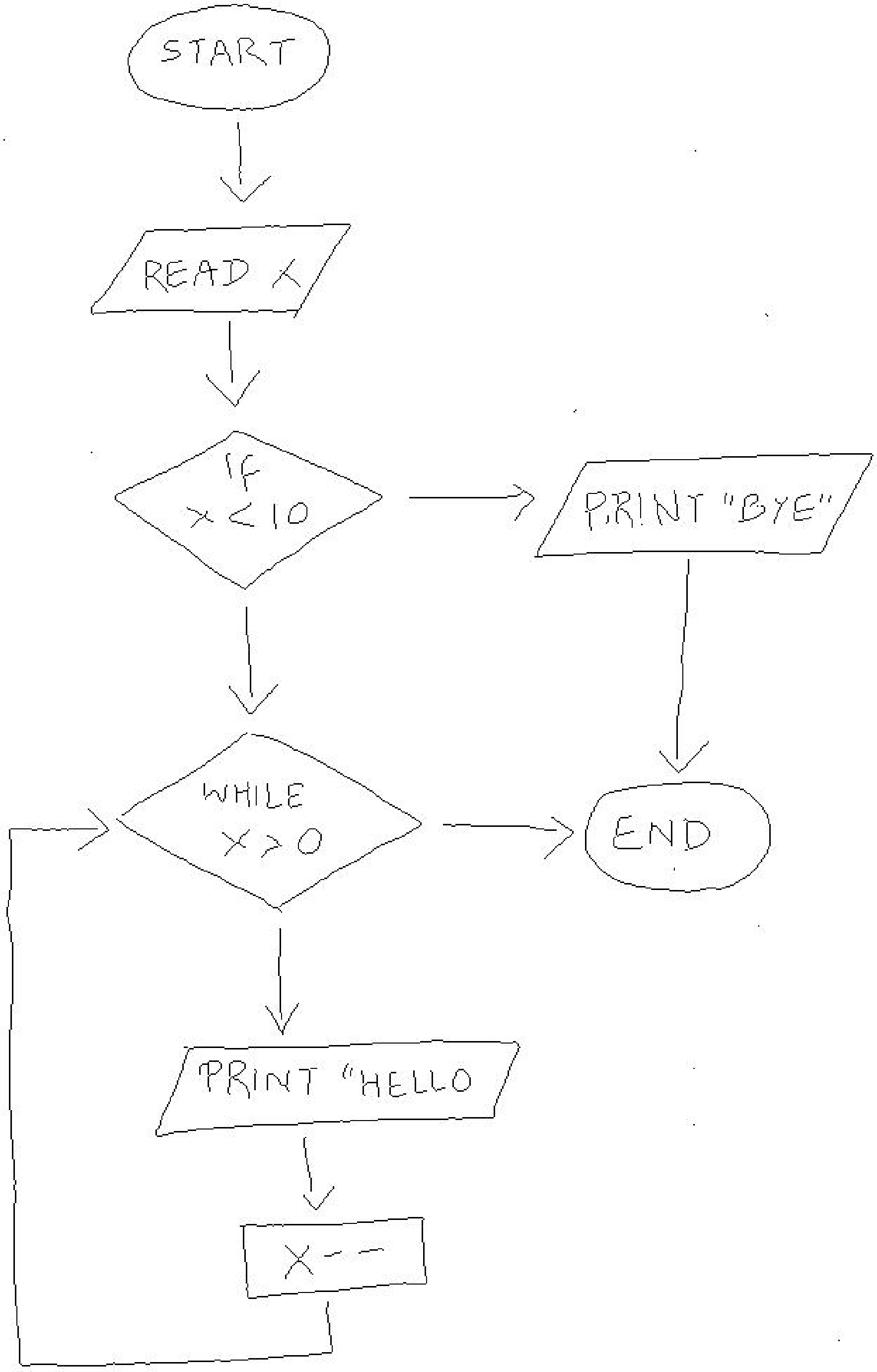
Did you have any troubles using the system?

