SOEN 6471-Advance Software Architecture

Summer 2023

Moodle **S**oftware **A**rchitecture

Deliverable **2**

Declaration

We, the members of the team, have read and understood the Fairness Protocol and the Communal Work Protocol, and agree to abide by the policies therein, without any exception, under any circumstances whatsoever.

Group C

Prashant **B**anavali
Shubham **J**agdishbhai **B**handeri
Rutvij **B**hatt
Poornaa **H**imadri **B**hattacharya
Khushboo **S**araf

GitHub Address:
SOEN6471_Moodle_TeamC

Table of Contents

4. Quality Attributes and Evaluation of Moodle

Moodle is an open-source learning management system (LMS), designed to offer a variety of features and functionalities that foster effective online learning and education. Its primary objective is to prioritize different quality attributes to deliver a superior learning experience.

4.1. Quality Attributes Aimed to Satisfy

Some of the quality attributes that Moodle strives to achieve are:

4.1.1. **Scalability**: Scalability refers to the system's ability to handle varying workloads by adjusting the system's resources and costs accordingly.[1] Moodle exhibits scalability as a key characteristic, with its ability to accommodate an expanding user base and an increasing number of courses. Its design ensures that the system can effectively handle substantial growth without compromising performance or functionality.

4.1.2. **Usability**: Usability refers to how effectively and efficiently a software product can be used by intended users to accomplish specific goals within a specific context, while also ensuring user satisfaction.[2] By emphasizing an intuitive interface, streamlined course management, content creation tools, role-based access control, customizability, user support, and accessibility, Moodle aims to enhance usability and provide a positive user experience for both administrators and learners.

4.1.3. **Reliability**: Paraphrase: Reliability refers to the degree to which a software product effectively performs pre-established functions under predetermined circumstances and for a specified period.[2] By prioritizing system stability, high availability, data backup, security measures, error handling, monitoring, and community support, Moodle aims to deliver a reliable and dependable learning management system.

4.1.4. **Modifiability**: Modifiability is the extent to which a software product can be modified effectively and efficiently without introducing errors or compromising the overall quality of the existing software.[2] Through its modular architecture, plugin system, theming capabilities, APIs, customization options, and support from the developer community, Moodle offers modifiability, enabling administrators and developers to tailor the system to specific requirements and extend its functionality according to their unique needs.

4.1.5. **Interoperability**: Interoperability refers to the extent to which multiple software products can seamlessly exchange information and effectively utilize the exchanged information.[2] By adhering to interoperability standards, supporting LMS integrations, providing APIs and web services, enabling authentication and enrollment from external systems, supporting content import/export, and following open standards, Moodle facilitates interoperability, allowing for seamless integration with other systems and the exchange of data and resources.

4.1.6. **Accessibility**: Accessibility refers to the extent to which a software product can be utilized by individuals with diverse characteristics and capabilities, enabling them to accomplish specific goals within a defined context of use. [2] Moodle promotes accessibility, by making significant efforts to cater to users with disabilities. It adheres to accessibility standards and incorporates features that accommodate various learning needs. By prioritizing accessibility, Moodle aims to provide an inclusive learning environment where all users, regardless of their abilities, can participate and engage effectively in educational activities.

4.2. Quality Attributes Not Satisfied

While Moodle satisfies these important quality attributes, there are a few that it does not which are emphasized further.

4.2.1. **Responsiveness**: The degree to which the response and processing times and throughput rates of a software product, when performing its functions, meet requirements. [2] The responsiveness of Moodle can also be influenced by factors outside of the Moodle software itself, such as server configuration, network infrastructure, and user device capabilities. Administrators can take steps to optimize the performance and responsiveness of Moodle by ensuring adequate server resources, optimizing database queries, and implementing caching mechanisms.

4.2.2. **Transparency**: Transparency refers to the degree to which the inner workings, processes, and decision-making of a system are visible and understandable to its users. The transparency in certain areas of Moodle may be limited or dependent on the specific implementation and customization. While Moodle provides visibility into its source code and encourages community involvement, the decision-making processes, governance structure, and long-term development roadmap may not be entirely transparent to all users.

4.2.3. **Orthogonality**: Orthogonality refers to the property of system components or features being independent and having minimal interactions or dependencies on each other. Moodle's functionality relies on the integration of various modules, plugins, and components. While these modules are designed to be modular and encapsulated, there can still be interdependencies between them. For example, certain features may require specific plugins or modules to be installed and configured, and removing or modifying one component may affect the functionality of related features.

