Transformation of sketchy UML Class Diagrams into formal PlantUML models

Monique Axt
Department of Communication, Quality Management and Information Systems
Mid Sweden University
Östersund, Sweden
mopa1801@student.miun.se

***Abstract*—Sketching software design models is a common and intuitive practice among software engineers. These informal sketches are transient in nature unless converted into formal models that can be reused and shared. Manual conversion, however, is time-consuming and redundant, and a method to automatically transform these sketches into permanent and formal software models is lacking. This study addresses this gap by creating and testing SketchToPlantUML, a sketch-recognition and transformation tool that reduces the effort of manually transforming static, sketched UML Class Diagrams (CDs) into formal models. The artefact uses the OpenCV library to preprocess images, segment UML elements, identify geometric features, classify relationships and transform the output into the equivalent, formal PlantUML model. Tested against a dataset of 70 sketched CDs, the artefact achieved overall Precision and Recall values of 88% and 86% respectively, scoring highest on classes (0.92 / 0.96) and lowest on association relationships (0.76 / 0.76). While the approach provides insight into image processing and object recognition using OpenCV, a more robust and generalised solution for automating the transformation of UML sketches into formal models is needed.**

***Index Terms*—UML, OpenCV, PlantUML, sketch-recognition**

I. Introduction

During early-stage design, developers often sketch to communicate and explore ideas. These designs are often visually represented using modelling languages that graphically model software systems, such as Unified Modelling Language (UML). A variety of traditional Computer Assisted Software Engineering (CASE) tools are available to create UML models digitally. The UML notation has a standardised set of symbols, rules and semantics that define relationships between components and how they should be represented, ensuring clarity and consistency. Consequently, UML CASE tools tend to enforce these rules, focusing on correctness and completeness. During early-stage design, however, sketching UML diagrams informally is more common [1].

The freedom from the rigidity and structure enforced by traditional CASE tools naturally allows for more open-ended exploration of ideas; sketching is faster, more flexible and can better support collaboration [2]. Recognising that sketching remains an integral and preferred method of developing and sharing ideas [3], sketching tools and environments have been developed to allow flexibility and informality when modelling designs. Such tools provide sketch-recognition capabilities, often in real-time, and beautify messy hand-drawn models.

However, these tools do not provide a means to transform sketches into formalised software models; the focus is placed on collaborative or real-time sketch recognition. During the transition from planning to implementation phases in Model-Based Software Development, informal models often need to be formalised. Additionally, the lack of focus on static images limits users on when and where design sketches can be made.

The benefits and ubiquity of informal sketching notwithstanding, formalisation aids in ensuring that the requirements and specifications of a software system are well-defined, unambiguous, and complete. Converting sketches to digital models, however, is a time- and labour-intensive process [4]. Combining the preference for sketching designs and the benefits of formalisation, automating the process from informal sketch to formal model is an important topic of research to support software development efficiency.

This study aims to address this problem by creating an approach that automatically transforms static UML sketches into formal software models. Publications by [4], [5] and [6] on the use of UML in software projects reveal Class Diagrams (CDs) as the most frequently used component and, based on these observations, this study is limited to converting sketchy CDs.

II. Purpose and Contributions

The initial design phase of the software development cycle is a crucial component of the development process and sketching is a fundamental activity during planning and communication. Despite modern tools simplifying the creation of diagrams, developers still feel that using these tools demands more time than they have available [6].

Many developers do not want to use formal tools or be constrained during early-stage design. Cherubini et al. [4] found limited adoption of formal UML among developers and that, antithetically, they often used rudimentary box-and-arrows diagrams instead. Related works have recognised the preference for sketching: e-whiteboards or similar computer-supported sketching environments have been developed, focusing either on informality (modelling language rules are enforced less strictly or not at all) or interpreting and beautifying (requiring more adherence to language rules for correct interpretation).

However, there is a lack of support for automating the actual static-sketch-to-software-model transformation and, con-

sequently, for improving the cost-benefit ratio of manual conversion. When designs are manually formalised, they are usually stored as static images [6]. Because many related tools focus on the collaborative aspect of early-stage design phases, there is little focus on interpreting static sketches. Moreover, analogue whiteboards are ubiquitous and economical. Conversely, the implementation and upkeep costs of replacing these with digital sketching environments are considerably high [4].

The goal of this study is to reduce the effort of manual static-sketch-to-software-model transformation by creating a tool that automates the conversion. The proposed tool aims to address the current gap in the field by focusing on static image input and the direct transformation of sketches into formal CASE software models. To achieve this objective, the problem of interpreting and classifying static images will be tackled using computer vision. The goal of this interdisciplinary field is to enable machines to perform tasks that usually require human visual perception. OpenCV (Open Source Computer Vision)[1] is a popular open-source library widely used for image recognition due to its numerous image-processing capabilities. The library has been successfully used by well-established companies for numerous detection and recognition applications.

RQ1 How can OpenCV be used to automatically interpret, classify and transform hand-drawn UML CDs into formal UML diagrams?

This research question aims to investigate the application of the library's computer vision techniques to the specific problem of sketchy diagram elements. Furthermore, the following sub-questions are explored:

1a) What are the key characteristics of sketchy UML CDs that need to be addressed by OpenCV?
1b) What are the potential benefits of using OpenCV?
1c) What are the limitations of this approach?

In addition to determining the advantages of using OpenCV for the specific task of this study and the limitations of the implemented approach, the objective is to acquire knowledge regarding the features of the sketched input that must be identified for effective analysis and manipulation.

RQ2 What is the effectiveness of the developed tool?

Interpreting sketched input poses a challenge in computer vision due to its inherent variability. The efficacy of the tool is expected to vary depending on both the condition of the original image and the quality of the preprocessing process. Precision and recall metrics will be used to evaluate the tool. Precision is the ratio of correctly identified UML CD elements to the total number of elements identified, whereas recall is the ratio of correctly identified elements to the total number of actual UML elements present in the image.

