Group Assignment Brief
CO3095 Software Measurement and Quality Assurance 2025-26 SEM1

0. AI Policy

You must not use any AI tools to create any of the gradable content/component in this group assignment. However, you may use AI tools to help you check for grammatical and spelling errors. If we detected any unauthorised use of AI tools in your group assignment, other than for checking of grammatical and spelling errors, it will be considered as a case of plagiarism, and you will be referred to the plagiarism officer. Additionally, you will get a zero mark in this group assignment immediately.

1. Overview of Group Assignment

The assignment requires students to work in teams to develop a fully functional software product/application, focused on the backend functionality and its overall software quality, based on the quality assurance techniques taught in this module. **There is no need for a GUI frontend. A simple text-based frontend is sufficient. There is also no need for any databases.** You can use a serialized text file for storage and persistence. Using a GUI frontend and a database will not give you additional marks, and if you implement those, it may cause you to slip the project since these consume time to implement (*you need to provide test cases to these too!*), and you may not have sufficient time for other required features (*check Marking Rubric below*). Additionally, if the graders cannot open and run your program in the computer labs with proper and comprehensive readme included in the submission, you will get a zero mark. If you do not follow the instruction, such as using other programming languages, tool, and techniques, etc. that were not introduced in this module, you will not get any marks as well. The development process should follow the agile software development methodology (**i.e., SCRUM framework**) and other software quality methods taught in lectures. There is also a research component (*see below*).

Please see below for the detailed requirements on the programming language, research component, development and planning tools to be used.

Programming Language	Only Python (*You must not use other languages as we are using some of the software quality tools developed in Python and you are required to use them in your group assignment, and this is also for marking standardization.*)
Research Component	Symbolic execution and concolic testing are similar White-Box testing techniques. You will research on both symbolic execution and concolic testing and apply these techniques to all your code functions you developed. You will need to explain how you perform symbolic execution and concolic testing, including drawing the symbolic tree, computing the values, other important elements related to these two techniques, etc in the report. This is in addition to all other taught testing techniques you must perform for all functions that you developed.

Development & Planning tools	We will be using PyCharm, GitHub, and other tools introduced in the module. For development tools, you must use **PyCharm** since all teaching staffs are familiar with this tool. We will load your PyCharm project in one of the lab computers, and if it not loadable, you will not get any marks! You must use GitHub for project management (*through GitHub Project*), code commits and code review. You must also add the GitHub ID of the Professor and all the TAs into your group assignment repository.

Each group must choose one (***no two groups are allowed to choose the same project***) project from the provided **CO3095 project list (*Section 8*)** and propose **fixed-sized** user stories (*for a group of 4 members, the number of user stories should be at least 9 for each member*) that are to be developed for the chosen project. You must explicitly state which member of the team is doing which user stories in the report. The professor will share a document to all groups in the lab session for all groups to indicate their chosen project topic. The choosing of the project topic is on a first come first serve basis.

Each user story should be done through appropriate market research and team discussion. You should bear in mind that this group project only last approximately one and a half month – You should not do something that is very complicated and that it will take more than one month. You should also start early and not wait until the last minute. **Note that** each group should first build the **backbone** of the project by defining the corresponding empty classes and functions based on the user stories. Within each sprint, each group member needs to implement the assigned empty classes or functions. Every user story (*issue*) in the GitHub Project needs to have a GitHub branch linked to it (*even though you may later merge this branch into the main branch, it should not be deleted to provide evidence of development*).

From the module learning outcomes, this assignment assesses:

1. Participation and utilization of software quality strategies that are taught in lectures, for the effective working of an agile team towards a common goal.
2. Implementation of agile methodologies in a group-based setting while engaging in a practical software project to produce a functional quality software product.
    a. There should be **at least 3 sprints (*each sprint is minimum of 1 to 2 weeks*)**.
    b. Each team member in the group should be assigned an **equal** number of **user stories**, in which each user story should be estimated with the appropriate **story points** via **Planning Poker**. The total story points for each team member in the group should also be relatively equal. For each sprint (based on a total of 3 sprints), each user should have at least 3 user stories with sufficient complexity for every user story e.g., the function to be developed for the user story should have greater than 3 conditions and branches and having a cyclomatic complexity of at least 10, etc.
    c. **PERT** should be used to draw the network diagram and measure the **critical path**.
    d. Estimate the cost using **COCOMO I and II.**
    e. **Track** the project with **burndown charts, velocity** and **earned value management techniques.**
3. Implementing test cases using the taught testing tools and techniques for **Black-Box testing** strategies/algorithms (***Specification-Based and Random-Based***) and **the White-Box testing** strategies/algorithms.

