Faculty of Technology, Design & Environment

Assessment cover

OXFORD BROOKES UNIVERSITY

Module No: COMP5047 Module title: Applied Software Eng

Assessment title:	Software Engineering of a Modern Computer Application

Due date and time:	23:00pm, 5th Dec. 2025

Estimated total time to be spent on assignment:	84 hours per student

LEARNING OUTCOMES

On successful completion of this assignment, students will be able to achieve the module's following learning outcomes (LOs):

1. Demonstrate an understanding of the role of requirements analysis and specification in software engineering and to be able to use this knowledge to create use case models and functional models of computer applications.
2. Demonstrate an understanding of the relationship between requirements and design and to be able to apply the knowledge to create structural and behavioural models of computer applications.
3. Critically evaluate and utilise design paradigms of object-oriented analysis and design, component-based design, and service-oriented design.
4. Use software modelling language such as UML and modelling tools in the context of model-driven software engineering.
5. Work in a group to apply the knowledge and skills developed in this module

Engineering Council AHEP4 LOs assessed	
C3	Select and apply appropriate computational and analytical techniques to model complex problems, recognising the limitations of the techniques employed
C5	Design solutions for complex problems that meet a combination of societal, user, business and customer needs as appropriate. This will involve consideration of applicable health & safety, diversity, inclusion, cultural, societal, environmental and commercial matters, codes of practice and industry standards
C6	Apply an integrated or systems approach to the solution of complex problems
C14	Discuss the role of quality management systems and continuous improvement in the context of complex

problems

C16 Function effectively as an individual, and as a member or leader of a team

GROUP 10:	

STUDENT NAMES

	Student Id:	Student Name:	Subsystem:
1.	19310301	Malachai Broderick	USU Student App
2.	19320199	Ben Crosfield	Student Union Management System
3.	19324320	Lucas Oliveira	USU Operation System
4.	19267298	Tania Reyes	Society Leader App

Statement of Compliance *(please tick to sign)*

☐ We declare that the work submitted is my own and that the work I submit is fully in accordance with the University regulations regarding assessments *(www.brookes.ac.uk/uniregulations/current)*

RUBRIC OR EQUIVALENT:

School of Engineering, Computing & Mathematics

APPENDIX I.

Introduction

This piece of work covers the practical application of some of the principles of software engineering as part of the course COMP5047 - Applied Software Engineering, the aim of this coursework is to document the project management structure and software quality requirements of the “Society Leader App”, which is part of the larger United Student Unions system. The document in fact provides a detailed account of a collaborative structure of my team and the technical design of one functional requirement which are all in accord with the modules' lectures.

APPENDIX II.

Project Management

1. Team Structure

The project team has four members. Each member is responsible for one subsystem of the United Student Unions (USU) platform. The structure used is in direct accord with a distributed subsystem responsibility method of ensuring the modularity of development and the subsequent economy of integration processes, being as follows:

- Member A – USU Student App (mobile application for students).
- Member B – Student Union Management System (web interface for university-specific student unions).
- Member C – USU Operation System (federation management subsystem).
- Author – Society Leader App (mobile and web interface for society leaders).

My project team agreed to work independently but we agreed to carry out weekly meetings in person and to maintain all project documentation in overall common use for the project team via a shared GitHub Repository.

2. Group collaboration

The project team arrived at some clearly defined media for collaboration to maintain consistency and equitable clarity. This consisted of:

- A GitHub repository used as the central hub for documentation.
- Weekly in-person meetings to coordinate subsystem integration and progress.

- A WhatsApp group for real-time communication.
- Version control practices to ensure traceability and accountability.