[1] opencv.org/about

The outlined requirements and subsequent contributions of this project are:

- A tool that provides direct transformation of UML sketches into a formal CASE tool model in the form of PlantUML output.
- A method to create an intermediate representation (IR) of the original sketched image that can be used as input for formalisation.
- Freedom from the constraint of having to use a tool while sketching by using static images as input.
- An approach that is easily reproducible by using open-source and freely available frameworks, i.e., without the use of proprietary software or specialised tools.
- Knowledge about how OpenCV can be used to recognise and classify UML class diagrams.

III. Background

This section describes the tools and terms that are relevant to the rest of the study.

A. OpenCV

OpenCV (Open Source Computer Vision) is an open-source computer vision and machine learning software library. The interdisciplinary field of computer vision deals with how computers can interpret, analyse and understand visual data, with the goal of enabling machines to perform tasks that usually require human visual perception, such as object recognition. OpenCV contains more than 2500 optimised algorithms for computer vision and machine learning applications.

B. PlantUML

Many tools are available that aid in creating formalised UML models and are categorised as drawing software or text-to-image software. An example of the former is Dia[2], a desktop application for drawing technical diagrams. PlantUML[3] is an example of the latter: an online tool that generates diagrams from simple and intuitive language. Formalising diagrams using text-based tools is generally simpler, faster and more portable than many dedicated diagramming applications.

C. Segmentation

Segmentation refers to dividing or extracting regions of interest (ROIs) in an image for further processing. In the context of this study, separating the different elements of a UML CD to recognise and classify them individually.

D. Morphological operations

Morphological operations refer to techniques that are used to analyse and manipulate the geometric structure of an image. A small shape called a structuring element (or kernel) is applied to each pixel in the original image to generate a corresponding pixel in the processed output. The resulting pixel value is determined by the specific morphological operation performed. Two common morphological operations

[2] dia-installer.de
[3] plantuml.com

are erosion, which shrinks object boundaries by removing pixels, and dilation, which expands object boundaries by adding pixels. These techniques can be used in preprocessing (to improve input for segmentation) and postprocessing (to remove imperfections in segmentation output).

E. Thresholding / Binarization

Thresholding is a fundamental method of segmenting images by converting greyscale or colour images into binary format. All pixels with intensity values above or below a specified threshold are turned on, while all other pixels are turned off. This is often used to simplify image processing tasks such as edge detection, morphological operations and object recognition.

F. Noise reduction

Noise reduction or denoising is an essential preprocessing step that removes unwanted pixels, or noise, from an image. Filtering algorithms alter pixel values based on their context, which informs the algorithm which pixels should be changed, and how. Commonly used algorithms include *Gaussian Blur*, which smooths pixels using a Gaussian function; *Bilateral Filter*, which smooths an image while preserving its edges; and *Median Blur*, which uses a sliding window to replace each pixel with the median value of its neighbouring pixels.

IV. Related Work

Automatic recognition of UML diagrams is of wide interest, and tools have been created or proposed to solve this. These can be separated into two categories: tools that expect precise computer-generated UML models as input; and tools that focus on sketched diagrams as input. This study is interested in the latter. The difficulties in reliably classifying sketched diagrams is still a challenging task for computers; hand-drawn shapes and text are inconsistent and imprecise, and recognising the meaningful patterns implied in sketches is not trivial [7].

A. Interactive recognition approaches

A common approach to the problem of sketch recognition is using stroke data. The tools discussed in this section use a combination of temporal and spatial information, i.e., when and where strokes are made, and in what order. To interpret strokes, these approaches create and use a digital platform that aims to mimic and retain the benefits of analogue whiteboard sketching while providing the advantages of digital modelling.

Hammond et al. [8] developed a tool that focuses on the importance of sketching, allowing users to sketch on a tablet or e-whiteboard. It uses a multi-layer framework with multi-stroke sketch recognition: line segments are processed and interpreted; strokes are selected based on spatial and temporal data; and using geometric properties, a collection of strokes is classified as one of seven supported UML objects.

SUMLOW [3] is another e-whiteboard system that concentrates on a minimally interruptive sketching experience. It addresses the lack of data persistency when using traditional whiteboards by featuring the preservation of hand-drawn diagrams. These can later be automatically beautified at the request of the user and, notably, exported to a traditional UML CASE tool.

Focusing on the collaborative aspect of early-stage design, OctoUML [9] is an exploratory and interactive environment that supports the means to create both sketched elements and formal notations simultaneously. It features several modes of interaction (voice commands, multi-touch, remote collaboration) and offers the ability to transform sketched designs into computer-drawn representations.

B. Offline recognition approaches

While e-whiteboards and digital platforms are useful mediums that facilitate collaboration, using static images as input provides flexibility while sketching and the benefit of transforming already-drawn sketches. Since real-time stroke data cannot be extracted from static images, a modified approach is required.

To address the challenges of offline sketch recognition, Bhasin et al. [10] present a preprocessing method for static hand-drawn images to improve pattern recognition and classification. While the research is limited to alphabetic characters, the techniques apply to sketched diagrams. The method's pipeline consists of cropping, noise reduction, binary thresholding, inversion, normalisation and thinning.

Deufemia and Risi [7] present a sketch-recognition system for multi-domain hand-drawn diagrams intended to work on both interactive and offline sketches. Similar to other online algorithms, their interactive sketch recognition takes advantage of the temporal and spatial data of strokes. To interpret static sketches, the method's pipeline is amended to include an image preprocessing step and recognition is performed without using stroke data.

The tool proposed in this study aims to integrate three key features not combined in related tools. Specifically, freedom from using a specialised platform, sketch recognition on static images, and direct transformation of sketches into a formal software model.

