Evaluating the Reuse Facilitation of a White-Label Architecture for Educational Quiz Games: A Developer-Centered Study

João Pedro Barros de Moura
Instituto Federal do Ceará (IFCE), Campus Maracanaú
Maracanaú, CE, Brazil
joao.moura04@aluno.ifce.edu.br

Paulo Afonso Pamplona de Carvalho Junior
Instituto Federal do Ceará (IFCE), Campus Maracanaú
Maracanaú, CE, Brazil
junior.pamplona07@aluno.ifce.edu.br

Francisco Yuri Carvalho de Oliveira
Instituto Federal do Ceará (IFCE), Campus Maracanaú
Maracanaú, CE, Brazil
francisco.yuri.carvalho08@aluno.ifce.edu.br

Bruno Correia da Silva
Instituto Federal do Ceará (IFCE), Campus Maracanaú
Maracanaú, CE, Brazil
bruno.silva@ifce.edu.br

Carlos Henrique Leitão Cavalcante
Instituto Federal do Ceará (IFCE), Campus Maracanaú
Maracanaú, CE, Brazil
henriqueleitao@ifce.edu.br

Abstract

Game engines and software development frameworks are widely available. Despite these advances, the development of digital educational games (DEG) still involves high costs and significant engineering effort, often requiring repeated implementation of similar functionalities across projects. This paper presents an evaluation of a proposed white-label architecture for quiz-based DEG. Two games were implemented using the proposed architecture. The first, *Grana*, was developed to validate the proposed architecture. The second, *Investe*, was implemented to evaluate the architecture's reusability in a different educational domain. The results indicate that approximately 89% of the Dart codebase was reused without structural modification, with adaptations concentrated in the data layer and the presentation layer. Furthermore, a preliminary user study assessed both usability and learning outcomes. Usability was evaluated using an adapted version of the System Usability Scale (SUS), achieving a score of 77.5. Additionally, the study describes a statistically significant improvement in investment knowledge after interacting with Investe, as confirmed by the Wilcoxon signed-rank test ($W = 4.0$, $p = 0.0209$). These findings indicate that the proposed architecture facilitates software reuse and contributes to reducing development effort, indicating a viable foundation for the efficient development of quiz-based digital educational games across different educational domains.

Keywords

Educational Digital Games; White-Label Architecture; Software Reuse

1 Introduction

The development of DEG is a complex and resource-intensive activity, typically requiring multidisciplinary teams involving software engineering, instructional design, and visual art [16, 22]. A recurring challenge is that DEG projects often need to reimplement functionally similar components such as quiz logic, level progression, reward systems, and user authentication from scratch for each new educational domain, even when the underlying software structure and gameplay mechanisms are largely unchanged.

White-label software architectures address this problem by encapsulating reusable core components and decoupling them from domain-specific content, enabling the generation of multiple applications from a single shared codebase [23, 24]. In the DEG context, the quiz engine, state management, and backend infrastructure can be preserved across projects while questions, assets, and configurations are adapted per domain.

This distinction is significant from a software engineering perspective. Architectural reusability is a design goal, but reuse facilitation, defined as the extent to which an architecture concretely reduces engineering effort for a new developer, is an empirical question that must be answered through evaluation [11]. Module boundary clarity, documentation quality, configuration complexity, and learning curve all influence whether a reusable architecture delivers on its promise in practice.

This paper addresses this gap by evaluating a proposed white-label architecture for quiz-based digital educational games. Two of them (*Grana and Investe*) were implemented using the proposed architecture: *Grana* to validate it, and Investe as a second instantiation, extending and adapting its components to support a different educational domain. The evaluation combines: (i) quantitative code reuse metrics obtained through static analysis; (ii) developer experience assessment using an adapted version of the System Usability Scale (SUS); (iii) open-ended qualitative feedback; and (iv) a preliminary user study on the application's educational effectiveness.

The main contributions of this work are:

(1) A white-label architecture for quiz-based educational digital games that enables the reuse of software components, game mechanics, and common functionalities across different applications, evaluated through the restructuring of the previously validated educational game (*Grana*).
(2) The development of a new educational game (*Investe*) as a second instantiation of the architecture, demonstrating its adaptability to a different educational domain.

(3) An empirical evaluation of the proposed architecture combining quantitative software reuse metrics, developer experience assessment, and preliminary educational effectiveness analysis.

This paper is organized as follows. Section 2 presents the background and related works. Section 3 describes the proposed architecture and the *Investe* game for financial literacy. Section 4 describes the evaluation methodology. Section 5 presents the results and discussion. Section 6 discusses threats to validity. Finally, Section 7 concludes the paper.