3. Meeting Schedule and Minutes

Week	Date	Type	Topics	Attendance	Time and Place
1	25/09	Kick-off meeting	Team structure, GitHub setup, subsystem allocation	All members	12:00–13:00, AB-115, Headington Campus
3	09/09	Progress meeting	Review case study, discuss quality attributes, integration plan	Two members	12:00–13:00, AB-115, Headington Campus
4	14/10	Planning meeting	Individual progress on Task 1 and 2, coordination for Week 5 delivery	Two members	12:00–13:00, AB-115, Headington Campus

The author participated in the first meeting and reviewed meeting notes from the third via GitHub.

4. Agenda Template

- Attendance check.
- Review of previous actions.
- Subsystem progress updates.
- Blockers and integration concerns.
- Action items and deadlines.

5. Action Tracking Example

Member	Action	Deadline
1	Define use cases and initial quality requirements for Student App subsystem	Week 5
2	Define use cases and quality attributes for Union Management System subsystem	Week 5
3	Define use cases and quality attributes for Operation System subsystem	Week 5
Author	Define use cases and quality attributes for Society Leader App (FR-SL-2: Event Creation)	Week 5
All	Review each other's quality requirement drafts and integrate feedback	Week 5

6. Project Management Approach

The agile methodology was adopted by the team, integrating short cycles of core team coordination with formal documentation practices. Responsibilities were decided and got defined as balanced by subsystem owners, provided so that they can focus on their own subsystem growth. The team meets weekly to enable fixing of engaging effort without overloading any team with workload.

APPENDIX III.

Software Quality Requirements

1. Functional Requirement Overview

FR-SL-2: The system will enable society leaders to create events by providing information on the events including the name of the event, theme of the event, details of the organising agent, time and date of the event, location of event, coverage of event promotion and promotion material. The successful creation of an event will tell us if the event will be up and promoted to the extent defined in the USU system. The suggested functionality relates to what is believed to be close to student engagement and total system calling into an event. This was analysed in the case study and the relative quality attributes referred to in lecture 2 (Zhu, 2025) sourced from ISO/IEC 25010 (2011).

2. Quality Requirements Specification

a) Security and Privacy Protection

Security and privacy are important aspects of user information gained from user effort and organisational databases. The requirements for this subsystem are considered from the principles outlined in lecture 2, and compatible with ISO/IEC 25010 (2011). The following events will take the following measures:

- Events creation activity action should only be performed by bona fide society leaders, and as such comprehensive action on the basis of access control to verified roles should be carried out.
- Data transmission will be harnessed using HTTPS protection on the basis of TLS 1.3.
- Event data will be stored on the basis of storage security assurances with encryption.
- Systemic action including events nor user information will go unlogged to the extent of total accountability. Auditability will be achieved.

b) Performance

Requirements for performance will lead to delivery of a fast experience from the Society Leader App for users, to the extent that the individual experience is a fast action, particularly meaningful during stressful peak usage periods. The Research guidelines of the lectures will be followed (Lecture 2):

- Request for event creation will have the confirmation returned to the user in the space of 2 seconds provided a normal load of request demands.
- Uploading for promotional material (≤ 10 MB) should occur in the space of 5 seconds.
- Asynchronous processing would be required for applications to enable user responsiveness.
- Naturally the fluidity of the UI should be available during data transmission.

c) Reliability

Reliability as such suggests that stability of platform and data systems are guaranteed during total event creation. The principles of lecture 2 should be followed:

- The availability of the system in question should be not less than 99.5% during academic periods.
- Total establishment of the event must be on the basis of atomic transaction type by event (the event creation).
- The results of failure will cause rollback of the result and clear notifications of specific fault necessitated.
- Event data will be stored with sources of information and data systems across Cloud Servers.

d) Scalability

Necessary are the methods to ensure growth in kind of utilisation to be gained at more and more universities. Considerations from lecture 1 and 2 are preferable:

- The system should cope accordingly with an abundance of (not more than) 100,000 concurrent society leaders.
- Versions of the systems infrastructure should expand on a horizontal basis to cover peak demand during event creations and tested estimated maximum periods.
- Processes for the calling of events should lead dynamic scalability in an audience of considerable range.
- The extent of cached data should be available from the system to ensure work performed will not lead to lapses in performance outcomes.

