Assignment 3 Report – Software Engineering

Group 1 Members:

- 1. Isaro Rubagumya Roxane (28762)
- 2. Ishimwe Billy Joshua (28455)
- 3. Joseph Mutangana (29061)
- 4. Rebeka Patience (26277)
- 5. Nsengiyumva George (28491)

Section A

Subsection 1: Nyabihu Solution Ltd – Learning Management System (LMS)

Question 1
a) Functional Requirements:

1. Course Creation – Teachers can create and manage courses. It defines what the system must do.

2. Assignment Upload – Teachers upload assignments for students to access.

3. Course Enrollment – Students can register for courses through the web or mobile platform.

4. Grade Recording – Teachers can enter and update student grades.

5. Payment Integration – The system connects to a payment gateway for fee transactions.

b) Non-Functional Requirements:

1. Performance – Supports up to 10,000 users without lag.

2. Security – Ensures authorized access to student data.

3. Usability – Interface should be responsive and intuitive.

4. Reliability – The system should have 99.9% uptime.

5. Scalability – Must support future growth of users.

Question 2
a) User requirements define what end-users (students and teachers) expect from the system in simple, user-understandable terms.

b) Student Requirements: Register for courses, Submit assignments, View grades.

Teacher Requirements: Create courses, Upload assignments, Record grades.

Question 3

a) System requirements specify detailed technical descriptions of system functions. They differ from user requirements as they are more precise and technical.

b) Example translations:

• User: Student should view grades → System: The system shall allow students to view grades through their dashboard.

• User: Teacher uploads assignments → System: The system shall provide a file upload module for teachers.

Question 4

a) Interface specifications define how users and other systems interact with the software.

b) i. Student Dashboard – Displays courses, grades, notifications.
ii. Teacher Course Page – Interface for course management and grading.
iii. Mobile Login Interface – Secure login page with username/password fields.

Question 5

a) An SRS documents all software requirements formally. It ensures clarity and shared understanding among stakeholders.

b) Sections: Introduction, Functional Requirements, Non-functional Requirements, Interface Specifications, and Constraints.

c) The SRS improves communication by providing a common reference for developers, testers, and clients.

Question 6

a) Prioritization Strategy: MoSCoW (Must, Should, Could, Won't).

b) Challenging Requirement: Handling 10,000 users due to performance and scalability complexity.

Question 7

a) Techniques: Load balancing and server clustering for performance.

b) Testing: Conduct stress and load tests using simulation tools.

Subsection 2: Goshen Microfinance Ltd – Mobile Banking App

a) Functional: Transfer funds, View statements, Pay bills, Manage accounts.

Non-functional: Security, Performance, Scalability, Platform compatibility, Reliability.

b) Software Model: Incremental Model – allows gradual module delivery while maintaining quality.

c) Interface Specification – Fund Transfer Page: Fields for sender, receiver, amount, PIN, and confirmation message.

Requirements documented in SRD with clear functional/non-functional categories, diagrams, and interfaces.

Section B

Subsection 1 – Neteka Tech
Q1: Hybrid Agile-Waterfall chosen for flexibility and documentation balance. Ensures customer satisfaction through adaptability and structured control.

Q2: Relevant SDLC stages:

1. Requirements: It is first stage where they analysis what will be needed.
2. Design: After getting requirements, you make design of them.
3. Implementation: After getting design, you get into coding.
4. Testing: Testing is crucial for making sure that id system/product id doing what it meant to.
5. Maintenance: After completing the product, you need to keep tracking it's wellbeing.

Question3:

Step 1: Identify the Conflict Clearly: E.g. Finance wants online payments, Academics want detailed grading logic.

Step 2: Understand Each Department's Goals: Why do they need these features? Are they blocking each other?

Step 3: Use Prioritization Techniques: Use MoSCoW (Must, Should, Could, Won't) or Kano Model to classify requirements.

Step 4: Facilitate a Joint Meeting: Bring both teams together with a neutral analyst facilitating to discuss trade-offs.

Step 5: Propose a Phased Solution: E.g. Implement basic payment + basic grading first → enhance both in next iteration.

Step 6: Document Agreements: Ensure signed-off agreement on what was decided and why.

b) Role of the requirement analyst in managing That conflict

Mediator: Act as a neutral third party between finance and academic departments.

Interpreter: Translate department-specific language into shared understanding for developers.

Prioritizer: Help stakeholders rank their needs based on value, feasibility, and urgency.

Negotiator: Suggest compromises **or** alternatives when two needs clash.

Documenter: Clearly record requirements, decisions, and justifications for future reference.