4. Measuring the **test coverage** to check how much code has been tested. For each testing technique, you should create a separate test package in your repository that will include **all the relevant test cases** for black-box and white-box with the naming convention "*your_studentid . test . blackbox . which_technique*" and "*your_studentid . test . whitebox . which_technique*", respectively.
5. Symbolic execution and concolic testing are similar White-Box testing techniques. You will research on both symbolic execution and concolic testing and apply these techniques to all your code functions you developed. You will need to explain how you perform symbolic execution and concolic testing, including drawing the symbolic tree, computing the values, other important elements related to these two techniques, etc in the report. This is in addition to all other taught testing techniques you must perform for all functions that you developed. You should also follow the same naming convention in point 4 for your symbolic test cases.

Deliverables to be submitted:

1. A video demonstrating how to run your final deliverable software product – this includes the source code and all the individual test cases. The video demonstration should:
    - Show how to compile and run the code.
    - Demonstrate the features of your software product.
    - Run all the test cases separately based on the different black-box and white-box techniques.
    - Show the critical path and its calculation.
    - Present the estimated story points via planning poker with evidence.
    - Present the cost computed via COCOMO, COCOMO II, EVM and their calculation.
    - For each sprint, show the following details in your SCRUM (GitHub Project) board:
        - The user stories in each sprint.
        - Who is responsible for each user story.
        - The start and end date of each user story.
    - Show the GitHub commit history of your project that is linked to the user stories.
    - Show and run your Black-Box test cases: What testing tools and techniques did you use? All the test cases and the test results (whether it passed or failed). If any of the test cases failed, you must provide an explanation – a failed test case means your code has bugs and they are not fixed! Ideally, you should not have any failing test cases.
    - Show and run your White-Box test cases (this includes the research component on Symbolic Execution): What testing tools and techniques did you use? All the test cases and the test results (whether it passed or failed). If any of the test cases failed, you must provide an explanation – a failed test case means your code has bugs and they are not fixed! Ideally, you should not have any failing test cases.
    - Show your test coverage of your code. For each testing technique, you should create a related test package, which includes **all the relevant test cases** that can be run without errors. The naming convention for blackbox and whitebox test cases should be "*your_studentid . test.blackbox . which_technique*" and "*your_studentid . test . whitebox . which_technique*", respectively.

2. A report (**no more than 5000 words**) with two main sections: Section A introduces the software/application features and development techniques (*you should emphasize what software qualities features or techniques that were taught in lectures are used here*), and Section B describes the development process (*you should emphasize the team structure and contribution*). The report structure (you must adhere to the report structure) is shown below:

Section A

- Introduction
- Software planning & design
  - Includes user stories, story points, burndown charts, velocity, PERT diagram, critical path, estimated costs via COCOMO, COCOMO II, EVM, etc.
- Implementation
  - Back-End
- Testing and Test Coverage Measurement
  - Black-Box Testing: What testing tool did you use? All the test cases and the test results (whether it passed or failed). If any test cases failed, you must give an explanation – a failed test case means your code has bugs and they are not fixed!
  - White-Box Testing: What testing tool did you use? All the test cases and the test results (whether it passed or failed). If any test cases failed, you must give an explanation – a failed test case means your code has bugs and they are not fixed!
  - Research Component on Symbolic Execution and Concolic Testing: The details on how you perform Symbolic Execution and Concolic Testing, including showing the symbolic tree, path conditions, etc. Show also the test cases and the test results (whether it passed or failed). If any test cases failed, you must give an explanation – a failed test case means your code has bugs and they are not fixed!
  - Test Coverage of the individual testing technique.
- Document with Evidence all the CMMI Level 2 process areas as shown in the lecture.
- Conclusion

Section B

- Introduction
- Team Structure
- Team Contribution & Reflection

Reference

Appendices

---

3. The source code (with all the test cases) and other relevant artifacts of your software product in a .zip file. Include a detailed README file on how to uncompress, import from the computer in Charles Wilson laboratory, compile and run all the code and test cases.