3. Summary Table of Quality Requirements

Attribute	Requirement Highlights	Reference
Security & Privacy	TLS 1.3, encryption, role-based access, logging	Lecture 2; ISO/IEC 25010 (2011)
Performance	≤ 2 s response, async processing, responsive UI	Lecture 2
Reliability	99.5 % availability, atomic transactions, error rollback, redundancy	Lecture 2.
Scalability	100 k concurrent users, horizontal scaling, distributed notifications, caching	Lecture 1 and 2.

APPENDIX V.

Implementation Plan

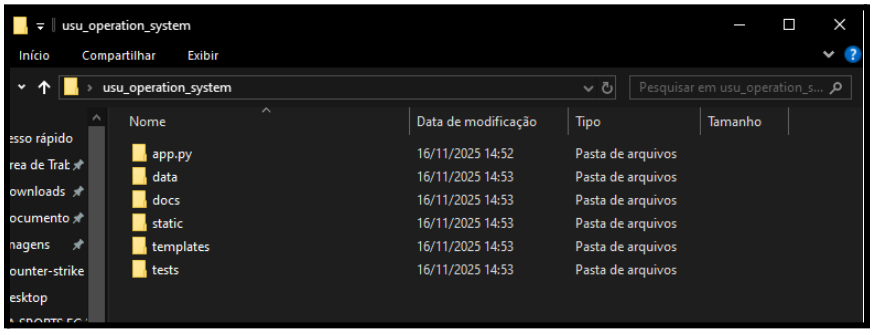
1. Subsystem Purpose

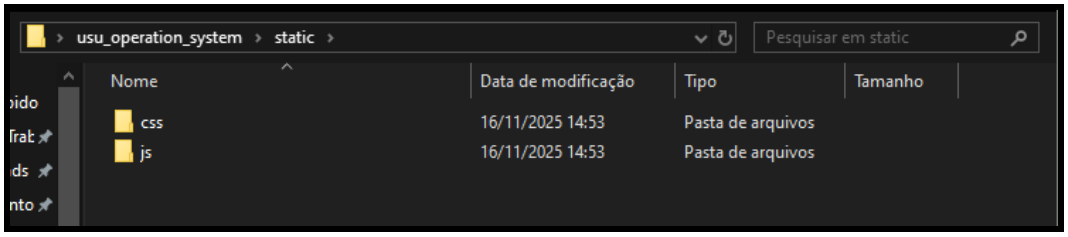
The USU Operation System is a subsystem that will serve as an administrative interface for all USU officers to oversee and operate the national network of university student unions. The subsystem will include the core functionalities of approval of union applications, creation and publication of events and synchronization between the subsystems (Student App, Union Management System and Society Leader App).

2. Technology Stack

Python 3 + Flask (web framework), SQLite (local database), HTML5/CSS/JavaScript (front-end), and GitHub for version control and collaboration.

3. Folder structure





4. Development timeline

Week	Goal
6 - 7	Set up Flask project skeleton and local DB schema for unions and events.
8 - 9	Implement core routes: login, view applications, approve/reject unions.
10	Add event creation and announcement features; begin integration tests.
11 - 12	Final integration with other subsystems and documentation.

5. Testing strategy

- Unit tests using Python's unittest for validating forms and performing operations on databases.
- Manual black box testing to confirm each functionality works as anticipated.
- Integration testing against mock API endpoints from other subsystems.

All team members maintain their own branches in github and commit to their branch after every functional change made and the commit message includes what was changed.

APPENDIX VI.

Prototype / Evidence of Progress

A minimal prototype has been implemented in Flask to demonstrate that the subsystem environment is functional, this prototype includes routes for the home page and an approval endpoint.

