SOFT3202/COMP9202 Assignment 1

Software Construction and Design 2

1 Summary

Weight: 10%
Due: April 1 2025 at 11:59 PM
Late policy: No work accepted after April 8;
Work past due date (April 1) has penalty of 5% of available marks per day (except with approved special consideration, special arrangements, or simple extension)
What to submit (students in SOFT3202): Two documents:
- URS.md document [scaffold you are given, with gaps filled in] and
- SRS.md document [scaffold you are given, with gaps filled in];
Each uploaded in Canvas at appropriate link
What to submit (students in COMP9202): Three documents:
- URS.md document [scaffold you are given, with gaps filled in] and
- SRS.md document [scaffold you are given, with gaps filled in] and
- Reflection.md;
Each uploaded in Canvas at the appropriate link.

Note: This task does not involve writing any code.
This assignment is not group work. Do not work together on this assignment, and do not share your answers with anyone (other than teaching staff).
Use of Generative AI (eg copilot is welcome, but it must follow guidelines.

Changelog:

2025-03-09 Released

2025-03-10 Spellos fixed, clarified markdown as filetype for Reflection.

2025-03-12 Added an additional example system test T3 in SRS.

- Renamed System Test T2 to T3 in Traceability Matrix to accommodate the above change (SRS).

- Renamed URS-04 to URS-08 in the Traceability Matrix (SRS).

- Renamed URS-05 to URS-09 in the Traceability Matrix (SRS).

- Clarified the acceptance criteria example in the URS, specifying that the GitHub repository URL refers to the GitHub repository name.

2025-03-13 Clarified rubric: no penalty on MD formatting

- In URS-02, URS-04, URS-05, fix typos and standardized use of "repository name (i.e., URL)" to reduce ambiguity about names

- Clarified that user stories do not capture all requirements

- Clarified that there is no penalty for covering non-functional cases

- Clarified the need for covering additional functional requirements related to authentication, and error handling for empty repositories and organizations.

2 Context

Imagine that you are in a team that is working for Advanced Medical Devices, which is a company with a dedicated software division. To motivate and recognize the engineers, the company aims to identify and reward the *most active contributor* in each team. The company hosts all software code in GitHub. To meet its aim, the company plans to use an external contracting group (Contract Hub) to develop a tool that determines the most active contributor in a given GitHub repository. Some work has already happened to capture and document requirements for this tool, however there are some gaps in the documents which have been produced so far.

The company notes that only authorized users should be able to use the tool. They prefer that any data that are stored on disk be encrypted, and any over network communication should occur over encrypted channel. The company expects the system to be reasonably fast, and expect a response to any query within 5 seconds. The company has thousands of engineers and it should be possible to accommodate parallel access from all of them.

3 Provided initial drafts

Work has already happened to create a draft *User Requirements Specification (URS)* and a draft Software Requrements Specification (SRS), and we provide these to you. There are still some gaps where more is needed. The drafts look as follows (source copies are provided in Canvas)

3.1 User Requirements Specification (URS)

1. **Introduction**

   (a) **Purpose**
   The purpose of this document is to define the user requirements for a tool that identifies the most active contributor in a given GitHub repository (the "what"). This tool will be used by Advanced Medical Devices to recognize and reward software engineers based on their contributions.

   (b) **Scope**
   The system will:
   - Authenticate via the GitHub API.
   - Retrieve commit data and analyze contributor activity.
   - Provide contributor rankings based on predefined metrics.
   - Export results in JSON format for integration with other tools.
   - Ensure security and compliance with company policies.

   (c) **Intended Audience**

      i. Software engineers at Advanced Medical Devices

      ii. Engineering managers and team leads

      iii. IT administrators

iv. Contractors and developers responsible for tool development

(d) **Definitions and Acronyms**

i. GitHub – A cloud-based platform for version control and collaboration.

ii. Contributor – A user who makes commits, pull requests, or other changes to a repository.

iii. Commit – A change recorded in a GitHub repository.

iv. Pull Request (PR) – A proposed change submitted for review before merging.

2. **System Overview**
The tool will connect to GitHub repositories, fetch contribution data, and determine the most active contributor based on metrics such as the number of commits, lines of code added or removed, and pull requests merged. It will generate reports and provide a ranking of contributors.