Important Dates:

- Handed out: **16-Oct-2025 at 2:00 pm UK Time**

- Deadline: **5-Jan-2026 by mid-day 12 pm UK Time** (*You must submit the zip file to the provided Turnitin Link in the CO3095 Sem 25-26 Blackboard: "Assessment and Feedback" →"CO3095 Group Assignment". Only 1 member of the group needs to submit. Do not submit the same copy multiple times by different group members. For every group, we will mark only the most recent uploaded copy, based on the uploaded time.*)

2. Other Important Information

- This assignment counts to **70%** of the module's mark.
- This assignment is a group work.
- Please remember to limit your submission file size to under 50MB and submit it before the deadline (any late submission will receive penalty in marks according to the university policy).
- If your video is larger than 50MB, please use the 'video compress' tool to compress it.
- Your submission should include:
    - A report in a PDF file, containing the following:
        - Academic integrity signature signed by all team members.
        - The main text of the report **(no more than 5000 words).** Detailed guidelines of the report can be found under the "Deliverables to be submitted" section.
    - The source code (with all the test cases) and other relevant artifacts of your software product. Detailed guidelines of the report can be found under the "Deliverables to be submitted" section.
    - A video demonstrating how to load and run your software/application (the source code and all the test cases) in the lab. Detailed guidelines of the video demonstrations can be found under the "Deliverables to be submitted" section.
- You can re-submit your work **multiple times before the deadline**. However, we will only grade the latest uploaded submission. If you have submitted after the deadline (*even if you have a previous submission before the deadline*), we will mark the latest uploaded submission, which is the latest submission after the deadline, and late penalty will be imposed!
- Each group should only submit once – choose among your group members who to submit. If multiple group members submit at the same time, we will mark the latest uploaded submission and if it is late submission, late penalty will be imposed!
- You could submit your report in your first attempt to check for similarity. Then, you can submit your final zip file in your future attempts.
- The similarity score of your submission should be **less than approximately 15%**. You should aim to keep the similarity score to less than 10%.
- The deadline is strict. Penalties in marks will be applied to late submissions according to the university policy.

3. Resit

- We **strongly** advise you to pass the exam via the normal group assignment route.

- If you fail, you will have to accomplish a resit exam **individually** at a much later resit period, which may delay your graduation date.
- In the resit exam, you need to answer some short answer questions, some long answer questions as well as some application-based questions.

4. Marking Rubric

Category & Expectation	Outstanding (85 - 100%)	Excellent (70 - 84%)	Competent (60 - 69%)	Satisfactory (50 - 59%)	Adequate (40 - 49%)	Compensated Fail (30 – 39%)	Fail (0 –29%)
Sprint Planning, CMMI, and Configuration Management (10 marks) *Group component except for user story estimation and individual member's configuration management.*	Presents more than three completed sprints (*evidenced in the group GitHub project*) with each sprint minimum of 1-2 weeks. Each user story has story points estimated via Planning Poker with evidence. All the CMMI Level 2 process areas are documented in great detailed and correctly. Every Sprint is following the Configuration Management strategy discussed in the lectures and	Presents three completed sprints (*evidenced in the group GitHub project*) with each sprint minimum of 1-2 weeks. Each user story has story points estimated via Planning Poker with evidence. All the CMMI Level 2 process areas are documented sufficiently and correctly. Every Sprint is following the Configuration Management strategy discussed in the lectures and	Presents two completed sprints (*evidenced in the group GitHub project*) with each sprint minimum of 1-2 weeks. Each user story has story points estimated via Planning Poker with evidence. Majority of the CMMI Level 2 process areas are documented correctly. Every Sprint is following the Configuration Management strategy discussed in the lectures and labs correctly (may have some minor	Presents two completed sprints (*evidenced in the group GitHub project*) with each sprint minimum of 1-2 weeks. Some user stories do not have story points estimated via Planning Poker with evidence. Majority of the CMMI Level 2 process areas are documented correctly. Every Sprint is following the Configuration Management strategy discussed in the lectures and labs correctly (may have some minor	Presents one completed sprint (*evidenced in the group GitHub project*) with each sprint minimum of 1-2 weeks. Some user stories do not have story points estimated via Planning Poker with evidence. Few CMMI Level 2 process areas are documented correctly. Some sprints are not following the Configuration Management strategy discussed in the lectures and labs correctly (have major errors) with evidence.	Presents one completed sprint (*evidenced in the group GitHub project*) with each sprint minimum of 1-2 weeks. Some user stories do not have story points estimated via Planning Poker with evidence. No CMMI Level 2 process areas are documented. The sprints are not following the Configuration Management strategy discussed in the lectures and labs correctly with evidence.	Presents no completed sprint (*evidenced in the group GitHub project*). The user stories do not have story points estimated via Planning Poker with evidence or there is no evidence of planning. No CMMI Level 2 process areas are documented. The sprints are not following the Configuration Management strategy discussed in the lectures and labs correctly with evidence.