Advisor: Offer insights about technical feasibility or user impact to guide discussions.

Q4: Steps to Implement Change:

1. **Impact Analysis**
    - Assess how this new integration will affect current modules, timelines, and resources.
    - Determine if additional APIs, authentication, or compliance steps are needed.
2. **Prioritize the Change**: Classify it as high/medium/low priority. If it's critical to stakeholders or legally required, it must be integrated.
3. **Adjust the Scope Using Time-Boxing**
    - Consider delivering core features first and plan this integration in a later sprint or phase**.**
    - Use agile iterations to avoid extending the entire project.
4. **Negotiate with Stakeholders**: Discuss options like: postponing less critical features or extending deadlines slightly (with justification).
5. **Update Project Plans**
    - Update the requirements, schedule, and cost estimates transparently.
    - Ensure all changes are documented and signed off.
6. **Communicate Clearly**: Keep both internal teams and clients informed to avoid misunderstandings.

Q5: Technical Feasibility

- Checks if the current technology, tools, and team can support development.
- E.g., Can the system handle thousands of student records and integrate with government systems?

2. Economic Feasibility

- Evaluates cost vs. benefits.
- Determines whether the budget is sufficient and if the system brings financial value**.**

3. Operational Feasibility

- Determines if the university and staff can effectively use and maintain the system.
- Includes training needs, ease of use, staff readiness.

4. Legal/Regulatory Feasibility

- Checks if the system complies with data protection, financial regulations, and education laws.
- Important for handling student data and government database integration.

Q6: Question 6b: Advantages & Limitations

Technique	Advantages	Limitations
Interviews	Allows one-on-one, detailed conversations. Easy to clarify answers.	Time-consuming, may miss broader views.
Prototyping	Stakeholders see a visual model; improves clarity.	May focus too much on design instead of full functions.
Workshops	Encourages collaboration and fast feedback from groups.	Difficult to manage if too many participants.

Question 7: SRS Outline for Student Registration Module

1. Introduction: Purpose, scope, and definitions for the registration module**.**

2. Functional Requirements: Actions like registering a student, adding/dropping courses, viewing status.

3. Non-Functional Requirements: Security (e.g., authentication), performance (e.g., handle 1000 students at once).

4. System Interfaces: How the module connects with grading, timetable, and payment systems.

5. Use Case Scenarios: Examples like: “Student registers for a course within the registration period.”

Question 8: Characteristics of Well-Defined Requirements

1. Clear and Unambiguous – No vague terms like "fast" or "easy".
2. Complete – Covers all expected behaviors and conditions.
3. Consistent – No contradictions between different requirements.
4. Testable – Can be verified through testing.
5. Traceable **–** Each requirement links to a business goal or function.

Poor-quality requirements lead to:

- Miscommunication
- Rework
- Project delays
- Bugs in system behavior

Question 9a: KPIs for Monitoring Software Process

1. Requirements Stability Index: Measures how often requirements change.
2. Defect Density**:** Number of bugs per 1,000 lines of code.
3. Velocity (for Agile Teams)**:** Amount of work completed in each sprint.

Question 9b: How KPIs Support Improvement

- Requirements Stability → Helps identify scope creep early.
- Defect Density → Highlights need for better testing or code reviews.
- Velocity → Helps with future sprint planning and workload balancing.

Question 10: Post-Implementation Review (PIR)

What It Involves:

- Review of the system's performance, stakeholder satisfaction, and project outcomes after deployment.

Main Objectives:

- Assess if goals were met
- Identify what went well and what didn't
- Gather feedback from users and developers

How It Guides Improvement:

- Improves future planning
- Enhances team performance
- Helps avoid repeating past mistakes

Subsection 2 – Real-time Patient Monitoring System

a. Best Software Process Model:

V-Model (Verification and Validation Model) or Spiral Model

Why?

- High reliability and safety required

- V-Model ensures testing at every phase (best for healthcare)
- Spiral allows risk analysis and iteration, good for integrating with old systems

---

b. Verification vs. Validation:

Process	In Patient Monitoring System	In Regular App
Verification	Rigorous testing at each level; simulations, unit tests, hardware-software integration	Usually basic functional testing
Validation	Includes real-life monitoring tests, alarms, and fault-tolerance checks	May only need basic usability or performance tests

---

c. Handling Late Feature Requests – Incremental/Iterative Models

- Allows adding new features in future increments without rebuilding entire system.
- The system grows in small, tested stages, so new features (e.g., new vital signs or alerts) can be added in later versions.
- Reduces risk and avoids delays to core deployment.

