Technical University of Munich
School of Computation, Information and Technology
Chair of Electronic Design Automation

Automating Printer Driver Operations: Scheduling-Based Grouping and Automatic Printing

Bachelor's Thesis

Anwar Oluwatofarati Raji

Technical University of Munich
School of Computation, Information and Technology
Chair of Electronic Design Automation

Automating Printer Driver Operations: Scheduling-Based Grouping and Automatic Printing

Bachelor's Thesis

Anwar Oluwatofarati Raji

Advisor :	Meng Lian, Hu Peng
Advising Professor :	Prof. Ulf Schlichtmann
Topic issued :	10.06.2024
Date of submission :	??.??.????

Anwar Oluwatofarati Raji
Auwaldgasse 5
80939 München

Abstract

The global demand for printed Electronics continues to exponentially increase every year. A key advantage is the quick fabrication of electronic components such as Transistors, Diodes and other Sensor applications without having to undergo etching process which is common with other Semiconductor manufacturing techniques. The advantage of printed Electronics is further emphasised on its compatibility with Polymers in comparison to classical microfabrication and the chemical complications which could arise through its combination. There is also the good scaling factor, the use of low cost substrates and better compatibility with flexible and stretchable materials. These methods however possess challenges as noted in these studies [2], [3] carried out by the EDA Team in the previous years. Among these issues are the drying periods along with the length of time it takes for the deposition of the printed Layers to be completed. These could take hours, demanding more time from the fabrication Team which in turn makes the process less efficient. To overcome this Limitation, a necessary step towards automating the printing process has to be taken. This ensures that the entire ensemble can be manufactured without requiring the presence of an external user to manually continue the process. It would be done through an automation script that takes into account various factors that can affect the workflow and making sure that at least the simplest of components can be manufactured at the behest of a User. This Script would utilize multiple GUI Altering functions that interact with the slice software as well as its printing counterpart.

Contents

1. Introduction

1.1. Introduction to Inkjet printing

The method through which the material would be deposited on the base during this fabrication is inkjet printing. In this method, a material cartridge containing the required emulsion, Silver (Ag) in our case, is ejected through a nozzle in droplets through the piezoelectric principle. This means that when an electric charge is applied to a crystal it deforms pushing the droplets out of the nozzle. This method has gained increasing attention in the area of printed electronics to create devices like antennas or even sensors and lab-on-chip devices. It has also a utility in the medical sector for the deposition of other materials such as proteins or cells onto substrates through which tissue can be manufactured. This method of printing has a good advantage over normal etching in circuit manufacturing it does not chemically consume as much material which cannot be recovered and also does not require expensive detailed masks, thereby making it more cost effective. Another advantage is it's material flexibility as it can applied on plastics, glass and silicon without chemical damage as observed in etching. In spite of its resolution limits compared to lithography which goes into the nanometer scale as well as the ink formulation management problem, which requires constant observation and care of the printing materials to be prevent issues with the deposition of droplets, there is still growing interest in Inkjet printing especially in the area of 3D microfabrication in microelectronics and biomedical engineering.

1.2. Design and Printing Automation

The Ensemble is the largest and most comprehensive part of the fabrication. It is divided into layers which are in turn split into objects, which are the smallest units of an ensemble, as seen in the below diagram. See Figure 1.2

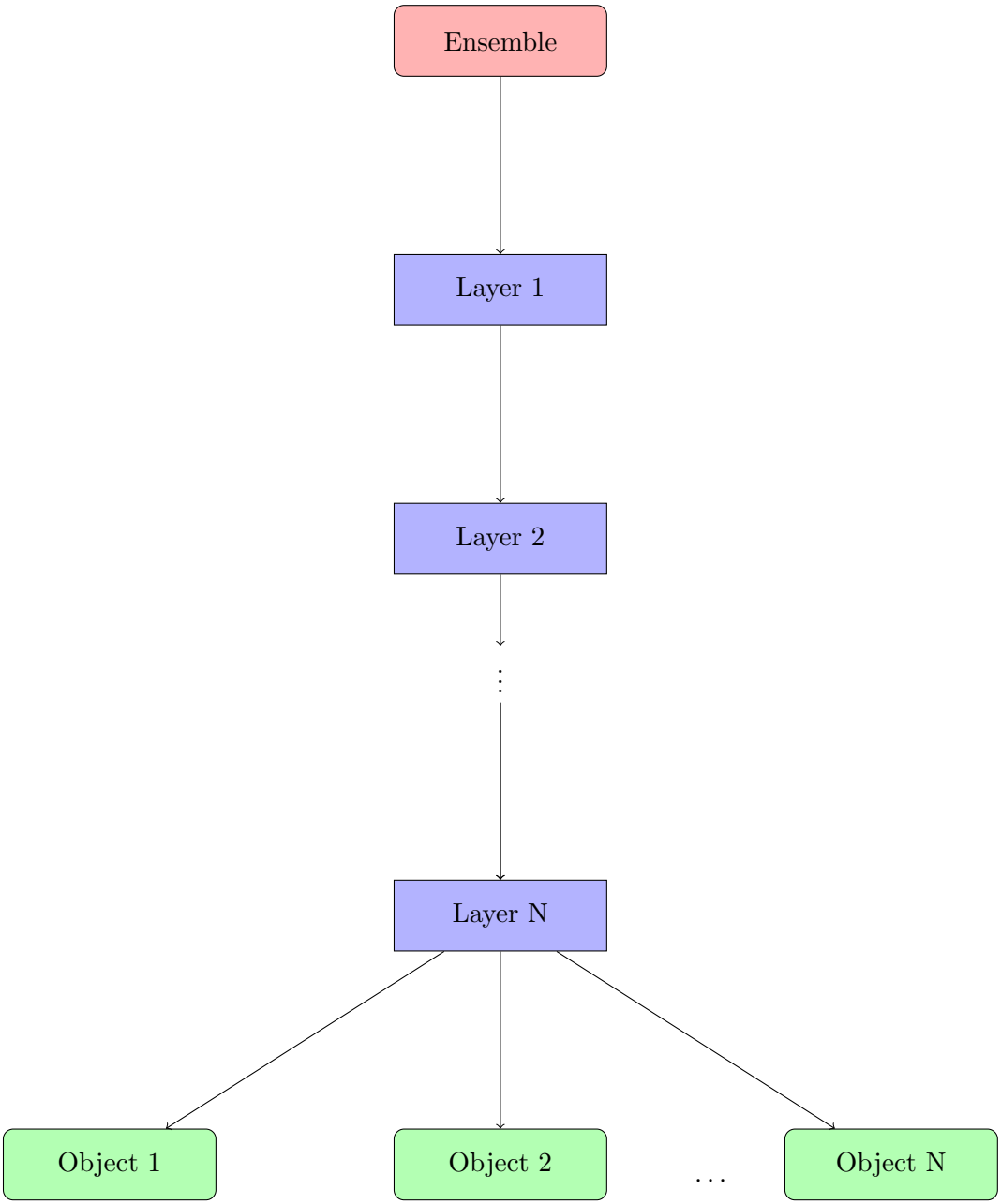


Figure 1.1.: This represents the hierarchy of the fabrication

Object

This is the smallest part of the the fabrication, represented by rectangles, circles or polylines.

Layer

This is the second smallest part of the fabrication, represented by a combination of objects.

Ensemble

The largest part of the fabrication, represented by a combination of multiple components/Layers.