Category & Expectation	Outstanding (85 - 100%)	Excellent (70 - 84%)	Competent (60 - 69%)	Satisfactory (50 - 59%)	Adequate (40 - 49%)	Compensated Fail (30 – 39%)	Fail (0 –29%)
	labs correctly with evidence.	labs correctly with evidence.	errors) with evidence.	errors) with evidence.			
Software Development Measurement (20 marks) *Group component*	Critical path is analysed using PERT and it is constructed correctly with Network Diagram, calculation and with very detailed explanation (*how it is estimated and calculated*) on all the components in the Network Diagram. The cost is estimated via COCOMO I correctly and documented in detail. The cost is estimated via	Critical path is analysed using PERT and it is constructed correctly with Network Diagram, calculation and with very detailed explanation (*how it is estimated and calculated*) on most of the components in the Network Diagram. The cost is estimated via COCOMO I correctly and documented sufficiently. The cost is estimated via	Critical path is analysed using PERT and it is constructed partially correct with Network Diagram, calculation and limited explanation on how each component in the Network Diagram was estimated and calculated. The cost is estimated via COCOMO I correctly and documented sufficiently. The cost is estimated via	Critical path is analysed using PERT and it is constructed partially correct with Network Diagram, calculation and limited explanation on how each component in the Network Diagram was estimated and calculated. The cost is estimated via COCOMO I correctly (with minor errors) and documented sufficiently.	Critical path is analysed using PERT and it is constructed partially correct with Network Diagram, calculation and limited explanation on how each component in the Network Diagram was estimated and calculated. The cost is estimated and documented via COCOMO I but have some errors. The cost is estimated and documented via	Critical path is analysed using PERT, but it is not constructed, or there are no critical path/PERT and their analysis. There is no or limited explanation on the components in the Network Diagram. The cost is estimated and documented via COCOMO I but have errors or there is no costing involved. The cost is estimated and	Critical path is not analysed using PERT. There is no or limited explanation on the components in the Network Diagram. The cost is estimated and documented via COCOMO I but have errors or there is no costing involved. The cost is estimated and documented via COCOMO II but have errors or there is no costing involved.

Category & Expectation	Outstanding (85 - 100%)	Excellent (70 - 84%)	Competent (60 - 69%)	Satisfactory (50 - 59%)	Adequate (40 - 49%)	Compensated Fail (30 – 39%)	Fail (0 –29%)
	COCOMO II correctly and documented in detail. All the Earned Value Management measurements taught in lectures are computed correctly and documented in detail. Burndown Chart for every sprint are computed correctly and documented in detail.	COCOMO II correctly and documented sufficiently. All the Earned Value Management measurements taught in lectures are computed correctly and documented sufficiently. Burndown Chart for every sprint are computed correctly and documented sufficiently.	COCOMO II correctly and documented sufficiently. All the Earned Value Management measurements taught in lectures are computed correctly (with minor errors) and documented sufficiently. Burndown Chart for every sprint are computed correctly (with minor errors) and documented sufficiently.	The cost is estimated via COCOMO II correctly (with minor errors) and documented sufficiently. All the Earned Value Management measurements taught in lectures are computed correctly (with minor errors) and documented sufficiently. Burndown Chart for every sprint are computed correctly (with minor errors) and documented sufficiently.	COCOMO II but have some errors. All the Earned Value Management measurements taught in lectures are computed correctly (with some errors) and documented sufficiently. Burndown Chart for every sprint are computed correctly (with some errors) and documented sufficiently.	documented via COCOMO II but have errors or there is no costing involved. Limited evidence on Earned Value Management measurements taught in lectures or they have been computed incorrectly and have limited documentation in most areas. Limited evidence on Burndown Chart for every sprint or they have been computed incorrectly and have limited documentation in most areas.	No evidence on Earned Value Management measurements taught in lectures or they have been computed incorrectly and have limited documentation in most areas. No evidence on Burndown Chart for every sprint or they have been computed incorrectly and have limited documentation in most areas.
Code snippet (20 marks) *Group component for able to*	≥ 85% of the (*Python*) code snippets are self-explanatory, easy to read, cover all the core functions, and	≥ 70% of the (*Python*) code snippets are self-explanatory, easy to read and cover all the core	≥ 60% of the (*Python*) code snippets are self-explanatory, easy to read and cover all the core	≥ 50% of the (*Python*) code snippets are self-explanatory, easy to read and cover all the core	≥ 40% of the (*Python*) code snippets are self-explanatory, easy to read and cover all the core	≥ 30% of the (*Python*) code snippets are self-explanatory, easy to read and cover all the core	< 30% of the (*Python*) code snippets are self-explanatory, easy to read and cover all the core