Section C

Question 2

a) How the Scrum Master can handle reduced team capacity during Sprint 2:

When like two developers fall sick, the Scrum Master should:

1. **Reassess Sprint Scope:**
    - Collaborate with the team to evaluate what work can realistically be completed.
    - Adjust the Sprint Backlog accordingly with the Product Owner**.**
2. **Communicate Transparently:** Inform stakeholders (especially the Product Owner) about the reduced capacity and its impact.
3. **Encourage Focus on Priority Tasks:** Ensure the team focuses on high-value user stories first.
4. **Support and Protect the Team:**
    - Prevent burnout among remaining members by not overloading them.
    - Facilitate team collaboration and help remove blockers quickly.

b) Scrum Events that Help the Team Reflect and Improve

1. **Sprint Retrospective**
    - Purpose: To reflect on what went well, what didn't, and how to improve future sprints.

    - Helps the team discuss how to better handle unexpected issues like sick leaves.
2. **Sprint Review**
    - Purpose: To present the increment to stakeholders and collect feedback.
    - Useful to assess the sprint outcome vs. plan, especially when team capacity is impacted.
3. **Daily Scrum**
    - Purpose: To inspect daily progress and adjust the work plan.
    - Keeps the team aligned and informed about each member's status and blockers.

c) Product Owner insists on adding a new story during a Sprint — Scrum Response

Correct response:
The Scrum Master should not allow adding new user stories mid-sprint unless it's a critical bug or urgent issue**.**

In Scrum, scope changes are only allowed between sprints**,** not during an active sprint.

Why?

- Maintains focus and protects the sprint goal**.**
- Sudden changes disrupt planning, velocity, and team stability.
- The new story can be added to the Product Backlog and prioritized for the next sprint**.**

d) Two Agile Principles in This Project and Their Application

1. **"Deliver working software frequently"**: Applied through sprints and incremental releases of features like login, course registration, notifications.

2. **"Customer collaboration over contract negotiation"**: The Product Owner (Head of IT) works closely with the team and users to refine and prioritize stories, ensuring the app meets real student needs.

Question 3 – ShopSmart Agile Adoption Issues

a) Three Challenges and Solutions

Challenge	Explanation	Solution
Unclear Requirements	Teams don't fully understand what to build	Conduct regular Backlog Refinement sessions
Unfinished Tasks	Poor sprint planning or overcommitting	Use past velocity **to** estimate realistically

Conflicts Over Priorities	Teams unsure what to build first	Product Owner should set clear priorities in the backlog

b) Importance of Product Backlog Refinement

- Ensures stories are:
  - Well-defined
  - Prioritized
  - Estimated
- Helps avoid confusion and delays during sprints.
- Keeps the team aligned with what the customer values most**.**

c) Improving Sprint Predictability

Scrum Master should:

1. Track Velocity Accurately**:** Use previous sprint data to guide future planning.
2. Encourage Better Story Sizing**:** Use story points and relative estimation.
3. Promote Focus**:** Protect team from distractions or mid-sprint changes.
4. Improve Definition of Done (DoD**)**: Ensure all tasks are completed to the same quality standard.

d) How Daily Scrum Improves Collaboration

- Encourages accountability: Everyone shares what they've done and what's next.
- Identifies blockers early: Team can quickly help one another.
- Aligns team goals: Keeps everyone working toward the same sprint objectives.

Question 4 – AUCA Online Registration System

a) Functional vs. Non-Functional Requirements

Type	Examples	Justification
Functional	Register for courses, pay fees online, approve registration	Specific tasks the system must perform
Non-Functional	Security, response time, availability, usability	Define how the system should perform (quality attributes)

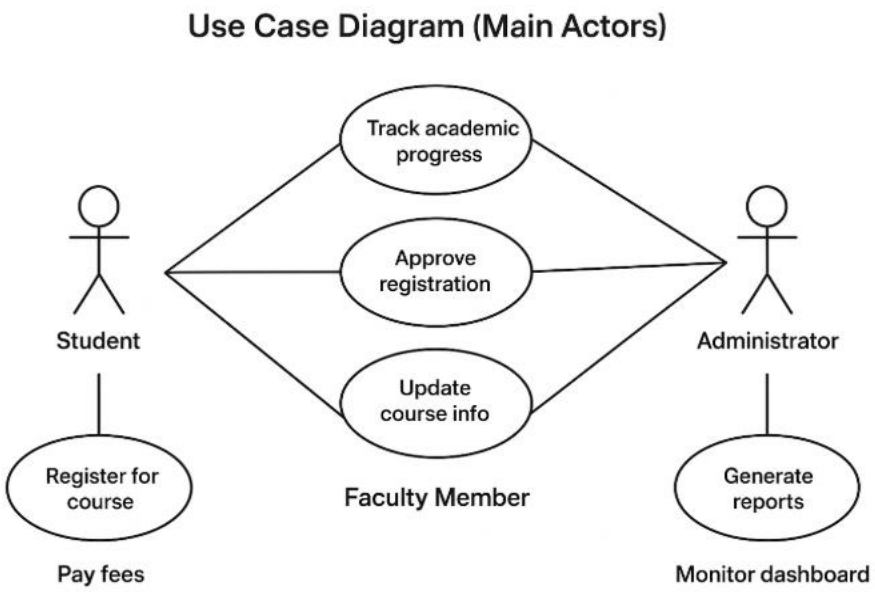
b) Use Case Diagram (Main Actors)