3. **Personas**

(a) **Name:** Sarah Thompson
Role: Engineering Manager
Background: Sarah is responsible for overseeing multiple software development teams across different products. She ensures that engineers are contributing effectively and wants to recognize top-performing developers to boost team motivation and engagement.
Goals: Identify the most active contributor across all company projects.
View top contributors for individual repositories to track team-level contributions.
Use this information to reward high-performing engineers and improve team morale.
Pain Points:
Lack of visibility into developer contributions across multiple projects.
Manually tracking activity across repositories is time-consuming.
Recognition of engineers is inconsistent due to difficulty in measuring contributions.
System Expectations:
- A report summarizing top contributors across all products.
- The ability to drill down into specific repositories for per-project rankings.
- Accurate data on commit activity.
- The ability to export contributor rankings for internal reporting and performance reviews.

(b) **Name:** Jane Walker
Role: Team lead
Background: Jane oversees a team of software engineers working on multiple repositories. She is responsible for ensuring that engineers contribute effectively and for monitoring their engagement in the development process. She collaborates with engineering managers to align team efforts with project goals.
Goals: Track team-level contributions within specific repositories.
Identify top contributors within her team to provide recognition and feedback.
Ensure team members are fairly assessed based on their contributions.
Pain Points: Difficulty in manually tracking individual contributions across multiple repositories.
Lack of clear visibility into team member activity over different project phases.
Inconsistent and subjective methods of evaluating developer performance.
System Expectations:
The ability to view contributor rankings within her specific team's repositories.
Filtering options to evaluate contributions over different project phases.
Access to detailed breakdowns of commit activity for individual team members.

Exportable reports for team performance reviews and internal discussions.

(c) **Name:** Samuel Davidson
Role: Software Engineer
Background: Samuel is a software engineer who actively contributes to multiple repositories. He primarily works on implementing new features, fixing bugs, and submitting pull requests. He is eager to understand how his contributions compare to those of his peers and seeks recognition for his work.
Goals:
Monitor his contribution ranking within different repositories.
Gain insights into his coding activity and areas for improvement.
Ensure that his contributions are recognized in performance evaluations.
Pain Points:
Limited visibility into how his contributions are measured and ranked.
Difficulty in tracking his progress across different phases of a project.
Lack of a standardized system for assessing individual developer contributions.
System Expectations:
A personal dashboard displaying his ranking in both lifetime and phase-specific contributions.
Ability to filter and analyze his contributions over different time periods.
Clear metrics showing the impact of his commits, pull requests, and code reviews.
Exportable data for personal records and performance reviews.

(d) **Name:** Sandy Hong
Role: IT Administrator
Background: Sandy is responsible for managing authentication, security, and access control for various tools used by the engineering teams. She ensures that systems comply with security policies and that only authorized users can access sensitive data. She also oversees API integrations and system performance.
Goals:
Ensure secure authentication and authorization for all users accessing the tool.
Maintain compliance with company security policies and encryption standards.
Support seamless integration of the tool with internal analytics and reporting systems.
Pain Points:
Managing authentication for a large number of users across different access levels.
Ensuring secure data storage and encrypted communication while maintaining performance.
Handling API rate limits and potential system failures in a scalable manner.
System Expectations:
Robust authentication and authorization mechanisms integrated with GitHub API.
Secure storage of authentication tokens using industry-standard encryption.
Encrypted network communication for all data transmissions.
Exportable reports in JSON format for integration with internal security and analytics tools.

4. **User Requirements** *(Note that user requirements can capture non-functional requirements as well.)*

URS-01 The system shall support authentication and authorization via GitHub API.

URS-02 The system shall allow users to input a GitHub repository name (i.e., URL) and retrieve contributor rankings.

URS-03 The system shall allow users to input a GitHub organization name (i.e., URL) and retrieve contributor rankings.

URS-04 The system shall allow users to input a GitHub repository name (i.e., URL) and retrieve their specific ranking for life time of the project.

URS-05 The system shall allow users to input a GitHub repository name (i.e., URL) and retrieve their specific ranking for the specific phase of the project.

URS-06 The system shall display the most active contributor(s) based on commit count.

URS-07 The system shall allow users to filter rankings by specific time periods (e.g., last week, last month, last year).

URS-08 The system shall process API data and provide results within five seconds under normal conditions.

URS-09 The system shall securely store authentication tokens if required for private repositories, using industry-standard encryption techniques.

URS-10 The system shall provide an export option to download contributor rankings in JSON format.

