# CS 255 Model Application Short Paper

Andrew Lemus

Andrew.lemus@snhu.edu

Southern New Hampshire University

**Process Model Application**  
To guide the development of the DriverPass system, I would apply a Spiralprocessmodel, which emphasizes iterative refinement and risk management. In the very first loop, I’d sit down with Liam and his team to hash out exactly what they need—like booking lessons, taking practice tests, and hooking up with the DMV API—and figure out any tech unknowns (for example, “Can we really integrate with the DMV in real time?”). Then I’d build a quick prototype of the scheduling feature just to make sure our assumptions hold up. Based on those findings, a prototype of the core scheduling module would be built to validate assumptions about in-memory versus persistent storage. User feedback from this prototype would inform the next cycle.

In the next round, I’d refine that scheduling module—adding automatic instructor/vehicle assignment, better error handling, and some basic UI mock‑ups. Development would proceed in increments: the second spiral would enhance the scheduling module by adding instructor and vehicle assignment logic, while the third would tackle the online course and testing components. Each cycle would conclude with a formal review—evaluating performance metrics (e.g., response time under load), security assessments (e.g., authentication workflows), and stakeholder satisfaction. This structure ensures that high‑risk elements (like secure role‑based access and DMV integration) are addressed early, reducing the likelihood of costly rework later in the project (Pressman, 2014).

By the fourth and fifth spirals, attention would shift to reporting features, cloud deployment pipelines, and user interface refinements. Continuous integration and automated testing (including JUnit for appointment services) would be integrated throughout each cycle to maintain code quality and accelerate feedback. The Spiral model’s flexibility accommodates evolving requirements—critical in a domain where DMV regulations and user expectations may change—while its emphasis on risk-driven prototyping keeps the project aligned with DriverPass’s business goals.

**Object Model Application**  
When it comes to the object model, I’d map out the real‑world pieces of DriverPass as classes. You’d have a base User class, then subclasses or roles like Customer, Instructor, and Admin. The Customer class would keep track of personal info plus a list of Appointment objects. Each Appointment would link to an Instructor and a Vehicle object. Then there’d be a Course class for the online modules, with Test objects inside it to store scores. For example, Instructor and Secretary might extend a base Staff class that defines shared attributes (e.g., staffId, name, role). The Customer class would encapsulate personal details, progress metrics, and a collection of Appointment objects.

The Appointment class, already defined in the milestone, would maintain associations to both an Instructor and a Vehicle object, reflecting the many‑to‑one relationship between lessons and instructors/vehicles. A separate Course class would represent online modules, each containing multiple Test objects to track practice exam results. The DMVIntegration class would encapsulate API‑specific methods for fetching regulation updates, while a Notification class would manage alerts and reminders. Finally, a ReportGenerator class would use data from these entities to produce financial and usage reports.

This object model facilitates encapsulation (each class manages its own data and behavior), modularity, and reusability—for instance, the same Notification class could serve both lesson reminders and security alerts. It also eases maintenance: if the DMV API changes, only the DMVIntegration class needs modification. UML class diagrams and sequence diagrams would be used during design to verify that the model accurately represents system behavior before coding begins (Sommerville, 2011).

**Process and Object Model Comparison**  
Each modeling approach offers distinct benefits and trade‑offs for the DriverPass scenario. The Spiralprocessmodel excels at riskmanagement and stakeholderengagement, ensuring that critical uncertainties—such as secure authentication, performance under load, and external integrations—are addressed early. Its iterative nature allows for frequent demonstrations for Liam’s team, fostering trust and enabling course corrections before substantial resources are committed. However, the Spiral model can be time**‑**consuming and documentation**‑**heavy, potentially slowing down development if cycles become overly bureaucratic.

In contrast, the objectmodel provides a clearmapping between real‑world entities (customers, lessons, vehicles) and software constructs, improving maintainability and enabling code reuse. By defining well‑encapsulated classes, developers can work in parallel on different components (e.g., scheduling, testing, reporting) with minimal conflict. Yet object modeling can become overlycomplex if too many classes or relationships are defined prematurely, leading to “analysis paralysis.” Moreover, an object model alone does not prescribe when and how to build features—it must be paired with a process model to guide actual development.

In practice, combining the two approaches yields the best results: the Spiral model directs the when and how of development cycles, while the object model defines the what of the system’s static structure. Together, they ensure that DriverPass is delivered on time, within budget, and aligned with both technical constraints and business objectives.

**References**  
Pressman, R. S. (2014). *Software engineering: A practitioner’s approach* (8th ed.). McGraw‑Hill.

Sommerville, I. (2011). *Software engineering* (9th ed.). Addison‑Wesley.

## References

[You were **not** requiredto use external resources for this report. If you did not use any resources, you should remove this entire section. However, if you did use any resources, including the textbook, to help you with your interpretation, you **must** cite them. Use proper APA format for citations. Two samples have been included below for you to use as a reference.

For a book:

Author, A. A. (Year of publication). *Title of work: Capital letter also for subtitle*. Location: Publisher.

For a website:

Author, A. A. (Year, Month Day). Title of article. *Title of Website*. Retrieved from  
https://www.homeaddress.com/