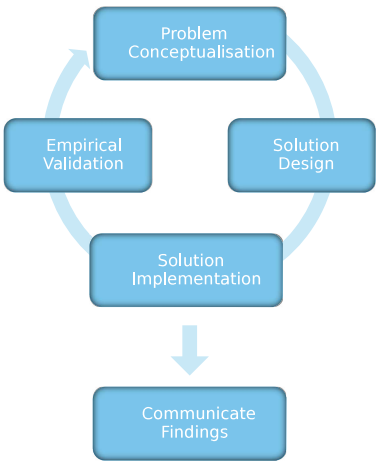


Fig. 1: The 5 steps of Design Science Research Methodology: Empowering iterative progress towards meaningful solutions and sharing the findings.

V. Research Methodology

This research aims to reduce the effort of manually transforming design sketches into formal diagrams by creating a tool that automates this process, thereby enhancing software development efficiency. The Design Science Research Methodology [11] is the design and investigation of artefacts in context and was used to pursue the goal of this study. The methodology's systematic and structured approach is well-suited for this study as it emphasises the creation of practical solutions to real-world problems. This paradigm involves a methodical process of identifying a problem, designing and implementing the solution, evaluating its effectiveness and accuracy, and communicating the findings (*see* Figure 1). This section details the design, implementation and evaluation method of the artefact.

A. Artifact

The created tool is developed using open-source software to make the approach reproducible, accessible and flexible. The OpenCV library is widely used for image recognition, is useable on almost any commercial system and supports interfaces for several popular programming languages. PlantUML is simple and portable: generated output can be saved and shared as plain text files. Both tools can be integrated with other platforms and libraries.

The code for the artefact can be found on GitHub[4] with installation and usage information in the README file[5]. The artefact recognises and classifies sketched UML CDs from static images and transforms the input into a formal PlantUML diagram. The process designed to achieve this involves five main steps: preparation, preprocessing, segmentation, classification and transformation.

1) Preparation

The scope of the artefact is limited to CDs, as [4], [5] and [6] cite these as the most frequently used diagram. Additionally, because CDs have several different geometric shapes but are still relatively simple[6] (*see* Figure 2), using these as input provides a reasonable range to test the capabilities of the OpenCV library within the time frame of this study. Similarly, given the time limitations and the primary emphasis on geometric object recognition, text recognition is excluded from the scope.

A dataset of 70 static images containing sketched UML CDs was selected to develop and evaluate the tool [12]. The dataset was created as part of a King's Undergraduate Research Fellowship (KURF) project[7] at King's College London [13]. The project uses machine learning to transform informal UML sketches into textual models. The dataset is a suitable match due to the similarity between the work done in the KURF project and the tool created in this study. Moreover, using an existing dataset reduces the effort of generating samples from scratch to evaluate an artefact. The selected dataset meets the key requirements of containing images and sketches of varying quality: images have bright, poor or non-uniform lighting, whereas the sketched diagrams differ in terms of stroke thickness, over-traced lines, and complete contours.

The focus of the study is on the geometric shapes that represent the components of a UML CD and text recognition is omitted. The relevant components are shown in Figure 2 (annotations in red) and are characterised as classes, inheritance relationships, and association relationships. Each component of the UML diagram is processed separately for analysis and recognition; several passes are made on variations of the original image to extract the geometric elements of each component. These elements are:

1) Quadrilaterals, representing classes
2) Closed-headed arrows, representing inheritance
3) Solid lines, representing association

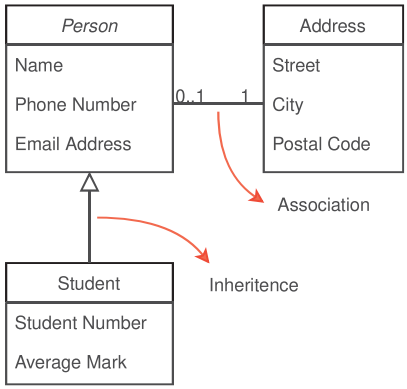


Fig. 2: A typical UML Class Diagram (annotations in red).

The identified constituents together with the differing qualities of the sketched dataset images are the characteristics of sketched UML CDs that need to be identified and addressed using OpenCV.

The final preparation step is template creation. Templates of arrows and asterisks are created based on the images in the dataset to identify these shapes using template matching. Due to the variability of sketched arrows, algorithms that rely on complete or perfect geometric properties to classify arrows do not provide high accuracy (*see* Section *V-A3 Segmentation*). In this implementation, all text (including UML multiplicity denotements) is treated as noise. Consequently, the asterisks templates are used to remove the matched symbols from the image.

2) Preprocessing

The goal of this step is to detect and store regions of interest (ROIs), reduce initial noise, binarize the image and prepare it for segmentation. The quality of the preprocessing greatly affects subsequent steps. Due to the scope of the study, text in the CDs is detected but not recognised. The detection is performed using a pre-trained `TextDetection` model using an implementation of the Efficient and Accurate Scene

[4] github.com/MoniqueAxt/SketchToPlantUML
[5] github.com/MoniqueAxt/SketchToPlantUML/blob/main/README.md
[6] Compared to, e.g., Component or Interaction Overview Diagrams.
[7] github.com/Nabboru/ModelSketch

Text Detector (EAST) algorithm[8]. As the model expects input image dimensions to be multiples of 32, images are resized and the resulting bounding box coordinates are adjusted to the corresponding regions on the original image.

Template matching is performed on the original image to identify and extract the arrow and asterisk ROIs. The former are later used for segmentation purposes. The image is then converted to greyscale, as morphological operations and edge detection algorithms typically expect this format. Before thresholding, the issue of non-uniform illumination in the dataset is addressed by removing the background using a light pattern. This process typically works well when external conditions are supervised and well-known (e.g., an empty background image is available). In this study, the light pattern is estimated by applying a blurring filter to the image with a large kernel size. As the illumination changes in the dataset are multiplicative (non-uniform scaling of brightness), the greyscaled image is then divided by the approximated light pattern.