The main goal in this project is to separate our ensemble circuit into layers and print the corresponding layers accordingly without having to trigger the process again through manual input by a user, noted as the final step in the abstract of Rinklin et al. [2]. The printing takes place by reading the DXF file in which the layers are already separated and then with the assistance of the Offset text files, which contain the coordinates of each corresponding Layer, to automatically translate this into set parameters in the CeraSlice software and the begin the physical printing through CeraPrint. The idea is to be implemented with the help of a python script which automatically triggers the printing process of the next layer after the estimated printing time of the previous layer is exceeded.

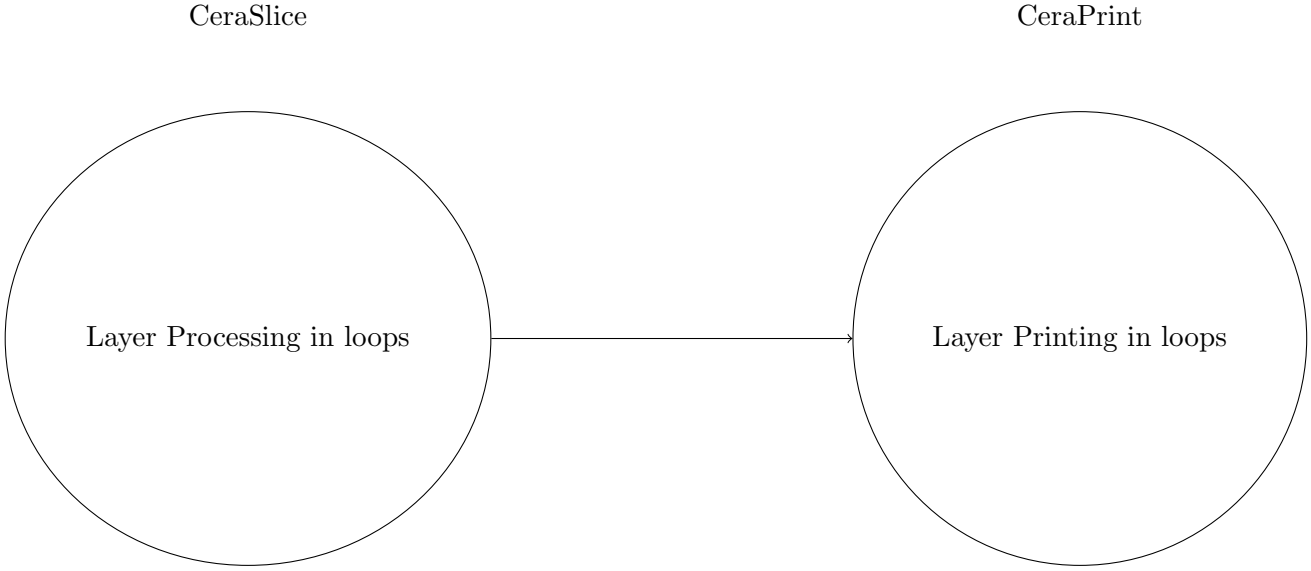


Figure 1.2.: A general overview of the working process

In this regard it is paramount to ensure that the algorithm;

- Is functional i.e. can be successfully compiled
- Takes input errors into account

- And makes sure to give comprehensible feedback from the printer and CeraDrop software with which it interacts.

The code is written in the Visual Studio Editor and executed through the Linux Ubuntu Virtual Subsystem, where the primary coding and management in accordance with GitHub, will be worked on.

1.3. Layer Separation

The Layers are separated through a algorithm already developed by the Team which is not the focus of this Task. It is instead to take the already separated Layers of the complete Ensemble along with its Parameters and have those as input into the CeraSlice Software before every printing process commences. This project will ensure that the DXF files are accurately and efficiently extracted in order to prevent errors in the script.

Within the scope of this project however, the concern will instead be mainly to ensure that the Layers are correctly located and gathered from their folders within the computer system and then compiled into a single object within CeraSlice and finally printed through CeraPrint through accurately scheduled automation, whereby all Layers are printed after another as intended according to their input expected durations without awaiting further user input which could extremely lengthen the printing process. The waiting time in between each Layer print operation would be estimated as determined by Tseng et al. [3].

2. Background

2.1. CeraSlice Software

This is the slicing software where our fabrication would be processed and prepared for printing. Within this application is where the bulk of the work of this project will lie, as;

- It is where the DXF files would be imported,
- It is where the objects and ensembles combined,
- It provides to opportunity to adjust and configure the print settings, such as deposition material, droplet displacement or number of nozzles in effect,
- It also provides the opportunity to observe the layer in a simulator before it is finally written as a print files ready to be taken in by the CeraPrinter software.

2.2. DriverCeraPrinter Software

To gain access to the Printer's machine control interface the DriverCeraPrinter software is required. This interface controls the printing process and it consists of the DropAnalyser/JetAnalyser (in the case of Aerosol), CeraPrint and FabAnalyser.

2.2.1. DropAnalyser

Within the DropAnalyser the pulses i.e. the piezoelectric voltage through which the nozzles eject the printed material, can be configured to modify the ejection frequency, amongst other functions such as Nozzle selection and activation. These can even be altered during fabrication.

2.2.2. JetAnalyser

In the case of the JetAnalyser, the deposition and displacement velocities, measured in mm/s, can be altered. As well as the substrate thickness.

2.2.3. FabAnalyser

The FabAnalyser has the function of examining and characterising the already printed fabrication. It uses an embedded camera in the position where the nozzle was.

2.2.4. CeraPrint

This tool is the most relevant for this Project. The Printing Jobs which are generated by the CeraSlice software are executed through this tool. It presents two modes to perform the aforementioned Task, as defined in the DriverCeraPrint Manual;

- Raster: to print files treated inkjet CeraSlice files in raster scan mode and CeraSlice Aerosol Jet® files
- Vector: to define design and printing settings for vector inkjet printing job with single nozzle or DMC cartridge

In this function the fabrication settings can be determined. Options such as fabrication speed, pulse selection and most importantly the objects of the ensemble can be individually selected to be printed.

In this project only the Raster mode will be relevant as that is where the print jobs are to be demonstrated and executed.

3. Implementation

Problem Statement

The demands of this automation project are:

- To select each Layer on the printing base and combine all its objects into a single ensemble,
- To apply all its settings such as number of nozzles, droplet distance etc,
- Write the new ensemble into a fabrication file and continue this loop for the other Layers,
- And print each ensemble, wait for the expected duration and loop to the next Layer until finished.

3.1. Choice of Programming Language

The selected Script and programming language of choice for this project will be Python. Due to its readily available "ezdxf" library which simplifies the process of working with and on the DXF Files that will be fed as input into the CeraPrinter F-Series model which is the printer used in this project. Its general hardware and software specifications can be found here... Python also has the pyautogui library which makes the GUI method much less complex to code, as compared to a language like C++ where the task would be more tedious and less accurate.