2 Background and Related Work

2.1 White-Label Architectures in Software Engineering

White-label architectures, also known as platform-based development strategies, separate domain-independent components, referred to as the platform core, from domain-specific components, referred to as the application skin, enabling rebranding and functional adaptation without structural code changes [23, 24]. The term originates from the commercial practice of producing a single product that different companies can brand as their own [24]. In software engineering, this translates into a shared codebase whose behavior, content, and visual identity are decoupled from its implementation, typically through configuration files, remote parameters, or asset replacement mechanisms.

The advantages of this approach have been documented in both industry and academic contexts. Umathay and Sinha [24] highlight that white-label strategies reduce development costs, shorten time-to-market, and support scalability by enabling multiple product variants from a single engineering investment. Sharma [23] further demonstrates that the applicability of design systems to white-label service development reduces the effort required for per-client customization while maintaining brand consistency. Heynen [14] validated this approach at scale in BMW's automotive infotainment systems, showing that a white-label digital platform successfully preserved core functionalities while adapting to a target domain. However, that study addressed a commercial non-educational context and conducted no developer-centered evaluation of the reuse process itself.

In the educational software domain, Software Product Lines (SPLs) represent a related but more heavyweight approach: they apply formal variability modeling to derive multiple application variants from shared infrastructure [8, 18]. While SPLs offer strong guarantees of systematic reuse, they require significant upfront investment in feature modeling and configuration management, which may be prohibitive for small teams developing mobile DEG. White-label architectures occupy a pragmatic middle ground: they rely on parameterization and asset substitution rather than formal variability models, making them more accessible to small development teams while still enabling meaningful reuse. The architecture evaluated in this work uses Firebase Remote Config as the primary customization mechanism to decouple all domain-specific content from the application core.

2.2 Software Reuse Assessment

Frakes and Kang [11] distinguish between reusability (a design property) and reuse (the actual act of using a component in a new context). Empirical evaluation of reuse facilitation requires quantitative metrics such as the percentage of reused lines of code (LOC) and the number of unmodified modules, alongside effort and experience measures [15]. Developer experience is commonly assessed via the System Usability Scale (SUS) [6], which has been adapted for API and architectural usability studies [17].

2.3 Educational Digital Games and Reusable Architectures

Battistella and Wangenheim [4] proposed a framework for quiz-based games in computing education emphasizing component reuse. Petri et al. [16] reviewed DEG development methods and noted the lack of systematic reuse strategies as a recurrent limitation. In the mobile space, Ruiz-Carhuamaca et al. [20] presented a gamified app for financial literacy; however, it was developed as a standalone system with no provision for reuse or domain adaptation.

2.4 The System Usability Scale and Its Adaptations

The System Usability Scale (SUS) is a widely used instrument for assessing perceived usability, consisting of ten Likert-scale items (scored 1–5) that alternately express positive and negative sentiments about a system [6]. The SUS score is computed by normalizing the sum of item contributions to a 0–100 scale, with scores above 68 considered above average and scores above 80.3 rated as "Excellent" [3]. Despite its simplicity, SUS has shown strong validity and reliability across a wide range of evaluation contexts [6].

A recognized strength of the instrument is its adaptability. Piccioni et al. [17] demonstrated that SUS items can be rephrased to target API usability, replacing end-user interaction language with developer workflow language while preserving the psychometric properties of the scale. In the educational games domain, Corrêa et al. [9] proposed a pilot adaptation of the SUS questionnaire for evaluating the usability of digital games with children and adolescents aged 10–14, replacing system references with game-specific terminology (e.g., substituting "using the system" with "playing the game") and simplifying language for non-expert respondents. That study demonstrated that contextual rephrasing preserves the instrument's capacity to discriminate usability levels while making it more accessible to the target population.

Building on this precedent, the present study adapts SUS to evaluate architectural usability from a developer's perspective. Rather than assessing how easy a system is for an end user to interact with, the adapted instrument assesses how easy and productive an architecture is for a developer to work with and adapt to a new domain. Items referencing "the system" and "using" are rephrased to reference "the architecture" and "working with" or "adapting," respectively, while the original 10-item structure, 1–5 Likert scale, and scoring formula are preserved unchanged.

3 White-Label Architecture

The architecture evaluated is a technical contribution to reduce the engineering effort required to develop quiz-based DEG for mobile devices. This section provides a sufficiently detailed description to contextualize the evaluation conducted in Sections 4 and 5.

3.1 Structural Overview

The architecture is organized into nine independent modules: `Core`, `Consent`, `About`, `InternetVerification`, `Levels`, `Menus`, `Tutorial`, `Shop`, and `Login`. Each module corresponds to a distinct functional area of the application and follows a layered internal structure comprising a UI layer and, where applicable, BLoC and Data layers: (i) the UI layer comprises pages and visual widgets responsible for user interaction; (ii) the BLoC layer contains Business Logic Components that manage state transitions through events and states; and (iii) the Data layer holds models and repositories responsible for data access and remote configuration retrieval. Not all modules include every layer; simpler modules (e.g., About and Menus) consist only of a UI layer, while modules without local persistence rely on the shared data models provided by the Core module. Figure 1 summarizes the layer composition of each module.