The resulting image is then binarized and inverted, using Ostu thresholding. This method separates the image into a foreground or background class by finding an optimal threshold value. The algorithm maximises the between-class variance of the image, defined as the sum of the foreground and background variances, weighted by their respective probabilities. In practice, this automates the determination of an optimal threshold value used to reduce noise and enhance contours of interest. Different types of binary transformations are available in OpenCV and the appropriate method to select depends on the input and desired outcome.

Finally, the image undergoes final preprocessing before segmentation. This involves further noise reduction using morphological operations, skeletonization and the closing of contour gaps. The accuracy of the segmentation process relies on closed contours to detect the quadrilaterals that represent UML classes, determine the direction of arrows and isolate the lines representing relationships between classes. Consequently, gap closing is a crucial step. To achieve this, kernels are created and used in Hit-Or-Miss morphological operations to join extreme pixels within a specified proximity. Defined as points connected to only one of their surrounding neighbours, the eight variations of contour-terminating pixels, illustrated in Figure 3, are used by the Hit-Or-Miss operations to detect these particular binary patterns of foreground and background pixels. Once found, the points are joined by drawing a line between them.

3) Segmentation

After preprocessing, the diagram undergoes segmentation and the separated elements are placed into lists. The output of this step is three lists containing the relevant pixel locations representing quadrilaterals, arrows and association lines.

[8]doi.org/10.48550/arXiv.1704.03155

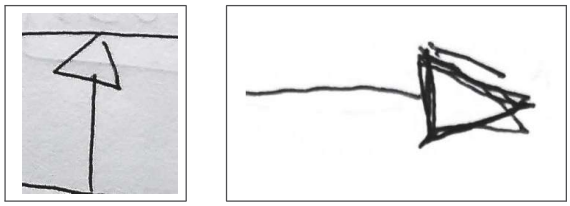
$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Fig. 3: Contour-terminating pixel variations: Visualizing the 8 kernel matrices representing possible 'loose end' pixel configurations in contours. Foreground pixels are denoted as 1, background pixels as 0.



(a) An arrow missing explicit endpoints

(b) Overdrawn lines creates an imprecise arrowhead

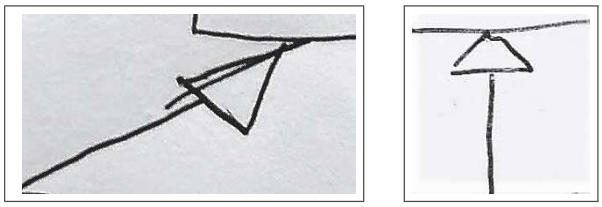
Fig. 4: Identifying sketchy input: a simple challenge for humans, a complex puzzle for computers.

Arrows

Arrows are isolated first to reduce misclassifications due to potential processing distortions later in the segmentation process. Using the ROI bounding boxes, arrows are extracted and the arrow-head and -shaft are segmented. The key points of both are determined using `goodFeaturesToTrack` and the pair of points furthest away from each other are marked as the arrow endpoints. The tip is classified as the point of the pair that is present in the isolated arrowhead, providing the direction of the arrow. This approach proved more accurate than using the traditional geometric properties of an arrow. As an example, the convex hull of an arrow will optimally contain two fewer points (the concave intersection of the shaft and arrowhead) than the approximate contour points. Using this method, however, would immediately misclassify a sketched arrowhead that is an imperfect triangle (e.g. more than or less than three corners). Figures 4 and 5 illustrate this computer vision problem of object classification that is trivial to solve using human perception.

Classes

Quadrilaterals representing classes are isolated by using the segmentation technique of watershedding. The `watershed`



(a) Connected components create non-standardised shapes

(b) Gaps and curved strokes are common in sketches

Fig. 5: Human perception is less reliant than computers on precise and structured data to interpret meaning.

algorithm separates an image into distinct regions, or segments, by treating the image as a topographic map, where the intensity values of the image are equivalent to the heights of the terrain. The map is then flooded from the lowest points, specified by markers, with the water level rising at each step until it reaches the highest point. This creates boundaries between, and thus segments, the different regions in the image.

To prepare the image for the algorithm, the contours in the image are filled and `distanceTransform` is applied, resulting in the value of each pixel being replaced by its distance to the nearest background pixel. The thresholded result of the transform marks the pixels within the filled contours as the foreground. Similarly, the image is dilated to obtain background pixels, and the differences between the foreground and background pixels are marked as unclassified. Unique labels are then applied to each region using the `connectedComponents` function, generating the markers needed for the `watershed` algorithm. To deal with quadrilaterals that extend beyond the image border and thus do not have a boundary marked, a white border is drawn around the initially skeletonised image, effectively creating a boundary, and this is used as the input image for watershedding.

Finally, UML classes are isolated by first removing potentially included arrowheads using the template ROIs and then extracting the external quadrilateral contours. Doing the former step before watershedding results in potential distortion of quadrilateral contours and less effective removal of these contours when isolating and detecting the lines representing relationships. The latter step is required because the watershed markers include the background contour and both internal and external contours of shapes. While visually the quadrilateral will appear isolated, the precise location and count of the contours are important for later use in the Classification step (Section *V-A4*).

Lines

Masking the resultant quadrilateral image with the preprocessed, skeletonised image isolates the lines and potential arrows. This image subsequently undergoes denoising to isolate only lines of interest. Firstly, arrows and text regions are removed. Secondly, noise due to gap closure and imperfect text and arrow removal is mitigated. Small lines below a threshold are removed using `connectedComponents`, while morphological closing, contour filling, morphological opening, masking, thinning and gap closing are used to remove larger residual noise while preserving lines. Each line's contours are retrieved and stored. While OpenCV offers several algorithms to identify lines and their features, sketched lines are often imperfect and lack the properties to be classified as a line, e.g., linearity, constant direction, collinearity or parallelism.

4) Classification

The classification step receives the segmented elements and creates an IR in the form of a tuple representing a relationship, i.e., two classes and the relevant relationship element.