URS-11 The system shall ensure encrypted communication for all data transmissions over the network.

URS-12 The system shall provide error handling mechanisms for API failures, including rate limit handling, access restrictions, and invalid repository names.

URS-13 The system shall support concurrent access by multiple users and handle parallel requests efficiently.

5. **User Stories** *(Not comprehensive. The user stories may not capture all requirements.)*

   - As a team lead, I want to see the *ranking of contributors* in my product. I also want to *filter contributor rankings by specific time periods* in my product so that I can evaluate contributions of engineers over different project phases.
   - As a software engineer, I want to know *my ranking* in both the life time of the project as well as the latest phase of the project.
   - As an IT administrator, I want the tool to generate exportable reports in JSON format so that I can integrate the data with other internal analytics tools.
   - As an engineering manager, I want to view the *most active contributor* in all our products, as well as individual active contributors in each project, so that I can recognize and reward top-performing engineers.

6. **Use Cases**

   (a) Identify the most active contributor overall.

   - **Actors:** Engineering Manager
   - **Preconditions:**
     - The user has access to the GitHub organization
     - The tool must have API access to retrieve organization data.
   - **Main Flow:**
     - The EM logs in to tool using their GitHub Credentials
     - The EM selects a Github organization by entering a URL
     - The tool fetches and processes contribution data
     - The system displays the most active contributor based on predefined metrics.
   - **Alternative Flow (1): Private organization**
     - The Manager is told that they cannot access the organization due to access restrictions.

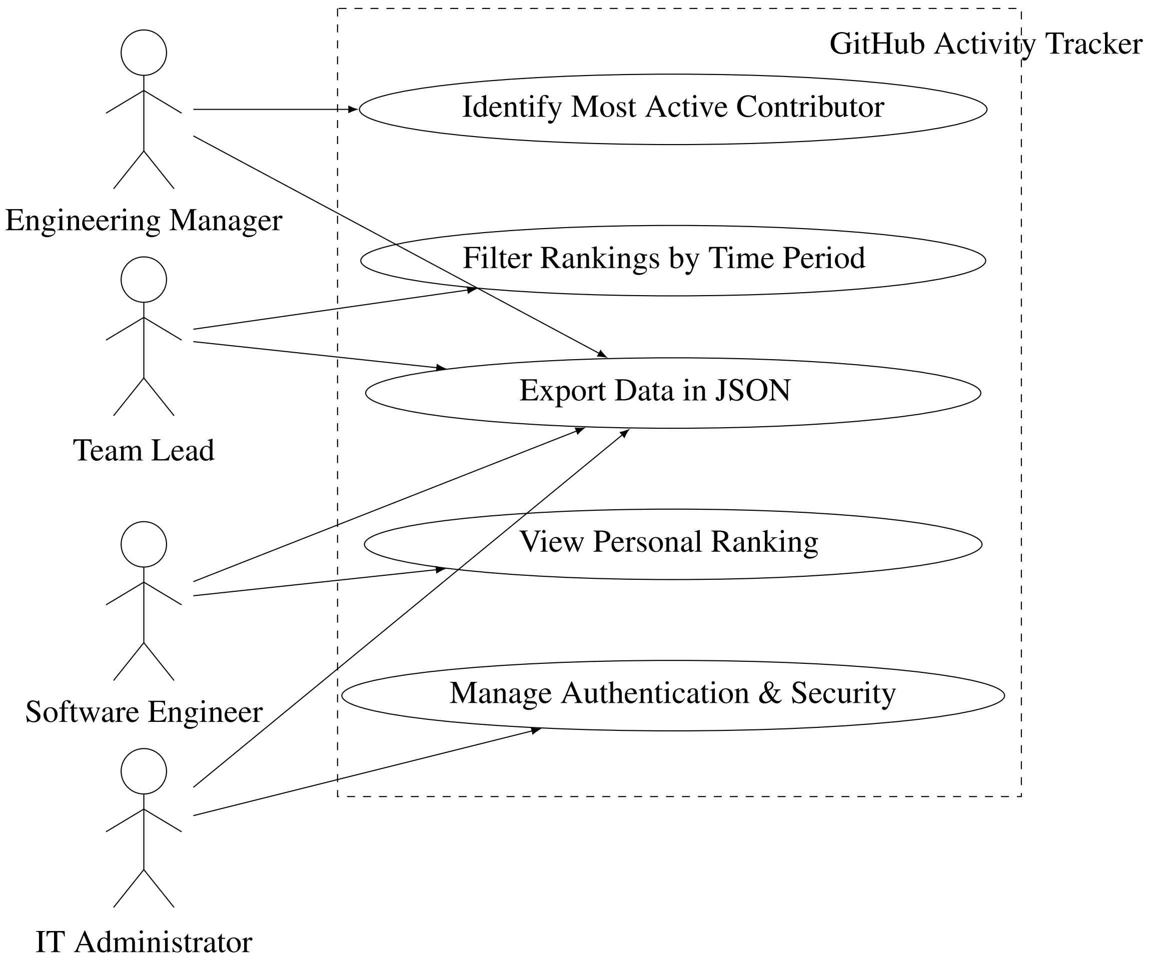
- **Alternative Flow (2): API rate limit exceeded**
  - The Manager is told that they cannot access the repository due to rate limit.
- **Post conditions:**
  - The most active contributor is displayed.