4.2.4. **Degradability**: Degradability refers to the ability of a software system to gracefully handle and recover from errors, failures, or adverse conditions, while still maintaining essential functionality and performance. It involves the system's ability to degrade its functionality in a controlled manner when faced with resource constraints or unexpected circumstances. Moodle supports integration with various external systems and plugins. However, if an integrated component or plugin experiences a failure or becomes unavailable, Moodle may not have built-in mechanisms to gracefully handle such failures. This can potentially lead to disruptions or errors in Moodle's functionality, impacting the overall user experience.

5. Agile Orthogonal Views of Moodle: A Comprehensive Perspective of Use

Orthogonal views provide different perspectives on a system or software. In the case of "Moodle," which is an open-source learning management system, we can consider three orthogonal views.

5.1. Functional View

The functional view focuses on the functionalities and features of Moodle. It captures the system's behavior and the interactions between its components. This view describes how Moodle supports teaching and learning activities, such as creating courses, managing user accounts, uploading resources, facilitating discussions, and conducting assessments. It emphasizes the functional requirements and capabilities of the system.

Here are some key aspects of the functional view of Moodle:

5.1.1. **User Management**: Moodle allows administrators to manage user accounts, roles, and permissions. It supports functions like user registration, authentication, and profile management. Users can have distinct roles, such as teachers, students, and administrators, each with specific privileges and access levels within the system.

5.1.2. **Course Management**: Moodle enables the creation, organization, and management of courses. Teachers or course creators can design and set up courses, including defining course content, resources, and activities. They can create modules, lessons, quizzes, assignments, and discussion forums, providing a structured learning experience.

5.1.3. **Content Management**: Moodle provides tools for managing and delivering content to learners. It supports the upload and organization of several types of resources, such as documents, presentations, videos, and interactive media. Teachers can create content pages, organize them in folders, and make them accessible to specific groups of learners.

5.1.4. **Communication and Collaboration**: Moodle offers communication and collaboration features to facilitate interaction between users. It includes discussion forums, messaging systems, chat rooms, and notifications. These tools enable learners and teachers to engage in discussions, seek clarifications, collaborate on group projects, and receive updates on course activities.

5.1.5. **Assessment and Evaluation**: Moodle supports various assessment methods and tools. Teachers can create quizzes, tests, and assignments with different question types, grading schemes, and time limits. The system provides mechanisms for automated grading, feedback provision, and gradebook management. It also supports peer assessment and self-assessment activities.

The appropriate viewpoint model for the Functional view of Moodle would be the *Behavioral Viewpoint Model*. This model focuses on capturing the system's functionalities, behavior, and interactions with external entities. The viewpoint model associated with this view could be a use case diagram or an activity diagram or sequence diagram.[3] [Figure 1]

5.2. Information View

The Information View of Moodle focuses on the underlying information and data structures within the system. It aims to provide an understanding of how data is organized, stored, and accessed within Moodle. The Information View encompasses the data entities, attributes, relationships, and their corresponding operations.

In Moodle, the Information View includes various aspects, such as:

5.2.1. **User Data**: This includes information about users within the Moodle system, such as their usernames, passwords, roles, and personal details. User data is crucial for managing user accounts, authentication, and access control.