The classification of relationships is based on the location of elements relative to each other and is highly dependent on the quality of segmentation and denoising in previous steps. Inheritance relationships are classified first to remove lines that are part of a forked arrow. This simplifies the subsequent classification of association relationships, as once all inheritance relationships have been determined, all remaining lines are considered associative.

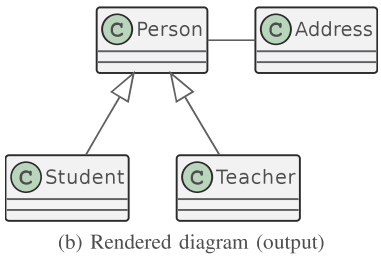
If arrows are present, the class closest to the arrow's tip is classified as a parent and, if the arrow is not forked (i.e., single-child scenarios), the child located on or near the arrow's shaft endpoint is found. Proximity is determined with the `pointPolygonTest` function. To determine whether an arrow represents a multi-child relationship, the locations of all detected lines relative to the arrow are calculated. If a line is within a specified proximity, it is identified as an extension of the forked arrow and all classes located on or near the line's endpoints are found, using the Hit-Or-Miss method discussed in Section *V-A2 Preprocessing*. The relationship between the relevant classes is then created and the line is removed from the list. Any remaining lines are categorised as associative and relationships are created based on the classes closest to its two endpoints, within a specific distance threshold.

5) Transformation

Text recognition is outside the scope of this study, and consequently, to distinguish between different classes in the textual output, short hashes based on the related components are created and used as class names. Each relationship is then generated using the relevant PlantUML syntax.

```
@startuml
Person <|-- Student
Person <|-- Teacher
Person - Address
@enduml
```

(a) Syntax (input)



(b) Rendered diagram (output)

Fig. 6: From text to graphic: The PlantUML syntax for each diagram is generated based on the identified relationships.

Associative relationships are denoted using a dash, whereas base/inherited relationships are expressed using a less-than sign, vertical bar and dash. The input syntax and resultant model are shown in Figure 6.

B. Evaluation

The objective of the evaluation is to assess the effectiveness of the developed tool. A systematic approach, informed by the Design Science Research Methodology, is conducted to determine whether the artefact meets the intended objectives (demonstrates utility) and furthers knowledge in the field (provides contribution). The process involves five key steps: identifying the evaluation criteria, selecting an appropriate evaluation method, conducting the evaluation, analysing the results, and drawing conclusions based on the findings. This section details the design of this process.

To assess the effectiveness of the tool, precision and recall metrics are used as evaluation criteria, in line with the evaluation method used in Deufemia and Risi's paper [7] discussed in Section *IV Related Work*. As opposed to using only accuracy, i.e., the overall correctness of predictions made by a model, precision and recall provides a more nuanced and comprehensive evaluation of effectiveness. Precision measures the ability of a model to identify only the relevant data points, defined as the proportion of true positives (TPs) to all positive predictions, including false positives (FPs). High precision, for example, indicates that when a positive prediction is made, it is likely to be correct, which is important when false positives are costly. Recall measures the ability of a model to find all the relevant data within a data set, defined as the ratio of true positives to all positive predictions, including false negatives (FNs). Also referred to as the sensitivity rate, high recall is crucial when minimising false negatives is a priority.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives}$$

Both measures are important regarding the aim of this research; resulting transformations should be accurate in identifying components, but a high amount of false positive predictions are also contrary to the aim. The use of these metrics provides an objective means to identify effectiveness as well as gaps in the model and, consequently, the viability of the approach used to address the issue of manual transformation.

Quantitative analysis is selected as the evaluation method. This method involves the use of numerical data which can be objectively measured and analysed, making it well-suited for evaluating the identified criteria.

The performance evaluation is conducted by manually comparing the sketched UML diagrams with the PlantUML models produced by the artefact. With the dataset diagrams as a benchmark, the comparison is used to measure the precision and recall of the tool, identifying UML elements that are misclassified, undetected or correctly identified.

To determine the contribution, utility and limitations of the artefact, the research questions are addressed, the data collected in the evaluation are analysed and the results of the tool's performance are indicated by the evaluation metrics. These are presented in Section *VI Results* and conclusions drawn based on the results are presented in Section *VII Discussion*.

VI. Results

The goal of this study is to reduce the effort of manually transforming sketchy UML CDs into formal PlantUML models by creating an artefact that automates this process using the OpenCV image processing library. This section details the identified characteristics of sketched UML diagrams and the artefact's performance results. Comparative analysis, limitations and validity are presented in Section *VII Discussion*.

To evaluate the effectiveness of the tool, precision and recall metrics were used. Performance was individually evaluated for each group of UML elements (i.e., classes, inheritance relationships and association relationships) and combined to provide an overall measure. Table I displays the raw numbers regarding correct classifications (True Positives), misclassifications (False Positives) and missed classifications (False Negatives). The corresponding precision and recall metrics are shown in Table II.

A. RQ1 a) What are the key characteristics of sketchy UML CDs that need to be addressed by OpenCV?

The automatic recognition of sketched input poses a significant challenge compared to human perception due to the inherent variability present. Consequently, the identification of features within sketches plays a crucial role in enabling effective detection and classification. The following section lists identified characteristics accompanied by a brief explanation.

Irregular lines Hand-drawn sketches exhibit lines that are curved and retraced, with varying stroke weights (*see* Figure 7). These features can result in regions of the line having varying contrasts, affecting which pixels are retained during binarization. Additionally, generalised morphological operations have different effects on thick

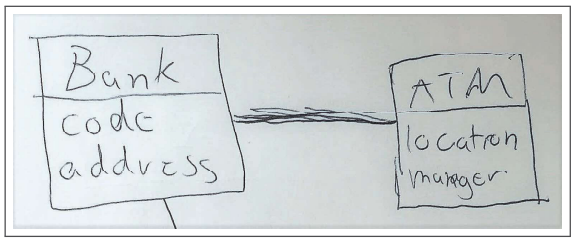
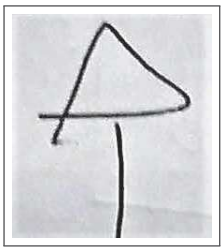
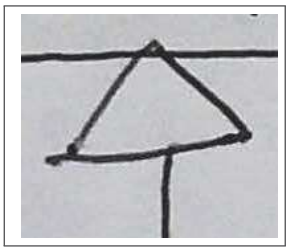


Fig. 7: Irregular lines: imperfect rectangles and the unconventional connection between classes in sketched UML Class Diagrams.