(b) Filter Contribution Rankings by Time Period
Complete this section

(c) Show My Contribution Ranking over Different Time Periods
Complete this section

(d) Export Contributor Data in JSON format
Complete this section

7. **Use case diagram**



8. **Non-Functional Requirements**

   (a) **Scalability Requirements:**
   The system should support simultaneous access by thousands of engineers.

   (b) **Performance Requirements:**
   The system shall process API data and return results within five seconds under normal operating conditions.
   The system shall efficiently manage API requests to prevent excessive delays or timeouts.

   (c) **Usability Requirements:**
   The system shall provide a clear and intuitive interface for users to input repository details and retrieve rankings.
   The system shall allow users to filter rankings by specific time periods to analyze contributions over different project phases.
   The system shall present error messages in a user-friendly manner, ensuring that users understand any issues and possible resolutions.
   The system shall allow exporting contributor rankings in a structured format for further analysis and reporting.

   (d) **Security Requirements:**
   The system shall enforce authentication and authorization mechanisms to restrict access

to authorized users only.
The system shall securely store authentication credentials and access tokens using industry-standard security practices.
The system shall ensure all data transmissions occur over an encrypted channel.
The system shall log authentication attempts and access activities for auditing and compliance purposes.
The system shall implement proper error handling to prevent unauthorized access to repository data.

9. **Risks & Mitigation Strategies**

   - **Technical Risk:** The tool may hit Github rate limits
     Mitigation: Implement caching and request optimization through batch processing.
   - **Security Risk: Risk: Unauthorized access to contributor data.**
     Mitigation: Implement strict authentication and authorization mechanisms to ensure that only authorized users can access the system. Enforce role-based access controls (RBAC) to limit data exposure.

     Risk: Exposure of sensitive authentication credentials.
     Mitigation: Store authentication tokens securely using industry-standard encryption techniques. Ensure that credentials are never stored in plain text and implement secure token management policies.

     Risk: Data interception during transmission.
     Mitigation: Enforce encrypted communication (e.g., TLS/SSL) for all data transmitted over the network to prevent man-in-the-middle attacks.

10. **Constraints:**
    Technology:
    - The company uses GitHub for its project.
    - The company standardizes on Python, and expects Python to be the language of choice.
    Regulatory and Compliance:
    - The company expects you to follow industry practice in any data handling

11. **Acceptance Criteria**

    (a) *Identify the most active contributor overall.*
    - Given that I have entered a valid GitHub repository name (i.e., URL), when I request contributor rankings, the system must display the name of the most active contributor based on commit count.
    - Given that a repository has multiple contributors with the exact same number of commits, when I request contributor rankings, the system must display all equally active contributors.

    Complete this section. You should provide enough acceptance criteria to validate each of the **use-cases.**

12. **Changelog**
    2025-02-24 — Created by (Contract Hub Customer Team)
    2025-02-25 — Signed off by (AMD Team lead)

3.2 Software Requirements Specification (SRS)

1. **Introduction**

   (a) **Purpose**
   The purpose of this document is to define the *formal technical specifications* (the "how") for the GitHub Activity Tracker tool built by Contract Hub for Advanced Medical Devices. This system will fetch contribution data from GitHub repositories, analyze commit activity, and determine the most active contributors.

   (b) **Scope**
   This tool will analyze GitHub repositories to determine the most active contributor based on predefined metrics. The tool will be designed to support multiple repositories and provide clear and accurate results. The system will be used by engineering managers and team leads.

