Part A

Functional Requirements:

1. **Animal Movement Monitoring**: The system must monitor the movements of all animals in the park by receiving signals from radio transmitters attached to the animals and calculating their positions within 0.1 seconds.
2. **Fence Monitoring and Alarm System**: The system must detect any damage to the electric fences surrounding the animal enclosures and raise an alarm when necessary.
3. **Veterinary Notification**: The system must automatically notify the veterinary staff (via a beep or alert) if any animal is injured.

Non-Functional Requirements:

1. **Performance**: The system must compute the position of each animal within 0.1 seconds to ensure timely monitoring and response.
2. **Usability**: The system's interface must be menu-based to ensure ease of use by staff in the control center.
3. **Reliability**: The system must be thoroughly tested before delivery to ensure safety, as it is monitoring dangerous animals. It must perform without failure, especially in emergency situations like animal escapes or fence breaches.

Part B

For this project, the **Incremental Model** would be the most suitable software development life-cycle. Here's why, along with the steps involved:

Why the Incremental Model?

1. **Complexity**: The system involves several complex and critical functions (like

monitoring animal movements, alarm systems, veterinary alerts, etc.). Building the entire system in one go (as in the Waterfall Model) might lead to risks and difficulties in meeting requirements.

2. Iterative Testing and Feedback: The Incremental Model allows for testing and gathering feedback on each increment, which is crucial for a safety-critical system where continuous improvements are necessary to ensure the effectiveness of the monitoring system for dangerous animals.
3. Flexibility: It allows adjustments to be made throughout the development process. For instance, based on feedback from initial increments, improvements can be made before the full system is deployed.
4. Time Sensitivity: The park's opening is scheduled for December, and the system must be ready by then. Developing the system in increments allows early delivery of critical features (like basic animal tracking) while continuing to refine other aspects (like the interface, alarms, and veterinary notifications) in subsequent increments.

Steps Involved in the Incremental Model:

1. Requirement Analysis:
    - Understand and document all the system’s functional and non-functional requirements.
    - Divide the requirements into different modules or components that can be developed incrementally.
2. Design:
    - For each increment, create the architectural design and detailed design of the specific modules being developed. For example, start with designing the animal movement tracking and position calculation module first.
3. Implementation (First Increment):
    - Develop the core functionalities such as receiving signals from transmitters, calculating animal positions, and displaying them.
    - Test this increment thoroughly for accuracy and performance.
4. Testing:
    - After implementing each increment, test it rigorously to ensure it meets the requirements.

    - Since this system monitors dangerous animals, testing each part is critical to ensure no failures in functionality like fence breaches or alarm triggers.
5. Delivery of First Increment:
    - Once the initial version of the system with core features is working (animal tracking, fence monitoring), deliver this to the client or stakeholders to gather feedback.
    - Provide partial but functioning systems to meet immediate needs while further increments are developed.
6. Repeat Steps for Subsequent Increments:
    - Add additional functionalities like the veterinary alert system, the staff control menu, and the alarm system for fence damage in subsequent increments.
    - Test, deliver, and gather feedback for each increment.
7. Integration and Final Testing:
    - After all increments are developed and tested individually, integrate them into the final system.
    - Perform comprehensive system-level testing to ensure all parts work together, especially testing the fail-safes and alarms since they are critical for a system involving dangerous animals.
8. Deployment and Maintenance:
    - Once the entire system is integrated, deploy it for use in the park.
    - Continue to maintain and update the system based on any operational feedback or necessary fixes.

Justification for Choosing the Incremental Model:

- Reduced Risk: By building the system incrementally, you minimize the risk of discovering major issues late in the development process (a key concern for safety-critical systems like monitoring dangerous animals).
- Early Delivery of Core Features: The Incremental Model allows you to deliver parts of the system (such as core animal monitoring) early, ensuring that critical functions are operational while the system is still under development.
- Flexibility for Changes: The model allows changes to be made in subsequent

increments based on feedback, which is useful for a system that requires thorough testing and might need adjustments after seeing real-time results.

- **Time Management**: Since the park has a strict deadline for opening, delivering functional increments allows some parts of the system to be operational while further features are still being developed.

In conclusion, the Incremental Model offers flexibility, reduces risk, and allows for the system's continuous development, testing, and refinement, which are essential for such a critical and safety-focused project.

Here's a tabular analysis comparing the **Waterfall Model**, **Reuse-Oriented Model**, and the **Incremental Model** in the context of developing the computer monitoring system for the park.

Criteria	Waterfall Model	Reuse-Oriented Model	Incremental Model (Recommended)
Project Complexity	Poor fit: Linear approach struggles with complex and evolving requirements. Once a phase is completed, it's difficult to make changes.	Moderate fit: Reusing pre-existing components reduces complexity, but it depends on the availability of suitable components.	Good fit: Allows for gradual development of complex systems, breaking it down into manageable increments.
Flexibility for Changes	Poor fit: Inflexible; any changes require going back to earlier stages, which is costly and time-consuming.	Moderate fit: Changes are limited to configuring or adapting reused components. Any new changes may require finding new components.	Excellent fit: Each increment can accommodate changes based on feedback from previous increments.