(a) Overextended lines



(b) Intersecting boundaries

Fig. 8: Characteristic features of sketching.

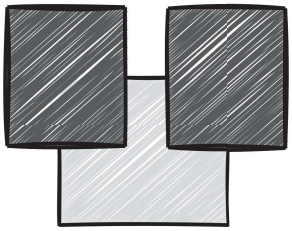


Fig. 9: Association lines forming a closed rectangle (light grey) are misclassified as UML classes (dark grey).

versus thin strokes. Curved lines also pose a challenge in shape classification: using strict geometric properties, a sketchy rectangle would be misclassified as it is considered to have more than four vertices, i.e. straight, connected lines.

Intersecting elements Effectively separating objects of interest for further analysis is complicated by intersecting contours common in freehand drawing, which can obscure the intended structure of the diagram (*see* Figure 8b and 9). This is particularly common when sketching relationships between classes using lines and arrows in UML diagrams. This can distort the geometric properties of individual elements and impact segmentation.

Line overextension Similarly, geometric distortions occur due to extensions of lines beyond intended boundaries, illustrated in 8a. This characteristic of sketching also obfuscates distinctions between meaningful and irrelevant contours. Association lines originating from class boxes, for example, demonstrate qualities analogous to overextended class lines, introducing the possibility of misclassifying elongated contours as a relationship.

Failure of co-termination Gaps in contours affect the successful extraction of relevant data. Many processing techniques rely on correct co-termination for precise results. Edge- and contour-detection, for example, will map disjointed contours as separate objects. Classification methods such as using the convex hull to identify an arrow result in FNs if strokes do not terminate at their endpoints. Segmentation methods like watershedding may under- or over-segment due to failures of co-termination.

Varying illumination In contrast with precise computer-generated diagrams, sketched input contains inconsistent lighting conditions, shading or shadows. This affects the quality of the image and the extraction of relevant details, primarily in thresholding processes, which are affected by uneven contrast, pixel intensity variations and illumination noise.

B. RQ1 b) What are the potential benefits of using OpenCV?

The OpenCV library is widely adopted, open-source and freely available under the BSD family of permissive free software licenses[9], making it accessible and cost-effective. Artefacts created using OpenCV are consequently reproducible. The library is platform-independent, supports multiple languages (C++, Python, Java, MATLAB) and integrates with other tools and libraries (such as NumPy, SciPy and Tensorflow), expanding the capabilities and possibilities for image processing and machine learning.

OpenCV offers a set of tools for both basic and advanced computer vision tasks, allowing developers to implement complex vision systems. The library is developed by a community of researchers, engineers and contributors who actively incorporate computer vision and image processing algorithms that have been validated and published in scientific papers. Moreover, OpenCV Extra Modules[10] contains algorithms that are not as extensively tested and available as an optional addition.

During the development of the artefact, several techniques were implemented and tested during each of the Research Methodology stages (Section *V*). Various methods exist to achieve a goal depending on the desired outcome, resulting in the ability to customise OpenCV's image processing to specific tasks. For example, the library offers multiple algorithms to perform noise reduction via blurring and thresholding. Combining or using several methods at different stages of the preprocessing pipeline is beneficial. In this study, `medianBlur` and `threshold` were used for illumination correction, while `GaussianBlur` and `THRESH_OTSU` were used to remove noise.

C. RQ2 What is the effectiveness of the developed tool?

Inheritance relationships represented by closed-headed arrows achieved high precision and recall. All three FN and FP predictions occurred due to imprecise templates and consequent misclassification of arrow-tip and -shaft endpoints, resulting in incorrectly identifying the roles of the relevant classes in the relationships.

Correct identification of classes relies on binarized images being denoised and the complete closure of gaps in the quadrilaterals. Almost 42% of faulty class predictions were caused by inadequate thresholding: imperfect correction of

[9] opencv.org/license

[10] github.com/opencv/opencv_contrib

UML element	*Qty*	*TP*	*FP*	*FN*
Classes	212	196	8	16
Inheritances	37	34	3	3
Associations	139	105	34	34
Total	**388**	**335**	**45**	**53**

TABLE I: Classification Results of UML Elements.
TP = true positives, FP = false positives, FN = false negatives.

UML element	*Precision*	*Recall*
Classes	0.92	0.96
Inheritances	0.92	0.92
Associations	0.76	0.76
Total	**0.86**	**0.88**

TABLE II: Precision and Recall Evaluation Metrics for UML Element Classification.

non-uniform background illumination, causing residual noise to be classified as a class, resulted in FPs, while FN predictions were due to contour degradation during binarization, resulting in gap distances that were above the threshold for closure. The approach used to segment classes by filling closed contours, however, accounted for 13 of the 16 FNs and 54.2% of all inaccurate classifications of this element, as the artefact does not distinguish between closed forms that represent classes and those that represent association relationships (*see* Figure 9).

The successful recall of classes also impacted the classification of relationships as, without two relevant classes, relationships cannot be formed. Only association relationships in the dataset were impacted by this, causing 39.7% of all incorrect predictions in this group. The majority of FN and FP predictions resulted from imprecise text detection and removal, which was responsible for 54.4% of incorrect association predictions. Notably, all association identification errors were due to imperfect processing in previous steps.

VII. Discussion

In this section, the artefact's performance is discussed and compared to similar work, proving a basis for assessing the contribution in context and situating this research within the existing literature. Additionally, the validity and limitations of the artefact are identified and ethical considerations are addressed.

