Problem Statement:

Driverless Cars, a technology firm specializing in self-driving car software development, aims to create an advanced software that empowers vehicles to autonomously navigate, park, and parallel park without human intervention. This encompasses the seamless execution of driving and parking maneuvers, prioritizing safety through the avoidance of collisions with pedestrians and other vehicles.

In this proposal, a comprehensive evaluation will be conducted to assess the merits and drawbacks of employing the Waterfall Method, Rational Unified Process (RUP), and Extreme Programming methodologies. The focus will be on developing a state-of-the-art software module specifically designed for parallel parking at slow speeds, leveraging an array of sensors, cameras, and motors.

Problem Description:

Developing a robust, safe, and efficient parallel parking module entails:

- Sensor fusion: Integrating data from diverse sensors like LiDAR, radar, cameras, and ultrasonic sensors.
- Environment perception: Accurately mapping surroundings including parked vehicles, curbs, pedestrians, and potential obstacles.
- Path planning: Calculating the optimal parking trajectory while accounting for real-time changes.
- Motor control: Precisely actuating steering, throttle, and brakes to execute the planned path smoothly and safely.
- Safety and reliability: Ensuring absolute safety and error-free performance, as any malfunction could risk collisions and damage.

Waterfall Methodology:

Process and Priorities:

In the context of developing parallel parking software for a driverless car using the waterfall methodology, the requirements phase would involve gathering and

comprehensively understanding the specific functional and performance requirements for the parallel parking system. This includes considerations such as sensor accuracy, environmental awareness, and precise control of the vehicle during parking maneuvers. In the design phase, the technical aspects of the solution would be meticulously planned involving scenarios like different parking space layouts, sensor integration, and the creation of detailed data models to facilitate the car's decision-making process. The implementation phase would involve coding the software according to the specified requirements and design specifications while taking into account the integration of sensor data and ensuring accurate vehicle movements during parking. Testing and verification would be crucial focusing on rigorous testing to eliminate any defects in the software that could compromise the safety and effectiveness of the parallel parking system. Finally, during the deployment and maintenance phase, the software would be released for real-world application in driverless cars, and ongoing maintenance efforts would be initiated to address any issues that arise post deployment/release, ensuring the continued reliability and efficiency of the parallel parking functionality. The waterfall method prioritizes planning everything in sufficient detail up front. It’s a very formal process with strict handoffs and serial execution.

Advantages:

Detailed Planning:

- Developing self-parking software for driverless cars is a complex task, and the waterfall methodology's emphasis on comprehensive planning allows for a thorough understanding of project requirements such as sensor integration needs upfront. This is particularly beneficial in domains where safety is crucial and where precise specifications are crucial. The formal process may aid in systematically integrating sensors like cameras, enhancing the overall reliability.

Formal Process:

- The formal and structured nature of the waterfall model ensures that each stage is completed before moving on to the next one. In the context of self-driving cars and their safety-critical nature, having a clear sequence of steps can instill a sense of control and assurance, which is vital for ensuring the reliability of self-parking functionality. Systematic development can enhance the reliability of mapping and localization components.

Documentation:

- Excessive documentation, often seen as a disadvantage in other contexts, becomes advantageous in safety-critical systems like self-driving cars. Detailed

documentation serves as a valuable reference for understanding the self-parking system's functionality, design decisions, and potential safety issues. It can be helpful in understanding things such as the vehicle's navigation logic during parking.

Disadvantages:

Rigidity in Adapting to Changes:

- The waterfall methodology's rigidity poses a challenge in accommodating changes once the project is underway. In the dynamic field of self-driving technology, where requirements may evolve due to new insights or emerging technologies, the waterfall model's lack of adaptability can hinder the development process.

Handoffs Between Stages and Communication Gaps:

- Strict handoffs between stages may lead to communication gaps, particularly in a dynamic project like developing self-parking software. Misinterpretation or oversight during handoffs can result in design flaws or incomplete implementations which may potentially affect the reliability of the self-driving and self-parking system.

Cost of Failure in Safety Critical Systems:

- While the waterfall model aims for accuracy from the start, the cost of failure can be exceptionally high in the context of self-driving cars. Errors in the initial stages might continue throughout the project leading to costly rework or safety issues in the self-driving and self-parking functionality.

