Project Title: System Verification and Validation Plan for Course Buddy

Team #5, Overwatch League
Jingyao, Qin
Qianni, Wang
Qiang, Gao
Chenwei, Song
Shuting, Shi

November 3, 2023

Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

[The intention of the VnV plan is to increase confidence in the software. However, this does not mean listing every verification and validation technique that has ever been devised. The VnV plan should also be a **feasible** plan. Execution of the plan should be possible with the time and team available. If the full plan cannot be completed during the time available, it can either be modified to "fake it", or a better solution is to add a section describing what work has been completed and what work is still planned for the future. —SS]

[The VnV plan is typically started after the requirements stage, but before the design stage. This means that the sections related to unit testing cannot initially be completed. The sections will be filled in after the design stage is complete. the final version of the VnV plan should have all sections filled in. —SS]

Contents

List of Tables

[Remove this section if it isn't needed —SS]

List of Figures

[Remove this section if it isn't needed —SS]

1 Symbols, Abbreviations, and Acronyms

symbol	description
T	Test

[symbols, abbreviations, or acronyms — you can simply reference the SRS (**?**) tables, if appropriate —SS]

[Remove this section if it isn't needed —SS]

This document ... [provide an introductory blurb and roadmap of the Verification and Validation plan —SS]

2 General Information

2.1 Summary

Course Buddy is a scheduling software that generates personalized study plans for students from course outlines that would be incorporated into the student's own schedule. The application also comes with social components that enable friend-collaborative study sessions.

2.2 Objectives

This VnV plan would direct us through the testing process to ensure that our system would accomplish all the intended functions correctly and efficiently. Adequate tests would be planned according to functional and non-functional requirements stated in our SRS to ensure our project exhibits adequate qualities and functions. We would prioritize and polish core functions including PDF unloading and analysis, AI scheduling algorithms, compatibility with other calendar applications, and those associated non-functional requirements. If time allows, we would also aim to develop tests for AI attention detection and the social component.

2.3 Relevant Documentation

2.3.1 Development plan

As part of the development process, the writing of this VnV plan implementation of all the test cases shall follow the development process specified in the development plan.

2.3.2 SRS

The functional and non-functional requirements and their corresponding fit criteria would lead the test plan and be used as a guideline to make sure tests could cover all the documented requirements.

2.3.3 Hazard Analysis

Additional requirements related to software failures should also be covered in testing.

2.3.4 VnV Report

The result from executing this VnV plan would be recorded in the VnV report. The VnV plan could be revised from the results.

3 Plan

[Introduce this section. You can provide a roadmap of the sections to come. —SS]

3.1 Verification and Validation Team

[Your teammates. Maybe your supervisor. You should do more than list names. You should say what each person's role is for the project's verification. A table is a good way to summarize this information. —SS]

3.2 SRS Verification Plan

[List any approaches you intend to use for SRS verification. This may include ad hoc feedback from reviewers, like your classmates, or you may plan for something more rigorous/systematic. —SS]

[Maybe create an SRS checklist? —SS]

3.3 Design Verification Plan

[Plans for design verification —SS]

[The review will include reviews by your classmates —SS]

[Create a checklists? —SS]

3.4 Verification and Validation Plan Verification Plan

[The verification and validation plan is an artifact that should also be verified. Techniques for this include review and mutation testing. —SS]

[The review will include reviews by your classmates —SS]

[Create a checklists? —SS]

3.5 Implementation Verification Plan

[You should at least point to the tests listed in this document and the unit testing plan. —SS]

[In this section you would also give any details of any plans for static verification of the implementation. Potential techniques include code walkthroughs, code inspection, static analyzers, etc. —SS]

3.6 Automated Testing and Verification Tools

[What tools are you using for automated testing. Likely a unit testing framework and maybe a profiling tool, like ValGrind. Other possible tools include a static analyzer, make, continuous integration tools, test coverage tools, etc. Explain your plans for summarizing code coverage metrics. Linters are another important class of tools. For the programming language you select, you should look at the available linters. There may also be tools that verify that coding standards have been respected, like flake9 for Python. —SS]

[If you have already done this in the development plan, you can point to that document. —SS]

[The details of this section will likely evolve as you get closer to the implementation. —SS]

3.7 Software Validation Plan

[If there is any external data that can be used for validation, you should point to it here. If there are no plans for validation, you should state that here. —SS]

[You might want to use review sessions with the stakeholder to check that the requirements document captures the right requirements. Maybe task based inspection? —SS]

