

## **MVC-ARENA: A Card-Based Representation for Reasoning About MVC Architecture**

**Md Jamiur Rahman Rifat**

Mathematics & Computer Science  
University of Lethbridge  
mdjamiurrahman.rifat@uleth.ca

**John Anvik**

Mathematics & Computer Science  
University of Lethbridge  
john.anvik@uleth.ca

### **Abstract**

Software engineering students often struggle to understand how application components interact within larger architectural patterns such as Model-View-Controller (MVC). This paper presents MVC-ARENA, a web-based multiplayer card game designed to support students' understanding of MVC architecture through interactive gameplay representations. Players construct software systems by placing cards into Model, View, and Controller regions, where gameplay mechanics externalize architectural responsibilities, component interactions, and defensive software engineering practices. The game uses component, defensive, and attack mechanics to encourage students to reason about separation of concerns, abstraction, maintainability, and system resilience. The paper also describes a pilot study investigating how students interpret these gameplay representations and whether the mechanics support understanding of MVC concepts and relationships between application layers and system responsibilities.

### **1. Introduction**

The Model-View-Controller (MVC) architectural pattern is widely used in software engineering for its support of modularity, maintainability, code reuse, and parallel development (Bucanek, 2009). As MVC and related patterns are commonly used in modern web and enterprise systems, it is important for students to develop a practical understanding of component responsibilities and system interactions (Bass, Clements, & Kazman, 2012). In particular, MVC requires students to reason about separation of concerns, component responsibilities, and interactions between different layers of an application (Seli'c, 2021). Although students may understand isolated programming constructs, understanding how functionality should be distributed across the Model, View, and Controller layers can be difficult. Students often misunderstand how responsibilities should be distributed across MVC layers and may incorrectly couple presentation, control, and data-management responsibilities (Kumar & Choppella, 2024).

Teaching architectural concepts presents several challenges in software engineering education (Ren et al., 2025; Shaw & Garlan, 1996). Architectural patterns are often introduced through lectures or static diagrams, which may not provide students with opportunities to actively explore how software components interact within a larger system. Visual representations can support reasoning about complex relationships by making structural dependencies and interactions more explicit to learners (Larkin & Simon, 1987). Prior work has shown that interactive and visual representations of gameplay can help learners reason about abstract programming concepts and software structures (Bromwich, Masoodian, & Rogers, 2004). Game-based learning (GBL) has previously been used to support the teaching of software engineering and computer science concepts, including programming, debugging, software project management, and cybersecurity (Günther, Kiesling, & Stummer, 2011). Prior work has explored the use of web-based card games for teaching programming and cybersecurity concepts (Anvik, Cote, & Riehl, 2019; Tareque, Deutekom, Anvik, & Bashir, 2024a, 2024b).

MVC-ARENA applies the web-based card-game approach to teaching software design and organization concepts. It is designed to help students reason about architectural responsibilities, software components, security mechanisms, and clean programming practices within MVC systems. MVC-ARENA is intended to externalize relationships between system elements and system responsibilities through gameplay mechanics that allow students to reason about separation of concerns, component responsibilities, and defensive design practices within layered applications. The game models software components and engineering practices using cards that players combine to construct software systems during gameplay.

The goal of this work is to investigate whether gameplay mechanics can help students better understand MVC architecture and the relationships between software components. The gameplay mechanics are designed to support the development of students' understanding of MVC architecture by making component interactions and architectural responsibilities visible during play. This paper presents the design of MVC-ARENA and proposes a pilot study to evaluate the game's effectiveness in supporting students' understanding of MVC concepts and component responsibilities in software systems.

## 2. Related Work

Game-based learning (Connolly, Boyle, MacArthur, Hainey, & Boyle, 2012) has been used to support the teaching of software engineering and computer science concepts, including programming, cybersecurity, testing, and software project management (Günther et al., 2011; Severo & Lelli, 2023; Annunziata, Lambiase, Palomba, & Ferrucci, 2024). Prior work has shown that gameplay representations can help students engage with abstract software concepts through interactive problem-solving activities (Severo & Lelli, 2023; Annunziata et al., 2024).