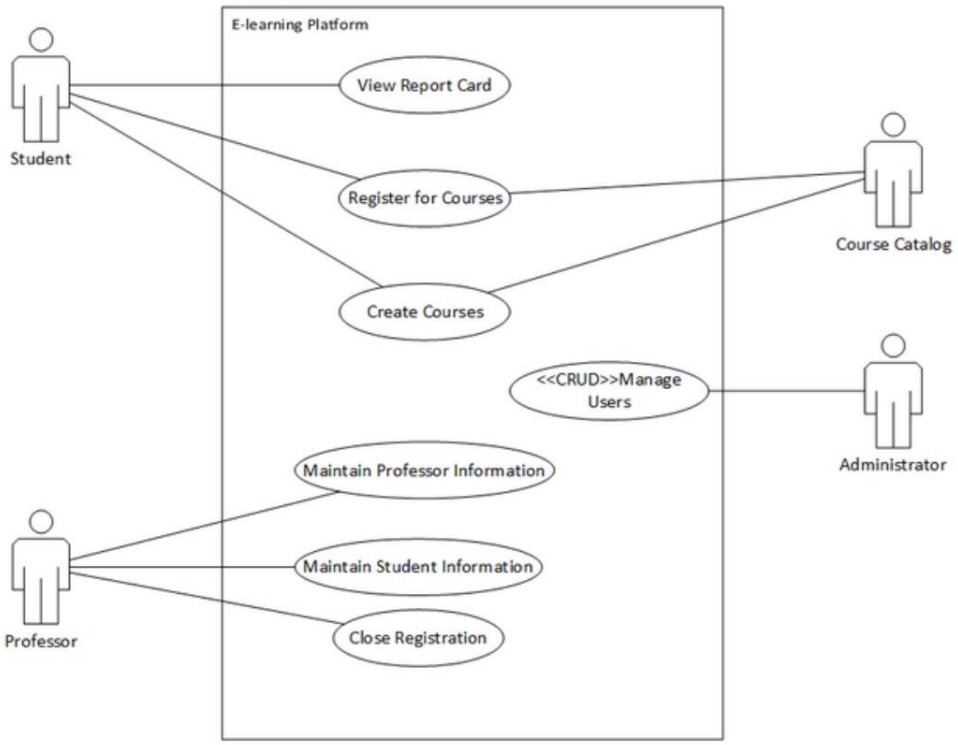

Figure 1: Use Case Diagram Viewpoint of Moodle

5.2.2. **Course Data**: Course data encompasses information about the courses offered in Moodle, including the course name, description, start and end dates, enrollment details, and associated instructors. This data is vital for organizing and managing courses within the system.

5.2.3. **Content Data**: Content data refers to the educational materials, resources, and activities available within courses. This can include files, documents, multimedia content, discussions, quizzes, and assignments. Content data is essential for delivering and interacting with learning materials.

5.2.4. **Assessment Data**: Assessment data relates to the evaluation and grading of students' performance within Moodle. This includes information about quizzes, assignments, tests, and grades. Assessment data is crucial for tracking student progress and generating reports.

5.2.5. **System Configuration Data**: System configuration data includes information about the Moodle installation and its settings. This can involve database configuration, system preferences, security settings, and plugin configurations.

The viewpoint model associated with this view could be *Data Model Style*. The data model style viewpoint of Moodle focuses on the structure, organization, and flow of data within the system. This viewpoint helps stakeholders comprehend how data is managed, stored, and utilized by Moodle. In the case of Moodle, an ERD or class diagram would depict the various entities or classes within the system and their relationships. [3] [Figure 2]

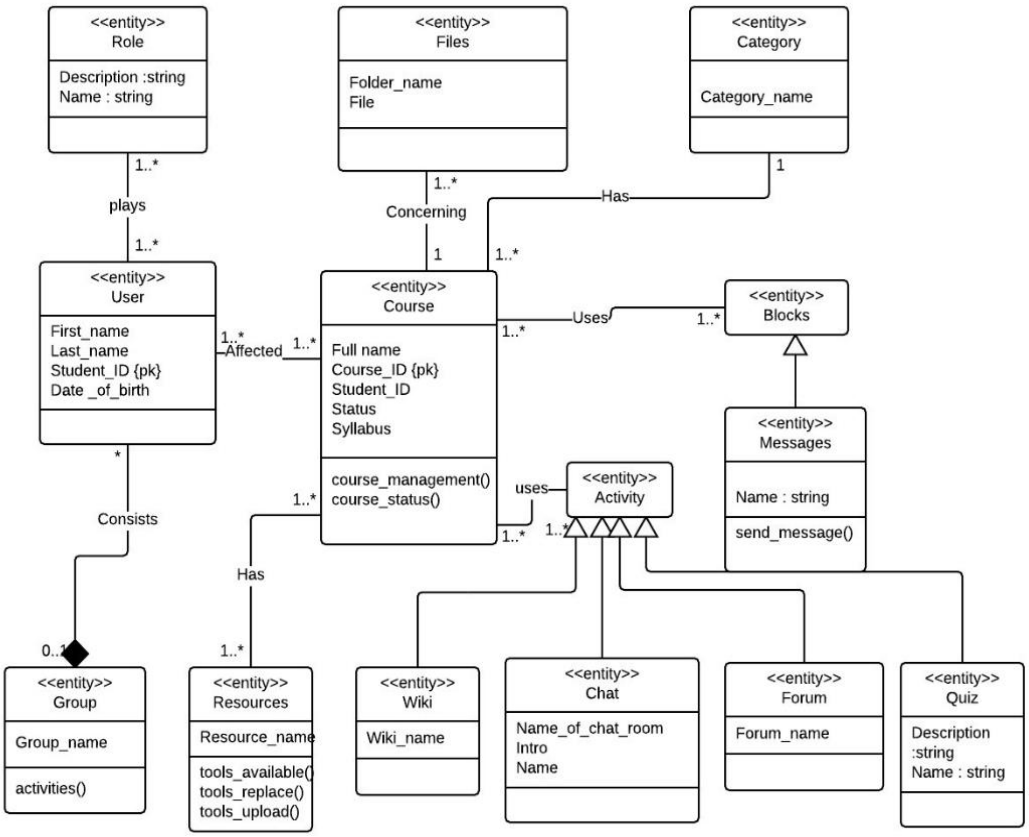

Figure 2: Data Model Style Viewpoint of Moodle

5.3. Deployment View

The Deployment View of Moodle focuses on the physical deployment and distribution of the system's components across different hardware and software environments. It provides an understanding of how Moodle is deployed, hosted, and distributed to ensure its availability and scalability. The Deployment View encompasses the infrastructure, servers, networks, and software configurations necessary for running Moodle.