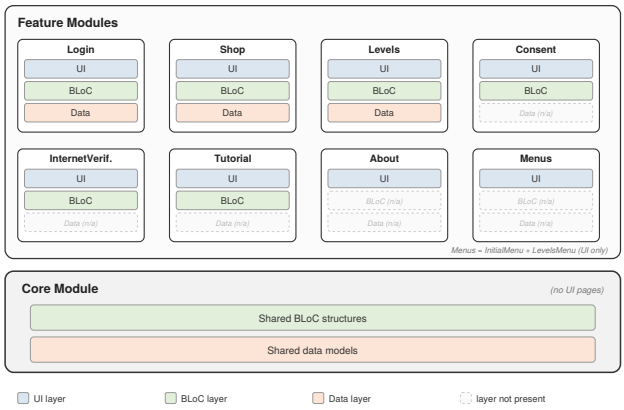

Figure 1: Module organization of the white-label architecture and dependencies among application modules.

The `Core` module occupies a special role: rather than containing UI pages, it centralizes the shared BLoC structures and data models consumed by all other modules, acting as the dependency hub of the architecture. As shown in Figure 2, all feature modules depend on the Core module. The `Menus` module is subdivided into two submodules, `InitialMenu` and `LevelsMenu`, each containing a single screen. This design enforces a strict boundary between domain-independent logic, concentrated in the BLoC layer and the `Core` module, and domain-specific content, concentrated in the UI and Data layers of each module, which is the structural basis for the white-label reuse strategy.

The BLoC pattern plays a central role in the architecture's white-label strategy. In BLoC, each component receives events as input and emits states as output, maintaining a strict unidirectional data flow: the UI layer dispatches events in response to user actions, the BLoC layer processes those events and produces new states, and the UI layer re-renders in response to state changes. The Data layer provides raw data to the BLoC layer through repositories, which abstract the source of data [2]. This separation is critical for white-label reuse: because BLoC components contain no domain-specific references, they can be carried unchanged into a new application instance simply by supplying different data from the Data layer and different visual assets from the UI layer. The evaluation in Section 5 confirms this property empirically: all nine BLoC modules were reused without structural modification. The BLoC pattern itself is not a contribution of this work. Its role is to provide a structured separation of concerns between presentation, business logic, and data layers, supporting the proposed domain-decoupling and reuse strategy.

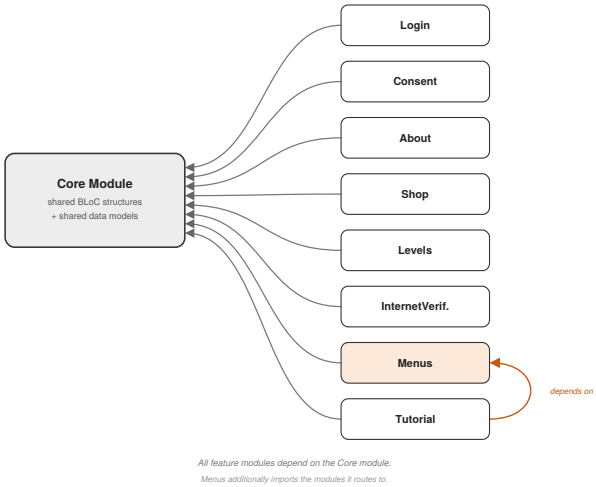

Figure 2: Dependency structure of the white-label architecture modules.

3.2 Technology Stack

Flutter [1] provides the cross-platform UI framework, targeting Android and iOS from a single Dart codebase. Its widget-based component model is particularly well-suited for white-label development: the `SizedBox` widget, for instance, automatically resizes visual components to the maximum space available on the device, enabling full interface reskinning through asset replacement without layout code changes. Firebase [12] supplies authentication (Google Sign-In), cloud storage for visual assets and audio, analytics (Firebase Analytics), and the Remote Config service. The BLoC pattern [2] structures application logic through events and states, ensuring a clean, testable separation between the presentation and business logic layers.

Key Flutter dependencies used in the implementation include: `flutter_bloc` for BLoC pattern support; `flutter_modular` for navigation and dependency injection between modules; `youtube_player_flutter` for in-level explanatory videos; `shared_preferences` and `hive` for local persistence of game state; `connectivity_plus` for internet availability checks; and the Firebase SDK packages for backend integration.

3.3 Remote Configuration and White-Label Mechanism

Firebase Remote Config is the primary white-label enabler of the architecture. It stores all domain-specific parameters in structured JSON documents, including: quiz questions and answers with hint texts (`application_quiz_questions`), game and level configuration (`application_config`), remote URLs for visual assets and audio (`application_assets_url`), and a versioning parameter (`application_assets_version`) that triggers automatic asset resynchronization when modified. At application startup, the system compares the local asset version against the remote one; if outdated, it downloads the latest assets transparently before presenting the main screen.