Prior work explored the use of web-based card games to teach programming and cybersecurity concepts through gameplay mechanics that model programming structures, software vulnerabilities, and defensive practices (Anvik et al., 2019; Tareque et al., 2024a, 2024b).

Although prior work has explored gameplay approaches for teaching programming, cybersecurity, testing, and project management, there appears to be limited work investigating gameplay approaches for teaching architectural patterns such as MVC. MVC-ARENA extends prior work by focusing on how gameplay mechanics can model component responsibilities, system interactions, and clean programming practices within layered software applications.

## 3. Game Design

MVC-ARENA is designed for two players, with each player competing against either another human or a computer-controlled opponent. The objective is to construct a software system that meets architectural requirements while earning enough points to win the game. The scoring system is intended to encourage players to construct balanced architectures that incorporate both functional and defensive software components. By requiring players to manipulate and organize architectural components during gameplay, MVC-ARENA encourages active reasoning about relationships between software structure, maintainability, and system behaviour.

Rather than focusing solely on individual programming constructs, the game emphasizes system organization, separation of concerns, and interactions among components within a larger software system. The cards and gameplay mechanics are designed to model common software engineering decisions and tradeoffs encountered during application development. These gameplay interactions are intended to provide students with concrete external representations (Zhang & Norman, 1994) of software design concepts that are otherwise difficult to visualize during traditional instruction (Storey, 2005). Manipulating these gameplay structures requires players to actively organize architectural responsibilities and explore relationships between application elements during play (Papert, 1980).

### 3.1. Component Cards

The game contains three primary categories of component cards corresponding to the Model (M), View (V), and Controller (C) layers of the MVC architectural pattern. These cards are used to construct the player's software system during gameplay.

The Model cards correspond to data management, business logic, and infrastructure responsibilities within a software system. Components such as Database, Object Relational Mapper (ORM), and Data Validation are intended to help players understand about how application data is managed, protected, and accessed. For example, the ORM card models the abstraction between application objects and relational database tables, while the Data Validation card reinforces defensive input handling practices.

The View cards model the presentation and interaction layer of the software system. Cards such as Web,

Mobile, and Command-Line Interface (CLI) illustrate different user interaction mechanisms. These cards are intended to make explicit that presentation concerns are separate from application logic and data management responsibilities.

The Controller cards focus on coordination, routing, and system control responsibilities within MVC systems. Cards such as Authentication, Authorization, and Rate Limiting model how requests are managed and how access to application resources is controlled. These cards are intended to help players reason about how controllers mediate communication between user interactions and backend functionality.

### 3.2. Defensive Cards

Defensive cards represent software engineering practices associated with abstraction, maintainability, code reuse, and system robustness. These cards increase the score multiplier of a component stack while also protecting the system from negative gameplay events. The mechanics are intended to encourage players to consider how defensive design decisions improve the long-term stability and maintainability of software systems.

Cards such as Interface, Inheritance, and Polymorphism externalize object-oriented design concepts and clean programming practices. These mechanics model how abstraction boundaries, extensibility, and code reuse can improve software organization and reduce system fragility. For example, the Inheritance card allows players to reuse and protect an existing stack, illustrating how functionality is reused across related software components. The stacking mechanics are intended to make relationships between abstraction, reuse, and system protection more visually explicit during gameplay. Organizing cards into protected stacks encourages players to reason about how abstraction and reuse contribute to software maintainability and resilience.

Other defensive cards emphasize operational reliability and resilience within software systems. Logging and Error Handling model practices that support debugging and runtime stability, while the Git card models version control and recovery from catastrophic failures. Together, these mechanics are intended to reinforce the role of defensive engineering practices in creating maintainable and resilient software architectures.