   (c) **Intended Audience**
   - Engineering teams at Contract Hub, responsible for development.
   - QA/Test teams, ensuring that the system meets functional and performance requirements.

   (d) **References**

      i. GitHub REST API Documentation

      ii. OAuth Authentication Guide

2. **System Overview**
   The GitHub Activity Tracker is a standalone tool designed to integrate with the GitHub API. It will be used by Advanced Medical Devices (AMD) to analyze commit history, rank contributors, and generate reports. The system does not modify repository contents but reads contribution data.
   System Functions
   The system will:

   - Authenticate with GitHub API using OAuth.
   - Retrieve commit history from specified repositories.
   - Process commit data to determine the most active contributor.
   - Provide ranking and filtering options based on time periods.
   - Export contributor rankings in JSON format.

   User Characteristics
   The primary users are:

   - **Engineering Managers** - View reports on the most active contributors.
   - **Team Leads** - Track and analyze team contributions.
   - **Software Engineers** - Check their own rankings and contribution levels.
   - **IT Administrators** - Manage authentication, security, and API access.

   Constraints

   - The system must use the GitHub REST API.
   - API requests are subject to GitHub's rate limits.
   - Authentication must be handled via OAuth tokens.
   - The system must return results within 5 seconds.

   Assumptions and Dependencies

   - The repository must exist and be accessible to the user.
   - Users must have the necessary permissions to retrieve commit history.

- The system depends on the availability of the GitHub API.

3. **Functional Requirements**

FR-01 (Medium) The system shall allow users to authenticate via the GitHub API by specifying a credentials file on the command line (map: URS-01).

FR-... (...) Fill in this section by providing a comprehensive set of requirements that covers the URS **User Requirements – URS Section 4**. Keep all remaining priorities *(Medium)*.

Beyond strictly functional requirements that correspond to system functionalities, we expect your functional requirements to cover the following topics.

- What happens when a user is not authenticated correctly?
- What happens when a repository is empty?
- What happens when an organization is empty?
- What happens if a contributor has no contributions?

4. **Interfaces**

   (a) **API Interfaces**

   - The system will interact with GitHub's REST API to fetch commit and contributor data.
   - Authentication will be handled via OAuth tokens.
   - API endpoints include:
     - GET /repos/owner/repo/contributors
     - GET /repos/owner/repo/commits
     - GET /users/username/events/public

   (b) **User Interface (UI)**

   - A command line interface for displaying ranked contributors.
   - Filters for time-based contribution rankings.
   - Export options for downloading JSON reports.

5. **Data Requirements**

   (a) **Collected Data**

   - Contributor name, GitHub username
   - Number of commits

   (b) **Stored Data**

   - Temporary caching of contributor rankings
   - User authentication tokens (secure storage)

6. **System Architecture**

   (a) **Architecture Overview**

   - The system follows a client-server architecture where the server is GitHub.

   (b) **Main Components**

   - **UI:** A command line UI using Python 3.12 standard libraries.
   - **Cache (Optional):** A temporary cache for API responses.
   - **Authentication Module:** OAuth authentication for secure API access.

7. **Failure Handling**

(a) **Error Handling**

- Graceful degradation in case of API failures (retry mechanisms, user notifications).
- Handling private repository access errors.
- Validation of user inputs (e.g., invalid repository names).

(b) **API Rate Limits**

- Implement exponential backoff for API requests.
- Cache recent results to reduce API calls.

8. **Performance Benchmarks**

(a) **Expected Response Times**

- The system shall process API data within five seconds under normal conditions.
- Fetching and ranking contributors shall not exceed 3 seconds for repositories with fewer than 10,000 commits.

(b) **Scalability Requirements**

- The system must handle concurrent requests from multiple users.
- The system should scale to support thousands of repositories.

(c) **Load Handling**

- Use load balancing for handling multiple concurrent API requests.
- Implement caching layers to minimize redundant API calls.

9. **System Tests**

T1 **Scenario:** Valid repository with multiple contributors
Expectation: Correct ranking of contributors displayed.

T2 **Scenario:** Valid repository with no contributors
Expectation: Error message: No contributions found.

T3 **Scenario:** Attempt to authenticate using a GitHub account
Expectation: System successfully authenticates and grant access.

