Fiyin Oyewole

Part 1 – Software Development Lifecycle & SCRUM (Written)

1. SCRUM Roles & Events
    - *Briefly describe how you would contribute to a SCRUM-based team as a student intern:*

      As a student intern, I would take on the role of a Development Team Member, where I'd contribute to the planning, development, and continuous improvement of features like the automatic flagging system. I am familiar with Agile principles from past academic projects and internships, including my experience as a Project Team Member for a Microsoft AI research project, where our team applied agile cycles to iterate on chatbot design and user feedback.

    - *What SCRUM ceremonies would you participate in, and what would be your input?:*
        - Sprint Planning: I would help break down user stories into manageable tasks, provide input on time estimates, and ask questions to clarify scope. From my past work on collaborative web redesign projects, I've learned how to turn broad objectives into clear development steps, which would be valuable here.
        - Daily Stand-ups: I'd provide concise updates on my progress, share any blockers, and listen actively to understand team priorities. During my time as a Web Design Intern, standups helped our remote team stay on the same page, especially when working across multiple features.
        - Sprint Reviews: I would demo my contributions—such as a new flagging logic feature or UI feedback panel—and receive input from team members and stakeholders. I believe showing what's been built and how it works promotes transparency and trust.
        - Sprint Retrospectives: I would offer honest feedback on what went well and areas we could improve—especially around communication or clarity of requirements. In my Microsoft project, these discussions often helped our team refine our working styles and share useful tools or shortcuts.
        - Backlog Refinement: I would assist the team by reviewing and discussing upcoming tasks, asking clarifying questions about edge cases (e.g., unusual flagging scenarios), and suggesting how we could break down complex stories.

2. Software Development Life Cycle Planning
    - *Describe how you would approach redesigning the automatic flagging system following the SDLC:*

      To redesign the automatic flagging system, I would start by understanding how the current system works and where it can be improved. This means reviewing the existing rules that trigger flags, looking at how often they are accurate, and talking to the people who use the system to learn what's helpful and what's frustrating. I would then look at real data to find patterns, such as which flags are often ignored or lead to false alarms. Based on what I learn, I would create a simpler and more flexible system where rules are easy to update and clearly explained. Throughout the process, I would share ideas and early versions with the team to get feedback and make sure the system meets real user needs. My goal would be to make the flagging system smarter, easier to manage, and more helpful for users.

- *Specify the phases (Requirements, Design, Implementation, Testing, Deployment, Maintenance) and what you would do at each stage:*

  1. Requirements Gathering:

  - I would start by meeting with stakeholders (e.g., admin users, support teams) to understand the current system's limitations.
  - I'd study existing data to see what kinds of flags are being triggered and whether they're accurate.
  - I'd define specific goals, such as reducing false positives, making flag rules more flexible, and improving transparency for users.
  - Drawing from my past user research experience, I'd also consider how admin staff interact with flags on the frontend and what could make their workflow smoother.

  2. Design:

  - I'd sketch out both the backend logic (e.g., rule-based triggers for flags) and frontend elements (e.g., red/yellow flag indicators, tooltip explanations).
  - I might propose using a rule engine or decision table to make the flagging logic more modular and easier to update.
  - If possible, I'd use wireframes to show how flagged applications would appear on the portal, including filters or override options.

  3. Implementation:

  - I'd code the new backend logic using technologies the team uses—such as Python, Node.js, or a framework like Django.
  - I'd create or update REST APIs to expose flag data and enable updates or dismissals of false flags.
  - On the frontend, I'd use a JavaScript framework (e.g., React or Vue) to highlight flagged cases and display relevant rule explanations. My web development internship has prepared me for building and styling such interfaces.

  4. Testing:

  - I'd write unit tests for the flagging rules and logic, ensuring each rule triggers correctly for specific data inputs.
  - I'd perform integration testing between the frontend and backend, possibly using Postman or Cypress.
  - I'd also simulate real scenarios to test how the system handles tricky edge cases (e.g., missing documentation or conflicting application data).

  5. Deployment:

  - I would work with the team to deploy the prototype in a test environment using a version control system (e.g., GitHub) and a CI/CD pipeline.

- I’d monitor how the system behaves post-deployment and check logs for unexpected behavior or errors.

6. Maintenance:

- I’d write clear documentation explaining how the new flagging rules work and how they can be updated.
- I’d encourage regular reviews of the flagged data to identify any patterns in false positives or gaps in rules.
- Based on feedback from users and stakeholders, I’d suggest incremental improvements—an approach I’ve followed in past projects where user feedback helped guide future sprints.