### 3.3. Attack and Destructive Event Cards

Attack cards and destructive event cards represent software vulnerabilities, operational failures, and development risks. These mechanics are intended to help players reason about the relationship between software design decisions and system robustness. By linking vulnerabilities to specific architectural and defensive components, the gameplay encourages players to reason about how design decisions affect software security and resilience. These interactions encourage players to connect software vulnerabilities with specific design and defensive decisions during gameplay.

Attack cards model common software vulnerabilities, such as SQL Injection, Denial-of-Service (DoS), and Malware. These cards adversely affect an opponent's system unless appropriate defensive or architectural mechanisms are in place. For example, the ORM card protects against SQL Injection attacks, while Rate Limiting and Caching reduce the effects of DoS attacks.

Destructive event cards simulate failures that can occur during software development and deployment. Bug and Disaster cards model runtime failures and operational disruptions that may affect a software system, regardless of the overall architecture's quality. These mechanics are intended to reinforce the need for defensive engineering practices and system resilience.

### 3.4. Gameplay

MVC-ARENA is designed for two players, with each player competing against either another human or a computer-controlled opponent. The objective is to construct a software system that meets architectural requirements while earning enough points to win the game.

Players build systems by placing cards in the Model, View, and Controller areas, which correspond to

the major architectural layers of an MVC application. To win, a player must not only reach the required point total but also satisfy minimum architectural requirements for each MVC layer. This mechanic is intended to reinforce the idea that a complete software system requires coordinated responsibilities across multiple architectural components. The spatial separation of these regions is intended to reinforce the separation of concerns between presentation, control, and data-management responsibilities within MVC systems. By requiring players to organize cards spatially across MVC regions, the gameplay makes dependencies and component responsibilities more visible during system construction.

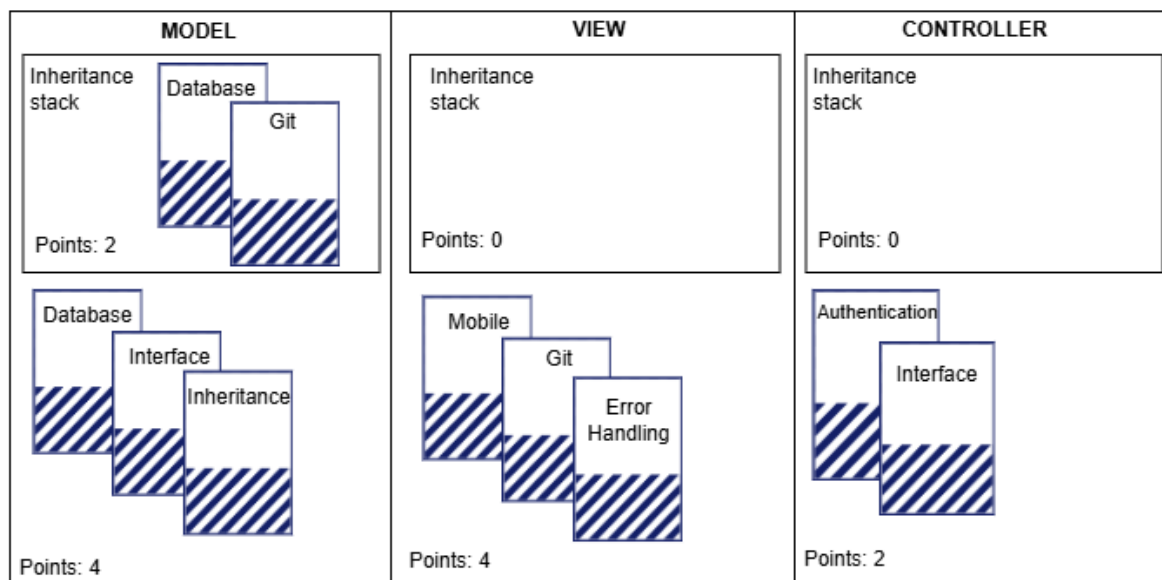


*Figure 1 – The MVC-ARENA play area illustrating the spatial separation of Model, View, and Controller responsibilities and the use of defensive card stacking to reinforce abstraction and system protection.*