T... Complete this section by describing a comprehensive set of system tests that cover all functional requirements. Please make sure to test for negative scenarios. Note that there is no penalty for covering non-functional requirements – there can be cross-cutting concerns between what is functional versus non-functional.

10. **Non-Functional Requirements**

(a) **Scalability Requirements:**
The system shall support simultaneous access by thousands of engineers without performance degradation.
The system shall be designed to handle multiple concurrent API requests efficiently using caching and rate-limiting mechanisms.
The system shall scale horizontally by distributing API requests and processing workloads across multiple instances if necessary.

(b) **Performance Requirements:**
The system shall process API data and return results within five seconds under normal

operating conditions.
The system shall fetch and rank contributors within three seconds for repositories with fewer than 10,000 commits.
The system shall optimize API calls by implementing caching and batching mechanisms where applicable.
The system shall include logging and monitoring tools to detect and resolve performance bottlenecks.

(c) **Usability Requirements:**
The system shall provide a user-friendly command-line interface (CLI) with clear instructions and minimal configuration steps.
The system shall allow users to filter contributor rankings by specific time periods (e.g., last week, last month, last year).
The system shall present error messages with meaningful descriptions and possible resolutions to guide users.
The system shall support exporting contributor rankings in JSON format for integration with external analytics and reporting tools.
The system shall provide detailed usage documentation for engineers, team leads, and IT administrators.

(d) **Security Requirements:**
The system shall use OAuth authentication for secure access to the GitHub API and shall not store user credentials in plain text.
The system shall encrypt all stored authentication tokens using industry-standard encryption techniques such as AES-256.
The system shall ensure all data transmissions occur over a secure channel using TLS 1.2 or higher.
The system shall log authentication attempts and API access activities for auditing and security monitoring.
The system shall implement rate limiting and request throttling mechanisms to prevent API abuse.
The system shall follow the principle of least privilege, ensuring that users can only access data relevant to their role.

11. **Traceability Matrix**

- **URS:** URS-01 Support authentication and authorization via GitHub API
  SRS: FR-01 System shall allow users to authenticate via GitHub API
  verification: System Test (T3)
- **URS:** URS-08: Process API data within five seconds
  SRS: Non-Functional Requirement: Performance
  verification: System Test (Performance Benchmark)
- Fill in this section by providing the complete matrix.
- URS: URS-09: Securely store authentication tokens
  Non-Functional Requirement: Security
  verification: Security Audit

12. **Changelog**
2025-02-25 — Created by (Contract Hub Engineering Team)
2025-02-26 — Signed off by (Contract Hub Customer Team lead)

4 Assignment Instructions

If you are enrolled in SOFT3202, you have *two key tasks*.

1. User Requirements Specification (URS) (4 points). You must complete the indicated missing material within the given structured URS document (by editing the provided URS.md file). The document has text in red to draw your attention to these gaps.
    - You must **convert the User Stories into Use Cases**. We have provided the first use case (a) as an example. Convert the remaining three (b), (c), and (d) yourself. [1 point]
    - You must **write the detailed acceptance criteria** that cover the complete set of **use-cases**. [3 points]
      Please carefully note the requirement. It should cover the **use-cases** not the **user requirements**.
2. Software Requirements Specification (SRS) (6 points). The user needs are translated to technical specifications by creating an SRS. The SRS defines the system's architecture, functional and non-functional requirements, and system design considerations. You are given the draft version of an SRS.
    - You are required to fill in the gap of missing functional requirements [2 points]
    - You are to provide textual descriptions of test cases, which (when eventually coded) will comprehensively cover the functional requirements [3 points]
    - You are to complete the traceability matrix, to show the correspondence between functional software requirements, and user requirements. [1 point]

If you are enrolled in COMP9202, you have *three key tasks*.

1. User Requirements Specification (URS) (3 points). You must complete the indicated missing material within the given structured URS document (by editing the provided URS.md file). The document has text in red to draw your attention to these gaps.
    - You must **convert the User Stories into Use Cases**. We have provided the first use case (a) as an example. Convert the remaining three (b), (c), and (d) yourself. [1 point]
    - You must **write the detailed acceptance criteria** that cover the complete set of use-cases. [2 points]
      Please carefully note the requirement. It should cover the **use-cases** not the **user requirements**.
2. Software Requirements Specification (SRS) (5 points). The user needs are translated to technical specifications by creating an SRS. The SRS defines the system's architecture, functional and non-functional requirements, and system design considerations. You are given the draft version of an SRS.
    - You are required to fill in the gap of missing functional requirements [1 point]
    - You are to provide textual descriptions of test cases, which (when eventually coded) will comprehensively cover the functional requirements [3 points]
    - You are to complete the traceability matrix to show the correspondence between functional software requirements and user requirements. [1 point]
3. You are to write a reflective discussion, and upload it as a markdown document Reflection.md to Canvas. [2 points] The discussion should be structured in three parts
    - Part I describes the process you followed in completing the URS and SRS;
    - Part II indicates what material you found useful in developing these skills [give sources for this material];