The `application_quiz_questions` parameter encodes each level as a JSON object containing a `videoURL` for the pre-level explanatory video and a `questions` array. Each question carries a `question` text field, an `answers` list, a `tip` hint string, and a `rightAnswer` index. The `application_products` parameter similarly encodes store items with `name`, `imageName`, and `value` fields. These parameters can be edited directly in the Firebase console, and the application will reflect the changes on its next initialization without requiring a code recompilation or app store update.

3.4 First Approach: *Grana*

The architecture was first validated through *Grana*, a quiz-based DEG for financial literacy targeted at rural school children. The game was initially developed using the Godot engine [7] and later reimplemented for iOS [19]. In that architectural instantiation, 16 existing features were mapped and 7 new features were implemented using the proposed white-label architecture. *Grana* demonstrated the feasibility of applying the architecture to develop a functional DEG. However, it did not empirically evaluate whether the architecture facilitates reuse across educational domains or supports adaptation in a new development scenario, which is the gap addressed by this work.

3.5 Second Approach: *Investe*

Investe is a mobile quiz DEG developed as a second instantiation of the white-label architecture (Section 3). Its educational goal is to teach basic concepts of financial investments, including stock markets, cryptocurrencies, risk, and return, to young adults and beginners through quiz-based gamified interaction.

3.5.1 Development Process: The adaptation was performed by a developer not involved in the original architecture design. Three phases were followed: (i) familiarization with the architecture and module mapping; (ii) domain adaptation that replaces quiz content, visual assets and Remote Config parameters for the investment domain; and (iii) integration and validation testing to confirm functional parity with the original architecture, covering navigation across all modules, quiz loading via Remote Config, the in-game shop, and the scoring flow. No structural changes were made to the BLoC layer; customization was concentrated in the Data and UI layers through Remote Config parameter replacement and visual asset substitution.

3.5.2 Educational Content and Gameplay Design: the quiz content covers five thematic areas: (i) basic investment concepts (risk, return, liquidity); (ii) fixed-income instruments (Treasury bonds, CDBs); (iii) variable-income instruments (stocks, REITs); (iv) cryptocurrencies; and (v) personal financial decision-making. Questions are organized into progressive levels, with earlier levels covering foundational concepts and later levels addressing more complex scenarios.

The gameplay follows the architecture's level-based structure: players answer multiple-choice questions, receiving immediate feedback and earning in-game currency for correct answers. Currency can be accumulated or spent in the in-game shop on power-up items (hints, skip, extra time, answer elimination). Short explanatory videos are presented between levels to introduce concepts for upcoming questions, providing contextual scaffolding aligned with quiz-game pedagogical principles [13].

3.5.3 Implemented Features. Table 1 presents the features implemented in the current version of *Investe*. The application inherits the feature set defined in the white-label architecture implementation used in *Grana*. Of the 23 features available in this implementation, 22 were carried forward to *Investe* with their implementations preserved or adapted, and one feature (F23, multi-language support) was deferred to future work.

Table 1: Feature mapping between the white-label architecture and the *Investe* adaptation, indicating feature origin and implementation status.

ID	Feature	Origin	Status
F01	Start Screen	Existing	Implemented
F02	Main Menu	Existing	Implemented
F03	Intro Screens	Existing	Implemented
F04	Avatar Selection	Existing	Implemented
F05	Shop Screen	Existing	Implemented
F06	About Screen	Existing	Implemented
F07	Level Map	Existing	Implemented
F08	Level Opening	Existing	Implemented
F09	Quiz Questions	Existing	Implemented
F10	Level Reward	Existing	Implemented
F11	GameOver Screen	Existing	Implemented
F12	Save Game	Existing	Implemented
F13	Help Buttons	Existing	Implemented
F14	Scoring System	Existing	Implemented
F15	Audio Manager	Existing	Implemented
F16	Remote Q&A	Existing	Implemented
F17	Internet Check	New	Implemented
F18	Consent Screen	New	Implemented
F19	Login Screen	New	Implemented
F20	Expl. Video	New	Implemented
F21	Answer Analytics	New	Implemented
F22	Remote Assets	New	Implemented
F23	Multi-Language	New	Planned

3.5.4 User Interface. The application's screens were built from the architecture's existing widget components with the visual identity

adapted to the investment theme via Remote Config and asset replacement. Primary screens include: Google authentication login, main menu, level map, quiz interaction (with feedback states for correct and incorrect answers), in-game shop, and end-of-game results displaying correct answers, errors, and accumulated currency. All screen content is managed remotely, enabling updates without application redeployment. Figure 3 contrasts the corresponding Grana and Investe screens for the main menu, level map, shop, and quiz interaction. The two games share the same interface structure and widget composition; only the visual assets and domain content differ, replaced through Remote Config and asset substitution without changes to the UI layout code.