Figure 1 illustrates the MVC-ARENA play area. Players construct software systems by placing cards into the Model, View, and Controller regions, corresponding to the major layers of an MVC application. The arrangement of cards externalizes relationships between system elements, component responsibilities, and defensive programming practices. The visual organization of the play area is intended to help players interpret how responsibilities are distributed across MVC layers and how defensive practices support system stability.

The example shown illustrates how defensive and infrastructural components can be combined across MVC layers to support abstraction, maintainability, and system resilience. The use of Interface and Inheritance cards reinforces abstraction and code reuse within the software architecture, while defensive cards such as Git and Error Handling highlight maintainability and system robustness. Attack and event mechanics are designed to encourage players to reason about how design decisions affect software security and resilience.

Players must therefore balance architectural growth, defensive practices, and responses to security threats and failures while building their software system. Attack cards create pressure to consider security and robustness, while defensive cards encourage players to think about maintainability and abstraction. The gameplay is therefore designed to model software development as a process involving architectural tradeoffs, component interactions, and continuous system maintenance.

#### 4. Proposed Pilot Study

The proposed pilot study will investigate how effectively MVC-ARENA supports students' understanding of MVC architecture concepts and the relationships among software components. In particular,

the study will focus on whether the gameplay representations help students reason about architectural responsibilities, component interactions, defensive design practices, and separation of concerns within MVC systems. In particular, the study will investigate how players interpret the spatial organization and stacking mechanics used throughout the game.

The participants in the study will be second- or third-year university students who have completed introductory Computer Science courses. Participants may have varying levels of familiarity with MVC and related software engineering concepts. The game is intended to support students' understanding of component organization, component responsibilities, and software engineering practices that are often difficult to visualize through traditional lectures or diagrams.

The evaluation will use pre- and post-game questionnaires to investigate how students interpret the gameplay mechanics and how effectively the game conveys relationships between application elements and architectural responsibilities. The questionnaires will include both quantitative and qualitative questions related to MVC concepts, component interactions, maintainability, abstraction, and security practices.

Quantitative analysis will be used to measure changes in students' understanding of MVC concepts before and after gameplay. Qualitative feedback will be used to explore how players interpret the gameplay representations and whether the mechanics successfully externalize architectural concepts and software engineering tradeoffs.

## 5. Conclusion

This paper presented MVC-ARENA, a web-based multiplayer card game designed to support students' understanding of MVC concepts through gameplay mechanics and interactions among software components. The game uses component, defensive, and attack mechanics to externalize architectural relationships and software engineering practices that are often difficult to visualize through traditional instruction. The proposed pilot study will investigate how students interpret these gameplay representations and whether the mechanics support understanding of architectural responsibilities and component interactions within MVC systems.

## 6. Acknowledgements

This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) through the Discovery Grants program.

Generative AI tools were used to assist with language refinement and editing during the preparation of this paper.

## 7. References

- Annunziata, G., Lambiase, S., Palomba, F., & Ferrucci, F. (2024). SERGE – Serious Game for the Education of Risk Management in Software Project Management. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training* (pp. 264–273). New York, NY, USA: Association for Computing Machinery.
- Anvik, J., Cote, V., & Riehl, J. (2019). Program Wars: A Card Game for Learning Programming and Cybersecurity Concepts. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 393–399). New York, NY, USA: Association for Computing Machinery.
- Bass, L., Clements, P., & Kazman, R. (2012). *Software architecture in practice* (3rd ed.). Addison-Wesley Professional.
- Bromwich, M., Masoodian, M., & Rogers, B. (2004). Visprocon: A visual programming environment for learning programming concepts. In *Proceedings of the sixth australasian conference on computing education* (pp. 239–243). Australian Computer Society, Inc.
- Bucanek, J. (2009). Model-view-controller pattern. In *Learn objective-c for java developers* (pp. 353–402). Berkeley, CA: Apress.
- Connolly, T. M., Boyle, E. A., MacArthur, E., Hainey, T., & Boyle, J. M. (2012, September). A systematic literature review of empirical evidence on computer games and serious games. *Comput.*