Criteria	Waterfall Model	Reuse-Oriented Model	Incremental Model (Recommended)
Risk Management	Poor fit: High risk due to late testing (after the entire system is developed). Issues discovered late are difficult to fix.	Moderate fit: Lower risk if high-quality components are reused. However, dependency on third-party components introduces risks.	Good fit: Risks are mitigated as each increment is tested and validated separately, allowing issues to be identified early.
Delivery of Core Features Early	Poor fit: System is only delivered once all phases are complete, so core functionalities are delayed until the end.	Moderate fit: Core features could be delivered early if relevant components are available for reuse.	Excellent fit: Core features can be delivered early as increments are completed. Allows early functionality like animal tracking to be operational while other parts are still under development.
Thorough Testing	Poor fit: Testing only occurs after the entire system is developed, increasing the risk of major issues being found late in the process.	Moderate fit: Reused components may be pre-tested, but overall system testing still happens at the end, increasing integration risks.	Excellent fit: Each increment is tested thoroughly before moving to the next, which is critical for a system handling dangerous animals.
System Integration	Poor fit: Integration happens late in the project, leading to potential	Moderate fit: Integration of reused components may simplify some aspects but could introduce new issues	Good fit: System integration happens progressively with each increment, reducing the risk of incompatibilities.

Criteria	Waterfall Model	Reuse-Oriented Model	Incremental Model (Recommended)
	problems if components don't work well together.	if the components aren't perfectly compatible.	
Time Sensitivity	Poor fit: Entire system must be completed before delivery. The December deadline might be hard to meet if unexpected delays occur.	Moderate fit: Could save time if reusable components are easily adaptable, but may face delays if significant customization is needed.	Good fit: Increments allow for partial delivery of functional systems, making it easier to meet deadlines while continuing development.
Budget Constraints	Moderate fit: Waterfall could lead to cost overruns due to the rigid structure and costly rework in case of late changes or issues.	Moderate fit: Reusing components may lower development costs, but licensing or customization costs for third-party components could increase the budget.	Good fit: Incremental development allows better budget management as development can be spread out and adjusted based on the progress and feedback.
Suitability for Critical Systems	Poor fit: Delayed testing and integration make it risky for safety-critical systems like monitoring dangerous animals.	Moderate fit: Reused components may be reliable, but full system reliability depends on component integration and customization.	Excellent fit: Each increment can be thoroughly tested, making it ideal for critical systems where safety and reliability are essential.

Criteria	Waterfall Model	Reuse-Oriented Model	Incremental Model (Recommended)
Dependence on External Components	N/A: Developed entirely in-house, with no reliance on external components.	High: Depends heavily on availability and compatibility of reusable components. Any failure in finding suitable components could derail the project.	Low: Mostly developed in-house with controlled use of external components, offering greater control over the system's functionality.

Summary of Why Waterfall and Reuse-Oriented Models Do Not Fit:

1. Waterfall Model:
    - Rigid structure: Since the system involves complex safety-critical components, any changes or issues discovered during development will be difficult and costly to address using Waterfall. This model does not offer flexibility, and testing is performed only at the end, which is too risky for such a critical project.
    - Late integration: The animal monitoring system must function seamlessly, and if issues arise during late-stage integration, they can cause delays that jeopardize the opening deadline.
    - Poor fit for evolving requirements: If requirements change during development (for instance, based on feedback from initial tests), the Waterfall Model would not accommodate these changes well.
2. Reuse-Oriented Model:
    - Dependency on existing components: This model relies heavily on the availability of reusable components, which may not exist for such a specialized system (e.g., monitoring dangerous animals). Customizing reusable components may also require significant effort, negating the time and cost benefits of reuse.
    - Integration issues: Reusing components from different sources may

cause integration challenges. In a system that requires real-time monitoring and safety measures, these challenges could introduce critical delays.
- Limited flexibility: While this model could save time and costs if the right components are found, it lacks the flexibility needed for evolving and custom requirements of the park's monitoring system.

Why Incremental Model is the Best Fit:

The Incremental Model provides the right balance of flexibility, risk management, and timely delivery. Its iterative approach allows for early delivery of critical features, continuous testing, and adaptability, making it the ideal choice for this project.

Part C

During requirements elicitation and analysis for the computer monitoring system for the park hosting dangerous animals, several difficulties may arise. Here are six potential challenges:

1. Incomplete or Ambiguous Requirements:

- Stakeholders may not have a clear understanding of what they need from the system. For example, they might provide vague descriptions of how the monitoring or alert systems should work. This can lead to incomplete requirements that need further clarification and can cause delays in the development process.

2. Diverse Stakeholder Needs:

- The project involves multiple stakeholders such as park managers, game wardens, veterinarians, safety officials, and possibly even tourists. Each group may have different needs and priorities, which can make it difficult to reconcile all of them into a cohesive system design.

3. Lack of Technical Knowledge from Stakeholders:

- Many stakeholders may not be familiar with the technical aspects of the system (e.g., radio signal transmission, real-time data processing, or software interface design). As a result, they may struggle to express their needs in terms that are actionable for the development team, leading to communication gaps and misunderstandings.