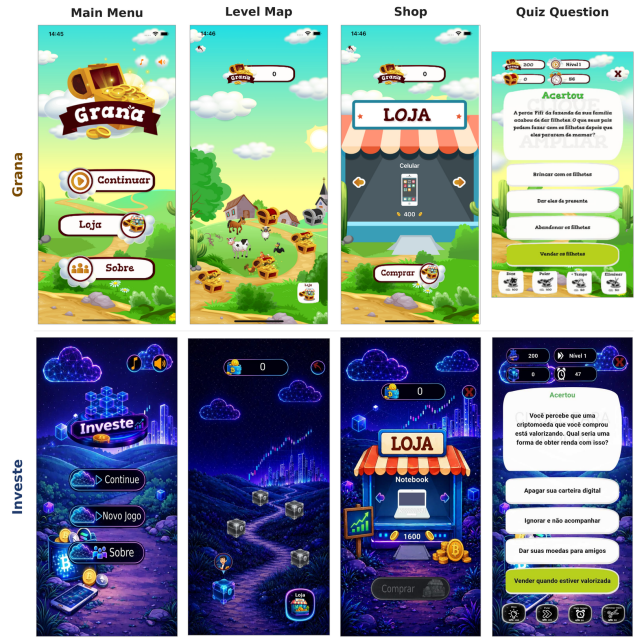

Figure 3: Corresponding screens (Main Menu, Level Map, Shop, and Quiz Interaction) of Grana (top) and Investe (bottom).

4 Evaluation Methodology

The evaluation adopts a case study design [21], using the *Investe* game as the unit of analysis and combining three complementary perspectives: (i) code reuse analysis; (ii) developer feedback on architectural reuse; and (iii) preliminary assessment of learning outcomes.

4.1 Code Reuse

Quantitative metrics were collected using `cloc` [10] and `git diff` applied to both the *Grana* and *Investe* repositories, excluding generated files and binary assets. We computed the total Dart LOC, the number of lines reused without modification, and the number of new or modified lines, together with the count of changed files, insertions, and deletions. Each module was also classified as fully or partially reused, describing the adaptation performed. In addition, the cyclomatic complexity per method was collected for both repositories using Dart Code Metrics.

4.2 Developer Evaluation on Architectural Reuse

After completing Investe's development, the developer responsible for adaptation completed a questionnaire with two parts: (1) an adapted version of the 10-item System Usability Scale (SUS), as described in Section 2; and (2) six open-ended questions targeting specific aspects of the adaptation process.

The open-ended questions covered: (a) which modules were most and least straightforward to adapt; (b) the perceived role of the Remote Config mechanism during the adaptation; (c) an estimate of development effort saved compared to a hypothetical from-scratch implementation of the same application; (d) the main sources of friction or unexpected complexity encountered; (e) which architectural decisions were most beneficial in practice; and (f) suggestions for improving the architecture or its documentation.

SUS scores were computed following the standard procedure [6] and interpreted using the adjectival rating scale of Bangor et al. [3]. Open-ended responses were analyzed through thematic analysis [5] by two researchers independently, with disagreements resolved by discussion until consensus was reached.

4.3 Preliminary Assessment of Learning Outcomes

A pre-test/post-test study was conducted with 11 volunteers aged 17–23, recruited by availability and interest in the topic. Participants answered a five-item multiple-choice pre-test on financial investments, interacted freely with Investe, then completed an equivalent five-item post-test. A Wilcoxon signed-rank test [25] was applied to the paired scores given the small sample size. This dimension serves as a functional validation check, assessing whether the application produced by the white-label adaptation process is educationally operative, not whether the architecture itself causes learning gains.

5 Results and Discussion

5.1 Code Reuse Metrics

Table 2 presents the Lines of Code (LOC) distribution across the *Investe* Dart codebase, excluding auto-generated files and asset binaries. The analysis classifies source code according to whether it was reused without structural modification or required changes to support the new educational domain. The results show that 7,109 out of 7,977 Dart LOC (89%) were directly reused from the original architecture, while 868 lines (11%) corresponded to new or modified code. These changes were primarily associated with application-specific adaptations, such as domain data and presentation elements, indicating that the proposed architecture enables substantial software reuse while allowing the incorporation of domain-specific requirements.

Across all nine modules, the BLoC components remained structurally unchanged, indicating that the separation between business logic and domain-specific content contributed to the architecture's

Table 2: Source code reuse metrics for *Investe* (Dart, `lib/`), measured with cloc 1.98 and `git diff`.

Category	LOC	Codebase (%)
Reused without modification	7,109	89%
New or modified (insertions)	868	11%
Total (Dart)	**7,977**	**100%**

Table 3: Module reuse and required adaptations during the development of *Investe*.