3.2. Script Implementation

The Python Script will be mostly written in Visual Studio Code and within a virtual Linux System as this has smooth integration with GitHub. The Code for this script will be documented thoroughly on GitHub where the Algorithm will be uploaded and maintained within its Life Cycle. This also functions as an opportunity for colleagues at the EDA to have a review and proper understanding of the code. The repository will of course be created within the LRZ subsystem of GitHub for security purposes. The Script however has to be run within a windows framework, especially in the case of the GUI Method. This is due to the mouse

and keyboard manipulations being unavailable in the Linux virtual system. The Script will be separated into the main function and various methods as seen here Git Repo

3.3. Layer-Printing Algorithm

With regards to printing each layer accordingly, there will also be measures to check on the status of the printer which is intended to be done through the use of an API (Application Programming Interface) should the CeraSlice software provide one. If not then serial communication with the printer via G-Codes would be required to observe its status and send printing commands. Under this status, the algorithm will leave enough time considering;

- If the printer is still actively printing a layer,
- And if the printer is to begin work on a new layer

Should the above mentioned options not work, the third possibility would be to implement the automation through the GUI (Graphic User Interface) method. This third method however proves a challenge as it is very likely unreliable and error prone, due to its dependence on the position of the Print Command Button on the computer screen, which does not take into account the possibility of a printer error (such as an empty nozzle) preventing the printing process from continuing or if the software's graphical coordinates are influenced by a User. In this project, two of the aforementioned methods will be approached, namely the; GUI and API Methods.

Regarding the API Method, an API provided by the CeraDrop exclusive software would be required for smoother implementation. Through this tool it would be convenient and efficient to;

- Monitor the printer status
- Get comprehensive feedback from the printer, in the case of an error
- Issue commands through the Script when necessary. This is a much more robust method and was the preferred approach when considering the requirements of this project.

In the case of the GUI Method however, there is much more wiggle room in terms of the possibility of its implementation i.e. it can always be done regardless of the existence of special tools such as an API. However as previously mentioned it is a very brute method and can be easily influenced if the screen of the software, where the printing is to be triggered, is influenced or if there is a windows error preventing access to the software graphically. The options in this scenario to write an efficient script that is error robust are limited.

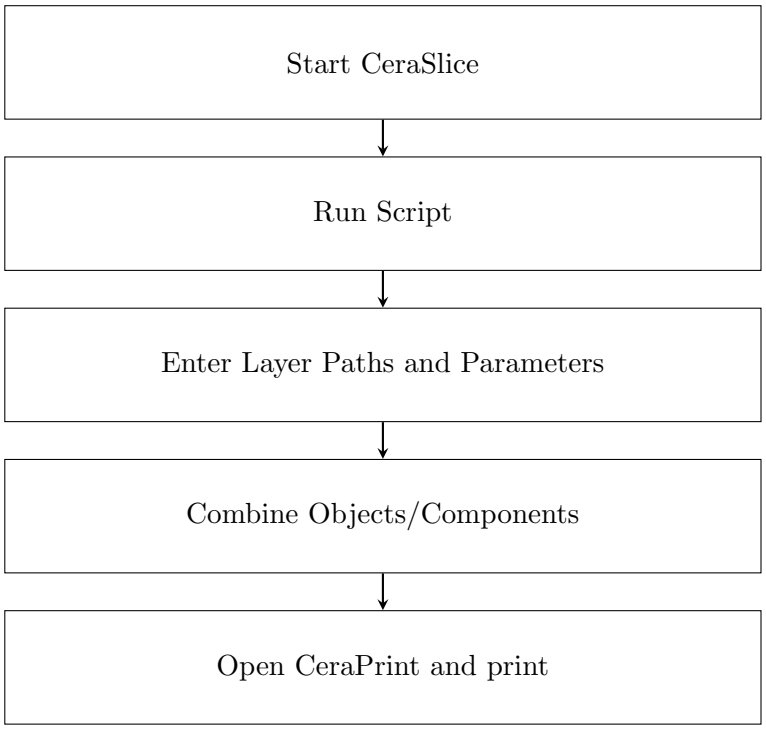
Henceforth, every other chapter for the rest of this Thesis would be divided into the two methods of Implementation which would be worked on.

3.3.1. API Method

After thorough consultation of the Team at CeraDrop it was conveyed that the Software has no API tool to interact with as it is an exclusive Software. This makes the scope of our project more limited as an interaction directly with the printer cannot take place, leaving us with the Graphical method only.

3.3.2. GUI Method

The script for the algorithm will pull from the python libraries; ezdxf to parse through the DXF files, pyserial to gain access to temporal libraries in python, and pyautogui to manipulate the windows mouse and keyboard functions. The idea is to ensure the CeraSlice software is active and the screen remains the same when the program is started. In order ensure there is enough ample time for the program to start before the algorithm begins to process, considering user error. About 20 seconds would be allocated in this regard. Along with an on-screen display indicating the Script has been activated and is currently running. After this runs, the algorithm will then estimate and print the screen size, before prompting the user to enter the file paths to the folder where each Layer is stored in the system. After that, the script prompts the user for the parameters required as input within the CeraSlice configuration window. Once the Layers and the parameters have been correspondingly added, the program will begin to select the icons on the CeraSlice GUI that are necessary to begin the printing process, this includes the tab where the Layer path that was selected before will be input, along with the parameter Tab where the previously input parameter would also be entered. And finally the Components would all be selected and compiled into a single object and saved as a CeraSlice file.



Once all the interactions are done, the algorithm will move to the CeraPrint interface, there the newly saved Layer in the CeraSlice would be available for selection. We would then finally be able to triggered the printer to start the deposition process. There are printed instructions to guide the user on how to utilize the script, preventing confusion and to add a user friendly aspect to this program, as well detailed comments in the code itself. There will also be a general Readme text to guide all potential users on how to use the script correctly here Git Repo. The script would be run through its own executable created through Pyinstaller.

The Algorithm is divided into six python scripts performing their own individual functions contributing to the whole project all available at Git Repo .

NOTE: These scripts represent the code in their final form and does not take into effect the changes and refactoring that were made to each script. Please consider the Gitlog here at Git Repo for this information

The main function: Named GUImain.py. It is the capital function of the algorithm and is what the user should run when attempting to utilize the automation script in its whole. It pulls functions from all previously declared and defined function which are stored in the same folder/repository as well as the required inbuilt libraries installed in Python 3. Along with

other auxiliary functions used to make the utilization of the codes efficient and understandable.

```
import time
import sys
import os
import tkinter as tk
from tkinter import simpledialog
from tkinter import filedialog
import pyautogui
from GUIclick import click_cycle
from dxfdirectory import scale_all_dxf_in_directory
from dxfprint import printselection
from windowsgui import switchapp

def click_ensemble(scaled_folder): ...
def get_directory_path(Directories): ...
def main(): ...
```

The scaling function: Named dxfscaler.py. This is the code that utilizes the ezdxf library to scale down/up the DXF layers according to their relative size as JPGs in the CeraSlice Application. In this case it is 0.35 times the size.

```
import os
import ezdxf

def scale_down_dxf(file_path, scale_factor, output_path):
    # Load the DXF file
    doc = ezdxf.readfile(file_path)
    msp = doc.modelspace()

    # Iterate over all entities in the modelspace and scale their
        geometric data
    for entity in msp:
        if entity.dxftype() == 'LINE':
            entity.dxf.start = tuple(coord * scale_factor for coord in
                entity.dxf.start)
            entity.dxf.end = tuple(coord * scale_factor for coord in
                entity.dxf.end)
        elif entity.dxftype() == 'CIRCLE':
            entity.dxf.center = tuple(coord * scale_factor for coord in
                entity.dxf.center)
            entity.dxf.radius *= scale_factor
        elif entity.dxftype() == 'POLYLINE':
            for vertex in entity.vertices:
                vertex.dxf.location = tuple(coord * scale_factor for
                    coord in vertex.dxf.location)
                # Continue with the scaling logic for POLYLINE and
                    other entities as before