The model expectantly classified inheritance relationships successfully, as the templates for identifying arrows via template matching were derived directly from the dataset. The identification of association relationships was predicted to achieve the lowest scores, primarily due to the challenging obstacle of distinguishing between (particularly short) association lines and irrelevant contours. While class classification performed well, it was anticipated that a generalised thresholding approach would not necessarily be suitable for every image due to differing light conditions. Surprisingly, the exclusive use of OpenCV's image processing techniques without machine learning resulted in higher overall metrics than anticipated. These observations are elaborated on in subsequent sections.

A. Constrasting results with related work

Similar to the results in Deufemia and Risi [7], the elements with the highest recognition rate are classes, whereas association relationships scored lowest, compared in Table III.

While the tools achieved similar values for classes, Deufemia and Risi analysed over double the number of class elements (480 versus 212) and, in addition to the elements analysed in this study, their tool supports recognition for UML packages (which were never confused for classes), and aggregation and composition relationships.

	SketchToPlantUML		**Deufemia & Risi**	
	Precision	Recall	Precision	Recall
Classes	0.92	0.96	0.99	0.95
Inheritance	0.92	0.92	0.92	0.89
Association	0.76	0.76	0.65	0.84

TABLE III: Comparison of Precision and Recall Metrics: SketchToPlantUML vs. Deufemia & Risi tool.

Direct comparisons to interactive sketch recognition approaches are difficult, as these methods differ significantly in implementation to offline or static sketch recognition, or lack evaluation data. As the SUMLOW [3] tool was evaluated using recognition rate (RR) metrics, the recall rates of SketchToPlantUML are used as a comparison, illustrated in Table IV. The SketchToPlantUML tool, however, supports recognition for fewer UML elements (three vs 15), and the need to differentiate between geometrically similar elements accounts for the differences in evaluation values. For example, components recognised by SUMLOW include classes, objects, packages, components and notes, which exhibit similar geometric properties. The approach used in this study would simply classify these components as classes.

The differences in association classification are attributable to SUMLOW's method of recognising only pre-determined symbols drawn in supported orderings. The comparison of SUMLOW's method to the SketchToUML method demonstrates the trade-off between accuracy and flexibility. While the former tool achieves higher accuracy when matched in the number of geometrically similar elements, the latter allows for more flexibility.

B. RQ1 c) What are the limitations of the approach?

The limitations of the artefact's design result from the dependence of each step on the preceding step's output, particularly in the segmentation process. Once isolated, the quadrilaterals representing UML classes are masked to isolate relationship elements. This caused two main drawbacks.

SketchToPlantUML vs. SUMLOW		
	Recall	Recogition Rate
Classes	96%	89%
Inheritance	92%	79%
Association	76%	90%

TABLE IV: Comparison of tools. Note that recall values are compared to recognition rate (RR) values.

Firstly, FN quadrilateral predictions caused disjointed class contours to be classified as association lines. A more robust approach might involve examining the UML elements in context: classifying a class-association-class relationship as a whole rather than as individual elements that are later associated. Secondly, the artefact is highly sensitive to noise and relies on perfect denoising to avoid FN association classifications: noise due to thresholding and inexact text removal caused over half of all incorrect predictions of all elements. However, there is a trade-off between precision and recall: a higher threshold removed all noise, but shorter association lines were inadvertently ignored, while a lower threshold resulted in the classification of all associations, but a larger amount of FP predictions. While this can be mitigated by additional processing before thresholding and more precise text detection, it must be noted that the dataset images themselves were relatively noiseless.

Thus, the approach would need to be adapted for sketches created in more noisy environments. A typical use case in real-world conditions would involve images of sketches created on a whiteboard, which is likely to contain more irrelevant markings than pen-and-paper drawings. As the goal of this study is to minimise the effort involved in transitioning from sketched planning diagrams to formal models, the limitation of the artefact regarding potential noise on a popular medium (the whiteboard) is important.

Moreover, the scope of the artefact is limited to CDs, which consist of geometrically different shapes that are also semantically different. In its current form, the artefact will classify geometrically similar shapes that are semantically different as the same UML component. For example, classes and packages.

C. Threats to validity

The limitations of the design and implementation assist in highlighting several potential threats to validity. While acknowledging the limitations, the artefact's performance should be interpreted within the context of the specific scope and conditions of the study, as further research would be needed to establish its effectiveness in other scenarios.

Construct Validity

Assumption of UML CD as input: The model was designed and developed to accurately predict UML CD elements and focuses on positive predictions. Accordingly, it does not account for images that are not specifically UML CDs. Given additional development time, this construct threat could have been addressed through additional implementation and testing efforts aimed at inspecting and classifying input before further processing.

Internal Validity

Reliance on developer proficiency: The artefact's performance relies on the proficiency and knowledge of the developer. Consequently, the evaluated effectiveness of the artefact is not necessarily reflective of the effectiveness of using OpenCV in achieving the goal of the study.

Manual evaluation process: The evaluation was conducted by a single evaluator visually inspecting the dataset images and manually comparing them to the artefact's output. Although efforts were made to mitigate potential human error and multiple evaluations were conducted on separate occasions, the manual nature of the process increases the likelihood of errors compared to an automated process. To mitigate this threat, future research could explore implementing automated evaluation techniques or utilising multiple evaluators to ensure inter-rater reliability. While the evaluation results are expected to be reliable and indicative of the artefact's effectiveness, it is prudent to recognise the limitations and potential sources of error introduced by the manual evaluation process.

External Validity

Generalisability: The artefact's current design is not necessarily generalisable to sketches drawn on alternative mediums or in noisier contexts (such as whiteboards). The findings may not accurately represent how the artefact would perform in similar but distinct conditions.