Module	Reuse	Adaptation Performed
Core	Full	No changes
Login	Full	No changes
InternetVerif.	Full	No changes
Menus	Full	No changes
Consent	Partial	Text via Remote Config
Levels	Partial	Parameters via Remote Config
Tutorial	Partial	Content via Remote Config
Shop	Partial	Values via Remote Config; widget skin replaced
About	Partial	Institution info and credits replaced

reuse capability. Table 3 summarizes the reuse of the *Investe* modules, indicating whether each one was fully or partially reused and describing the adaptations performed during the white-label instantiation.

A file-level comparison between the Grana and Investe repositories identified 30 changed Dart files, comprising 868 insertions and 680 deletions. Of the 7,977 physical Dart lines in Investe, 7,109 (89%) were carried over from Grana without modification, confirming that domain adaptation was concentrated in a small fraction of the codebase.

Beyond LOC, we computed the cyclomatic complexity per method for each module, in both Grana and Investe, using Dart Code Metrics. As shown in Table 4, complexity is uniformly low and virtually identical before and after adaptation (overall average 1.62 vs. 1.63), indicating that the domain adaptation did not degrade the structural health of the reused code.

5.2 Developer's Feedback on Architectural Reuse

The developer reported approximately 3 weeks of part-time work (~120 hours) to complete *Investe*. The self-estimated effort for an equivalent from-scratch implementation was 9-10 weeks (~360 – 400 hours), suggesting a perceived reduction of 66-70%. This estimate carries the inherent uncertainty of counterfactual reasoning and is reported as a perception indicator.

Table 5 presents the adapted SUS responses. The SUS score of 77.5 falls in the "Good" range [3] (68 – 80.3). Item 4 received a score of 1 (strongly disagree), reflecting that the developer did not require external assistance to use the architecture effectively, despite the initial learning curve associated with the Core module's shared dependencies.

Table 4: Average cyclomatic complexity per method, by module, before (Grana) and after (Investe) adaptation.

Module	Grana	Investe
Core	1.72	1.73
Login	1.40	1.48
InternetVerif.	1.22	1.27
Consent	1.90	1.90
Levels	1.63	1.63
Tutorial	1.45	1.45
Menus	1.47	1.51
Shop	1.78	1.76
About	1.62	1.62
Overall	**1.62**	**1.63**

Table 5: Adapted SUS responses for assessing architectural usability from the developer's perspective.

#	Adapted Item	Score
1	I would like to reuse this architecture frequently	4
2	I found the architecture unnecessarily complex	2
3	I found the architecture easy to work with	3
4	I needed assistance to use the architecture effectively	1
5	Architectural components were well integrated	5
6	There was too much inconsistency in the architecture	2
7	Most developers could adapt this architecture	5
8	The architecture was cumbersome to use	2
9	I felt confident working with the architecture	4
10	I needed to learn a lot before using the architecture	3
SUS Score		**77.5**

Thematic analysis of the developer's feedback identified four themes related to architectural reuse, adaptation effort, and development workflow.

Theme 1 — Remote Config as Primary Reuse Enabler. The developer identified Remote Config as the most impactful feature: "Changing questions, rewards, and levels without touching any code was a game changer. I could iterate on content independently from the build cycle." This finding highlights the role of Remote Config as a key mechanism for separating educational content from application logic, enabling content adaptation without requiring source-code changes.

Theme 2 — BLoC Layer as Stability Anchor. BLoC components provided a stable, predictable foundation: "Once I understood

the event–state flow in the Levels module, adapting others was straightforward — they all follow the same pattern." This observation reinforces that the architectural consistency of the BLoC layer contributed to reducing adaptation effort when extending the application.

Theme 3 — Onboarding Friction in Module Dependency Navigation. Understanding the Core module's role as a shared dependency provider was the primary initial challenge. The developer suggested a module dependency diagram and a step-by-step adaptation guide as valuable additions. This finding indicates that reusable architectures require supporting documentation to reduce the learning curve associated with module relationships.

Theme 4 — Asset Pipeline as Minor Bottleneck. Preparing and uploading domain-specific visual assets to the cloud storage location referenced by Remote Config was identified as a manual, error-prone step. CLI tooling for asset synchronization was suggested. This finding indicates that, although the software architecture reduced implementation effort, some operational activities outside the codebase, such as asset management, remained potential sources of manual effort.

5.3 Preliminary Educational Effectiveness

Table 6 presents individual pre-test and post-test scores collected before and after interaction with *Investe*, allowing an initial assessment of changes in participants' investment knowledge.

Table 6: Participants' characteristics and pre-test/post-test scores on investment knowledge assessment.

