Project: PDF to DOCX Converter

Python

Study Course

B144 Software Design & Modeling

By

Chehab Hany Mohamed Elsayed Elsayed Elmenoufi

Student Number: GH1034223

Under the Guidance of

Prof. Mazhar Hameed

GitHub URL: https://github.com/Chippo90/Software-Design-and-Modeling

Video Recording URL: https://youtu.be/WhVEsJ50ZKM

Gisma University of Applied Sciences

Berlin, Germany

September 2025

Table of Contents

1. Introduction

This project presents a simple desktop application designed to convert PDF documents into editable DOCX files. Built using Python and Tkinter, the tool provides a user friendly interface for selecting files, tracking conversion progress, and viewing word and page counts.

By leveraging libraries such as PyMuPDF and python-docx, the application ensures efficient text extraction and formatting. The project demonstrates core software engineering principles including design, threading for responsive UI, and integration of reusable functionalities.

2. Team Introduction

2.1. Project Manager: Chehab Hany

Developing the timeline, tracking tasks, and mitigating risks. For this project, task decomposition, milestone definition like file selection, threading, documentation, and time management were incorporated within a personal project schedule.

2.2. Software Engineer: Chehab Hany

Writing Python code for both the application and the reusable library component. This included implementing file handling, conversion and GUI interaction with tkinter.

3. Idea Pitching

3.1. Concept

The PDF to DOCX Converter is a desktop application designed to quickly and efficiently convert text based PDF files into editable Word documents. It provides users not only with the converted document but also useful statistics such as total word count, total page count, and conversion progress. Unlike many online converters, this application runs free and locally on your computer ensuring privacy and security of your documents.

3.2. Purpose

The primary goal of this application is to simplify the process of extracting editable content from PDF files, which are often hard to modify directly. It allows users to:

- Edit and reuse content from reports, research papers, or official documents.
- Quickly analyze large PDF files by counting words and pages.
- Track conversion progress for large documents in real time.

3.3. Intended Users

The application is designed for a wide range of users, including:

- **Students**: for converting lecture notes, assignments, or research papers.
- **Professionals**: for converting business reports, contracts, or presentations.
- **Writers and Editors**: for repurposing content from PDFs.
- **Anyone** who wants a fast, offline, and easy PDF conversion tool without relying on online and paid services.

4. Project Planning

S/N	Phase	Description	Tools	Number of Days
1	Idea & Requirements	Checking assessment brief and searching for ideas	Search Engines	2
2	Setup	Defining the coding language and installing it	PyCharm	1
3	Architecture	Drafting UML and drawing them	Draw.io	4
4	Features	Implementing the desired features	PyCharm	4
5	GUI	Implementing a GUI library	Python tkinter	2
6	Github	Final package and readme file	Github	1
7	Demo	Video recording showing my project	YouTube	1

5. Requirements Elicitation

5.1 User Story #1:

As a user, I want to select a PDF file from my computer so that I can easily convert it

Acceptance Criteria:

- Can open a file dialog to browse PDF files.
- File selection is seen and confirmed in the opened window.

5.2 User Story #2:

As a user, I want to convert the selected PDF file so that I can edit the contents in Microsoft Word

Acceptance Criteria:

- Convert button starts the conversion.
- The required DOCX file is created in the same directory.

5.3 User Story #3:

As a user, I want the conversion process to run in the background so that the application remains responsive.

Acceptance Criteria:

- During conversion the GUI remains interactive.
- Progress is updated visually in the main window.

5.4 User Story #4:

As a user, I want to see the total numbers of words and pages in the PDF file so I can analyze the document.

Acceptance Criteria:

- After selection, word and page count are displayed in the GUI.

5.5 User Story #5:

As a user, I want to monitor conversion progress and receive feedback on completion so that I can know if there's any error.

Acceptance Criteria:

- Progress status is displayed during conversion.
- Notification is displayed after completion.

5.6 User Story #6:

As a developer, I want the conversion logic to be implemented as a reusable library so that it can be imported and used in other projects.