4. Complex Safety Requirements:

- Since the system involves monitoring dangerous animals, the safety requirements are critical and must be very precise. Determining the exact parameters for alarms, fence monitoring, and emergency protocols may be difficult because there is little room for error. A slight misunderstanding in safety protocols could lead to significant system failures, compromising safety.

5. Evolving Requirements:

- As the park nears completion or during system development, stakeholders may realize that certain aspects of the system need to change. For example, they might decide to add more animals or introduce new tourist areas, which would affect the monitoring system. These evolving requirements can make it difficult to finalize the system specifications.

6. Feasibility Constraints:

- Balancing the requirements with the project's budget (Rs 6 million) and time constraints (deadline in December) might be challenging. Stakeholders may have high expectations for the system's features (such as very detailed animal tracking or advanced alarms), but some of these requirements may not be feasible within the given cost and timeline. Determining which requirements to prioritize within these constraints can be difficult.

These challenges make it essential to maintain clear and ongoing communication with stakeholders, engage in iterative requirements gathering, and use techniques like prototypes and user stories to refine and clarify the system's needs.

Part D

As the project manager, the Software Requirements Specification (SRS) document should possess certain desired characteristics to ensure clarity, precision, and completeness in defining the system's functionality and constraints. Below are the desired characteristics of the SRS document along with explanations:

1. Correctness

- The SRS must accurately reflect the actual needs and requirements of the stakeholders. Every requirement listed should be necessary, and there should be no incorrect or outdated information. The correctness of the SRS ensures that the system will meet stakeholder expectations and deliver the intended functionality.

2. Completeness

- The SRS should contain all necessary information about the system, including all functional and non-functional requirements. This means every system behavior, constraint, and assumption is documented, and nothing important is left out. A complete SRS reduces the risk of missing features and ensures the development team understands the full scope of the project.

3. Unambiguity

- Each requirement in the SRS must be stated clearly and unambiguously. There should be no room for multiple interpretations of the same requirement. For example, a statement like “The system should be user-friendly” is ambiguous because “user-friendly” is subjective. Instead, precise, measurable criteria should be used (e.g., “The system shall allow a trained user to perform task X within Y minutes”).

4. Consistency

- The SRS must ensure that there are no conflicting requirements. For example, if

one part of the document specifies a system should be accessible from mobile devices while another part limits access to desktop computers, it creates inconsistency. Consistency means all requirements, terminology, and assumptions throughout the document should be in agreement.

5. Verifiability

- Every requirement in the SRS must be written in a way that allows for it to be tested or verified. A verifiable requirement provides clear criteria for determining whether the system meets that requirement. For example, “The system shall compute the position of each animal within 0.1 seconds” is verifiable because it can be measured and tested.

6. Modifiability

- The SRS should be structured in a way that allows changes to be made easily without requiring extensive rework. This is important because requirements may evolve during the development process. A modifiable SRS should have clear sections and subsections, making it easy to locate and update specific requirements.

7. Traceability

- The SRS must allow for easy traceability of requirements, meaning it should be possible to trace each requirement from its origin (e.g., a stakeholder need) through its design and implementation stages, and finally to the testing phase. Traceability ensures that no requirement is overlooked during development and testing, and that all functionality is covered by test cases.

8. Prioritization

- The SRS should indicate the priority of each requirement, especially when there are constraints on time and budget. High-priority requirements are those that must be implemented for the system to function properly, while low-priority requirements can be deferred or dropped if necessary. This helps manage stakeholder expectations and guide development focus.

9. Understandability

- The SRS should be easy to read and understand by all stakeholders, including non-technical stakeholders such as park management. It should avoid overly technical language and jargon whenever possible and should provide explanations or diagrams where necessary to enhance understanding.

10. Feasibility

- The SRS should only include requirements that are technically and practically feasible within the project's constraints (budget, time, and available resources). Unfeasible requirements that are impossible or overly costly to implement should not be included. This helps prevent unrealistic expectations and scope creep.

11. Conciseness

- While the SRS should be comprehensive, it should also be concise and to the point. Extraneous information should be avoided, and requirements should be stated in a clear and succinct manner. Conciseness helps reduce confusion and makes the document easier to review and maintain.

12. Maintainability

- The SRS should be organized and written in a way that makes it easy to update as the project progresses. Maintainability ensures that the SRS remains a living document, reflecting any changes in scope, requirements, or system constraints over time. Well-structured documents with version control facilitate this.

13. Scalability

- The SRS should account for potential future expansion or scaling of the system. For example, if the park plans to add more animals in the future, the system should be designed with scalability in mind. This means that future requirements can be accommodated without significant system redesign or rework.

Conclusion:

An SRS document is the foundation for successful software development and must be **clear, precise, and thorough**. A well-prepared SRS not only guides developers but also ensures that all stakeholders have a shared understanding of what the system will do, reducing the risk of project failure due to misunderstandings or missing requirements. By ensuring the SRS adheres to these desired characteristics, the project manager can ensure the system is developed efficiently, within budget, and meets the park's needs for safety and functionality.