Category & Expectation	Outstanding (85 - 100%)	Excellent (70 - 84%)	Competent (60 - 69%)	Satisfactory (50 - 59%)	Adequate (40 - 49%)	Compensated Fail (30 – 39%)	Fail (0 –29%)
uncompress and import/open on the Charles Wilson Lab Computer without any error, readme file, and working GitHub link. Individual components for all others.	do not have any code issues. All relevant code (*including all the test cases*) is committed by the members working on their assigned user stories in GitHub. The submitted compressed project must be able to be uncompressed and imported into PyCharm on the Charles Wilson Lab Computer without any error. There is a very detailed README file on how to uncompress, load from the computer in Charles Wilson laboratory, compile and run all the code and test cases. There is also a working public	functions, and do not have any code issues. All relevant code (*including all the test cases*) is committed by the members working on their assigned user stories in GitHub. The submitted compressed project must be able to be uncompressed and imported into PyCharm on the Charles Wilson Lab Computer without any error. There is a detailed README file on how to uncompress, load from the computer in Charles Wilson laboratory, compile and run all the code and test cases. There is also a	functions, and do not have any code issues. All relevant code (*including all the test cases*) is committed by the members working on their assigned user stories in GitHub. The submitted compressed project must be able to be uncompressed and imported into PyCharm on the Charles Wilson Lab Computer without any error. There is a somewhat detailed README file on how to uncompress, load from the computer in Charles Wilson laboratory, compile and run all the code and test cases.	functions, and do not have any code issues. All relevant code (*including all the test cases*) is committed by the members working on their assigned user stories in GitHub. The submitted compressed project must be able to be uncompressed and imported into PyCharm on the Charles Wilson Lab Computer without any error. There is a somewhat detailed README file on how to uncompress, load from the computer in Charles Wilson laboratory, compile and run all the code and test cases.	functions, and do not have any code issues. Some (*incomplete*) relevant code (*including all the test cases)* is committed by the members working on their assigned user stories in GitHub. The submitted compressed project must be able to be uncompressed and imported into PyCharm on the Charles Wilson Lab Computer without any error. There is a less detailed README file on how to uncompress, load from the computer in Charles Wilson laboratory, compile and run all the code and test cases.	functions, and do not have any code issues. Limited relevant code (*including all the test cases*) is committed by the members working on their assigned user stories in GitHub. The submitted compressed project could not be uncompressed and imported into PyCharm on the Charles Wilson Lab Computer without any error. There is no detailed README file on how to uncompress, load from the computer in Charles Wilson laboratory, compile and run all the code and test cases. There is no	functions, and do not have any code issues. No relevant code (*including all the test cases*) is committed by the members working on their assigned user stories in GitHub. The submitted compressed project could not be uncompressed and imported into PyCharm on the Charles Wilson Lab Computer without any error. There is no detailed README file on how to uncompress, load from the computer in Charles Wilson laboratory, compile and run all the code and test cases. There is

