Notexio — CSE323 (Operating Systems Design) Project Report

Project name: Notexio Text Editor
Course: CSE323 — Operating Systems Design
Student name: Nafisa Tabassum
Student ID: 2221466042
Section: 14
Instructor: Dr. Safat Siddiqui
Semester: Fall 2025
Submission date: 23.12.25
GitHub repository: https://github.com/nafisatabassum30/Notexio
GitHub Pages project page: https://nafisatabassum30.github.io/Notexio/
Demo video (2–5 min): https://youtu.be/5XnIDQdSf2A

---

1) Executive Summary

Notexio is a lightweight, customizable text editor built with **Python + Tkinter** to demonstrate operating-system concepts through a practical GUI application. The project focuses on **file I/O**, **state consistency (unsaved changes)**, **crash recovery via recovery files**, **cross-platform behavior (Windows/Linux input + printing)**, and **safe user experience patterns** (warnings, recent files, settings persistence).

---

2) Project Overview

2.1 Goals

bulletBuild a usable text editor with core “Notepad-like” operations.

bulletApply OS concepts in a real program: file handling, recovery, background work, resource management, and cross-platform compatibility.

bulletProvide a clean and modern UX: toolbar, status bar, themes, shortcuts.

2.2 Key Features Implemented (from the codebase)

bullet**File I/O:** New/Open/Save/Save As, recent files list, unsaved-changes warning.

bullet**Editing:** Undo/Redo, find/replace, go-to-line, clipboard operations.

bullet**View/UX:** Zoom, fullscreen, optional line numbers, status bar showing cursor position and counts.

bullet**Safety:** Recovery files + cleanup, optional auto-save loop.

bullet**Themes:** Light/dark mode and customizable colors.

bullet**Export:** Export document as **PDF** using ReportLab.

bullet**Platform support:** Windows drag-and-drop (optional `tkinterdnd2`), OS-specific printing (`win32print` on Windows, `lp/lpr` on Linux/macOS).

---

3) Architecture & Module Responsibilities

Notexio is organized as a modular Tkinter application where UI events call into feature managers:

bullet`main.py`: App composition and menu/shortcut wiring.

bullet`src/editor.py`: Main text widget, modified-state tracking, and core UI container.

bullet`src/file_manager.py`: File open/save logic + recent files persistence.

bullet`src/safety_features.py`: Recovery file creation/cleanup + background auto-save loop.

bullet`src/theme_manager.py`: Theme application across widgets.

bullet`src/ui_components.py`: Toolbar, status bar, line numbers UI.

bullet`src/edit_operations.py`: Find/replace/go-to-line and clipboard operations.

bullet`src/misc_features.py`: Print / print preview / export as PDF / drag & drop.

bullet`src/settings_manager.py`: JSON config read/write (`config/settings.json`).

Design choice: This separation makes each feature set isolated and testable, and mirrors OS design thinking (clear responsibilities, controlled interaction points).

---

4) Operating Systems Concepts Demonstrated

4.1 File I/O, Encoding, and Error Handling

Notexio reads and writes files using explicit encoding (`utf-8`). This touches OS-level concerns:

- bullet**File descriptors / handles:** OS resources that must be opened/closed correctly (Python context managers handle this reliably).
- bullet**Encoding correctness:** Prevents corrupted text and ensures consistent storage across machines.
- bullet**Failure modes:** Permission errors, missing files, locked files — surfaced to the user as dialogs.

4.2 Concurrency and UI Thread Safety (Auto-save / Recovery)

Tkinter requires UI updates to happen on the **main thread**. Notexio's auto-save runs in a **background thread** (daemon), which periodically writes recovery files.

Theory: GUI toolkits generally are not thread-safe because widget state is shared; touching it from multiple threads can cause race conditions or crashes. A safe design is:

- bulletBackground thread does **file I/O only** and avoids mutating widgets directly.
- bulletUI changes should be scheduled using the event loop (e.g., `root.after(...)`) if needed.

4.3 Crash Recovery and Data Durability

Recovery files are a simplified durability mechanism:

- bulletRegular snapshots of unsaved content are stored in `recovery/`.
- bulletOn startup, the app scans for `.recovery` files and offers restore.
- bulletOld recovery files are cleaned up to avoid disk growth.

Theory: This imitates OS/file-system reliability ideas (periodic checkpoints; reducing loss after abnormal termination).

4.4 Cross-platform Differences (Input + Printing)

- bulletMouse wheel events differ across OSs (`event.delta` vs `Button-4/5`).

bulletPrinting pipelines differ: Windows typically uses Win32 APIs; Linux/macOS uses `lp/lpr`.

This is an OS-design reality: identical user features often require platform-specific implementations.