[For those capstone teams with an external supervisor, the Rev 0 demo should be used as an opportunity to validate the requirements. You should plan on demonstrating your project to your supervisor shortly after the scheduled Rev 0 demo. The feedback from your supervisor will be very useful for improving your project. —SS]

[For teams without an external supervisor, user testing can serve the same purpose as a Rev 0 demo for the supervisor. —SS]

[This section might reference back to the SRS verification section. —SS]

4 System Test Description

4.1 Tests for Functional Requirements

[Subsets of the tests may be in related, so this section is divided into different areas. If there are no identifiable subsets for the tests, this level of document structure can be removed. —SS]

[Include a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. —SS]

4.1.1 Area of Testing1

[It would be nice to have a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. If a section covers tests for input constraints, you should reference the data constraints table in the SRS. —SS]

Title for Test

1. test-id1

   Control: Manual versus Automatic

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

2. test-id2

   Control: Manual versus Automatic

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

4.1.2 Area of Testing2

...

4.2 Tests for Nonfunctional Requirements

4.2.1 Look and feel

1. TAR-1

   Type: Dynamic, Manual

   Initial State: The web application is launched.

   Input/Condition: Users are asked to read the text on each page.

   Output/Result: `MIN_UNDERSTAND%` of users reporting that all the texts are legible.

   How test will be performed: The web application displayed with screens with various resolutions and sizes. `MIN_TESTER_NUM` users are asked to read the texts and report any unclear typography.

2. TAR-2

   Type: Dynamic, Manual

   Initial State: The web application is launched.

   Input/Condition: Users are asked to observe color on each page.

   Output/Result: `MIN_UNDERSTAND%` of users comfortable with the color used.

   How test will be performed: `MIN_TESTER_NUM` users are asked to observe the color and report any distracting or overwhelming color palette.

3. TAR-3

   Type: Dynamic, Manual

   Initial State: The web application is launched.

   Input/Condition: Users are asked to interpret icons and graphics on each page.

   Output/Result: `MIN_UNDERSTAND%` of users able to understand the meaning of each icon and graphic as designed.

How test will be performed: MIN_TESTER_NUM users are asked what they think each icon and graphic means. Any mismatch from the UI designer's intention would be recorded.

4. TAR-4

   Type: Dynamic, Manual

   Initial State: The web application is launched.

   Input/Condition: Users are asked to observe the layout of each page.

   Output/Result: MIN_UNDERSTAND% of users comfortable with the payout.

   How test will be performed: The web application is launched with different devices including smartphones, tablets, and desktops. MIN_TESTER_NUM users are asked to point out any unresponsive, out-of-margin, or compressed elements.

5. TSTR-1

   Type: Dynamic, Manual

   Initial State: The web application is launched.

   Input/Condition: Users are asked to point out the menus and navigation bars.

   Output/Result: MIN_UNDERSTAND% of users able to locate menus and navigation tools without confusion.

   How test will be performed: MIN_TESTER_NUM users are asked to point out the menus and navigation bars without assistance.

6. TSTR-2

   Type: Dynamic, Manual

   Initial State: The web application is launched.

   Input/Condition: Users are asked to point out inconsistent UI elements.

   Output/Result: MAX_ELEMENT_BAD inconsistent UI elements found.

   How test will be performed: MIN_TESTER_NUM users are asked to explore the application and point out any inconsistent UI elements that break the harmony.

7. TSTR-3

   Type: Dynamic, Manual

   Initial State: The web application is launched.

   Input/Condition: Users are asked to click each interactive UI element.

   Output/Result: `MAX_ELEMENT_BAD` interactive UI elements being unresponsive.

   How test will be performed: `MIN_TESTER_NUM` users are asked to explore the application and report any unresponsive interactive elements.

8. TSTR-4

   Type: Dynamic, Manual

   Initial State: The web application is launched.

   Input/Condition: Users are asked to adjust font size.

   Output/Result: `MAX_UNSATISFIED%` of users reporting unsuitable font size.

   How test will be performed: `MIN_TESTER_NUM` users are asked to report how they like the default font size and attempt to adjust the font size to their preference. Report if the adjusted font still causes difficulty in reading.

9. TSTR-5

   Type: Dynamic, Manual

   Initial State: The web application is launched.

   Input/Condition: Users are asked to perform a task while an animation is displayed.

   Output/Result: `MIN_UNDERSTAND%` of users who could perform the task distracted.