Category & Expectation	Outstanding (85 - 100%)	Excellent (70 - 84%)	Competent (60 - 69%)	Satisfactory (50 - 59%)	Adequate (40 - 49%)	Compensated Fail (30 – 39%)	Fail (0 –29%)
	GitHub link to the project.	working public GitHub link to the project.	There is also a working public GitHub link to the project.	There is also a working public GitHub link to the project.	There is also a working public GitHub link to the project.	working public GitHub link to the project.	no working public GitHub link to the project.
Black-Box and White-Box Testing and Coverage (20 marks) *Individual Component*	All functions have test cases. ≥ 85% of the test cases passed. ≥ 85% of the test cases are documented properly indicating what test techniques are used, what testing tools are used and the expected and actual test results. The average test coverage for all the black-box and white-box testing coverage is ≥ 85%. The black-box test cases must be in a package "*your_studentid . test.blackbox . which_technique*"	≥ 90% of the functions have test cases. ≥ 70% of the test cases passed. ≥ 70% of the test cases are documented properly indicating what test techniques are used, what testing tools are used and the expected and actual test results. The average test coverage for all the black-box and white-box testing coverage is ≥ 70%. The black-box test cases must be in a package "*your_studentid .*	≥ 80% of the functions have test cases. ≥ 60% of the test cases passed. ≥ 60% of the test cases are documented properly indicating what test techniques are used, what testing tools are used and the expected and actual test results. The average test coverage for all the black-box and white-box testing coverage is ≥ 60%. The black-box test cases must be in a package "*your_studentid .*	≥ 70% of the functions have test cases. ≥ 50% of the test cases passed. ≥ 50% of the test cases are documented properly indicating what test techniques are used, what testing tools are used and the expected and actual test results. The average test coverage for all the black-box and white-box testing coverage is ≥ 50%. The black-box test cases must be in a package "*your_studentid .*	≥ 60% of the functions have test cases. ≥ 40% of the test cases passed. ≥ 40% of the test cases are documented properly indicating what test techniques are used, what testing tools are used and the expected and actual test results. The average test coverage for all the black-box and white-box testing coverage is ≥ 40%. The black-box test cases must be in a package "*your_studentid .*	≥ 50% of the functions have test cases. ≥ 30% of the test cases passed. ≥ 30% of the test cases are documented properly indicating what test techniques are used, what testing tools are used and the expected and actual test results. The average test coverage for all the black-box and white-box testing coverage is ≥ 30%. The black-box test cases are not in a package "*your_studentid .*	< 50% of the functions have test cases. < 30% of the test cases passed. < 30% of the test cases are documented properly indicating what test techniques are used, what testing tools are used and the expected and actual test results. The average test coverage for all the black-box and white-box testing coverage is < 30%. The black-box test cases are not in a package "*your_studentid .*

Category & Expectation	Outstanding (85 - 100%)	Excellent (70 - 84%)	Competent (60 - 69%)	Satisfactory (50 - 59%)	Adequate (40 - 49%)	Compensated Fail (30 – 39%)	Fail (0 –29%)
	with clear comments in all the black-box test cases code explaining which black-box testing technique is used for which black-box test cases. The black-box test cases must be able to be executed using the taught techniques in the lab showing the test coverage on the Charles Wilson Lab Computer without any error. The white-box test cases must be in a package "*your_studentid . test . whitebox . which_technique*" with clear comments in the white-box test cases code explaining which white-box testing technique is	*test.blackbox . which_technique*" with clear comments in all the black-box test cases code explaining which black-box testing technique is used for which black-box test cases. The black-box test cases must be able to be executed using the taught techniques in the lab showing the test coverage on the Charles Wilson Lab Computer without any error. The white-box test cases must be in a package "*your_studentid . test . whitebox . which_technique*" with clear comments in the white-box test cases code explaining	*test.blackbox . which_technique*" with clear comments in all the black-box test cases code explaining which black-box testing technique is used for which black-box test cases. The black-box test cases must be able to be executed using the taught techniques in the lab showing the test coverage on the Charles Wilson Lab Computer without any error. The white-box test cases must be in a package "*your_studentid . test . whitebox . which_technique*" with clear comments in the white-box test cases code explaining	*test.blackbox . which_technique*" with clear comments in all the black-box test cases code explaining which black-box testing technique is used for which black-box test cases. The black-box test cases must be able to be executed using the taught techniques in the lab showing the test coverage on the Charles Wilson Lab Computer without any error. The white-box test cases must be in a package "*your_studentid . test . whitebox . which_technique*" with clear comments in the white-box test cases code explaining	*test.blackbox . which_technique*" with clear comments in all the black-box test cases code explaining which black-box testing technique is used for which black-box test cases. The black-box test cases must be able to be executed using the taught techniques in the lab showing the test coverage on the Charles Wilson Lab Computer without any error. The white-box test cases must be in a package "*your_studentid . test . whitebox . which_technique*" with clear comments in the white-box test cases code explaining	*test.blackbox . which_technique*" and there are no clear comments in all the black-box test cases code explaining which black-box testing technique is used for which black-box test cases. The black-box test cases could not be executed using the taught techniques in the lab showing the test coverage on the Charles Wilson Lab Computer. The white-box test cases must be in a package "*your_studentid . test . whitebox . which_technique*" with clear comments in the white-box test cases code explaining which white-box testing technique is	*test.blackbox . which_technique*" and there are no clear comments in all the black-box test cases code explaining which black-box testing technique is used for which black-box test cases. The black-box test cases could not be executed using the taught techniques in the lab showing the test coverage on the Charles Wilson Lab Computer. The white-box test cases must be in a package "*your_studentid . test . whitebox . which_technique*" with clear comments in the white-box test cases code explaining which