In the context of Moodle, the Deployment View includes the following aspects:

5.3.1. **Web Server**: Moodle requires a web server to handle incoming requests from users and serve web pages. The web server software, such as Apache or Nginx, hosts and serves the Moodle application.

5.3.2. **Database Server**: Moodle relies on a database server to store and manage data, including user information, course content, and system configurations. Commonly used database systems for Moodle include MySQL, PostgreSQL, or MariaDB.

5.3.3. **File Storage**: Moodle may require a dedicated file storage system for storing course files, multimedia content, and user uploads. This can be implemented using network-attached storage (NAS) or a cloud storage solution.

5.3.4. **Load Balancer**: In large-scale deployments, a load balancer can be employed to distribute incoming user requests across multiple web servers. This ensures high availability, load distribution, and scalability of the Moodle system.

5.3.5. **Network Infrastructure**: The network infrastructure connects the different components of the Moodle deployment. This includes switches, routers, firewalls, and other networking devices that enable communication between users, servers, and databases.

5.3.6. **Backup and Disaster Recovery**: Moodle deployments should incorporate backup and disaster recovery mechanisms to ensure data integrity and system availability. Regular backups and replication strategies help mitigate the risk of data loss or system failure.

The viewpoint model for the Deployment View can be represented using *Deployment Diagrams*. Deployment diagrams in UML depict the physical components and their relationships, showing how software artifacts are deployed on hardware nodes. [Figure 3]

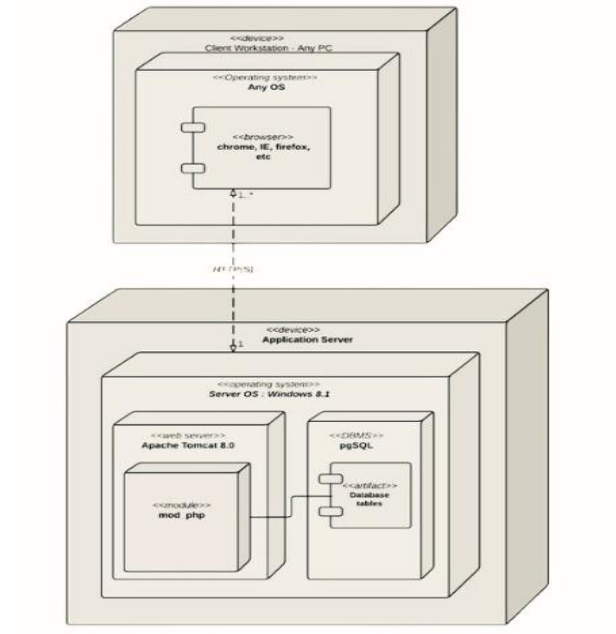

Figure 3: Deployment Viewpoint of Moodle

6. Architectural Patterns in Moodle

6.1. (6A) Exploring the Architectural Patterns in Moodle: Enabling Flexibility and Customization

Moodle follows a modular and extensible architecture, allowing for customization and flexibility. While Moodle does not strictly adhere to a specific architectural pattern, it incorporates various design principles and patterns to achieve its goals.

Here are some notable architectural aspects and patterns [4] used by Moodle:

6.1.1. **Modular Design**: Moodle employs a modular design approach, where different functionalities are organized into discrete modules. This allows for easier development, maintenance, and customization of specific features without impacting the entire system.

6.1.2. **Model-View-Controller (MVC)**: Moodle loosely follows the MVC pattern, which separates the application logic into three components: Model, View, and Controller.

The Model represents the data and business logic, the View handles the presentation and user interface, and the Controller manages the interaction between the Model and View.

6.1.3. **Plug-in Architecture**: Moodle utilizes a plug-in architecture, which allows developers to extend the functionality of the system through the addition of plug-ins. These plug-ins can add new features, activities, themes, and more, without modifying the core codebase. This modular approach enhances the system's scalability and promotes community contributions.

6.1.4. **Service-Oriented Architecture (SOA)**: Moodle adopts a service-oriented architecture, where services are designed as independent components that communicate with each other using standard protocols (e.g., web services). This enables interoperability and allows integration with external systems or tools.

6.1.5. **Event-Driven Architecture**: Moodle incorporates an event-driven architecture, where events are triggered by user actions or system processes. These events can be subscribed to by other components or plug-ins, allowing them to respond and perform specific actions. This decoupled communication mechanism promotes flexibility and extensibility.