   How test will be performed: `MIN_TESTER_NUM` users are asked to perform a task that involves the animation display part of the screen. They are asked to report if they feel the animation is intrusive or distracting.

4.2.2 Usability and Humanity

1. TUHR-1

   Type: Dynamic, Manual

   Initial State: The web application is launched.

   Input/Condition: Users are asked to navigate to a required page.

   Output/Result: `MIN_UNDERSTAND%` of users able to navigate to a required page without confusion.

   How test will be performed: `MIN_TESTER_NUM` users are asked to navigate to a required page without assistance within a reasonable amount of time undergoing minimum trial and error.

2. TUHR-2

   Type: Dynamic, Manual

   Initial State: The web application is launched.

   Input/Condition: Users are asked to perform a core function.

   Output/Result: `MIN_UNDERSTAND%` of users able to perform a core function.

   How the test will be performed: `MIN_TESTER_NUM` users are asked to perform a core function like generating a study plan without assistance within a reasonable amount of time undergoing minimum trial and error.

3. TUHR-3

   Type: Dynamic, Manual

   Initial State: The web application is launched.

   Input/Condition: Users are asked to inspect the grammar of texts on the page.

   Output/Result: `MAX_BAD_GRAMMAR` grammar mistakes found.

   How test will be performed: `MIN_TESTER_NUM` users are asked to report grammar mistakes in all the text visible in this web application.

4. TUHR-4

Type: Dynamic, Manual

Initial State: The web application is launched.

Input/Condition: Users are asked to inspect for offensive messages on the page.

Output/Result: `MAX_OFFENSIVE` offensive messages found.

How test will be performed: `MIN_TESTER_NUM` users are asked to report offensive messages in all the text visible in this web application.

5. TUHR-5

   Type: Dynamic, Manual

   Initial State: The web application is launched.

   Input/Condition: Users are asked to observe color combinations on the page.

   Output/Result: `MAX_COLOR_AMBIGUOUS` indistinguishable color combinations found.

   How test will be performed: `MIN_TESTER_NUM` users are asked to report indistinguishable color combinations in this web application. This user group should include people with color blindness.

4.2.3 Performance

1. TSLR-1

   Type: Dynamic, Automatic

   Initial State: The web application is launched.

   Input/Condition: The processing time for a core function.

   Output/Result: It takes `MAX_TIME_PROCESS` for the application to finish a core function.

   How test will be performed: An automated script is used to perform and time a core function, including uploading syllabuses, generating tasks, and prioritizing tasks.

4.2.4 Safety-Critical

1. TSCR-1

   Type: Dynamic, Automatic

   Initial State: The web application is launched.

   Input/Condition: A pair of new credentials are added.

   Output/Result: Format of credentials in database.

   How test will be performed: An automated script is used to add a pair of new credentials and check whether plain text is stored in the database.

4.2.5 Precision and Accuracy

1. TPAR-1

   Type: Dynamic, Manual, Automated

   Initial State: The web application is launched.

   Input/Condition: A course outline is uploaded.

   Output/Result: `MIN_PRECISION%` of coincidence between algorithm result and human result.

   How test will be performed: An automated script is used to upload a course outline and categorize and prioritize extracted tasks. `MIN_TESTER_NUM` users are asked to categorize and prioritize extracted tasks from the same course outline. The computed result is compared with human decisions.

4.2.6 Robustness and Fault-Tolerance

1. TRFTR-1

   Type: Dynamic, Automated

   Initial State: The web application is launched.

   Input/Condition: Incorrect user input.

   Output/Result: `MIN_OPERABLE%` of the system operating.

How test will be performed: An automated script is used to put false input in every interactive element, including wrong file format, special characters, out-of-boundary data, and malicious commands.

4.2.7 Capacity

1. TCR-1

   Type: Dynamic, Automated

   Initial State: The web application is launched.

   Input/Condition: Massive user operations.

   Output/Result: `MIN_OPERABLE%` of the system operating.

   How test will be performed: An automated script is used to log in a large amount of users and repetitively upload and download files.

2. TCR-2

   Type: Dynamic, Automated

   Initial State: The web application is launched.

   Input/Condition: Massive course data input.

   Output/Result: It takes `MAX_TIME_PROCESS` to finish a core function.

   How test will be performed: An automated script is used to inject a large amount of course data. Run TSLR-5.

4.2.8 Operational and Environmental

1. TOER-1

   Type: Dynamic, Automated

   Initial State: The web application is launched.