Category & Expectation	Outstanding (85 - 100%)	Excellent (70 - 84%)	Competent (60 - 69%)	Satisfactory (50 - 59%)	Adequate (40 - 49%)	Compensated Fail (30 – 39%)	Fail (0 –29%)
	used for which white-box test cases. The white-box test cases must be able to be executed using the taught techniques in the lab showing the test coverage on the Charles Wilson Lab Computer without any error.	which white-box testing technique is used for which white-box test cases. The white-box test cases must be able to be executed using the taught techniques in the lab showing the test coverage on the Charles Wilson Lab Computer without any error.	which white-box testing technique is used for which white-box test cases. The white-box test cases must be able to be executed using the taught techniques in the lab showing the test coverage on the Charles Wilson Lab Computer without any error.	which white-box testing technique is used for which white-box test cases. The white-box test cases must be able to be executed using the taught techniques in the lab showing the test coverage on the Charles Wilson Lab Computer without any error.	which white-box testing technique is used for which white-box test cases. The white-box test cases must be able to be executed using the taught techniques in the lab showing the test coverage on the Charles Wilson Lab Computer without any error.	used for which white-box test cases. The white-box test cases could not be executed using the taught techniques in the lab showing the test coverage on the Charles Wilson Lab Computer.	white-box testing technique is used for which white-box test cases. The white-box test cases could not be executed using the taught techniques in the lab showing the test coverage on the Charles Wilson Lab Computer.
Research (20 marks) *Individual Component*	Detailed explanation on how Symbolic Execution and Concolic Testing are performed on every function. This includes showing the construction of the symbolic trees correctly, showing the construction of the path conditions correctly, showing the construction of	Detailed explanation on how Symbolic Execution and Concolic Testing are performed on every function. This includes showing the construction of the symbolic trees correctly, showing the construction of the path conditions correctly, showing the construction of	Detailed explanation on how Symbolic Execution and Concolic Testing are performed on every function. This includes showing the construction of the symbolic trees correctly, showing the construction of the path conditions correctly, showing the construction of	Detailed explanation on how Symbolic Execution and Concolic Testing are performed on every function. This includes showing the construction of the symbolic trees correctly, showing the construction of the path conditions correctly, showing the construction of	Detailed explanation on how Symbolic Execution and Concolic Testing are performed on every function. This includes showing the construction of the symbolic trees correctly, showing the construction of the path conditions correctly, showing the construction of	Detailed explanation on how Symbolic Execution and Concolic Testing are performed on every function. This includes showing the construction of the symbolic trees correctly, showing the construction of the path conditions correctly, showing the construction of	There is limited evidence on applying symbolic execution.