Participant	Age	Pre-test	Post-test
P1	19	2	3
P2	22	3	4
P3	17	1	3
P4	22	2	4
P5	21	3	2
P6	22	2	2
P7	21	2	3
P8	22	4	5
P9	23	1	2
P10	18	2	3
P11	21	3	3
Mean	**20.7**	**2.27**	**3.09**

The mean score increased from 2.27 to 3.09, representing an average gain of +0.82 on a 0–5 scale. The Wilcoxon signed-rank test indicated a statistically significant difference between pre-test and post-test scores ($W = 4.0$, $p = 0.0209$), providing preliminary evidence that interaction with *Investe* may contribute to improved investment knowledge. This exploratory evaluation should be interpreted as an initial assessment of the game's educational potential rather than a conclusive measure of learning effectiveness. The limited sample size and the absence of a control group represent important limitations, and further studies with larger and controlled samples are required to investigate the educational effectiveness of the game.

5.4 Discussion

The results provide empirical evidence that the white-label architecture supports the development of quiz-based educational games in a new educational domain. The 89% code reuse rate, absence of structural modifications across all BLoC components, SUS score of 77.5, and statistically significant learning gain provide complementary evidence that the architecture achieves its intended reuse and adaptability goals in a real-world development scenario.

Remote Config emerges as an important architectural mechanism that facilitates software reuse. By externalizing domain-specific content, it transforms content adaptation from a programming task into a data management task, reducing the need for source-code modifications. The identified friction points, namely onboarding and the asset pipeline, appear addressable through documentation and tooling improvements without requiring architectural restructuring, representing feasible directions for future refinement.

6 Threats to Validity

Construct Validity. The SUS was adapted from system to architectural developer experience assessment. While such adaptations are established in API evaluation [17], the adapted items may not capture all relevant dimensions of architectural usability. Validation with multiple respondents is recommended.

Internal Validity. The effort reduction estimate (67–70%) is a counterfactual self-assessment subject to recall bias and anchoring effects and is reported as a perception indicator only. The educational study lacks a control group, precluding causal attribution of learning gains to the application alone.

External Validity. A single developer and a single new domain limit generalizability. The developer's prior Flutter and Firebase experience may have reduced onboarding friction relative to a less experienced developer. Replication across developers and domains is necessary.

Reliability. LOC and complexity metrics were collected with a reproducible toolchain (`cloc`, `git diff`, and Dart Code Metrics) under a documented counting policy (excluding generated files and assets). Qualitative analysis was conducted by two researchers independently, with disagreements resolved by discussion.

7 Conclusion

This paper presented a developer-centered empirical evaluation of a white-label architecture for quiz-based DEG, using the *Investe* application, a quiz-based digital educational game for financial investment education developed as part of this work, as the evaluation case.

The results provide empirical evidence that the architecture supports software reuse and adaptation: 89% of the Dart codebase was reused without structural modification; no structural changes were required in any BLoC module; domain-specific content customizations were handled through Remote Config; the developer experience assessment achieved a SUS score of 77.5 ("Good"); and interaction with *Investe* was associated with a statistically significant improvement in participants' financial investment knowledge ($W = 4.0$, $p = 0.0209$), providing preliminary evidence of its educational potential.The evaluation also identified improvement

opportunities: (i) a module dependency diagram and step-by-step adaptation guide to reduce onboarding friction; and (ii) CLI tooling to automate the asset synchronization pipeline.

As future work, we plan to address the identified improvement opportunities through additional documentation and tooling, and replicate the evaluation with additional developers across different educational game domains.

Artifact Availability

The materials supporting this study (representative source-code excerpts of *Investe* paired with the corresponding *Grana* excerpts, the adapted SUS questionnaire, and the anonymized open-ended developer responses) are available from the authors upon reasonable request.

Acknowledgments

The authors thank the Instituto Federal do Ceará (IFCE) for the institutional support that made this work possible.

Generative AI tools were used only to improve manuscript organization and readability. They were not used to generate technical content, data, analyses, or results. The authors reviewed all outputs and take full responsibility for the final manuscript.

References

[1] Salah Y. Ameen and Dana Y. Mohammed. 2022. Developing Cross-Platform Library Using Flutter. *European Journal of Engineering and Technology Research* 7, 2 (2022), 18–21.

[2] Felix Angelov. 2024. BLoC Library Documentation. https://bloclibrary.dev. Accessed: 2026.

[3] Aaron Bangor, Philip Kortum, and James Miller. 2009. Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale. *Journal of Usability Studies* 4, 3 (2009), 114–123.

[4] Paulo Eduardo Battistella and Christiane Gresse von Wangenheim. 2016. Games for Teaching Computing in Higher Education: A Systematic Review. *IEEE Technology and Engineering Education* 9, 1 (2016), 8–30.

[5] Virginia Braun and Victoria Clarke. 2006. Using Thematic Analysis in Psychology. *Qualitative Research in Psychology* 3, 2 (2006), 77–101.