   Input/Condition: Send requests to supported calendar APIs.

   Output/Result: `MIN_API_SUCCESS%` of successful API requests.

   How test will be performed: An automated script is used to send requests to calendar APIs and record the result.

2. TOER-2

   Type: Dynamic, Automated

   Initial State: The web application is launched.

   Input/Condition: Run regression test suites.

   Output/Result: `MIN_REGRESSION_PASS%` of passed regression tests.

   How test will be performed: An automated script is used to run regression tests.

4.2.9 Maintainability and Support

1. TSPR-1

   Type: Dynamic, Manual

   Initial State: The web application is launched.

   Input/Condition: Users are asked to contact support.

   Output/Result: `MAX_SUPPORT_STEP` steps needed to reach help.

   How test will be performed: `MIN_TESTER_NUM` users are asked to reach the helpdesk through email, phone, and chatbot and record the steps it takes from the main page to help.

2. TSPR-2

   Type: Dynamic, Automatic

   Initial State: The web application is launched.

   Input/Condition: User feedback is inputted.

   Output/Result: User input in database.

   How test will be performed: A script is used to input user feedback and check if this feedback gets stored in the database.

4.2.10 Security

1. TSR-1

   Type: Dynamic, Automatic

   Initial State: The web application is launched.

   Input/Condition: Wrong credential pairs.

   Output/Result: Account not logged in.

   How test will be performed: A script is used to input the wrong credential pairs and check that an error message indicating bad credentials is displayed and the account is not logged in.

2. TSR-2

   Type: Dynamic, Automatic

   Initial State: The web application is launched.

   Input/Condition: Create a new account.

   Output/Result: Account data format.

   How test will be performed: A script is used to create a new account and check that the account data is encrypted before transmission.

3. TSR-3

   Type: Dynamic, Manual

   Initial State: The web application is launched.

   Input/Condition: Users are asked to modify the database.

   Output/Result: `MAX_ATTACK_SUCCESS` users successfully modified the database.

   How test will be performed: `MIN_TESTER_NUM` users are asked to taint the database as unauthorised entity.

4. TSR-4

   Type: Dynamic, Automatic

   Initial State: The web application is launched.

Input/Condition: A series of user actions.

Output/Result: An audit log.

How test will be performed: A script is used to perform a series of user actions and check that the activities are recorded with time stamps and relevant metadata.

5. TSR-5

   Type: Dynamic, Automatic

   Initial State: The web application is launched.

   Input/Condition: Login attempts.

   Output/Result: Account restricted.

   How test will be performed: A script is used to perform a brute-force attack to log in to a test account.

4.2.11 Complience

1. TCPR-1

   Type: Dynamic, Manual

   Initial State: Development container is launched.

   Input/Condition: A PR containing local changes is pushed.

   Output/Result: Workflow result.

   How test will be performed: A PR containing local changes is pushed, which triggers a workflow linter check.

4.3 Traceability Between Test Cases and Requirements

All requirements refer to SRS.pdf.

Test Case #	Requirement #				
	FR1	**FR2**	**FR3**	**FR4**	**FR5**
TFR1-A1	X				
TFR1-A2	X				
TFR2-A3		X			
TFR3-A4			X		
TFR4-A5				X	
TFR5-A6					X
TFR5-A7					X

Table 1: Traceability Matrix: Functional Requirements - Authentication

Test Case #	**Requirement #**									
	FR6	**FR7**	**FR8**	**FR9**	**FR10**	**FR11**	**FR12**	**FR13**	**FR14**	**FR15**
TFR6-UI1	X									
TFR6-UI2	X									
TFR7-UI3		X								
TFR8-UI4			X							
TFR9-UI5				X						
TFR10-UI6					X					
TFR11-UI7						X				
TFR11-UI8						X				
TFR11-UI9						X				
TFR12-UI10							X			
TFR13-UI11								X		
TFR14-UI12									X	
TFR15-UI13										X

Table 2: Traceability Matrix: Functional Requirements - User Input

Test Case #	**Requirement #**								
	FR16	**FR17**	**FR18**	**FR19**	**FR20**	**FR21**	**FR22**	**FR23**	**FR24**
TFR16-17-D1	X	X							
TFR18-D2			X						
TFR19-D3				X					
TFR20-D4					X				
TFR21-D5						X			
TFR22-D6							X		
TFR23-D7								X	
TFR22-23-D8							X	X	
TFR24-D9									X
TFR24-D10									X