Limited Flexibility for Evolving Requirements:

- The waterfall model is not well-suited for projects with evolving requirements due to its prescriptive, linear, and sequential approach. In the case of self-driving cars, the technology is dynamic and new sensor technologies or regulatory requirements may emerge during the development process.

Time-Consuming Nature and Timely Delivery:

- The sequential nature of the waterfall model can be time consuming and may delay the delivery of the self-parking software. In an industry that advances rapidly such as autonomous vehicles, timely delivery is crucial to staying competitive and the waterfall model's sequential approach may create challenges in meeting tight timelines.

Rational Unified Process Methodology

Process and priorities:

The Rational Unified Process project lifecycle phases are Inception, Elaboration, Construction, and Transition. These phases occur for various disciplines such as business modeling, requirements, analysis and design, implementation, testing, deployment, configuration and change management, project management, and environment.
In the inception phase of developing driverless car parallel parking software, the emphasis is on defining project scope and estimating costs, with a focus on specific requirements for the feature. The elaboration phase involves refining the use case model and designing a robust software architecture, addressing challenges like sensor inaccuracies and unexpected obstacles. The construction phase includes implementing algorithms for real-time sensor data processing, decision-making, and rigorous testing for accuracy and reliability in diverse parking scenarios. Finally, during the transition phase, the software is delivered to customers with a deployment plan, ensuring integration into the overall driverless car system and ongoing monitoring for performance issues and resolutions. Throughout, safety, regulatory compliance, and user experience are critical considerations in this iterative development process. RUP prioritizes iterative development and a focus on the highest risk items during each iteration. It also emphasizes requirements management through the use of use cases and scenarios. Software architecture must also be component based and software is modeled visually in UML. Verification and validation is an important aspect as is change management.

Advantages:

Develop Iteratively:

- Advantage: Breaking down the project into iterative chunks allows for a focused approach on specific aspects, enabling early involvement of customers and accommodating changes in requirements.
- In the context of driverless cars, iterative development allows for quick adaptation to evolving technologies and changing regulatory requirements.

Don’t Need to Wait Until Phase is 100% Done Before Starting Another Phase:

- The advantage of this is different aspects of the driverless car software can be developed in parallel. It’s also possible to validate and test components of the software as they are completed.

Manage Requirements:

- RUP emphasizes managing requirements effectively which is crucial in the development of a complex system like autonomous parallel parking.

- Given the safety requirements in autonomous driving, managing and tracing requirements is essential for compliance and reliability.

Using Component-Based Architectures:

- It promotes reusability of software components which effectively reduces development time and effort. It aligns with the goal of not rewriting everything from scratch for each project.
- As technology evolves, using reusable components can accelerate the development of specific functionalities like parallel parking without compromising on reliability.

Model Software Visually (UML):

- Visual modeling with UML aids in capturing both the structure and behavior of the system. It provides a clear understanding of the system and its interactions.
- Visualizing complex algorithms and interactions involved in autonomous parking can enhance the design and communication among team members.

Continuously Verify Software Quality:

- Integrating verification and validation throughout the process ensures a focus on reliability, functionality, and performance from the early stages.
- Given that safety is crucial for self-driving technology, continuous quality verification is vital to prevent accidents and ensure a high level of performance.

Control Changes:

- Actively managing change requests allows for controlled, tracked, and monitored changes, preventing chaos in the development process.
- In a rapidly evolving field, managing changes is crucial to adapting to new sensor technologies, camera technologies, regulations, or safety standards.

Disadvantages:

Structured Phases:

- The structured phases in RUP may lead to potential delays, especially if a phase must be 100% complete before moving to the next one.
- Given the uncertainty in developing cutting-edge technology, rigid phase dependencies may hinder the ability to quickly respond to emerging challenges or opportunities.

Potentially Lengthy Phases:

- The time frames for the inception and elaboration phases can vary and might be lengthy, depending on the project complexity.

- In a fast-paced industry such as autonomous technology, extended durations for initial phases may delay the launch of autonomous parking solutions.

Resource Intensive:

- RUP requires dedicated resources for disciplines such as configuration and change management, project management, and environment management.
- Allocating resources for these disciplines might divert focus and resources away from the core development of autonomous parking functionality.

