Software Design & Architecture Project

Project Overview

For this project, you will be presented with an existing preliminary prototype code base, and you will need to decide on patterns to improve it, while also identifying anti-patterns and removing them. This project will give you some hands-on experience in implementing patterns, identifying anti-patterns and also designing software architectures. You can complete this project either individually or as a group of no more than **three** people.

Learning Outcomes

When completed successfully, this project will enable you to:

- Pinpoint bad design, problematic source code, and bad coding practices in need of improvement
- Identify and justify opportunities for including software design patterns
- Implement software design patterns in an existing code base
- Utilize object-oriented design principles in this process
- Utilize UML, both structural and behavioral diagrams, to represent software designs
- Utilize domain-driven design principles to model a system and develop a high-level microservices architecture

Project Description

Understanding, updating and extending from an existing codebase is a crucial skill to develop if you plan to work in the software industry. A substantial portion of what you will work on will be fixing, maintaining and extending from existing source code. It can be quite rare to code directly from scratch, and even in such cases, your own code will evolve over time and demand more polish and refinement. Key to this process of improving software quality is the identification and implementation of design patterns, handy solutions to common software problems. As code becomes more unwieldy, we can use patterns to shore it up. Structural patterns might enhance reusability, architectural patterns can reduce duplication, and behavioral patterns simplify objects and divvy up responsibilities. All patterns bring something to the table. The trick is knowing when to employ them and to justify their existence, not simply using them for the sake of it.

On the other hand, just like patterns are tools for best practices, there are also “anti-patterns”. These are poor coding practices and design choices that can complicate and weaken source code, making it difficult to read and prone to errors. Some common “anti-patterns” include:

- Poor/non-explicit variable names
- Lack of comments or useless comments
- Passing mutable arguments
- Returning different types in a function
- Clumsy, unnecessary loop statements
- Superfluous/lengthy/nested if statements
- Global variables
- Using try/except blocks without handling exceptions
- Broad import statements
- Deprecated, unused code blocks (i.e. “Dead Code”)
- Unnecessary abstractions

The source code you will be using for this project is of an initial prototype parking lot manager application for a single parking lot. It is an initial prototype of parking lot management company EasyParkPlus developed internally, and they have now contacted you as a software engineering expert. Download and run the baseline source code for the Parking Lot Manager here.

Take some time to explore the code and understand what it is doing. You should draw out some of the code’s structure, logic and flow of data using appropriate UML diagrams;

this can help clarify areas of overlap and confusion. Once you are comfortable with the operations of the program, begin improving it, utilizing at least **two distinct patterns** from the OO-related ones you have studied so far. In addition, remove any anti-patterns and poor coding elements you encounter, while adding in your own comments and clarification. These fixes and improvements should not simply be cosmetic; they should provide significant structural and architectural improvements to the system. In a separate document, prepare a **written justification** outlining the fixes you have made and why you chose your selected patterns.

Additionally, EasyParkPlus, is also planning to scale up this prototype application to handle their operations across multiple of their parking lot facilities. The parking lot company will also add a new business activity related to electric vehicle charging and add a new feature to the parking lot application: **Electric Vehicle (EV) Charging Station Management.**

To extend their system you will consider their business and system from a domain-driven design perspective, with an eye to developing a scalable microservices architecture-based solution. In architecting this software, including the new EV charging capability, you should use a domain-driven design approach, and propose a resultant microservices-based architecture to support the overall application—you are asked to **describe the microservice-based software architecture**, but you are not required to develop the extended microservices software implementation at this stage.

To apply domain-driven design:

- Identify the core domain and core subdomains
- Define bounded contexts
- Design sample ubiquitous language for each context
- Model domain entities, value objects and aggregates

To help you with any questions you may have about the technical and operational details and language and vocabulary used, the Technical Manager at EasyParkPlus, Michael, has conveniently made himself available here: Michael, Technical Manager.

Design a preliminary microservices architecture:

- Identify services (align with bounded contexts)
- Identify key responsibilities of each service
- Describe APIs/endpoints (external facing and service-to-service endpoints)
- Identify separate DBs per service

Tips & Resources

- To complete this activity, the latest version of Python 3 is required. You will be responsible for making sure your assignments work with Python. If you're unsure what version of python you are using, you can always check by typing `python3 --version` on the command line.
- If you find that any libraries are missing, you may need to use the pip command to install them. More on that here: https://packaging.python.org/en/latest/tutorials/installing-packages/
- The written justification does not have a page requirement, but it should adequately address your chosen patterns, why they were included, and how you fixed the anti-patterns.
- The written portion describing the microservices-based architecture does not have a page requirement, but should provide a high level architecture as per the submission instructions
- The Gang of Four – Design Patterns: https://springframework.guru/gang-of-four-design-patterns/
- Refactoring: Improving the Design of Existing Code: https://books.google.com/books?hl=en&lr=&id=2H1_DwAAQBAJ
- When and Why Your Code Starts to Smell Bad: https://ieeexplore.ieee.org/abstract/document/7817894
- Domain-Driven Design & Microservices: https://www.geeksforgeeks.org/domain-oriented-microservice-architecture/

Submission Guidelines

Your final submission should consist of a packaged **.zip or .rar file** containing your written justifications for your design choices, including:

- Two UML diagrams (one structural, one behavioral) representing the design of the code you first downloaded
- Two UML diagrams (one structural, one behavioral) representing the re-designed code you have developed and submitted
- Your updated source code for the parking application
- Screenshots (or brief video) of the application running on your computer
- Written document of how you used domain-driven design to model the Parking Management System, including the Electric Vehicle (EV) Charging Station Management extension to the system, and your proposed high-level microservices-based architecture of the system including:
    - A high-level bounded context diagram
    - Basic domain models for parking management and EV charging

    - A proposed microservices architecture diagram including services, APIs/endpoints and per service DBs

To submit your project, please click on the "Submit Project" button on your dashboard and follow the steps provided in the Google Form. If you are submitting your Software Design and Architecture project as a group, **please ensure only ONE member submits on behalf of the group**. You will also be prompted to upload the final page of your Group Project Agreement, which must be completed and signed by all group members. Please reach out to msse+projects@quantic.edu if you have any questions. Project grading typically takes about 3-4 weeks to complete after the submission due date. There is no score penalty for projects submitted after the due date, however grading may be delayed.

Plagiarism Policy

Here at Quantic, we believe that learning is best accomplished by “doing”—this ethos underpinned the design of our active learning platform, and it likewise informs our approach to the completion of projects and presentations for our degree programs. We expect that all of our graduates will be able to deploy the concepts and skills they’ve learned over the course of their degree, whether in the workplace or in pursuit of personal goals, and so it is in our students’ best interest that these assignments be completed solely through their own efforts with academic integrity.

Quantic takes academic integrity very seriously—we define plagiarism as: “Knowingly representing the work of others as one’s own, engaging in any acts of plagiarism, or referencing the works of others without appropriate citation.” This includes both misusing or not using proper citations for the works referenced, and submitting someone else’s work as your own. Quantic monitors all submissions for instances of plagiarism and all plagiarism, even unintentional, is considered a conduct violation. If you’re still not sure about what constitutes plagiarism, check out this two-minute presentation by our librarian, Kristina. It is important to be conscientious when citing your sources. When in doubt, **cite**! Kristina outlines the basics of best citation practices in this one-minute video. You can also find more about our plagiarism policy here.

Project Rubric

Scores 2 and above are considered passing. Students who receive a 1 or 0 will not get credit for the assignment and must revise and resubmit to receive a passing grade.

Score	Description
5	● Addresses all of the project requirements, including but not limited to: ○ Design and code improvements appropriately use two relevant design patterns ○ Written report is detailed and documents changes made and reasons the patterns were utilized ○ Original design is represented correctly using two appropriate UML diagrams ○ Redesign is represented correctly using two appropriate UML diagrams ○ All bad coding practices present in the code base are identified ○ Appropriate improvements are made to each of these bad coding examples ○ Appropriate bounded context diagram is provided ○ Detailed DDD-based domain models are provided ○ High-quality microservices architecture diagram is provided ○ Your submission has been correctly submitted in the format requested, including the updated source code, screenshots of the application running on your computer and the written report
4	● Addresses most of the project requirements, including but not limited to: ○ Design and code improvements appropriately use two relevant design patterns ○ Written report is detailed and explains changes made and reasons the patterns were utilized ○ Original design is represented correctly using two appropriate UML diagrams ○ Redesign is represented correctly using two appropriate UML diagrams ○ Most bad coding practices present in the code base are identified ○ Appropriate improvements are made to each of these bad coding examples ○ Appropriate bounded context diagram is provided

	○ Good DDD-based domain models are provided ○ Good microservices architecture diagram is provided ○ Your submission has been correctly submitted in the format requested, including the updated source code, screenshots of the application running on your computer and the written report
3	● Addresses some of the project requirements, including but not limited to: ○ Design and code improvements appropriately use some relevant design patterns ○ Written report explains some changes made and reasons the patterns were utilized ○ Original design is represented mostly correctly using two appropriate UML diagrams ○ Redesign is represented mostly correctly using two appropriate UML diagrams ○ Some bad coding practices present in the code base are identified ○ Appropriate improvements are made to some of these bad coding examples ○ Adequate bounded context diagram is provided ○ Adequate DDD-based domain models are provided ○ Adequate microservices architecture diagram is provided ○ Your submission has been correctly submitted in the format requested, including the updated source code, screenshots of the application running on your computer and the written report
2	● Addresses few of the project requirements, including but not limited to: ○ Design and code improvements did not appropriately use some relevant design patterns ○ Written report explains few changes made and reasons the patterns were utilized ○ Original design is represented using at least one appropriate UML diagrams ○ Redesign is represented using two appropriate UML diagrams ○ Only a few bad coding practices present in the code base are identified ○ Appropriate improvements are made to only few of these bad coding examples ○ Basic bounded context diagram is provided ○ Adequate DDD-based domain models are provided ○ Adequate microservices architecture diagram is provided

	○ Your submission has been correctly submitted in the format requested, including the updated source code, screenshots of the application running on your computer and the written report
1	● Addresses the project but is missing most requirements, including but not limited to: ○ Design and code improvements did not appropriately use some relevant design patterns ○ Written report is provided but does not explain changes made and/or reasons the patterns were utilized ○ Original design is not represented correctly ○ Redesign is not represented correctly ○ Most bad coding practices present in the code base are not identified ○ Did not make appropriate improvements to bad coding examples ○ Minimal bounded context diagram is not provided ○ Minimal DDD-based domain models are not provided ○ Minimal microservices architecture diagram is not provided ○ Your submission has been correctly submitted in the format requested, including the updated source code, screenshots of the application running on your computer and the written report
0	● The student either did not complete the assignment, plagiarized all or part of the assignment, or completely failed to address the project requirements.