Table 3: Traceability Matrix: Functional Requirements - Data

Test Case #	**Requirement #**						
	FR25	**FR26**	**FR27**	**FR28**	**FR29**	**FR30**	**FR31**
TFR25-S1	X						
TFR26-S2		X					
TFR27-S3			X				
TFR28-S4				X			
TFR29-S5					X		
TFR30-S6						X	
TFR31-S7							X

Table 4: Traceability Matrix: Functional Requirements - Scheduling

Test Case #	Requirement #																							
	AR1	AR2	AR3	AR4	STR1	STR2	STR3	STR4	STR5	UHR1	UHR2	UHR3	UHR4	UHR5	SLR1	SCR1	PAR1	RFTR1	CR1	CR2	SER1	OER1	OER2	OER3
TAR-1	X																							
TAR-2		X																						
TAR-3			X																					
TAR-4				X																				
TSTR-1					X																			
TSTR-2						X																		
TSTR-3							X																	
TSTR-4								X																
TSTR-5									X															
TUHR-1										X														
TUHR-2											X													
TUHR-3												X												
TUHR-4													X											
TUHR-5														X										
TSLR-1															X									
TSCR-1																X								
TPAR-1																	X							
TRFTR-1																		X						
TCR-1																			X					
TCR-2																				X				
TOER-1																						X		
TOER-2																							X	

Test Case #	Requirement #																						
	MR1	MR2	MR3	SPR1	SPR2	SPR3	ADR1	ADR2	SR1	SR2	SR3	SR4	SR5	SR6	SR7	SR8	SR9	CR1	CTR1	CTR2	CPR1	CPR2
TSPR-1				X																		
TSPR-2						X																
TSR-1									X													
TSR-2											X											
TSR-3												X										
TSR-4															X							
TSR-5																	X					
TCPR-1																						X

Table 5: Traceability Matrix: Non-Functional Requirements

5 Appendix

This is where you can place additional information.

5.1 Symbolic Parameters

parameter	value	unit	description
MIN_UNDERSTAND%	95	N/A	the minimum percentage of testers who can understand among all testers
MAX_UNSATISFIED%	5	N/A	the maximum percentage of testers reporting uncomfortable
MAX_TRIAL_TIME	1200	s	the maximum allowed trial time
MIN_TESTER_NUM	20	N/A	the minimum number of testers needed
MAX_BAD_GRAMMAR	0	N/A	the maximum occurrence of grammar mistakes allowed
MAX_OFFENSIVE	0	N/A	the maximum occurrence of offensive messages allowed
MAX_ELEMENT_BAD	0	N/A	the maximum occurrence of inconsistent or unresponsive elements allowed
MAX_ATTACK_SUCCESS	0	N/A	the maximum occurrence of successful attack
MAX_COLOR_AMBIGUOUS	0	N/A	the maximum occurrence of indistinguishable color combinations allowed
MAX_TIME_PROCESS	2	s	the maximum processing time for a core function
MIN_PRECISION%	95	N/A	the minimum precision of the algorithm
MIN_OPERABLE%	95	N/A	the minimum percentage of system being operable

MIN_API_SUCCESS%	95	N/A	the minimum percentage of successful API calls
MIN_REGRESSION_PASS%	100	N/A	the minimum percentage of successful API calls
MAX_RESPONSE_TIME	24	h	The maximum issue resolve response time
MIN_LANGUAGE	5	N/A	The minimum number of languages that can be translated
MAX_SUPPORT_STEP	5	N/A	The maximum steps needed for asking for support

5.2 Usability Survey Questions

1. Which part of the typography do you find confusing?
2. Do you like the color choice of the page?
3. What do you think this icon/graphic means?
4. Which part of the layout do you find awkward?
5. Could you point at the menu/navigation tool on this page?
6. Which element do you think is inconsistent with the rest?
7. Which interactive element do you find unresponsive?
8. Are you comfortable with the default font size? If not, could you adjust it to your preference?
9. Do you find this animation intrusive or distracting?
10. Could you navigate to the progress page?
11. Could you make a schedule with this course outline?
12. What grammar mistakes could you find?
13. What offensive message could you find?

14. Which color combination on the page do you find indistinguishable?
15. How many steps do you need to reach support?
16. Are you able to modify this database?

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning. Please answer the following questions:

Appendix — Reflection

[This section is not required for CAS 741 —SS]

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning. Please answer the following questions:

1. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage etc. You should look to identify at least one item for each team member.

2. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?