It includes three basic routes that demonstrate the core behaviour of the subsystem:

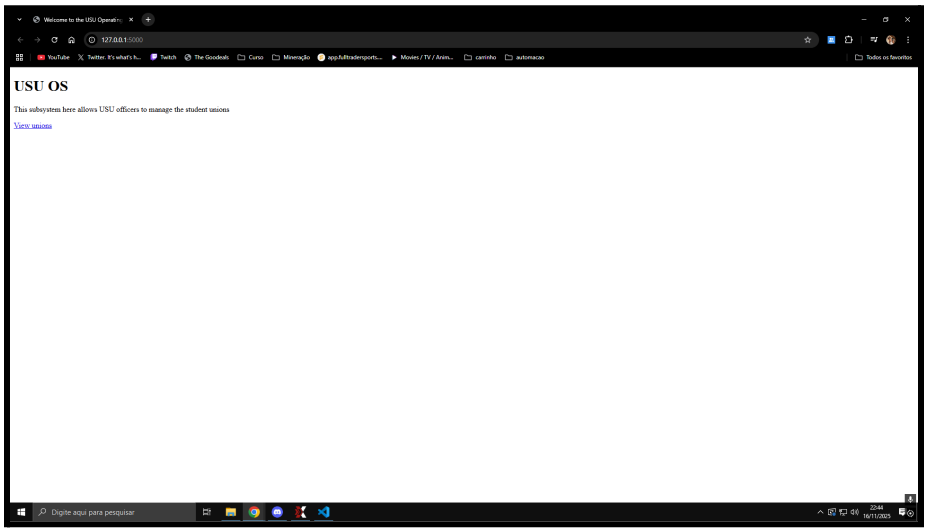
- / - Shows the home page of the USU Operation System.

- /unions - Displays a list of student unions stored in memory.

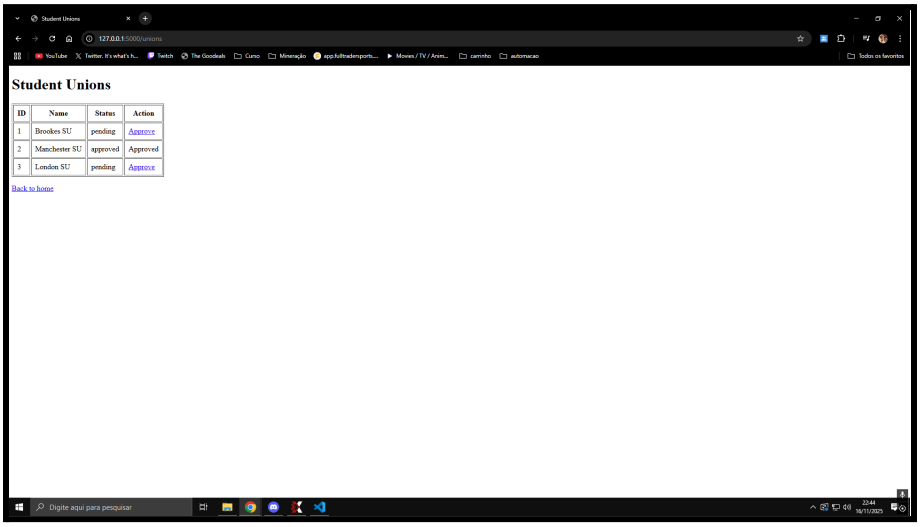
- /approve/<id> - Updates the status of a selected union from “pending” to “approved” and redirects the user back to the list.

These routes confirm that the subsystem can start correctly, respond to requests and modify its internal state. Although the data is currently stored in memory, this behaviour is consistent with the Implementation Plan and provides a foundation for connecting to SQLite in the next development phase.