6.1.6. **Caching**: Moodle employs caching techniques to improve performance and reduce database load. It caches frequently accessed data or pre-computed results to serve subsequent requests faster. Caching can be implemented at various levels, such as database query caching, object caching, and page caching.

6.1.7. **Distributed Architecture**: Moodle can be deployed in a distributed architecture to handle high traffic and provide scalability. It supports load balancing and clustering techniques to distribute the workload across multiple servers.

6.2. (6A) The Crucial Role of Architectural Knowledge in the Design and Development of Moodle: A Comprehensive Overview

The use of architectural knowledge in Moodle's design and development process is essential for several reasons.

6.2.1. **Scalability**: By adopting modular and extensible architectures, Moodle can handle a growing user base and increasing demands for new features. The use of patterns such as plug-ins and SOA enables the system to scale both vertically and horizontally.

6.2.2. **Flexibility**: The architectural patterns employed in Moodle allow for customization and flexibility. Educators and developers can tailor the system to suit their specific needs by extending existing functionalities or creating new plug-ins. This empowers users to adapt the LMS to their unique requirements.

6.2.3. **Maintainability**: Clear architectural patterns promote code organization and maintainability. The modular design allows for easier code maintenance, bug fixes, and enhancements. It also facilitates collaboration among developers, as they can work on specific modules or plug-ins independently.

6.2.4. **Interoperability**: Moodle's adoption of service-oriented architecture facilitates integration with external systems and tools. This enables seamless data exchange, interoperability with other educational software, and the ability to leverage existing systems within the Moodle ecosystem.

6.2.5. **User-centric Design**: Moodle focuses on user experience and usability. The architecture supports customizable user interfaces, personalization features, and

accessibility standards to ensure a positive and inclusive learning experience for students, teachers, and administrators.

6.2.6. **Security**: Moodle emphasizes security and implements various measures to protect user data and system integrity. This includes secure authentication and authorization mechanisms, encryption protocols, data privacy controls, and adherence to best practices for secure software development.

Overall, the architectural patterns used by Moodle provide a foundation for building a flexible, scalable, and customizable learning management system. They enable educators, developers, and administrators to create and adapt an educational platform that meets their specific requirements while maintaining a robust and well-organized codebase.

6.3. (6B) Identifying Undesirables in Moodle's Software Architecture: Anti-Patterns, Smells, and Their Impact

Here are some common anti-patterns or undesirables that can be found in software architecture [4][5][6], including their potential impacts on Moodle:

6.3.1. **Monolithic Architecture**: Moodle's architecture traditionally follows a monolithic design, where the entire application is bundled into a single codebase. This can lead to challenges in scalability, maintainability, and extensibility. It becomes difficult to modify or add new features without impacting the entire system, and scaling the application horizontally can be problematic.

6.3.2. **Tight Coupling**: Tight coupling refers to a situation where modules or components within the system are highly dependent on each other, making it difficult to modify or replace individual parts without affecting the entire system. In Moodle, tight coupling can lead to difficulties in upgrading or swapping out modules, inhibiting flexibility, and agility.

6.3.3. **Lack of Modularity**: Modularity refers to the division of a system into separate, independent components or modules. A lack of modularity in Moodle's architecture can make it challenging to understand and reason about the system. It can also limit the reusability of components and hinder the ability to easily test and maintain the system.

6.3.4. **Limited Scalability**: Moodle's traditional architecture may face challenges in terms of scalability, particularly with regards to handling high volumes of users or courses. Scaling Moodle horizontally (across multiple servers) can be complex due to the monolithic nature of the system.

6.3.5. **Performance Bottlenecks**: In large-scale Moodle installations, certain architectural decisions, such as inefficient database queries or excessive file I/O operations, can lead to performance bottlenecks. These bottlenecks can result in slow response times, degraded user experience, and reduced system performance overall.

6.3.6. **Lack of API-driven Architecture**: Moodle's architecture historically has not placed a strong emphasis on providing comprehensive APIs (Application Programming Interfaces). This limits the ability for developers to integrate Moodle with external systems or build custom applications on top of Moodle.

6.3.7. **Inefficient Data Storage and Retrieval**: In some cases, Moodle's architecture may lead to inefficient data storage and retrieval mechanisms. This can result in slow database queries, excessive disk I/O operations, or redundant data storage, impacting overall system performance.

6.3.8. **Lack of Separation of Concerns**: Separation of Concerns (SoC) is a fundamental principle in software architecture, aiming to keep various aspects of a system logically separated. In Moodle, there might be instances where concerns such as business logic, presentation, and data access are tightly coupled, making the system harder to understand, modify, and maintain.

