

Learning Software Architectural Patterns and Quality Attributes Through a Digital Educational Game

Enzo Rigazzo¹, Fernando Yang¹, Alessandra de Oliveira²,
Renan Vinicius Aranha¹, Pedro Henrique Valle¹

¹Universidade de São Paulo - USP – São Paulo, SP – Brazil

²Universidade Federal de Juiz de Fora - UFJF – Juiz de Fora, MG – Brazil

enzorigazzo@usp.br, fernandoyang@ime.usp.br
alessandra.oliveira@ufjf.br, renanvinicius@usp.br
pedrohenriquevalle@usp.br

Abstract. Introduction: Teaching software architectural trade-offs is challenging because students must go beyond memorizing Architectural Patterns and quality attributes to reason about their relationships in concrete scenarios. **Objective:** This work presents *TotalMeltdown*, a Digital Educational Game (DEG) designed to support undergraduate computing students in understanding how architectural decisions influence software quality attributes. **Methodology:** The game was designed based on commonly discussed trade-offs between Architectural Patterns and software quality attributes. Its educational design draws on Problem-Based Learning by presenting players with contextualized software scenarios that require decision-making. **Results:** The resulting game provides an interactive environment that operationalizes architectural trade-offs as software project decisions. It also integrates an in-game notebook and immediate feedback to support learning. **Conclusion:** The proposed game transforms abstract software architecture concepts into contextualized decision-making activities. Future work includes heuristic inspection, motivation, and engagement instruments such as IMI and IMMS, and player experience and learning assessment using MEEGA+. **Keywords** DEG, Quality Attributes, Architectural Software Patterns

1. Introduction

Software architecture education involves abstract and context-dependent concepts. Students must understand not only the definitions of Architectural Patterns and software quality attributes, but also how design decisions may improve some attributes while negatively affecting others. Such relationships are commonly referred to as architectural trade-offs and are central to architectural decision-making [Farshidi et al. 2020, Lago et al. 2019].

Traditional teaching approaches may present these concepts through lectures, examples, and theoretical exercises. Although these approaches are important, they may limit students' opportunities to experiment with decisions and observe their consequences in a contextualized environment. Digital Educational Games (DEGs) can complement these approaches by promoting active participation, immediate interaction, and engagement during learning activities [Li et al. 2024, Sailer e Homner 2019].

This work presents *TotalMeltdown*, a DEG designed to support the learning of architectural trade-offs in Software Engineering. The target audience consists of undergraduate students in computing-related programs who already possess basic

knowledge of software development and quality attributes. The game addresses Architectural Patterns and quality attributes, as well as other skills, since players must interpret requirements, compare alternatives, and justify architectural decisions.

The main objective of this work is to describe the game design and its technological implementation. The proposed approach combines architectural decision-making with management mechanics, allowing players to select patterns and development teams while balancing project requirements. The work contributes a game-based learning environment focused on autonomous exploration of trade-offs and reports the main issues identified during its initial heuristic evaluation.

2. Related Work

Several games have been proposed to support Software Architecture education. DecidArch focuses on architectural decision-making and explicitly addresses trade-offs among quality attributes [Lago et al. 2019, de Boer et al. 2019]. However, as a physical multiplayer game, it depends on scoring mechanisms that may be difficult for students to interpret without prior knowledge.

ATAM-RPG uses role-playing and simulation to teach concepts related to the Architecture Tradeoff Analysis Method [Montenegro et al. 2017]. Its scenario-based approach promotes realism and collaborative discussion, but it is mainly intended to reinforce previously acquired knowledge. Similarly, Smart Decisions employs the Attribute-Driven Design method to teach architectural design through a catalog of solutions [Cervantes et al. 2016]. Despite its structured learning process, the game requires facilitator support and may pose difficulties with scoring interpretation.

Other initiatives include the LEARN Board Game [Sousa e Marques 2020] and its digital counterpart, D-LEARN [Caminha e Marques 2024]. These approaches introduce Architectural Patterns and quality attributes through playful activities; however, their progression is mainly based on theoretical questions. In contrast, *TotalMeltdown* focuses on decision-making through project contracts. Instead of advancing primarily by answering questions, players must analyze requirements and combine architectural patterns with developer characteristics to achieve quality goals. The game also provides an in-game notebook that offers contextual access to theoretical information during gameplay. This design aims to reduce reliance on instructor supervision while enabling students to connect theoretical knowledge with practical decisions.

3. Game Design

This section describes the design of *TotalMeltdown*, a DEG developed to support the learning of trade-offs in software architectural decisions. The development process was organized into two complementary activities: conceptual grounding and gameplay design. First, an exploratory study was conducted to identify commonly discussed relationships and trade-offs between architectural patterns and software quality attributes. This conceptual basis was used to define the attributes represented in the game and the effects associated with each Architectural Pattern.

These identified concepts were incorporated into two main game elements. First, they were represented through the contract resolution system, in which players must select an architectural pattern and development teams according to project requirements. Second, they were organized into an in-game notebook, which provides background explanations of quality attributes, architectural patterns, and their associated trade-offs.