```

```
    # Save the scaled DXF file
                doc.saveas(output_path)
    # Example usage
#directory_path = r"//nas.ads.mwn.de/ga78nog/TUM-PC/Desktop/Anwar/
    Layers"
#output_directory = r"//nas.ads.mwn.de/ga78nog/TUM-PC/Desktop/Anwar/
    Scaled Layers"
#scale_factor = 0.35
```

The folder scaling function: Named dxfdirectory.py. This ensures that all Layers kept within a folder are scaled down according to the chosen scaling factor.

```
import os
from dxfscaler import scale_down_dxf

def scale_all_dxf_in_directory(directory_path, scale_factor,
    output_directory ): ...
```

The clicking function: Named GUIclick.py. This function is utilized to click through the CeraSlice Application performing all required tasks for each layer.

```
import time
import tkinter as tk
import pyautogui
import pyperclip
def click_cycle(file_location, fabname): ...
```

The printing function: Named dxfprint.py. This function contains the clicking cycle for the CeraPrint section of the project and takes the fabrication number and input printing duration as arguments.

```
import pyautogui
import time
def printselection(fabno, duration): ...
```

The windows function: Named windowsgui.py. This function is used to switch from CeraSlice to Ceraprint.

```
import pyautogui
import pygetwindow as gw
import time

def switchapp(window_title):
        window = gw.getWindowsWithTitle(window_title)
```

```
7           if window:
8                   window[0].activate()
9           else:
10                  print("No window found")
11
12  window_title = "Fabrication"
13  switchapp(window_title)
```

3.4. Testing through TDD (Test Driven Development) Cycle Principle

This Program is to be written in accordance with the TDD method Beck [1]. This ensures a simplification of the coding of complex projects. All individual functions are tested as soon as they are written to ensure smooth and correct functionality. Once a function produces satisfactory results, it is immediately pushed to the GitHub Repository, accompanied by an appropriate comment describing the alterations made to the code body, to ensure the individual coder as well as the other team members within a project can follow through on the steps made. Should the function or algorithm persist in its incorrectness, the Code must be continuously refactored until it passes the written Tests and can be successfully pushed to the Git Repo as seen in the figure below.

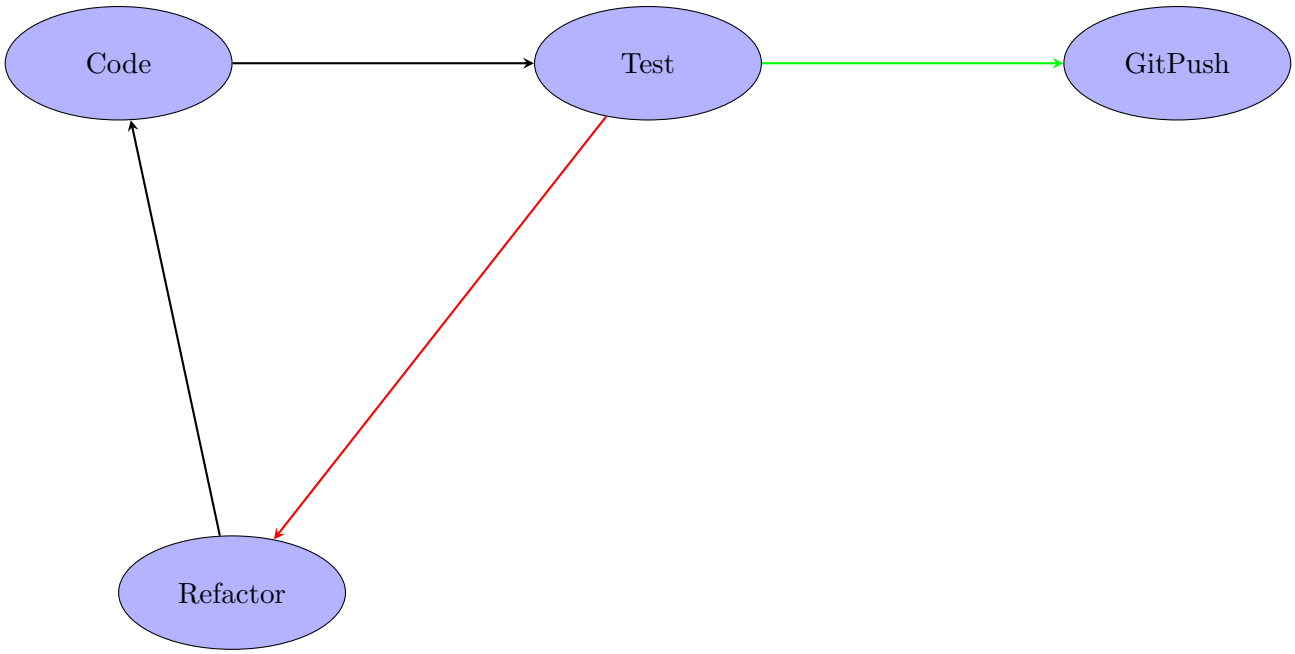


Figure 3.1.: This shows the TDD Cycle as implemented within this Project. The red arrow indicates a failed Test and the green a positive one.

Code

Within this Phase the code is initially written with the goals in mind.

Test

The code is then tested through a test function which gives a positive or negative defined by how the coder expects the outcome to be.

Refactor

Should the test produce a negative result, the code must be altered to fix whatever bugs might have occurred. This is known as "refactoring".

GitPush

When a code body produces a positive test result it is pushed to the Git Repository and merge with the main branch.

4. Results

4.1. Code refactoring

Through the TDD method as seen in the previous chapter, we have had to test the Code every moment a function was written. Most functions expected and passed the Test functions written through the aid of the Test functions. During the implementation of the screen manipulation however it was quickly but as expected established that the Desktop screen dimensions deviated from that of the home notebook. Corresponding changes had to be made to the algorithm, where the code was refactored to take the size of the Desktop screen into consideration. This aspect is vital to our algorithm as it ensures the correct Icons are selected in proper order and that the parameters are entered correctly into the given input tabs within the CeraSlice GUI. Further attempts at Code refactoring will be specifically noted throughout the rest of this chapter whenever they come into relevance.

It was also decided to make the script more user friendly. To this end a prompt box using the "tkinter" library was implemented to indicate the status of the script in real-time, along with immediate printed instructions that come up in any terminal where the Python script would be activated.

4.2. CeraSlice Application

This section details all attempts at running the script in tandem with the CeraSlice Software. It must be ensured the correct fabrication files are selected or created, the correct layers imported and the proper nozzle and print settings are sliced before the newly prepared Ensembles are finally written as cdp files and are ready to be accessed in CeraPrint.

4.2.1. Initial Application

The following lists in the subsequent sections represent each goal and their success when applying the code. They are individually tested to fix certain bugs and collectively tested as required by the algorithm as a whole.

The simple goal at first application is detailed by following automation steps and their implementation results:

- Opening CeraSlice and preparing the screen for the script presented no issues at all.

- Creating a new fabrication file was also successful.

- Hiding the stage and importing the DXF file presented issues as the stage did not disappear as expected as the imported DXF file was too large in its JPG form.

- Selecting all objects of this Layer and combining them into an ensemble did not work as the stage was also included.