6.3.9. **Limited Support for Microservices**: Moodle's traditional architecture does not inherently embrace a microservices-based approach. While there have been efforts to introduce service-oriented architecture (SOA) concepts, the transition to a fully microservices-oriented architecture may require further development and refactoring.

6.3.10. **Limited Event-Driven Architecture**: Moodle's architecture historically has limited support for event-driven architecture, where components communicate through events rather than direct dependencies. This can make it challenging to decouple modules and introduce asynchronous, loosely coupled communication patterns.

6.3.11. **Lack of Comprehensive Testing Infrastructure**: Ensuring the quality and reliability of a complex system like Moodle requires a robust testing infrastructure. However, the architecture may not always provide comprehensive support for automated testing, making it harder to maintain and validate the system's behavior during development and updates.

Code Smell	Description
Monolithic Design	Moodle's architecture may exhibit a monolithic design, with tightly coupled components, making it challenging to scale, modify, and introduce new features.
Lack of Separation of Concerns	In certain areas, Moodle's architecture may lack proper separation of concerns, resulting in tangled dependencies and difficulty in maintaining and understanding the codebase.
Limited Extensibility	The architecture of Moodle may have limited extensibility, making it challenging to integrate third-party systems or add custom functionality without modifying the core codebase.
God Object	Instances where a class or module in Moodle takes on excessive responsibilities, violating the principle of separation of concerns and making code maintenance difficult.
Inconsistent Naming Conventions	In certain areas of Moodle's codebase, there may be inconsistencies in naming conventions, leading to confusion and difficulties in understanding the code.
Code Duplication	Moodle's codebase may contain instances of code duplication, where the same or similar code appears in multiple places, increasing the risk of bugs and making maintenance harder.
Dead Code	Code in Moodle that is no longer executed or serves no purpose, cluttering the codebase and potentially causing confusion.

6.4. (6B) Unveiling the Unwanted: Exploring Moodle's Architecture Undesirables and Anti-Patterns for Valuable Insights

Understanding these undesirable and anti-patterns in Moodle's architecture can be valuable for various purposes.

6.4.1. **Improvement and Optimization**: By identifying and addressing these issues, the Moodle development team can enhance the system's performance, scalability, and maintainability.

6.4.2. **Extension Development**: Developers creating Moodle plugins or customizing the system can be aware of these undesirables to avoid common pitfalls and adhere to best practices.

6.4.3. **Evaluation and Selection**: Educators, administrators, or organizations considering Moodle as a learning management system can assess the impact of these architectural concerns on their specific requirements and make informed decisions.

7. Enhancing the Software Architecture and Description of Moodle

7.1. Evidence-based Recommendations Using the Question Framework for Architectural Description Quality Evaluation

Improving the software architecture and software architecture description of Moodle can contribute to enhanced maintainability, scalability, performance, security, and user experience. Following are some reasoned-based suggestions for improving the architecture and its documentation:

7.1.1. **Modularize the Architecture**: *Issue*: Many components/plugins are interdependent, and callbacks are not concurrent.

- Moodle can continue its efforts to modularize the architecture, further separating concerns and decoupling components. This allows for easier maintenance, extensibility, and scalability. Well-defined modules with clear interfaces promote encapsulation and facilitate independent development and deployment. Modularization simplifies the integration of Moodle with external systems and services. Each module can define well-defined interfaces and APIs, enabling seamless integration with other learning tools, content repositories, authentication systems, or analytics platforms [7][8].

7.1.2. **Enhance API Design and Documentation**: *Issue*: Lack of linear development flow and clear instructions or storyline for modules or APIs.

- Improving the design and documentation of APIs within Moodle can provide clear guidelines for developers and encourage the development of robust integrations and extensions. Well-documented APIs with consistent naming conventions, clear usage examples, and comprehensive documentation contribute to better code readability, maintainability, and developer experience. Proper versioning and compatibility management help ensure smooth upgrades and reduce the impact on Moodle users and developers [9].

7.1.3. **Adopt Microservices Architecture**: *Issue*: While Moodle itself is inherently built as a monolithic architecture, it has large codebases, which can be cumbersome to manage over time.

- Introducing a microservices architecture like other platforms such as A+ and Stratum can provide more flexibility, scalability, and fault isolation in Moodle [10][11]. Decomposing the system into smaller, independently deployable services allows for better management of resources, independent scaling, and the ability to adopt different technologies and deployment strategies for each service.

7.1.4. **Refine User Experience and UI Design**: *Issue*: Moodle has been developed and contributed to by a large community of developers and contributors over time. This

decentralized development process can sometimes result in inconsistencies in UI design across different sections or plugins within Moodle, making it harder for users to develop a consistent mental model of the platform.