Actors:

- Student
- Faculty Member
- Administrator

Use Cases:

- Register for course
- Pay fees
- Track academic progress
- Approve registration
- Update course info
- Generate reports
- Monitor dashboard



c) Suitable Software Process Model: Agile Scrum

- The system involves multiple modules and evolving requirements.
- Stakeholder feedback (e.g., students, faculty) is essential.
- Agile supports incremental delivery and fast adaptation.

d) Sample Interface Specification – Course Registration Page

Feature	Description
Login Field	Enter student ID and password
Course List	Display available courses with status
Register Button	Allows student to enroll
Drop Button	Allows course removal
Credit Tracker	Displays total registered credits
Error Message	If course is full or registration is closed

e) Main Components of the Software Requirements Document (SRD)

1. Introduction – Project overview and objectives
2. Functional Requirements – What the system should do
3. Non-Functional Requirements – Performance, security, etc.
4. System Architecture – Design overview and modules
5. Interface Specifications – UI and external systems
6. Use Cases/User Stories – Scenarios describing user interaction
7. Glossary & Appendices – Definitions, references

Question 5 – Banking Loan Management System

a) Sprint Review vs. Sprint Retrospective

Event	Purpose
Sprint Review	Review what was built, get stakeholder feedback
Sprint Retrospective	Reflect on how the team worked, identify process improvements

b) Three Key Retrospective Discussion Points

1. Why some features didn't meet acceptance criteria
2. Improving testing practices and Definition of Done
3. Fixing communication between developers and testers

c) How Continuous Feedback Helps Alignment

- Regular check-ins with the Product Owner clarify expectations.
- Demoing partial features early allows for real-time feedback.
- Prevents surprises at the end of the sprint.

Question 6 – Embedded Systems for Smart Thermostats

a) Challenges: Traditional vs. Agile Models

Challenge	Traditional Model	Agile/Iterative
Flexibility	Rigid; hard to change after design	Agile adapts quickly
Hardware Constraints	Difficult to integrate late-stage hardware changes	Iterative allows hardware-software co-development
Testing	Testing is late	Agile supports continuous integration/testing

b) Suggested Hybrid Model: Spiral + Agile

Justification:

- Spiral supports risk analysis and hardware constraints.
- Agile enables iteration, continuous delivery, and customer feedback.
- Together, they balance predictability with flexibility**.**

c) Requirements Elicitation with Hardware Constraints

1. Joint Workshops: Involve both hardware and software teams.
2. Prototyping: Build early versions to validate design.
3. Use Case Scenarios: Focus on real-world usage (e.g., thermostat changes settings via app).
4. Constraints Analysis: Identify power limits, memory, compatibility with IoT protocols.

References

Gashema, G. (2025). *Lecture 1: Software process and software process models* [Lecture slides]. AUCA.

Gashema, G. (2025). *Lecture 2 & 3: Software process and software process models* [Lecture slides]. AUCA.

Gashema, G. (2025). *Requirements: Software process and software process models* [Lecture slides]. AUCA.

Pressman, R. S., & Maxim, B. R. (2014). *Software engineering: A practitioner's approach* (8th ed.). McGraw-Hill Education.

Sommerville, I. (2016). *Software engineering* (10th ed.). Pearson Education.

Schwaber, K., & Sutherland, J. (2020). *The Scrum Guide: The definitive guide to Scrum*. https://scrumguides.org

IEEE. (1998). *IEEE recommended practice for software requirements specifications* (IEEE Std 830-1998). https://ieeexplore.ieee.org/document/720574

Project Management Institute. (2017). *A guide to the project management body of knowledge (PMBOK guide)* (6th ed.). Project Management Institute.

Ambler, S. W. (2021). *Agile modeling and agile unified process*.

https://www.agilemodeling.com