- Part III discusses in what ways these documents do (or do not) relate to your previous experiences of documents that capture requirements for software.

5 Deliverables

5.1 User Requirements Specification (URS)

All students must upload in Canvas a URS document that fills in each of the red coloured sections, starting from the draft URS.md document we provide. Please do not modify any other sections of the URS.md other than those indicated.

5.2 Software Requirements Specification (SRS)

All students must upload in Canvas a SRS document that fills in each of the red coloured sections, starting from the draft SRS.md document we provide. Please do not modify any other sections of the URS.md other than those indicated.

5.3 Reflection – Only COMP9202 students

Students enrolled in COMP9202 must upload in Canvas the reflection as a pdf document structured in 3 parts.

Detailed Rubric for Individual Tasks

SOFT3202 (Undergraduate) - Total: 10 points

URS Assessment (4 points)

Use Cases (1 point)

- Converting user stories into use cases (b), (c), and (d)
- Complete and detailed use cases: 1 point
- Partially complete or missing coverage: -0.5 points
- Incorrect format (does not follow actor, precondition, main flow, alt flow, post-condition structure): -0.5 points

Acceptance Criteria (3 points)

- Complete, testable, and well-formulated criteria: 3 points
- Missing coverage for some use cases: deduct up to 1 point
- Technical implementation details in criteria that is better suited for SRS: deduct up to 0.5 points
- Non-testable criteria: deduct up to 1 point
- Ambiguous or subjective language (e.g., “should work well” instead of “should process within 2 seconds”): deduct up to 0.5 points

Overall coherence and alignment

- Clarity: up to -1 point

SRS Assessment (6 points)

Functional Requirements (2 points)

- Complete coverage of all missing functional requirements: 2 points
- Inclusion of non-functional requirements in functional requirements: deduct up to 0.5 point
- Incomplete coverage: deduct up to 0.5 points
- Missing technical precision: deduct up to 0.5 points
- Missing negative cases: deduct up to 0.5 points

Test Cases (3 points)

- Comprehensive test cases covering all functional requirements: 3 points
- Missing coverage of functional requirements: deduct up to 1 point
- Missing negative test cases: deduct up to 1 point
- Non-testable scenarios: deduct up to 1 point

Traceability Matrix (1 point)

- Complete traceability matrix showing correspondence between functional SRS requirements and user requirements: 1 point
- Incomplete matrix (i.e. missing or extra items): deduct up to 1 point
- Incorrect mapping between user and system requirements: deduct up to 0.5 points
- Inconsistent naming (e.g. FR1 vs FR01) between URS, SRS, and test cases: deduct up to 0.5 points

Overall coherence and alignment

- Clarity: up to -1 point

COMP9202 (Graduate) - Total: 8 points

URS Assessment (3 points)

Use Cases (1 point)

- Same criteria as SOFT3202

Acceptance Criteria (2 points)

- Same evaluation criteria as SOFT3202, but maximum is 2 points

Overall coherence and alignment

- Clarity: up to -1 point

SRS Assessment (5 points)

Functional Requirements (1 point)

- Same evaluation criteria as SOFT3202 but maximum is 1 point

Test Cases (3 points)

- Same criteria as SOFT3202

Traceability Matrix (1 point)

- Same criteria as SOFT3202

Overall coherence and alignment

- Clarity: up to -1 point

Reflection

- Complete reflection addressing all required parts
- Missing reflection part would result in proportionate point deduction

General Advice

- **Review Tutorials and Lectures:** Begin by reviewing the tutorials and lectures. Remember that everything you need for each component has already been covered in this unit.
- **Understand the Fundamentals:** Go through the revision slides on Ed and make sure that you understand all of the content covered so far.
- **Ask Questions:** If you have any questions or uncertainties about the material covered, don't hesitate to ask on Ed for clarification and a TA will get back to you shortly.