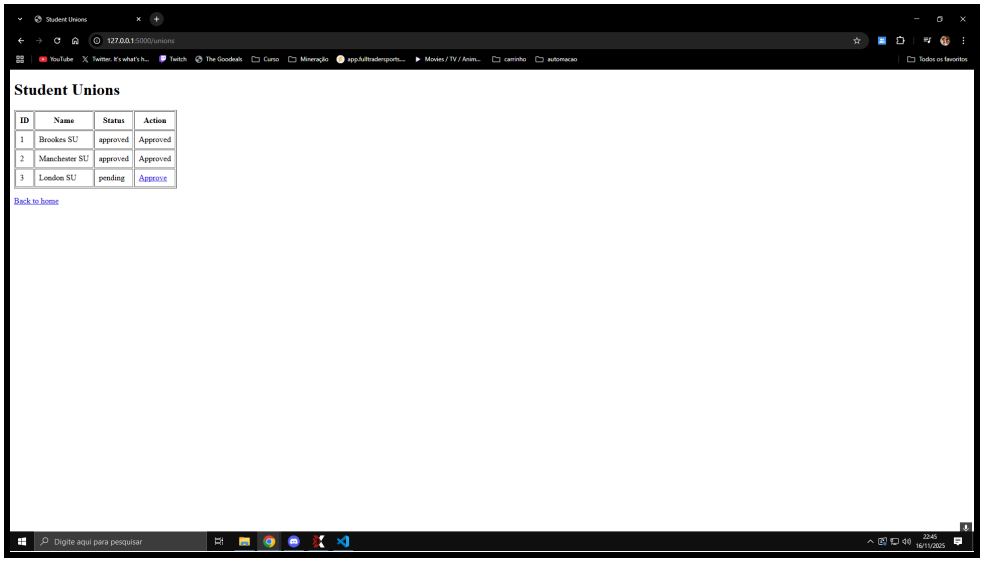
Screenshots demonstrating the prototype environment, route execution and server output are included below.



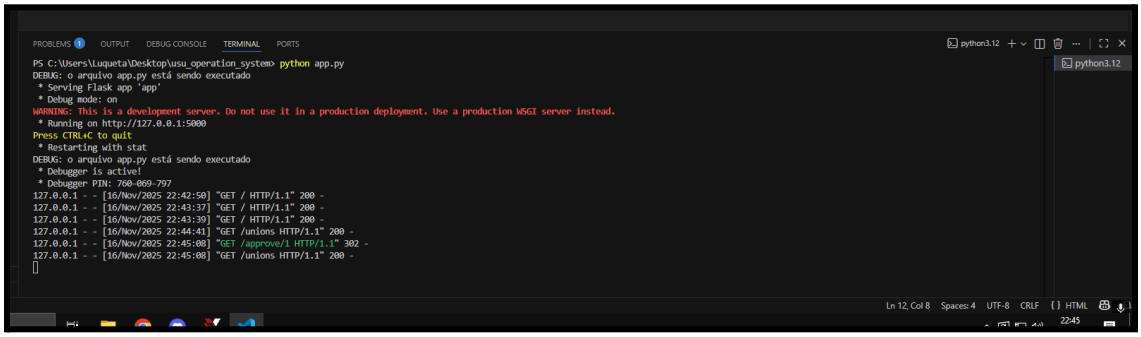
Screenshot 1 - USU Operation System home page.



Screenshot 2 - Student unions list before approving.



Screenshot 3 - Student unions list after approving.



Screenshot 4 - Flask server running in the terminal.

APPENDIX VII.

Reflection and Evaluation

1. Project Management Reflection

In the first part of the semester, we used an agile method to coordinate our work together via GitHub and WhatsApp. After that point, we began to use this process less frequently; however, we continued to document our work and create plans so we could continue where we left off at that time. At that point, my focus changed to ensuring my portion of the project remained on schedule and to produce the appropriate outputs by myself. In doing so, I was able to illustrate how important it is to have clearly defined roles within a group, to follow best practices when using version control systems, and to communicate regularly so you do not experience delays when integrating individual pieces of a larger project.