Did you found the drawing guidelines to be too difficult to follow?

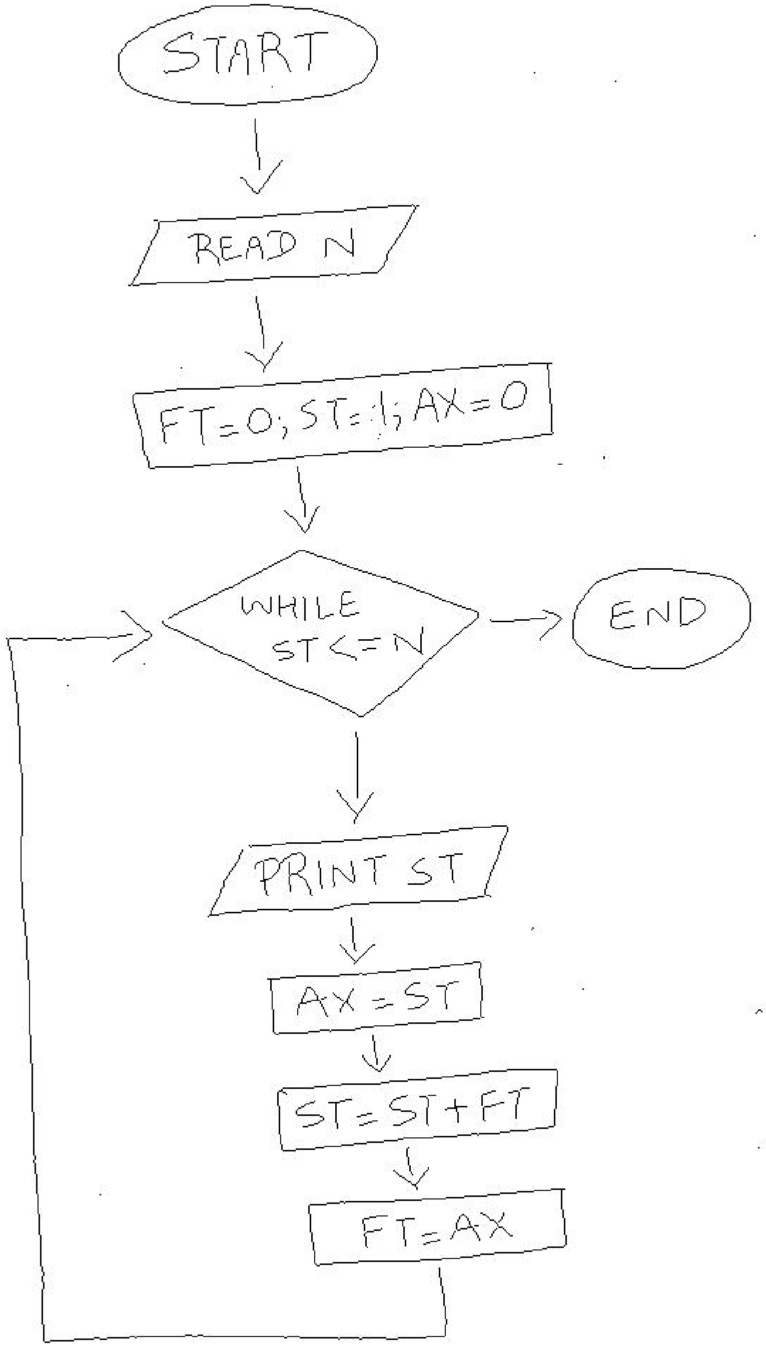
Is there anything that you disliked about the system itself?

Do you have any final comments or recommendations?