The game was developed for Web platforms using the Godot Engine¹, and the main game architecture is shown in Figure 1. Its structure separates the user interface, game logic, pedagogical evaluation, feedback system, and a JSON-based repository² for storing and representing contracts, developers, patterns, and notebook pages. This organization supports future expansion of game content without requiring substantial modifications to the core mechanics.

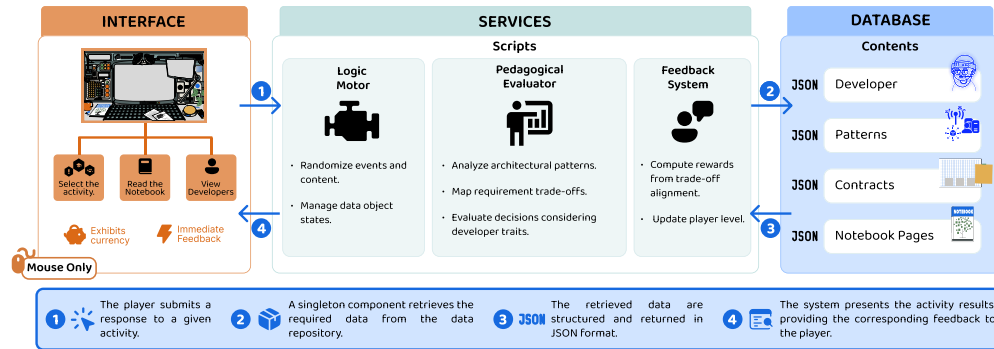


Figura 1. Overview of the *TotalMeltdown* architecture.

The game design was guided by the objective of transforming architectural trade-offs into decision-making situations. Instead of presenting concepts solely through direct explanations or theoretical questions, *TotalMeltdown* places players in project management scenarios where they must interpret requirements, compare alternatives, and face the consequences of their decisions. The main in-game activity of solving contracts was inspired by a non-educational game called Dispatch³.

4. The *TotalMeltdown* Game

This section presents *TotalMeltdown*, including its gameplay mechanics, interface, and educational principles for learning software architectural trade-offs.

4.1. Overview

In *TotalMeltdown*, the player assumes the role of a software production manager responsible for completing software project contracts. Each contract presents a fictional software project scenario with functional and quality-related requirements, requiring the player to identify the quality attributes to prioritize. The central learning objective is to illustrate that Architectural Patterns do not represent universally optimal solutions. Instead, each pattern has benefits and limitations that may affect software quality attributes differently, depending on the project's context. Therefore, players are encouraged to reason about the solution of the contract.

4.2. Core Gameplay Mechanic

The gameplay involves three main activities: contract resolution, developer recruitment, and learning Architectural Patterns. During contracts, players select an Architectural

¹More information available in: <https://godotengine.org/>

²JavaScript Object Notation (JSON) is a lightweight text-based format used to store and represent structured data (key/value).

³Developed by AdHoc Studio: <https://www.adhocla.com/>

Pattern and assemble a development team. Different developers represent distinct technical profiles, encouraging players to balance team composition with architectural choices when solving contracts. Developers contribute different attribute values, while patterns modify them based on their inherent trade-offs. The final project score is determined by combining developers' attributes with the modifiers introduced by the selected Architectural Pattern. Each contract defines target quality attributes, and success depends on how well the resulting configuration satisfies these requirements.

For example, Figure 2 illustrates the contract interface, in this particular case describing a system that requires high levels of security and reliability. Although the scenario does not impose strict architectural constraints, the player may select a pattern that favors these attributes, such as a Layered Architecture. The selected developers' security and reliability values are combined, and the resulting values are multiplied by the chosen pattern. These final scores are then compared with the minimum values required by the contract. The closer the selected configuration is to meeting or exceeding the required attributes, the greater the player's reward.

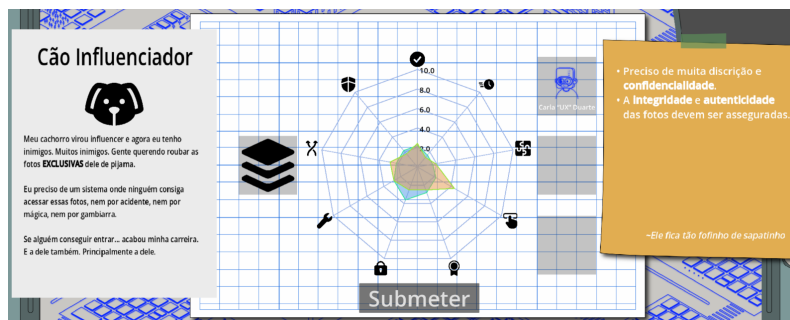


Figura 2. Contract details window.

Before submitting a solution, players may inspect developer profiles and consult an in-game notebook that explains quality attributes and Architectural Patterns. The system then evaluates the selected configuration against the contract requirements and provides feedback on the outcome. Successful contracts reward players with more in-game currency, which can be used to recruit additional developers and learn new patterns.