---

5) Challenges & Fixes (STAR Format)

The course requirement asks for challenges described using **STAR**:

bullet**Situation** (context)

bullet**Task** (what needed to be done)

bullet**Action** (what I did)

bullet**Result** (outcome)

Challenge 1 — Recent Files menu opened the wrong file (late-binding lambda bug)

bullet**Situation:** I added an “Open Recent” submenu. Clicking a recent file sometimes opened the *last* file in the list, not the one clicked.

bullet**Task:** Fix the menu so every item opens its correct path.

bullet**Action:** I fixed the classic Python late-binding closure issue by capturing the current filepath in the lambda default argument:

- `command=lambda fp=filepath: open_file(fp)` instead of `command=lambda: open_file(filepath)`

bullet**Result:** Each menu entry consistently opens the correct file, improving usability and correctness.

Theory (why it happens): In Python, closures capture variables by reference. When the callback runs, `filepath` has already changed to the last loop value unless you bind it at definition time.

Challenge 2 — PDF export crashed on special characters like `&` or `<`

bullet**Situation:** Exporting to PDF failed on some documents; ReportLab raised parsing errors when the text had special characters.

bullet**Task:** Make “Export as PDF” robust for real text content.

bullet**Action:** I escaped reserved XML/HTML characters before feeding text into `reportlab.platypus.Paragraph`:

- & → `&amp;`, < → `&lt;`, > → `&gt;`

bullet**Result:** PDF export became stable for normal programming text and notes that include symbols.

Theory: ReportLab's `Paragraph` expects XML-like markup; unescaped characters can break parsing.

Challenge 3 — Auto-save could freeze or behave unpredictably (UI thread-safety)

bullet**Situation:** I added auto-save / recovery to protect user work. Early versions risked touching UI state from a background thread.

bullet**Task:** Keep auto-save reliable without breaking Tkinter's single-thread UI rules.

bullet**Action:** I designed auto-save as a background loop that performs **file writes only**, keeping UI rendering and widget updates inside the main event loop.

bullet**Result:** Auto-save runs without UI freezes, and recovery snapshots are produced in the background.

Theory: Tkinter widgets are not thread-safe; UI changes must happen on the main thread to avoid race conditions and crashes.

Challenge 4 — Mouse wheel scrolling didn't work consistently across Windows/Linux

bullet**Situation:** Scrolling worked on Windows but failed on Linux (or scrolled in the wrong direction).

bullet**Task:** Support scrolling across platforms with consistent behavior.

bullet**Action:** I implemented dual handling:

- Windows/macOS: `event.delta` - Linux: `Button-4` and `Button-5`

bullet**Result:** Scrolling works across platforms and improves the "feels like Notepad" UX.

Challenge 5 — Recovery files could grow unbounded and waste disk space

bullet**Situation:** Recovery snapshots are valuable, but frequent snapshots can flood disk storage over time.

bullet**Task:** Keep recovery useful while avoiding uncontrolled disk growth.

bullet**Action:** I implemented cleanup logic to keep only the newest N recovery files (default 10), deleting older ones based on modification time.

bullet**Result:** Recovery stays effective without filling storage, matching responsible OS resource management.

Theory: Storage is a finite OS resource. Good applications apply retention policies and clean up old artifacts.

Challenge 6 — Modified/unsaved state management (false positives and UX correctness)

bullet**Situation:** In text editors, “unsaved changes” must be accurate; false warnings annoy users, and missing warnings can cause data loss.

bullet**Task:** Track modified state correctly and integrate it with window title and exit prompts.

bullet**Action:** I used Tkinter’s `<<Modified>>` virtual event and maintained an `is_modified` flag to:

- add `*` to the title when content changes - prompt on exit or before opening a new file

bullet**Result:** Users get correct warnings and visual indicators, reducing accidental data loss.

---

6) Testing & Validation (What I Verified)

bulletOpen/Save/Save As for `.txt` and arbitrary extensions.

bulletRecent files list updates and opens correct file.

bulletUnsaved-changes prompt appears when expected.

bulletRecovery files are created and older ones are cleaned up.

bulletTheme toggles apply to major UI components.

bulletExport as PDF works for normal text and text containing <, >, &.

bulletMouse wheel works on Windows/Linux event models.

---

7) Limitations & Future Work

bulletAdd atomic-save strategy (write to temp + rename) to avoid partial writes if the app crashes mid-save.

bulletAdd unit tests for file operations and settings persistence.

bulletImprove print preview to render like a real page layout.

bulletPackage for distribution (PyInstaller) with icon + assets bundled.

---

8) Submission Checklist (What included on GitHub)

bullet`docs/index.md` (GitHub Pages landing page)

bullet`docs/Notexio_CSE323_Report.pdf` (the report PDF)

bullet`docs/report.md` (same report in Markdown for quick reading)

bulletA demo video link (YouTube/Drive) embedded/linked on `docs/index.md`

---

9) References

bulletTkinter documentation (event loop and `<<Modified>>` usage)

bulletReportLab documentation (`SimpleDocTemplate`, `Paragraph`)

bulletOS printing concepts (`lp/lpr`, Win32 print pipeline)