- Selecting the nozzle settings to 1-12 and printing material to Silver(Ag) and validate was functional.

- Going into settings and under ensemble, select splat diameter to 50 um, overlap to 20 um, priming width to 0 and calculate also worked as required.

- Setting the detection resolution to 15um, and tolerance to 3um, calculating and validating, worked with no issues.

- Under the settings Tab, setting the object displacement to 25um and apply, functioned properly.

- Under the settings Tab, setting the component priming distance to 0 and apply, was also functional

- Navigating to the slice option and calculating, worked with no issues.

- Observing the Layer under the simulator and selecting detailed filling along with viewing the raster of objects, posed challenges as the mouse had to navigate to the different boxes on the screen, which the code did not account for.

- And finally moving to the fabrication option and write the file was dysfunctional because

the mouse missed the location of the 'write' button.

- Waiting for the file to be written for 20 seconds did not work as well, as the writing did not even begin.

Encountered Issues

During the first test phase there were immediate issues with the integration of the main function with the methods. The other functions are written in separate python files stored in the same folder as the main.py script. They had to be properly imported along with the specific functions within them. Along with this it was also discovered that the DXF files were too large for the stage which cause massive issues when writing the fabrications. The "Ctrl+a" command to select all layers also proved to be disastrous, causing the algorithm to compile the entire base instead of the smaller and less detailed Layer on its surface. There were also issues with the coordinates of the dialog box to select the path of the folder containing each Layer, which consistently kept changing positions with every new fabrication. And finally the on screen coordinates of the other icons had to be carefully tested and calibrated in the algorithm.

4.2.2. Second Application

There were some changes made to the code requirements to improve on the initial attempt.

The corrected automation steps to be executed on this software and their implementation results are as follows:

- Opening CeraSlice and preparing the screen for the script presented no issues at all.

- The Script is to be run and the folder containing the Layers first selected in a dialog prompt, followed by a selection of the folder where the newly descaled Layer DXF files are to be stored. The scaling function can be found here Git Repo. This function correctly scaled down the Layer size.

- Then the script generated a new fabrication file, which will contain the main stage where the printing will take place as its base, this functioned correctly.

- The stage was be hidden and the newly resized layer will be completely selected via a drag box and combined into an object. This phase worked perfectly as the stage had

disappeared and the Layer was properly scaled.

- Selecting all objects of this Layer and combining them into an ensemble, worked as required.

- Selecting the nozzle settings to 1-12 and setting printing material to Silver(Ag) and validating, were also successful.

- Going into settings and under ensemble, setting splat diameter to 50 um, overlap to 20 um, priming width to 0 and calculating, all worked properly.

- Setting the detection resolution to 15um, and tolerance to 3um, calculating and validating, functioned properly.

- Under the settings Tab, setting the object displacement to 25um and apply, functioned properly.

- Under the settings Tab, setting the component priming distance to 0 and apply, was also functional

- Navigating to the slice option and calculating, worked with no issues.

- Observing the Layer under the simulator and selecting detailed filling along with viewing the raster of objects, had finally proved successful after the clicking coordinates were altered.

- And finally moving to the fabrication option and write the file was dysfunctional because the mouse missed the location of the 'write' button.

- Waiting for the file to be written for 20 seconds did not work as well, as the writing did not even begin.

Solving previous Issues

The issues containing the selection of all objects without including the stage was simply implemented through a Dragbox function available through pyautogui as seen in 4.1. This code was refactored 3.3.2

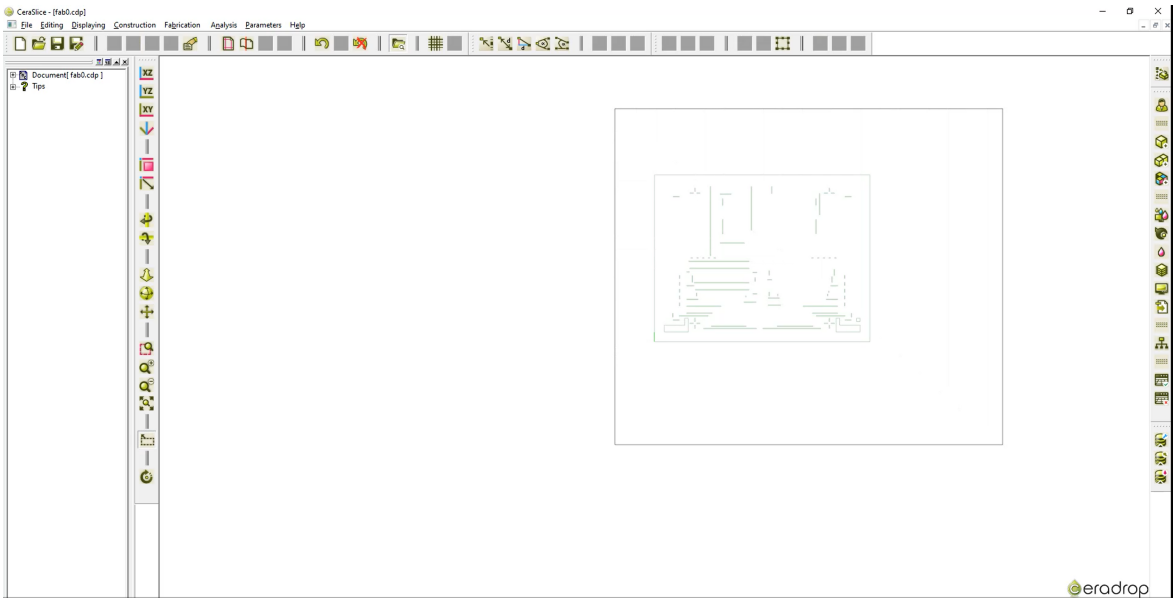


Figure 4.1.: The dragbox selecting all objects.

Encountered Issues

Before we load our DXF file into this software, we need to feed it into the algorithm where we observe the layer size and correspondingly scale its size to ensure it can fit on the base to be printed, it is to be implemented via the ezdxf library as noted in 3.1. This code was altered 3.3.2. And then the loading of this converted DXF file into our fabrication file takes place. To import all DXF files without depending on moving coordinates, a hotkey option would also have to be utilized. To this end, the "Alt" key would be utilized, to highlight the menu tab and then the folder path would be input.

4.2.3. Third Application

The again improved automation steps to be executed on this software and their implementation results are as follows:

- Opening CeraSlice and preparing the screen for the script presented no issues at all.

- The Script is to be run and the folder containing the Layers first selected in a dialog prompt, followed by a selection of the folder where the newly descaled Layer DXF files are to be stored. The scaling function can be found here Git Repo. This function correctly scaled down the Layer size.

- Then the script generated a new fabrication file, which will contain the main stage where the printing will take place as its base, this functioned correctly.

- The stage was be hidden and the newly resized layer will be completely selected via a drag box and combined into an object. This phase worked perfectly as the stage had disappeared and the Layer was properly scaled.

- Selecting all objects of this Layer and combining them into an ensemble, worked as required.

- Selecting the nozzle settings to 1-12 and setting printing material to Silver(Ag) and validating, were also successful.

- Going into settings and under ensemble, setting splat diameter to 50 um, overlap to 20 um, priming width to 0 and calculating, all worked properly.

- Setting the detection resolution to 15um, and tolerance to 3um, calculating and validating, functioned properly.