Dataset representativeness: The selected dataset contains input diagrams sketched by only one individual on a limited sketch medium (paper). This lack of variation in sketching style, medium, or authorship may restrict the generalisability of the artefact's performance to a broader population of sketchy UML diagrams and subsequently may not fully capture the range of variability present in real-world scenarios.

Efforts to mitigate these external threats involved inspecting the dataset to ensure variation in lighting conditions, sketch variability and diagram complexity. Further research might consider collecting or creating an expanded dataset, incorporating diagrams sketched by multiple individuals on different sketching mediums.

D. Ethical considerations

OpenCV and PlantUML are open-source, allowing full access to the source code. The former is distributed under the 3-clause BSD license[11] which imposes minimal restrictions on the use and distribution of software, while PlantUML is distributed under the copyleft GNU General Public License[12], requiring derived works to remain open source. The use of

[11] opencv.org/license

[12] gnu.org/licenses/gpl-3.0.en.html

these libraries and the intended goal of this research pose no ethical concerns.

However, caution should be exercised when using external or online tools in conjunction with the developed artefact. The artefact outputs the relevant PlantUML textual model but does not automatically render the corresponding graphical model. Data privacy and confidentiality should be considered when using external or online tools to render graphical PlantUML models from their textual representations.

VIII. Conclusions

The objective of this study is to reduce the effort of manually transforming sketched UML CDs into formal PlantUML models, by developing a reproducible artefact called SketchToPlantUML that automates this process. The created artefact interprets static sketches, without needing to use a specialised or interactive tool, and directly transforms informal diagrams into formal models. An intermediate representation of the sketched CD is implemented as a set containing related elements.

Important characteristics of sketched input and UML diagrams were identified, complementing the existing body of research on sketch interpretation and object recognition using OpenCV, particularly in the context of hand-drawn UML diagrams.

The tool is suitable for transforming sketched UML CDs containing the supported elements and can successfully reduce the effort of manual transformation. The evaluation indicated overall precision and recall values of 86% and 88%, respectively. Noting that this study had a narrower scope, the results are comparable to related work. The main causes of inaccurate predictions are ascribable to noise (caused by incomplete text removal and imperfect thresholding) and the segmentation method used (misclassifying closed association lines as rectangles representing classes).

OpenCV was an appropriate and useful tool in achieving the goal of the study and, assuming sufficient experience with the library, addressing the causes of the inaccurate predictions is an approachable goal that would considerably improve the artefact's classification performance. Exploring methods such as image resizing to reduce execution time (primarily related to template matching) would also be useful.

Nevertheless, a combination of image processing in conjunction with machine learning would likely result in a more robust and generalisable method of classification that is less susceptible to the variabilities of static images and free-hand sketching. While this approach was not explored, further research that combines these methods would provide a valuable contribution to the goal of supporting developers by reducing the effort of manually transforming sketched diagrams into formal models.

References

[1] M. Petre, "Uml in practice," in *2013 35th international conference on software engineering (icse)*. IEEE, 2013, pp. 722–731. [Online]. Available: https://doi.org/10.1109/ICSE.2013.6606618

[2] G. Goldschmidt, "The backtalk of self-generated sketches," *Design issues*, vol. 19, no. 1, pp. 72–88, 2003. [Online]. Available: https://www.jstor.org/stable/1512057

[3] Q. Chen, J. Grundy, and J. Hosking, "Sumlow: early design-stage sketching of uml diagrams on an e-whiteboard," *Software: Practice and Experience*, vol. 38, no. 9, pp. 961–994, 2008. [Online]. Available: https://doi.org/10.1002/spe.856

[4] M. Cherubini, G. Venolia, R. DeLine, and A. J. Ko, "Let's go to the whiteboard: how and why software developers use drawings," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2007, pp. 557–566. [Online]. Available: https://doi.org/10.1145/1240624.1240714

[5] B. Dobing and J. Parsons, "How uml is used," *Communications of the ACM*, vol. 49, no. 5, pp. 109–113, 2006. [Online]. Available: https://doi.org/10.1145/1125944.1125949

[6] F. Chen, L. Zhang, X. Lian, and N. Niu, "Automatically recognizing the semantic elements from uml class diagram images," *Journal of Systems and Software*, vol. 193, p. 111431, 2022. [Online]. Available: https://doi.org/10.1016/j.jss.2022.111431

[7] V. Deufemia and M. Risi, "Multi-domain recognition of hand-drawn diagrams using hierarchical parsing," *Multimodal Technologies and Interaction*, vol. 4, no. 3, p. 52, 2020. [Online]. Available: https://doi.org/10.3390/mti4030052

[8] T. Hammond and R. Davis, "Tahuti: A geometrical sketch recognition system for uml class diagrams," in *ACM SIGGRAPH 2006 Courses*, 2006, pp. 25–es. [Online]. Available: http://dx.doi.org/10.1145/1185657.1185786

[9] B. Vesin, R. Jolak, and M. R. Chaudron, "Octouml: an environment for exploratory and collaborative software design," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 2017, pp. 7–10. [Online]. Available: https://doi-org.proxybib.miun.se/10.1109/ICSE-C.2017.19

[10] D. Bhasin, G. Goyal, and M. Dutta, "Design of an effective preprocessing approach for offline handwritten images," *International Journal of Computer Applications*, vol. 98, no. 1, 2014. [Online]. Available: http://dx.doi.org/10.5120/17147-7179

[11] P. Runeson, E. Engström, and M.-A. Storey, "The design science paradigm as a frame for empirical software engineering," *Contemporary empirical methods in software engineering*, pp. 127–147, 2020. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-32489-6_5

[12] L. Piucco, "Handwritten uml class diagrams," Sep 2021. [Online]. Available: https://www.kaggle.com/datasets/eb72f3b6dfddec0fdf05a6e2f461116cce40e5d0d5e15f0532389d24d9f09fb70

[13] K. C. London. King's undergraduate research fellowships (kurf). [Online]. Available: https://keats.kcl.ac.uk/course/view.php?id=106869&section=6#tabs-tree-start