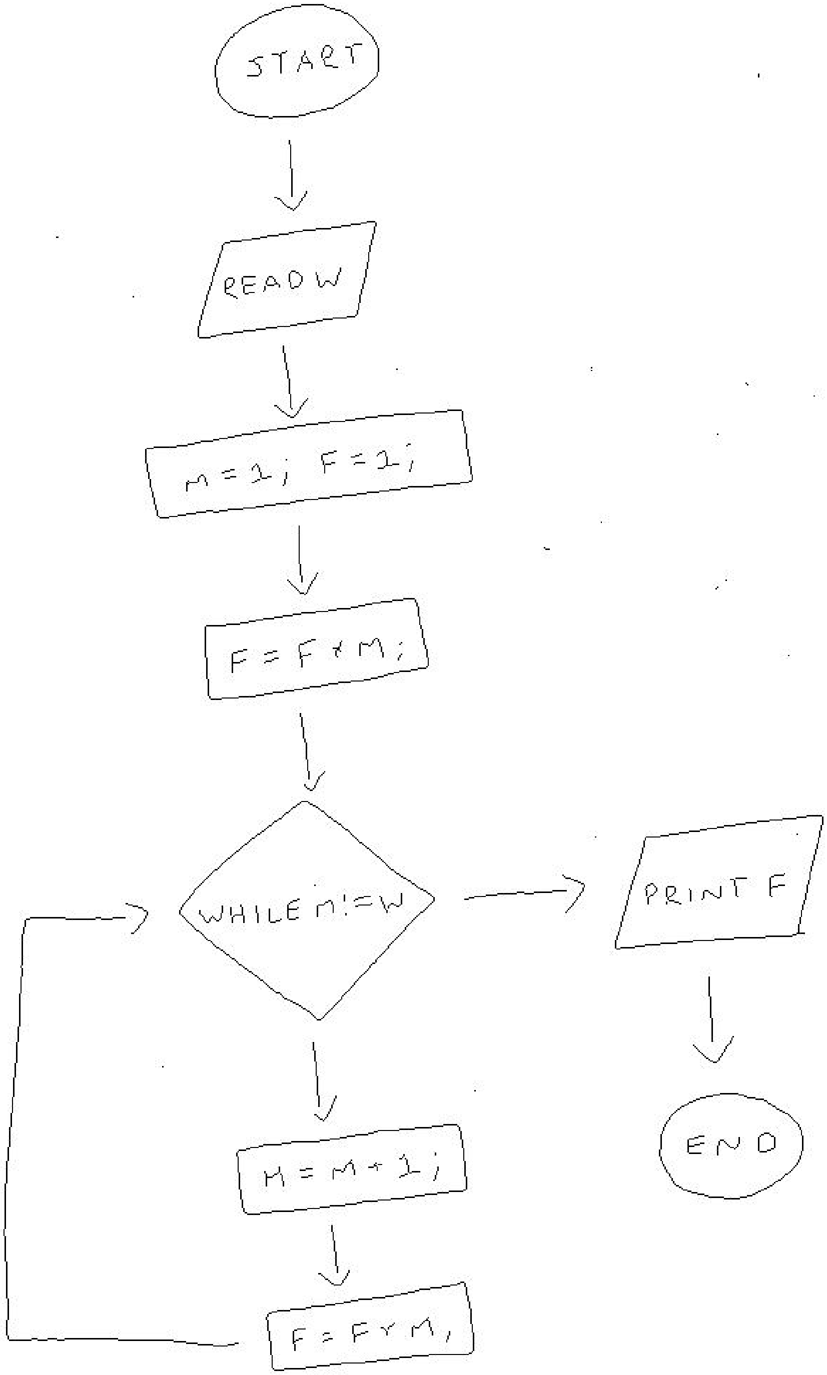
APPENDIX F



APPENDIX G


START
READ N
FT=0; ST=1; AX=0
WHILE ST<=N
END
PRINT ST
AX=ST
ST=ST+FT
FT=AX

APPENDIX H


START
READ W
m=1; F=1;
F=F*M;
WHILE m!=W
PRINT F
END
M=M+1;
F=F*M,