- Focusing on consistent user experience and UI design principles can enhance the usability and satisfaction of Moodle users. Conducting user research, usability testing, and incorporating user feedback can help identify pain points and improve the user interface, navigation, and accessibility aspects of the system [12].

7.1.5. **Strengthen Security Measures**: Moodle handles a vast amount of sensitive data, including user profiles, grades, assignments, and communication between students and instructors. Ensuring the security of this data is crucial to protect the privacy and confidentiality of users and comply with data protection regulations. A secure architecture instills trust among users, including students, instructors, administrators, and educational institutions. When users have confidence in the security of the platform, they are more likely to engage in online learning activities and entrust their data to Moodle, which ultimately contributes to its reputation as a reliable and trustworthy learning management system. Continuously monitoring security threats, conducting regular security audits, and staying updated with security patches are crucial to maintaining a secure Moodle platform. Implementing comprehensive security measures, such as secure coding practices, encryption, two-factor authentication, and access control mechanisms, helps protect user data and prevent unauthorized access or breaches [13].

7.1.6. **Document Architectural Decisions**: *Issue*: Moodle is an open-source project that relies on contributions from a diverse community of developers and users. Documentation efforts are often community-driven, and the level of documentation can vary based on individual contributions and available resources. The decentralized nature of community-driven documentation can sometimes result in gaps or inconsistencies.

- Providing comprehensive documentation of architectural decisions helps in understanding the rationale behind design choices, facilitating communication among developers and stakeholders. Documenting architectural decisions, trade-offs, and the underlying reasoning helps future maintainers, supports knowledge sharing, and enables better collaboration within the development community.

7.1.7. **Conduct Performance Optimization**: *Issue*: Feature rich services of organizations or institutions may exhaust servers and hinder UI.

- Many institutes have specific service level agreements (SLAs) or performance expectations for Moodle. Performance optimization ensures that Moodle meets these requirements and provides a reliable and performant learning platform. ‘Identify and address performance bottlenecks within Moodle by conducting performance testing, analyzing resource utilization, and optimizing critical areas such as database queries, caching mechanisms, and resource-intensive operations’ [14]. Utilize caching strategies, implement efficient algorithms, and consider scalability requirements to ensure optimal system performance It is particularly important for activities like page loading, accessing course materials, submitting assignments, and participating in discussions.

7.1.8. **Embrace Continuous Integration and Delivery**: *Issue*: With flexibility of designing individual modules, if one program component must be updated, other elements may also require rewriting, and the whole application must be recompiled and tested.

- Implementing continuous integration and delivery practices in the development process enables faster feedback cycles, early bug detection, and easier deployment. Automating testing, ensuring code quality, and utilizing deployment pipelines contribute to more reliable releases and smoother updates for Moodle. ‘Upgrade API has already been developed and continues to improve’ [15].

A few more suggestions for improving the software architecture and software architecture description of Moodle:

7.1.9. **Implement Event-Driven Architecture (EDA)**: Adopting event-driven architecture can enhance the scalability and extensibility of Moodle [16]. Events can be used to trigger actions, notify components of changes, and facilitate loose coupling between modules. This allows for better integration with external systems, asynchronous processing, and improved flexibility in adding new features.

7.1.10. **Apply Containerization and Orchestration**: Leveraging containerization technologies like Docker and container orchestration platforms like Kubernetes can simplify deployment, improve scalability, and enable better resource utilization and provide isolation.

7.1.11. **Incorporate Machine Learning and Data Analytics**: Integrate machine learning and data analytics capabilities into Moodle to provide personalized learning experiences, predictive analytics, and intelligent recommendations. Leveraging techniques such as natural language processing and data-driven insights can enhance Moodle's effectiveness as a learning platform.

7.1.12. **Enhance Documentation with Visualizations and Emphasize Maintenance**: Augment the software architecture description of Moodle with visualizations such as UML diagrams, system context diagrams, sequence diagrams, or component diagrams. Visual representations aid in understanding the system's structure, interactions, and dependencies, making it easier for developers and stakeholders to grasp the architecture quickly. Regularly review and update the software architecture documentation as the system evolves. Ensure that architectural decisions, changes, and updates are reflected in the documentation to keep it relevant and accurate. Encourage collaboration and contributions from the development community to improve the comprehensiveness and accuracy of the architecture description.

7.1.13. **Foster Community Engagement**: Actively engage with the Moodle community and leverage their expertise to improve the software architecture. Encourage discussions, feedback, and contributions from developers, users, and stakeholders. Building an active and collaborative community around Moodle fosters innovation, identifies architectural issues, and facilitates the sharing of best practices.