- Under the settings Tab, setting the object displacement to 25um and apply, functioned properly.

- Under the settings Tab, setting the component priming distance to 0 and apply, was also functional

- Navigating to the slice option and calculating, worked with no issues.

- Observing the Layer under the simulator and selecting detailed filling along with viewing the raster of objects, had finally proved successful after the clicking coordinates were altered.

- And finally moving to the fabrication option and write the file was functional because the mouse found the location of the 'write' button.

- Waiting for the file to be written for 20 seconds did not work as well, as more time needed to be added for a safe window before looping to a new fabrication.

Solving previous Issues

To fix the problem of finding Layer paths from the previous Application a direct path search within their folder for every layer was implemented in the code. This seeks every layer within a folder in an iterative manner and uses the keyboard to enter and select them. This clicking code 3.3.2 was relevant to this aspect.

Encountered Issues

The looping of the remaining Layers was not successful due to an obstruction caused by various Tabs that were opened before when configuring the Layers and Nozzles.

4.2.4. Fourth Application

The finally improved automation steps to be executed on this software and their implementation results are as follows:

- Opening CeraSlice and preparing the screen for the script presented no issues at all.

- The Script is to be run and the folder containing the Layers first selected in a dialog prompt, followed by a selection of the folder where the newly descaled Layer DXF files are to be stored. The scaling function can be found here Git Repo. This function correctly scaled down the Layer size.

- Then the script generated a new fabrication file, which will contain the main stage where the printing will take place as its base, this functioned correctly.

- The stage was be hidden and the newly resized layer will be completely selected via a drag box and combined into an object. This phase worked perfectly as the stage had

disappeared and the Layer was properly scaled.

- Selecting all objects of this Layer and combining them into an ensemble, worked as required.

- Selecting the nozzle settings to 1-12 and setting printing material to Silver(Ag) and validating, were also successful.

- Going into settings and under ensemble, setting splat diameter to 50 um, overlap to 20 um, priming width to 0 and calculating, all worked properly.

- Setting the detection resolution to 15um, and tolerance to 3um, calculating and validating, functioned properly.

- Under the settings Tab, setting the object displacement to 25um and apply, functioned properly.

- Under the settings Tab, setting the component priming distance to 0 and apply, was also functional

- Navigating to the slice option and calculating, worked with no issues.

- Observing the Layer under the simulator and selecting detailed filling along with viewing the raster of objects, had finally proved successful after the clicking coordinates were altered.

- And finally moving to the fabrication option and write the file was functional because the mouse found the location of the 'write' button.

- Waiting for the file to be written for 20 seconds finally work as well, as a buffer time of 5 seconds was added for a safe window before looping to a new fabrication.

Solving previous Issues

To ensure that all Layers were able to be looped the clicking algorithm 3.3.2 was altered to include the deselection of all previous Tabs. There was also added time to ensure the fabrications were written before another new Layer was to begin being processed. Through all these newly refactored codes, the script was able to successfully process all Layers by requesting the locations where they were stored by iterating through all of them before moving through to the next phase.

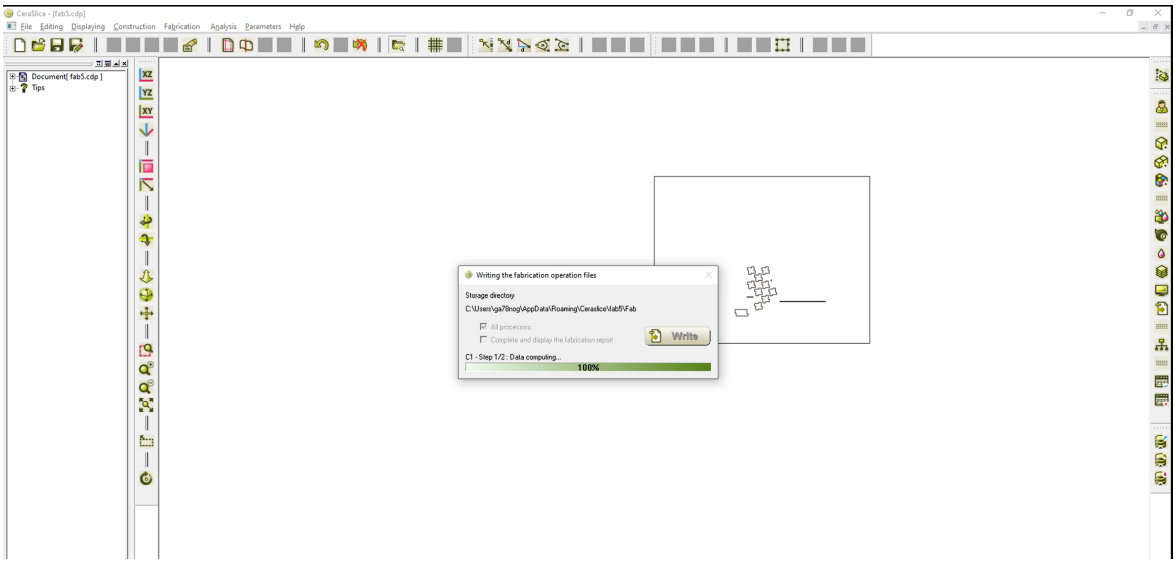


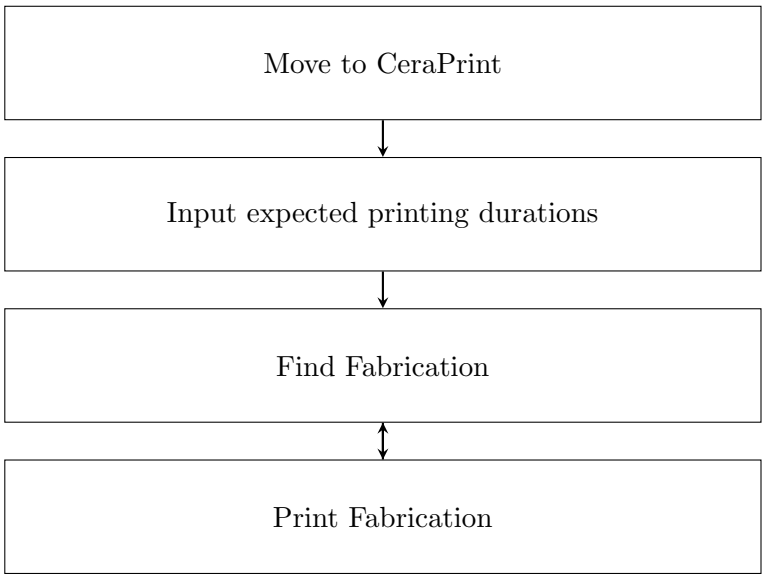
Figure 4.2.: Iterating through all Layers successfully.

4.3. CeraPrint Application

To this end the written files in the previous process through the slicing software are then sequentially selected and printed within give time frames. This function will utilize its very own separate printing function which will be written separately as the other script functions in the previous chapters, before being imported and applied by the main.py script. It will of course be uploaded and documented on Git Repo. Before utilization on the printer itself there will also be test runs of the printing functions per layer to ensure each process carries on smoothly and timely without having to waste the CeraPrinter's resources through false printing and refactoring. To achieve this a Test function will also be written and compiled before the being refactored into an actual physical printing code.

4.3.1. CeraPrint Tests

The general idea in this scenario since there wont be any actively printed circuits is to properly ensure and demonstrate that the script, in a scenario of Application is indeed robust enough to start the printing process and await the duration of each Layer's printing before beginning the next.