2. Design and Quality Evaluation

The core entities of the Operation System (Union, Officer, and Event) were modeled with encapsulation, abstraction, and inheritance to enhance maintainability. The architecture follows the single responsibility principle within a framework.

Evaluation of quality follows the ISO 9126/25010 model focusing on function, security, and reliability. We did acknowledge trade-offs. For instance, while encrypting and authenticating all data can be very efficient and secure, it does provide a reduction in efficiency; however, it is a requirement for protecting user's data. The design follows the concepts of separation of concerns and abstraction as outlined in Lecture 8 to manage the complexity of the system.

3. Implementation Reflection

The Implementation Plan (Task 3a) provided a structure for the project layout of the Flask project and the primary routing for the project (/ and /approve). Task 3b will demonstrate the creation of a prototype to show how the routing functions in the application when connected to SQLite and using HTML templates. This is consistent with the module's definition of detailed design, which progresses from the overall architecture to the specific algorithms and interfaces to implement the architecture. Once the routes are created, unit testing and manual validation will be performed to confirm the routes return the anticipated response(s). The results will be included in the final evaluation.

4. Software Quality Assessment

Quality Attribute	Design decision / Evidence	Reflection
Functionality	CRUD operations and synchronisation with other subsystems.	Core features match requirements; integration pending.
Reliability	Flask error-handling logic and transaction consistency.	Adopts Lecture 8 patterns (Circuit Breaker, Recovery) for fault isolation.
Usability	Simple HTML templates for officers.	Meets user needs while minimising interface complexity.
Efficiency	Lightweight SQLite database and parameterised queries.	Expected response < 2 s under normal load.
Maintainability	Modular folders (models, controllers, tests).	Facilitates future updates and debugging.
Security	Gate Keeper / Access Token concept for authentication.	Reinforces Lecture 8 security patterns and KF-5 privacy goal.

5. Learning and Future Improvements

This project has enhanced my knowledge of requirements engineering, modular architecture, and domain driven design. By applying the principles of microservices and object oriented design from Lecture 8, I was able to translate quality criteria into actual design decisions. Continuing to develop the project independently improved my self-management and documentation skills. Future revisions of the project should incorporate automated testing, peer review, and deployment monitoring to help me enable continuous integration and resiliency.

APPENDIX IV.

References

The references listed below reflect the core content of the COMP5047 lectures and coursework, with ISO/IEC 25010 (2011) cited as supporting material for software quality attributes.

Zhu, H. (2025) COMP5047 Lecture 1: The Nature of Software. Oxford Brookes University. [Powerpoint slides].

Zhu, H. (2025) COMP5047 Lecture 2: Software Quality. Oxford Brookes University. [Powerpoint slides].

Zhu, H. (2025) COMP5047 Lecture 3: Use Case and Scenario Analysis. Oxford Brookes University. [Powerpoint slides].

Zhu, H. (2025) COMP5047 Lecture 4: Activity Diagrams. Oxford Brookes University. [Powerpoint slides].

Oxford Brookes University (2025) COMP5047 Coursework Specification. Oxford Brookes University. [Module document].

Oxford Brookes University (2025) COMP5047 Case Study - Student Unions. Oxford Brookes University. [Module document].

ISO/IEC (2011) Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models (ISO/IEC 25010:2011). Geneva: International Organization for Standardization.

Brooks, F. P. (1987) No Silver Bullet: Essence and Accidents of Software Engineering. IEEE Computer, pp. 10–19.

Evans, E. (2003) Domain-Driven Design – Tackling Complexity in the Heart of Software. Addison-Wesley Professional.

ISO/IEC 9126-1 (2001) Software Engineering — Product Quality — Part 1: Quality Model. Geneva: International Organization for Standardization.

Zhu, H. (2025) COMP5047 Lecture 8: Software Design – Structural Design. Oxford Brookes University. [Powerpoint slides].