4.3. Game Interface

The game's main interface is organized as an interaction hub that contains the game's principal activities and mimics a programmer's futuristic workstation. As illustrated in Figure 3, players can access contracts, inspect their available developers, and consult the in-game notebook from the same environment. The notebook serves as an in-game reference, allowing players to review Architectural Patterns, quality attributes, and their associated trade-offs at any moment during gameplay. It functions as a mini-wiki with clickable table of contents allowing players to navigate and revisit concepts. The text provides formal definitions of quality attributes and their subattributes, as well as the Architectural Patterns unlocked by the player. For each pattern, it presents its main trade-offs and a radar chart summarizing its attribute multipliers. The interface was designed to make the main sources of information available during gameplay, ensuring the player can make informed decisions while maintaining the flow of interaction.



Figura 3. Main interaction hub of *TotalMeltdown*. Players access contracts (1), inspect developers (2), and consult the in-game notebook (3).

4.4. Educational Principles

The learning design is based on four main principles. First, contracts use problem-based learning by presenting situations that require analysis rather than direct recall. Second, players engage in active decision-making because every choice influences the project outcome. Third, the system provides feedback after each contract, encouraging reflection on the consequences of decisions. Finally, the notebook provides contextualized support by allowing players to review theoretical concepts as needed.

5. Final Considerations

This work presents *TotalMeltdown*, a Digital Educational Game designed to support learning about software architectural trade-offs. The game combines project analysis, architectural pattern selection, team composition, and contextualized documentation to encourage students to reason about the consequences of software design decisions.

The main contribution of the proposed game lies in combining Architectural Pattern selection with team composition and quality attribute trade-offs. Rather than treating patterns as isolated solutions, *TotalMeltdown* represents them as context-dependent choices whose effects vary according to project requirements and the capabilities of available developers. The in-game notebook and feedback mechanisms further support this process by allowing players to consult theoretical concepts and reflect on the results of their decisions.

As this work focuses on presenting the game concept and its design, future work will validate the game across different dimensions. First, a heuristic evaluation using AHJED [Azevêdo et al. 2018] will be conducted to identify usability, gameplay, and pedagogical issues. Second, the Intrinsic Motivation Inventory (IMI) and the Instructional Materials Motivation Survey (IMMS) will be used to investigate players' motivation and engagement. Finally, MEEGA+ will be applied with the target audience to investigate the perceived educational impact and the player experience of the game.

Use of Artificial Intelligence

Generative Artificial Intelligence tools were used only for language review and minor editorial support. ChatGPT and DeepSeek were used for text refinement, and Grammarly for minor proofreading. All methodological decisions, analyses, and conclusions remain the sole responsibility of the authors.

Referências

- Azevêdo, M., Rousy, D., e Siebra, C. (2018). Ahjed - avaliação heurística para jogos educacionais digitais. In Sánchez, J., editor, *Nuevas Ideas en Informática Educativa*, volume 14, pages 126–136, Santiago de Chile. Universidad de Chile.
- Caminha, L. e Marques, A. (2024). D-learn: A digital game for software architecture education. In *Anais do XX Simpósio Brasileiro de Sistemas de Informação*, Porto Alegre, RS, Brasil. SBC.
- Cervantes, H., Haziyeve, S., Hrytsay, O., e Kazman, R. (2016). Smart decisions: An architectural design game. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, pages 327–335.
- de Boer, R. C., Lago, P., Verdecchia, R., e Kruchten, P. (2019). Decidarch v2: An improved game to teach architecture design decision making. In *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, page 153–157. IEEE.
- Farshidi, S., Jansen, S., e van der Werf, J. M. (2020). Capturing software architecture knowledge for pattern-driven design. *Journal of Systems and Software*, 169:110714.
- Lago, P., Cai, J. F., de Boer, R. C., Kruchten, P., e Verdecchia, R. (2019). Decidarch: Playing cards as software architects. In *Proceedings of the 52nd Hawaii International Conference on System Sciences*, HICSS. Hawaii International Conference on System Sciences.
- Li, Y., Chen, D., e Deng, X. (2024). The impact of digital educational games on student's motivation for learning: The mediating effect of learning engagement and the moderating effect of the digital environment. *PLOS ONE*, 19(1):e0294350.
- Montenegro, C. H., Astudillo, H., e Gómez Álvarez, M. C. (2017). Atam-rpg: A role-playing game to teach architecture trade-off analysis method (atam). In *2017 XLIII Latin American Computer Conference (CLEI)*, pages 1–9.
- Sailer, M. e Homner, L. (2019). The gamification of learning: a meta-analysis. *Educational Psychology Review*, 32(1):77–112.
- Sousa, T. A. S. e Marques, A. B. S. (2020). Learn board game: A game for teaching software architecture created through design science research. In *Proceedings of the XXXIV Brazilian Symposium on Software Engineering, SBES '20*, page 834–843, New York, NY, USA. Association for Computing Machinery.