- Educ.*, 59(2), 661–686. Retrieved from <https://doi.org/10.1016/j.compedu.2012.03.004> doi: 10.1016/j.compedu.2012.03.004
- Günther, M., Kiesling, E., & Stummer, C. (2011). Game-based Learning in Technology Management Education: A Novel Business Simulation. *International Journal of Emerging Technologies in Learning (iJET)*, 6(1), 20–25.
- Kumar, M., & Choppella, V. (2024). Enhancing mvc architecture pattern description using its system of systems model. In *Proceedings of the 17th innovations in software engineering conference*. New York, NY, USA: Association for Computing Machinery. Retrieved from <https://doi.org/10.1145/3641399.3641410> doi: 10.1145/3641399.3641410
- Larkin, J. H., & Simon, H. A. (1987). Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, 11(1), 65-100. Retrieved from <https://www.sciencedirect.com/science/article/pii/S0364021387800265> doi: [https://doi.org/10.1016/S0364-0213\(87\)80026-5](https://doi.org/10.1016/S0364-0213(87)80026-5)
- Papert, S. (1980). *Mindstorms: children, computers, and powerful ideas*. USA: Basic Books, Inc.
- Ren, Z., Lhatshok, U., Gyamtsho, P., Dema, T., Rabgay, T., & Awasthy, R. (2025). Strategies and struggles in software architecture teaching: A systematic literature review. In *2025 international conference on education technology and computers (icetc)* (p. 760-766). doi: 10.1109/ICETC66579.2025.11387435
- Seli’c, B. (2021). On the precise semantics of the software layering design pattern. *Journal of Object Technology*, 20(2), 1–?? Retrieved from <http://dx.doi.org/10.5381/jot.2021.20.2.a6> doi: 10.5381/jot.2021.20.2.a6
- Severo, J., & Lelli, V. (2023). Testing Maze: an educational game for teaching functional testing. In *Proceedings of the XXXVII Brazilian Symposium on Software Engineering* (pp. 407–415). New York, NY, USA: Association for Computing Machinery.
- Shaw, M., & Garlan, D. (1996). *Software architecture: perspectives on an emerging discipline*. USA: Prentice-Hall, Inc.
- Storey, M.-A. (2005). Theories, methods and tools in program comprehension: past, present and future. In *13th international workshop on program comprehension (iwpc’05)* (p. 181-191). doi: 10.1109/WPC.2005.38
- Tareque, M. H., Deutekom, S., Anvik, J., & Bashir, M. (2024a). Program Wars v.2.0 : Improving a Game-based Learning Approach for Teaching Fundamental Programming Concepts. In *Proceedings of the 26th Western Canadian Conference on Computing Education* (pp. 1–7). New York, NY, USA: Association for Computing Machinery.
- Tareque, M. H., Deutekom, S., Anvik, J., & Bashir, M. (2024b). You Hacked My Program! Teaching Cybersecurity using Game-based Learning. In *Proceedings of the 26th Western Canadian Conference on Computing Education* (pp. 1–7). New York, NY, USA: Association for Computing Machinery.
- Zhang, J., & Norman, D. A. (1994). Representations in distributed cognitive tasks. *Cognitive Science*, 18(1), 87-122. Retrieved from <https://www.sciencedirect.com/science/article/pii/S0364021394900213> doi: [https://doi.org/10.1016/0364-0213\(94\)90021-3](https://doi.org/10.1016/0364-0213(94)90021-3)