[6] John Brooke. 1996. SUS: A "quick and dirty" usability scale. *Usability Evaluation in Industry* (1996), 189–194.

[7] Carlos Henrique Leitão Cavalcante et al. 2021. Grana — Educação financeira para crianças de escolas rurais através de um jogo para dispositivos móveis. In *Anais do XXXII Simpósio Brasileiro de Informática na Educação (SBIE)*. SBC, Porto Alegre, 360–370.

[8] Sridhar Chimalakonda and Kesav Vithal Nori. 2020. A Family of Software Product Lines in Educational Technologies. *Computing* 102, 8 (2020), 1765–1792. doi:10.1007/s00607-019-00740-1

[9] Ana Grasielle Dionísio Corrêa, Giovana Tinasi Goulart, Douglas Fabiano Lourenço, Silvana Maria Blascovi-Assis, and Bruno da Silva Rodrigues. 2025. Versão Adaptada do Questionário SUS (System Usability Scale) para Avaliação de Usabilidade de Jogos Digitais com o Público Infantojuvenil: estudo piloto. In *Anais do XXIV Simpósio Brasileiro de Jogos e Entretenimento Digital (SBGames)*. Sociedade Brasileira de Computação, Salvador, BA, Brasil, 826–836. doi:10.5753/sbgames.2025.9837

[10] Al Danial. 2024. CLOC — Count Lines of Code. https://github.com/AlDanial/cloc. Accessed: 2026.

[11] William B. Frakes and Kyo Kang. 2005. Software Reuse Research: Status and Future. *IEEE Transactions on Software Engineering* 31, 7 (2005), 529–536.

[12] Google LLC. 2023. Firebase Documentation. https://firebase.google.com/docs. Accessed: 2026.

[13] Juho Hamari, Jonna Koivisto, and Harri Sarsa. 2014. Does Gamification Work? – A Literature Review of Empirical Studies on Gamification. In *Proceedings of the 47th Hawaii International Conference on System Sciences (HICSS)*. IEEE, 3025–3034.

[14] Kobus Christian Heynen. 2024. *Launching a Digital Platform Ecosystem in the Automotive Industry: A Case Study of Integrating a White Label Digital Platform for BMW's In-Vehicle Infotainment System.* Master's thesis. University of Innsbruck.

[15] Maurizio Morisio, Michel Ezran, and Colin Tully. 2002. Success and Failure Factors in Software Reuse. *IEEE Transactions on Software Engineering* 28, 4 (2002), 340–357.

[16] Giani Petri, Christiane Gresse von Wangenheim, and Adriano Ferreti Borgatto. 2019. An Evolution of a Model for the Evaluation of Educational Games. In *Proceedings of the 2019 Conference on Human Factors in Computing Systems (CHI)*. ACM.

[17] Marco Piccioni, Carlo A. Furia, and Bertrand Meyer. 2013. An Empirical Study of API Usability. In *Proceedings of the 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. ACM, 5–14.

[18] Klaus Pohl, Günter Böckle, and Frank van der Linden. 2005. *Software Product Line Engineering: Foundations, Principles and Techniques.* Springer, Berlin, Heidelberg.

[19] Albert RQ Queiroz, Bruno C da Silva, and Carlos H Leitao. 2022. Desenvolvimento de um jogo de educação financeira utilizando elementos da ruralidade do nordeste brasileiro para plataformas Apple. In *Simpósio Brasileiro de Jogos e Entretenimento Digital (SBGames)*. SBC, 1455–1459.

[20] Ayme Nori Ruiz-Carhuamaca, Jair Alexander Yauricasa-Seguil, and José Carlos Morales-Arevalo. 2024. Design of a Mobile Learning App for Financial Literacy in Young People Using Gamification. *International Journal of Advanced Computer Science and Applications (IJACSA)* 15, 12 (2024), 95–98.

[21] Per Runeson and Martin Höst. 2009. Guidelines for Conducting and Reporting Case Study Research in Software Engineering. In *Empirical Software Engineering*, Vol. 14. 131–164.

[22] Rafael Savi, Christiane Gresse von Wangenheim, Vania Ulbricht, and Tarcisio Vanzin. 2010. Proposta de um Modelo de Avaliação de Jogos Educacionais. In *Anais do RENOTE*, Vol. 8.

[23] Chirag Sharma. 2020. *Applicability of Design System to White-Label Service Development.* Master's thesis. Aalto University.

[24] Ananthan Umathay and Ashish Kumar Sinha. 2016. User Experience Strategy for White Labeling a Software Product. In *Proceedings of the 8th Indian Conference on Human-Computer Interaction (IndiaHCI)*. ACM, 102–110.

[25] Frank Wilcoxon. 1945. Individual Comparisons by Ranking Methods. *Biometrics Bulletin* 1, 6 (1945), 80–83.