Acceptance Criteria:

- Core PDF to DOCX function is in a separate module.

5.7 User Story #7:

As a user, I want to select the destination directory for the converted file so that I can organize my files.

Acceptance Criteria:

- User can browse and select any folder.

6. Refinement and Specification

To implement a functional application, several Python libraries were integrated:

- **Tkinter**: provides the graphical user interface, enabling intuitive user interaction.(*tkinter — Python interface to Tcl/Tk*, no date)
- **Filedialog and Messagebox**: support file selection and feedback dialogs,
- **Fitz**: handles PDF parsing and text extraction.(*Tutorial - PyMuPDF documentation*, no date)
- **Python-docx**: library is used to generate and write DOCX files.(*python-docx — python-docx 1.2.0 documentation*, no date)
- **Threading:** ensures the conversion process runs in the background without freezing the interface.(Python, no date)

The main logic is in my custom class "MyPDFConverter", which includes file handling and the conversion workflow. It manages user input, document transformation, and updates the interface with real-time progress and final statistics.

The features are as below in the table:-

S/N	Feature	Description	Input	Output
1	PDF Selection	Select PDF file from file browser	PDF file path	Path sotred in system
2	Text Extraction	Extract text from each page	PDF file	Page wise texts
3	Page Count	Count number of pages manually	PDF file	Total page count
4	Word Count	Count number of words manually	Page text	Total word count
5	Conversion to DOCX	Save extracted text in word document	Extracted text	File saved locally
6	Progress Status	Show progress percentage	Current page and total pages	Progress status shown in GUI
7	Display result	Show final stats and location of file	Page count, word count, and path	GUI popup with results