4.3.2. Initial Application

Within this section the testing of the algorithm within this frame would be listed. Certain sections of the code, each with their goal, might achieve their aims and other might not as they are tested individually. The automation goals and results in the CeraPrint section are as follows:

- Minimizing the CeraSlice app did not function the CeraPrint app was not yet active.
- Requesting for print duration for all Layers had not worked as well. This was due to there being no array in the code to hold the input times for each layer.
- Properly navigating to CeraPrint was not yet written in the automated code at all.
- Finding all fabrications proved challenging, as they were not immediately seen in the CeraPrint Tab.

- The code for looping the printing time was also not yet written.

Encountered Issues

The CeraSlice Application which was previously active was no longer required as the CeraPrint software would now come into relevance. Automating the exit of CeraSlice to CeraPrint proved challenging. An Issue concerning the length of time to wait for each Layer to be printed was also raised. For all refactoring in this subsection we will only deal with this code 3.3.2

4.3.3. Second Application

The improved automation steps relative to the initial application are:

- Minimizing the CeraSlice app did function through the use of the "alt"+"tab" hotkey combination.

- Requesting for print duration for all Layers had not worked as well. This was due to there being no array in the code to hold the input times for each layer.

- Properly navigating to CeraPrint was not functional as the hotkey solution proved inefficient.

- Finding all fabrications proved challenging, as they were not immediately seen in the CeraPrint Tab. This however was solved by creating an array for all fabrication names where they were stored as they were created in CeraSlice.

- The code for looping the printing time was also not yet written.

Solving Encountered Issues

For navigating into CeraPrint, the user has to ensure that the app is already opened and not minimized before running the script from the beginning. To leave the app it was decided to automate the keyboard to press the "alt" + "Tab" buttons to navigate to CeraPrint. To find all fabrications, the code was altered to iterate add all fabrication names into a list at the

moment of creation in CeraSlice and then iterate through them later on in CeraPrint as seen in 4.3.

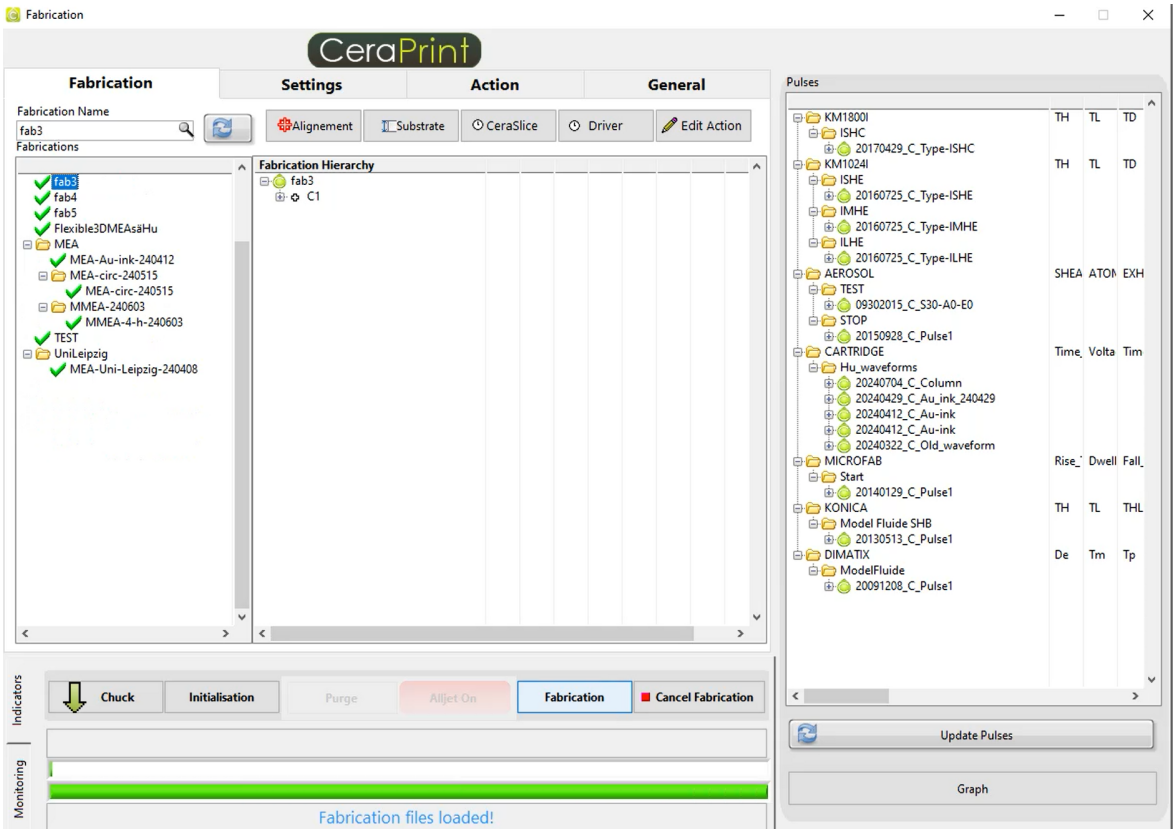


Figure 4.3.: Iterating through the Fabrications.

Encountered Issues

Automating the movement from CeraSlice to CeraPrint still proved challenging as the above mention solution proved unreliable as the position of the CeraPrint software in the selection menu kept changing.

4.3.4. Third Application

The automation goals within this application remain the same as the previous:

- Minimizing the CeraSlice app did function through the use of the ”alt”+”tab” hotkey combination.

- Requesting for print duration for all Layers had not worked as well. This was due to there being no array in the code to hold the input times for each layer.

- Properly navigating to CeraPrint was now functional. As the hotkey solution proved inefficient, a windowsgui function was utilized to jump to the already opened CeraPrint Tab instantaneously

- Finding all fabrications proved challenging, as they were not immediately seen in the CeraPrint Tab. This however was solved by creating an array for all fabrication names where they were stored as they were created in CeraSlice.

- The code for looping the printing time was also not yet written.

Solving Encountered Issues

The "alt"+"tab" solution was completely abandoned in place of a windowsgui function. Through this method the script switched directly to the Fabrication screen in CeraPrint without issue, provided the screen is active and not minimized.

Encountered Issues

The duration times, how and when they are to be input was still an issue.

4.3.5. Fourth Application

The automation goals within this application remain the same as the previous:

- Minimizing the CeraSlice app did function through the use of the "alt"+"tab" hotkey combination.

- Requesting for print duration for all Layers had finally worked as well. This came after an array was created to hold the printing durations and the code now requested the durations for all layers consecutively via a dialog box.

- Properly navigating to CeraPrint was now functional. As the hotkey solution proved inefficient, a windowsgui function was utilized to jump to the already opened CeraPrint Tab instantaneously

- Finding all fabrications proved challenging, as they were not immediately seen in the CeraPrint Tab. This however was solved by creating an array for all fabrication names where they were stored as they were created in CeraSlice.

- The code for looping the printing time was also not yet written.

Solving Encountered Issues

To input the duration times a manual process was created through a dialog box that requests the user to input the duration times per Layer and subsequently adds them to a list.

Encountered Issues

The matching of the Layers to the duration times without mixing them up still proved a challenge.