Category & Expectation	Outstanding (85 - 100%)	Excellent (70 - 84%)	Competent (60 - 69%)	Satisfactory (50 - 59%)	Adequate (40 - 49%)	Compensated Fail (30 – 39%)	Fail (0 –29%)
	the test cases on symbolic execution correctly, etc. ≥ 85% of the test cases passed. The average test coverage testing coverage is ≥ 85%.	the test cases on symbolic execution correctly, etc. ≥ 70% of the test cases passed. The average test coverage is ≥ 70%.	the test cases on symbolic execution correctly, etc. ≥ 60% of the test cases passed. The average test coverage is ≥ 60%.	the test cases on symbolic execution correctly, etc. ≥ 50% of the test cases passed. The average test coverage is ≥ 50%.	the test cases on symbolic execution correctly, etc. ≥ 40% of the test cases passed. The average test coverage is ≥ 40%.	the test cases on symbolic execution correctly, etc. ≥ 30% of the test cases passed. The average test coverage is ≥ 30%.	
Video Demonstration of System (10 marks) *Group Component*	The presentation covers all the points under "Deliverables to be submitted" on page 3 and is very organized.	The presentation covers majority of the points under "Deliverables to be submitted" on page 3 (missing 1-2 minor points) and is very organized.	The presentation covers majority of the points under "Deliverables to be submitted" on page 3 (missing 2-3 points) and is very organized.	The presentation covers majority of the points under "Deliverables to be submitted" on page 3 (missing 2-3 points) and is very organized.	The presentation covers some of the points under "Deliverables to be submitted" on page 3 (missing 3-4 points) and is very organized.	The presentation covers little of the points under "Deliverables to be submitted" on page 3 and is very disorganized.	The presentation covers none of the points under "Deliverables to be submitted" on page 3 and is very disorganized.

5. Marking and Feedback

- TAs and the module conveners will work together to mark all submissions.
- TA will help in distributing feedback.
- A paragraph of "Overall comment" will be given at the end of the feedback.

6. Cover Sheet of Academic Integrity

Module Code: _____

Assignment: _____

I understand that this is a piece of coursework. I confirm that I handed in a signed Declaration of Academic Honesty Form (available at https://campus.cs.le.ac.uk/ForStudents/plagiarism/) and that I am fully aware of the statements contained therein. I understand if the submission will be checked by Turnitin Software. Any submission with similarity scores greater than approximately 20% may be further assessed individually and may be reported to Plagiarism Office. You should aim to keep the similarity score to less than 10%.

	Member 1	Member 2	Member 3	Member 4
Last, First Name: (in CAPITALS)				
Student ID: (e.g., ab123)				
Date:				
Signature:				

7. Resources

- Agile Scrum development: https://www.atlassian.com/agile/scrum.
- Development tools:
  - GitHub: https://github.com.
- Video compress tool: https://www.veed.io/tools/video-compressor.
- Report templates:
  - IEEE Xplore: https://ieeexplore-ieee-org.ezproxy3.lib.le.ac.uk/Xplore/home.jsp
  - Science Direct: https://www.sciencedirect.com/
  - Scopus: https://www-scopus-com.ezproxy3.lib.le.ac.uk/search/form.uri?display=basic#basic
- Textbooks (*non-exhaustive*)
  - Software quality assurance consistency in the face of complexity and change, by Neil Walkinshaw
  - Software Testing and Analysis - Process, Principles, and Techniques, by Mauro Pezze and Michal Young

The A-Z of databases and electronic resources can be found at the URL of https://le.ac.uk/library/search-collections/databases-az

8. CO3095 Project List

1) Multiple choice question and answer desktop application.
2) A Credit scoring tool.
3) A Financial investment assistant.
4) A Job Recruiter Assistant application.
5) A Crowdsourcing Tool for Route Optimisation.
6) A Timetable Management application.
7) A film recommendation system.
8) An application for campus navigation.
9) A platform for evaluating schools.
10) Football league scheduling application.
11) Flight Journey application.
12) Job Finder Application.
13) Asset Management System.
14) Multidisciplinary Project Management application.
15) News monitoring and analytics application.
16) Smart Shopping Trolley application.
17) Software Support for Car Sharing.
18) Stock Market Trading System.
19) Takeaway Menu System.
20) Unit Job Scheduling.
21) Virtual Chat Assistant.
22) A tool to manage pension benefits.
23) Medical Screening Assistant.
24) Placements Management System.

25) Social Network Data Capture and Analysis Tool.

26) A tool for tracking placement applications, approvals, assessments, and visits.

27) An application for managing chronic patients' health.

28) Ticket booking system.

29) A scientific conference organisation assistant.

30) A tool for managing software projects.

31) Software Bug Tracking and Reporting Tool.

32) Examination paper management system.

33) Clinical research management tool.

34) Software Code Review Tool.

35) Music Player Application.

36) Text/Code Editor/Compiler.

37) An Automated Collect and Analysis Tool for Real Estate.

38) Stock Control System.

39) Teach-SPL: A teaching tool for Software Product Lines.

40) A judge system for programming competitions.