Not Tailored for Rapid Prototyping:

- RUP may not be as well-suited for rapid prototyping, which is often crucial in emerging technology domains.

No Explicit Time Windows:

- RUP does not provide explicit time windows for phases, which may lead to uncertainty in project planning.
- Uncertain time frames can be challenging for project scheduling and coordination, especially when dealing with external factors like regulatory changes.

Extreme Programming Methodology:

Process and Priorities:

Developing self-driving car software using Extreme Programming follows a strategic approach. There will be a planning stage where user stories are developed that address autonomous parking, emphasizing features like space detection and safe navigation during parking maneuvers and prioritizing safety in release plans. Releases will be small. Iteration planning will address challenges like refining sensors and enhancing decision-making. The next phase is managing which will involve optimizing workspaces for safety and collaboration, maintaining a steady pace, measuring project velocity and allocating expertise for autonomous parking. The designing phase will address self-driving challenges, emphasizing simplicity in design. During this stage a metaphor is agreed upon by all members of the team. Safety features will be prioritized early and there will be refactoring practices to enhance the adaptability and maintainability of the self-driving car codebase, especially in response to evolving safety standards. During the coding phase there will be tailored coding standards which prioritizes safety and reliability. Prioritizing Test Driven Development involves creating unit tests for robust functionality. Pair programming and serial integration ensure collaborative

problem-solving and controlled updates. Frequent code integration in a dedicated environment will address unique challenges such as spatial awareness, sensor limitations, and precision maneuvering. Fostering collective ownership enables team members to contribute, ensuring safety and reliability. Testing is an integral aspect of the XP methodology. Unit tests for all code components emphasizes a focus on safety-critical functionalities. It is crucial to ensure that all code passes these unit tests before release, validating that changes do not compromise safety or introduce defects. In the context of identifying bugs, tests are developed promptly to enable quick identification and resolution within the self-driving car system. Furthermore, running acceptance tests frequently plays a crucial role in evaluating the overall performance and safety of the self-driving car in various real-world scenarios and with real-world driving data. Extreme programming prioritizes planning, small releases, metaphors, simple design, refactoring, testing, pair programming, collective ownership, continuous integration, a sustainable pace, whole team and coding standards.

Advantages:

Close collaboration between developers and customers:

- This ensures a thorough understanding of the specific requirements and expectations of customers in the development of self-driving car features. Regular communication allows for real-time feedback from customers which enables developers to make adjustments and improvements promptly.

Small, Valuable Releases:

- This enables the development of parallel parking capabilities and allows for regular updates and improvements without overwhelming the system.

Automated Testing:

- This ensures that the software controlling parallel parking is thoroughly tested in various simulated and real-world scenarios which reduces the amount of defects, the risk of accidents, or the car malfunctioning.

Refactoring:

- This contributes to maintaining and enhancing the simplicity of the software which is crucial for reliability and safety.

Collaborative Ownership:

- This facilitates cross-functional collaboration so that developers, safety experts, and algorithm specialists collectively contribute to the success of the parallel parking module.

Continuous Integration:

- Regular integration and testing help identify and fix issues early in the development process. This is important for ensuring the safety and reliability of the self-driving car's parking capabilities.

Pair programming:

- Increases collaboration and the speed of problem solving. It can also result in enhanced code quality and reduced defects.

Disadvantages:

On-site Work Requirement:

- In the development of autonomous vehicle software experts in various domains, such as sensor technology and safety standards, may not always be co-located. This can pose challenges for on-site collaboration.

Time Consuming:

- Rigorous testing and continuous integration may extend the development timeline. This may potentially delay the deployment of self-parking capabilities to end-users.

Voluntary Coding Standards:

- In a safety-critical environment, ensuring consistent coding standards is crucial. Depending on voluntary adoption may result in inconsistencies that could impact the overall reliability and safety of the software.

Simple Design:

- Driverless car software can be extremely complex, involving sophisticated algorithms for perception, decision-making, and control. Extreme programming is generally considered more suitable for smaller to medium-sized projects, and the complexity and scale of autonomous systems may pose a challenge to the simple design advocated by XP.