4.3.6. Fifth and Final Application

The finally improved automation goals within this final application are listed as such:

- Minimizing the CeraSlice app did function through the use of the "alt"+"tab" hotkey combination.

- Requesting for print duration for all Layers had finally worked as well. This came after an array was created to hold the printing durations and the code now requested the durations for all layers consecutively via a dialog box.

- Properly navigating to CeraPrint was now functional. As the hotkey solution proved inefficient, a windowsgui function was utilized to jump to the already opened CeraPrint Tab instantaneously

- Finding all fabrications proved challenging, as they were not immediately seen in the CeraPrint Tab. This however was solved by creating an array for all fabrication names where they were stored as they were created in CeraSlice.

- The code for looping the printing time was finally written. It takes in the durations and fabrication names already input previously and iterates through the lists matching each index to its corresponding counterpart.

Solving Encountered Issues

The finally issue was solved by sequentially matching the Layers and durations with the same list index in the code as seen in 4.4.

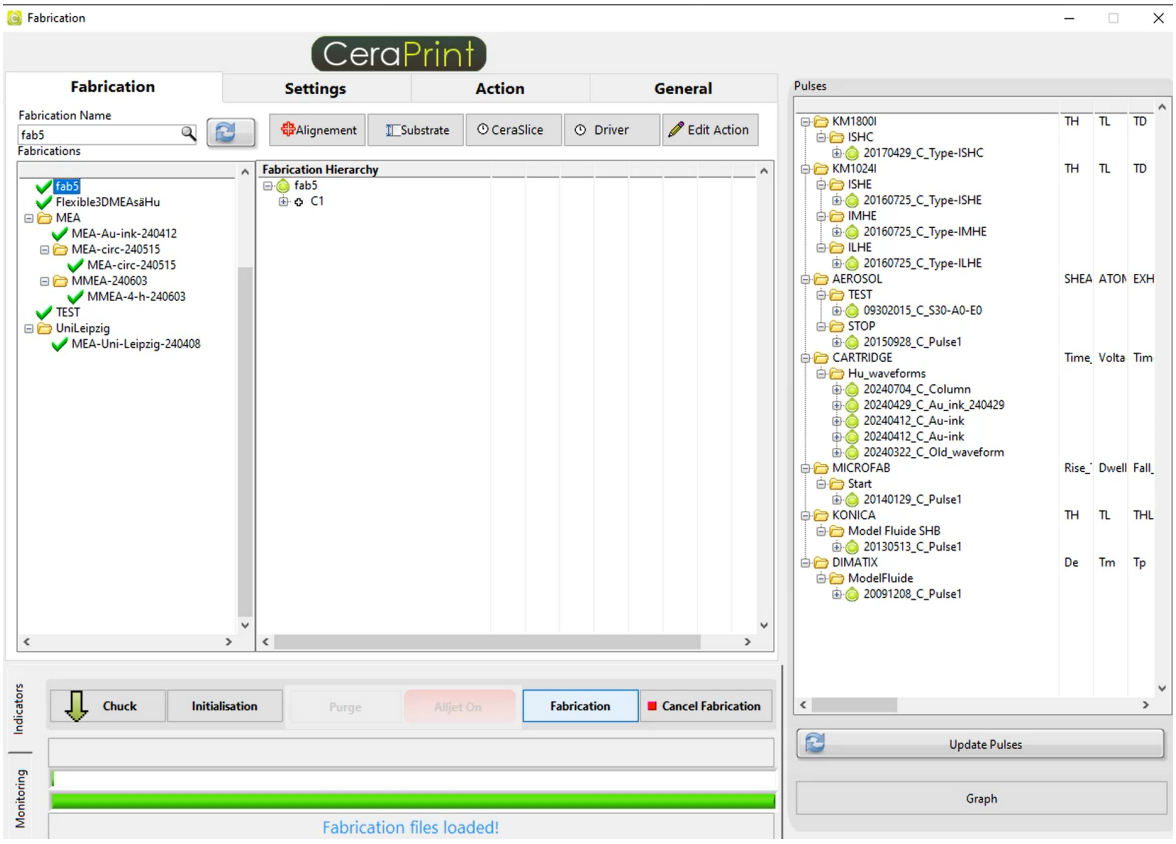


Figure 4.4.: All waiting times were accurate per Fabrication window

5. Conclusion

5.1. Algorithm Robustness and Application Limits

As indicated in the previous Chapter 3.3, the GUI Method leaves us with quite limited options which instead depend slightly more on the user than it would have in the case of an API. This means that all who utilize this script must strictly heed to the instructions provided in the Readme here Git Repo to prevent disruptive errors in the printing process, as the process is indeed a blind automation, leaving much less room for error and manipulation.

The User of this algorithm has to ensure that the CeraSlice software is activated and working without any hindrances before activating the automation script at all as there is no pausing it once it is running per Layer. There is however a pause prompt added to the code that allows the user to pause the process in the case an error was made in the layer selection phase. Should any error arise when adding the paths of the file layers, there is a pause function also inbuilt into the code, with an option to restart and add the proper path into the array where they are all collected, thereby ensuring the printing process is still salvaged.

They must also ensure that all Layers that are to be printed are available, accessible and in most cases stored within the same Folders in the system before being carefully selected with their paths copied, as the CeraSlice GUI has to compile these Layers accordingly and prints them all sequentially. Should any error arise when adding the paths of the file layers, there is a pause function also inbuilt into the code, with an option to restart and add the proper path into the array where they are all collected, thereby ensuring the printing process is still salvaged. It must also be made sure that once the script is triggered that no external influence is allowed to manipulate the keyboard or mouse of this Desktop.

Should however the printer have hardware issues or the nozzles do not print the material properly or if there are any physical errors as noted in Rinklin et al. [2], the algorithm has no means of correcting such malfunctions, as it is neither in the scope of this project to use cameras within the nozzle to check and for printing errors and then potentially make decisions in the algorithm, nor is it to interact with the printer directly without an API getting feedback before continuing the printing operation. It is thereby recommended that these factors all need to be controlled for before the script is utilized with the software to prevent unnecessary errors and wastage of material and power.

5.2. Possible Improvements

Implementation through a proper API would result in a much more efficient project. Should it be available for this software package, the API should have access to the CeraSlice software to slice the ensembles, the CeraPrint driver to select fabrications and print them and the F-series Printer to observe its status if an error has occurred along with a coded possibility to resolve them.

List of Figures

Bibliography

[1] Beck, Kent [Nov. 2002]: Test Driven Development, Addison Wesley Professional.

[2] Rinklin, Philipp, Tseng, Tsun-Ming, Liu, Cai, Li, Mengchu, Terkan, Korkut, Grob, Leroy, Adly, Nouran, Zips, Sabine, Weiß, Lennart, Schlichtmann, Ulf and Wolfrum, Bernhard [2019]: Electronic design automation for increased robustness in inkjet-printed electronics, Flexible and Printed Electronics **4**(4): 045002.
`https://dx.doi.org/10.1088/2058-8585/ab4c1c`

[3] Tseng, Tsun-Ming, Lian, Meng, Li, Mengchu, Rinklin, Philipp, Grob, Leroy, Wolfrum, Bernhard and Schlichtmann, Ulf [2021]: Manufacturing cycle-time optimization using gaussian drying model for inkjet-printed electronics, IEEE/ACM International Conference on Computer-Aided Design (ICCAD).
`https://mediatum.ub.tum.de/download/1638643/1638643.pdf`