7. Modelling and Architecture

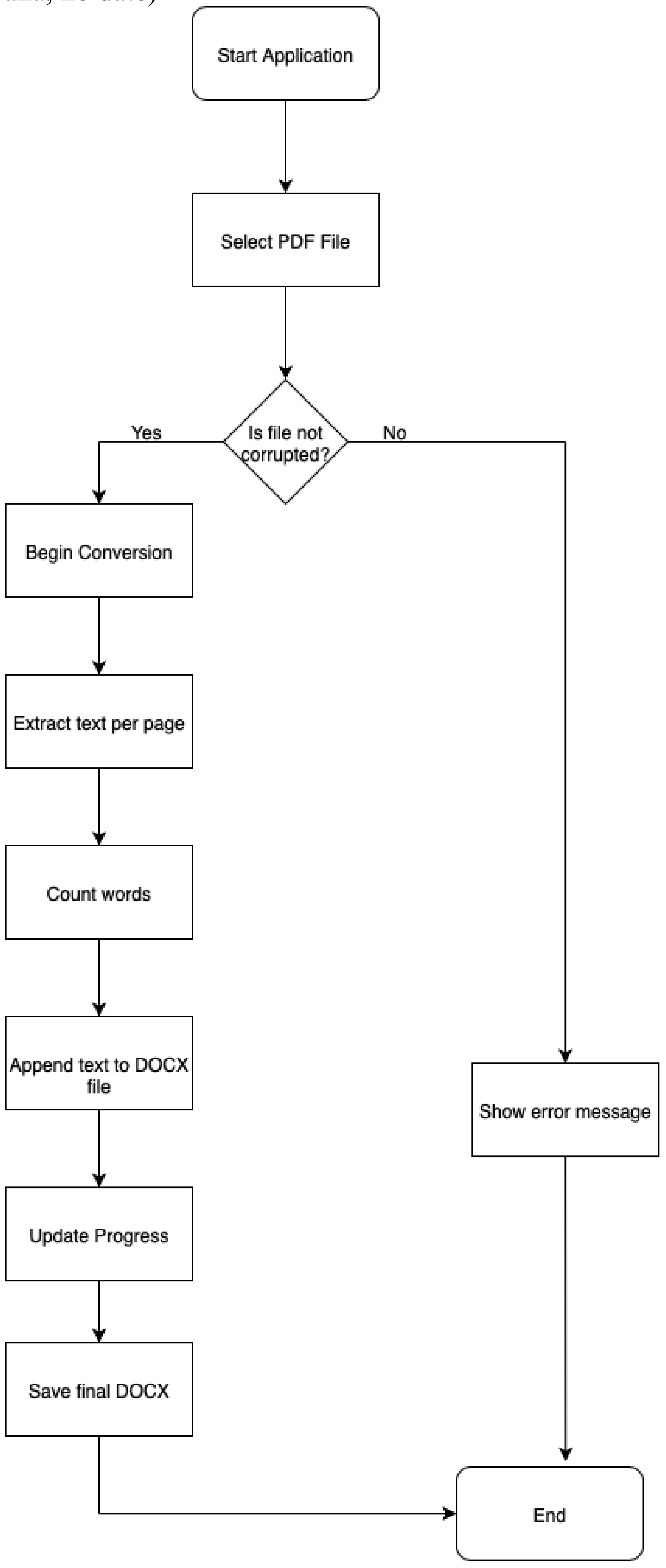
7.1 Architecture Overview

The system is divided into two main layers:

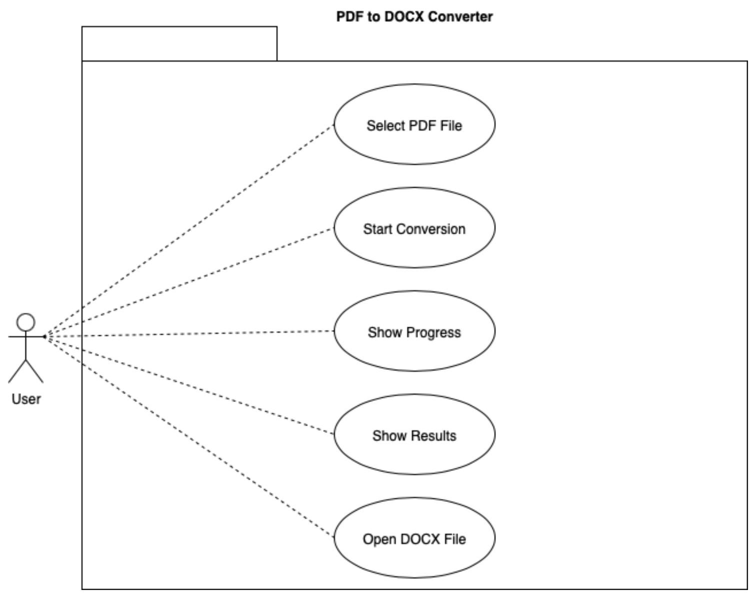
- Core Library (MyPDFConverter)
  - Handles the conversions process.
  - Can be reused in other python scripts.
- GUI Layer (Tkinter App)
  - Provides a user friendly interface for selecting PDF files.
  - Displays progress updates.

7.2 Workflow Diagram

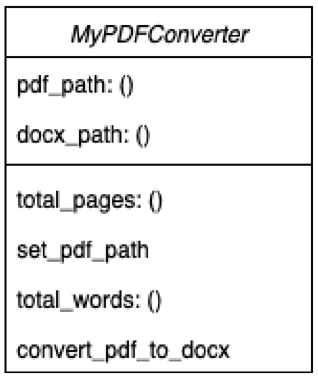
(Asana, no date)