7.1.14. **Conduct Regular Architecture Reviews**: Perform periodic architecture reviews to identify potential issues, assess architectural decisions, and ensure alignment with evolving requirements and industry trends. Architecture reviews help identify areas for improvement, validate design choices, and make necessary adjustments to keep the architecture robust and aligned with organizational goals.

Implementing these suggestions can help Moodle evolve into a more adaptable, scalable, secure, and user-centric learning management system. By continuously improving the software architecture and its description, Moodle can stay at the forefront of educational technology and provide an excellent learning experience for its users.

References

[1] M. Erder, P. Pureur, E. Woods, **Continuous Architecture in Practice: Software Architecture in the Age of Agility and DevOps**, *Addison-Wesley*, 2021.

[2] [ISO/IEC, 2011] ISO/IEC 25010:2011. **Systems and Software Engineering -- Systems and Software Quality Requirements and Evaluation (SQUARE) -- System and Software Quality Models**. *The International Organization for Standardization (ISO)/The International Electrotechnical Commission* (IEC). 2011.

[3] R. C. Martin, M. Martin, **Agile Principles, Patterns, and Practices in C#**, *Prentice-Hall*, 2006.

[4] A. Koenig, **Patterns and Antipatterns**, *Journal of Object-Oriented Programming*, Vol 8, No 1 (1995), 46-48.

[5] D. Guo, H. Wu, **A Review of Bad Smells in Cloud-based Applications and Microservices**, *The 2021 International Conference on Intelligent Computing, Automation and Systems (ICICAS 2021)*, Chongqing, China, December 29-31, 2021.

[6] K. Alkharabsheh, **An Empirical Study on the Co-Occurrence of Design Smells in the Same Software Module: God Class Case Study**, *The 2021 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT 2021)*. Amman, Jordan. November 16-18, 2021.

[7] "Components" – docs.moodle.org https://docs.moodle.org/dev/Communication_Between_Components#Components (accessed: June 17, 2023)

[8] "Communication Channels" – docs.moodle.org https://docs.moodle.org/dev/Communication_Between_Components#Communication_Channels (accessed: June 17, 2023)

[9] "API Guides" – moodledev.io https://moodledev.io/docs/apis (accessed: June 17, 2023)

[10] Niemelä, Pia, and Heikki Hyyrö. "**Migrating learning management systems towards microservice architecture.**" (2019). https://trepo.tuni.fi/bitstream/handle/10024/129800/paper2a.pdf?sequence=1&isAllowed=y

[11] Riekkinen, Markku. "**Integrating stratum and a+ functionalities in moodle: Architecture and evaluation.**" *Master's thesis*, 2017. (PDF)

[12] "New Future Development" – moodledev.io https://moodledev.io/general/development/process#new-feature-development (accessed: June 17, 2023)

[13] "Security" – moodledev.io https://moodledev.io/general/development/policies/security (accessed: June 17, 2023)

[14] "Performance recommendations" – docs.moodle.org https://docs.moodle.org/402/en/Performance_recommendations (accessed: June 17, 2023)

[15] "Plugin Upgrades" – moodledev.io https://moodledev.io/docs/guides/upgrade (accessed: June 17, 2023)

[16] Campo, M., Amandi, A. & Biset, J.C. **A software architecture perspective about Moodle flexibility for supporting empirical research of teaching theories**. *Educ Inf Technol 26*, 817–842 (2021). https://doi.org/10.1007/s10639-020-10291-4

Team Member	**Contributions**
Prashant Banavali	• Analyzed and concluded selection of Moodle architecture patterns. • Gathered relevant information on patterns. • Discussed with the team to identify different architectural patterns. • Worked on documenting *Problem 6* in collaboration with Shubham.
Shubham Jagdishbhai Bhanderi	• Worked on *Problem 6* in collaboration with Prashant. • Conducted an in-depth analysis of the software architecture employed by Moodle. • Identified and discussed various patterns, principles, styles, tactics, undesirables, anti-patterns, and smells present in the system.
Rutvij Bhatt	• Explored similarities between architectural patterns of different e-learning platforms and distinguished uniqueness of Moodle. • Referred Moodle Community Forums and developer documentation for existing architectural concerns in Moodle. • Concluded *Problem 7* regarding suggestions for improving Moodle Architecture. • Documented Deliverable 2.
Poornaa Himadri Bhattacharya	• Researched all possible Quality Attributes. • Analyzed which quality attributes were satisfied by Moodle. • Found rationale for quality attributes not supported by Moodle. • Worked on *Problem 4.*
Khushboo Saraf	• Analyzed view and viewpoints of Moodle. • Researched on UML diagrams that will describe the viewpoints. • Worked on *Problem 5*.