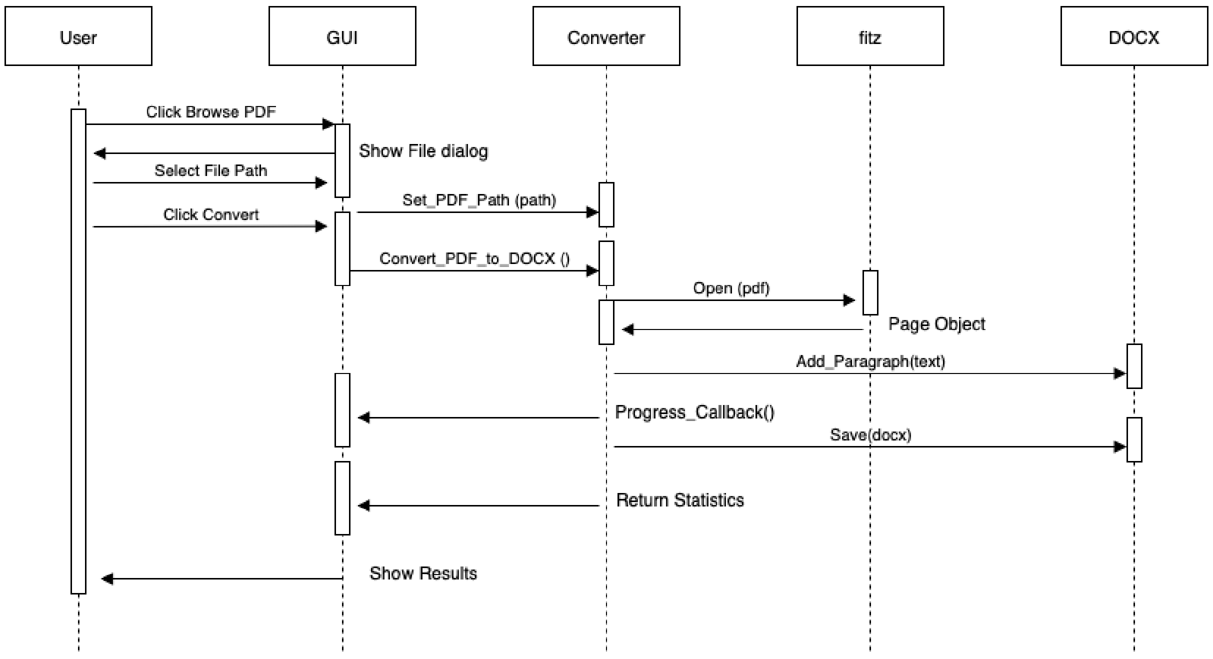
7.3 Use Case Diagram



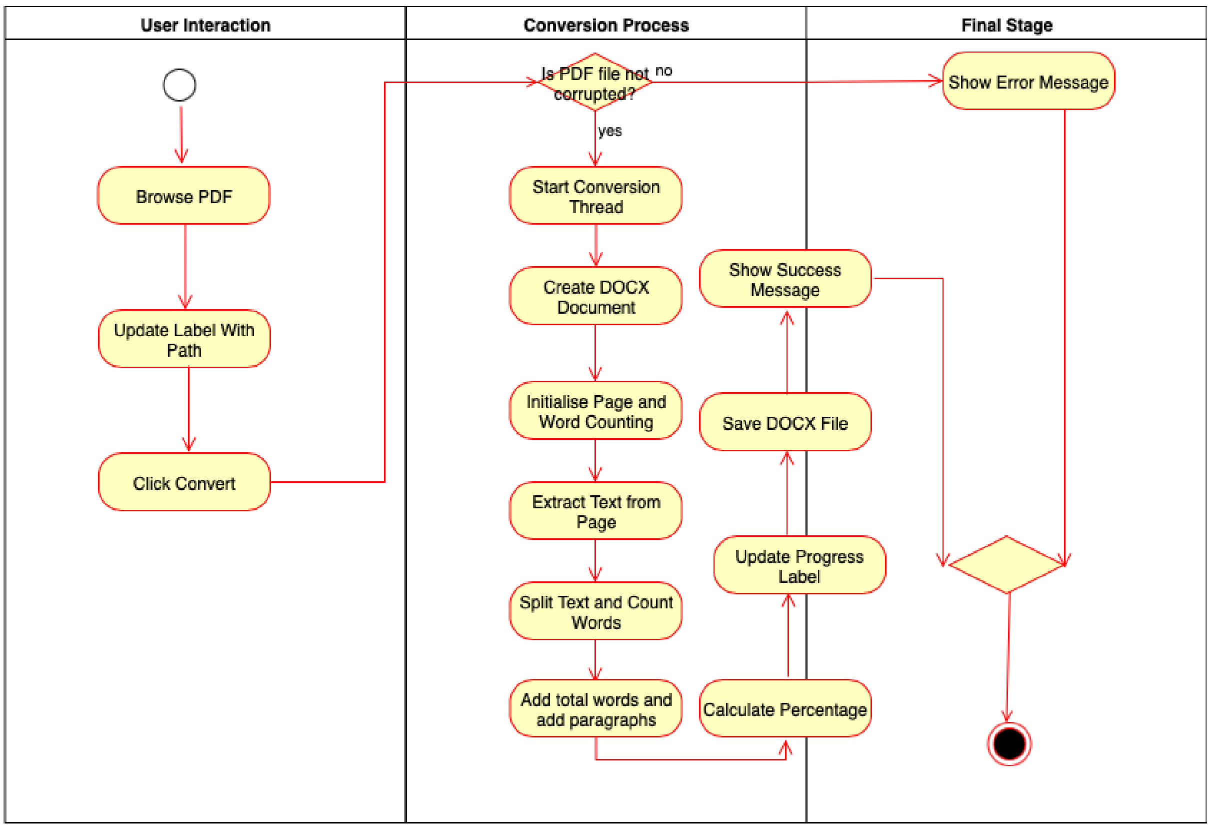
7.4 Class Diagram



7.5 Sequence Diagram



7.6 Activity Diagram



8. Reusable Code Library

8.1 Purpose

The MyPDFConverter class is designed not just as part of the GUI application, but as a standalone, reusable library for PDF to DOCX conversion. The main goal is to provide a simple interface to convert text based PDFs into editable Word documents, while also offering useful statistics such as total pages and total word count.

This design allows the core functionality to be used independently of the GUI, enabling other applications or scripts to leverage the same conversion logic.

8.2 Functionality

The library provides the following capabilities:

- **Manual Page Counting**: goes through page by page and count them without relying on any other dependency.
- **Custom Word Counting**: counts words using python built in (split() + len()).
- **Progress Feedback**: supports a callback function to report conversion progress which is useful for GUI updating.
- **Statistics Reporting:** results total words and pages count after conversion.

8.3 Reusability

The MyPDFConverter library can be reused in various ways:

- Standalone Python Scripts
- Integration in GUI Applications: GUI interacts with the library with a simple interface.
- Automation: Can be imported to automatically convert multiple PDF documents without any GUI.
- Extendable for Future Features: Can be extended for image and table conversion and to other formats without changing the GUI.

9. Conclusion

The PDF to DOCX converter successfully addresses the need for accessible document transformation by combining simplicity with functionality. Through design and implementation, the application showcases effective use of GUI development, background processing, and external libraries. This project not only fulfills its technical objectives but also reflects an understanding of software architecture, user experience, and maintaining code practices.

10. References

Asana (no date) *Workflow Diagrams Help Visualize Business Processes [2025]*, *Asana*. Available at: https://asana.com/resources/workflow-diagram (Accessed: September 17, 2025).

Python, R. (no date) *An Intro to Threading in Python – Real Python*. Available at: https://realpython.com/intro-to-python-threading/ (Accessed: September 17, 2025).

python-docx — python-docx 1.2.0 documentation (no date). Available at: https://python-docx.readthedocs.io/en/latest/ (Accessed: September 17, 2025).

tkinter — Python interface to Tcl/Tk (no date) *Python documentation*. Available at: https://docs.python.org/3/library/tkinter.html (Accessed: September 17, 2025).

Tutorial - PyMuPDF documentation (no date). Available at: https://pymupdf.readthedocs.io/en/latest/tutorial.html (Accessed: September 17, 2025).