# Software Architecture Requirements for Procedural Content Generation in Digital Games: A Systematic Mapping

**Arthur R. Pinheiro So[1], Francisco Carlos M. Souza[2], Alinne C. Corrêa Souza[2], José Augusto Fabri[1],**

[1]Federal University of Technology – Paraná – Cornélio Procópio, PR, Brasil

[2]Federal University of Technology – Paraná – Dois Vizinhos, PR, Brasil

arthurriuiti@alunos.utfpr.edu.br

{franciscosouza, alinnesouza, fabri}@utfpr.edu.br

***Abstract. Introduction****: Procedural Content Generation (PCG) has become essential in game development to support designers and manage growing system complexity. However, there is a lack of dedicated software architectures for PCG-based systems.* ***Objective:*** *In this study, we aim to identify and analyze software architectures and frameworks used in games that employ PCG techniques.* ***Methodology****: A systematic mapping study reviewed 121 publications. From these, we selected nine studies—four through database search, three via snowballing, and two from manual search.* ***Results:*** *Most architectures support multi-content generation and integrate user experience metrics. Standard components include data storage, multi-source inputs, and modular generators. The selected studies highlight trends toward modular, scalable, and user-centered architectures for PCG, offering a foundation for future research and development in this domain.*

***Keywords*** *procedural content generation, software architecture, systematic mapping, software requirements*

## 1. Introduction

Procedural content generation (PCG) has been broadly used in the games industry for various reasons, such as reducing development time and costs [Togelius et al. 2013]. The increased use of PCG techniques and the growth in the complexity of digital games make it necessary to consider which architectures are being used, since the architecture is known to play a major role in software quality [Wasserman 1996]. Architectures must also be considered in PCG systems, but as investigated in this work, only a few works have explored the area of architectures in the domain of PCG.

In this context both [Pereira et al. 2022] and [Olsen Peter 2014] present architectures for integrating multiple content generators, [Pereira et al. 2022] creates an architecture for orchestrating maps, quests, and enemies for top-down shooter and later to a 2D platformer game [Teoi et al. 2024], while [Olsen Peter 2014] proposes an architecture for integrating various types of terrain generators (mountains, constructions, forests for instance). Similar to ours, [Morelli e Nakagawa 2011] presents a systematic mapping of studies that describe the use of software architectures in video game development. In total, 31 studies were selected for analysis out of 276, which were

categorized according to the respective video game subsystem, topic, environment, and evaluation and use level.

Although studies delve into both the fields of software architecture for games and architectures for the field of PCG, few provide an investigation of architecture for PCG. Works from [Morelli e Nakagawa 2011] only offer a view of architectures for games. Works that propose architectures, such as [Olsen Peter 2014] and [Pereira et al. 2022], could benefit from a systematic investigation in this context. Thus, this work should stand out as a systematic mapping of software architectures for games that use PCG. Two research questions were defined to identify the types of software architectures used in games with PCG and the characteristics considered in designing these architectures. In total, nine studies were identified, which were classified according to the observed architectures' characteristics.

The remainder of the article is organized as follows: Section 2 presents the concepts related to the study in more depth. Section 3 details a systematic mapping carried out to identify similar works and gaps in research. Section 4 presents the results of the systematic mapping conducted, and finally, Section 5 concludes with possible future work.

## 2. Background

### 2.1. Procedural Content Generation

There are different types of content present within digital games: maps, quests, sounds, textures, and even the game rules are considered digital content [Togelius et al. 2013]. Creating this content is a creative process that involves both technical and artistic aspects, requiring significant effort and time on the part of the developers.

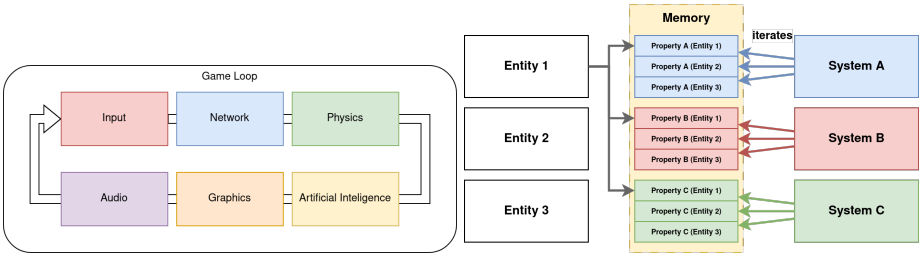
Procedural content generation, which means generating content automatically, with little or no human input [Togelius et al. 2013], emerged in the early 1980s when it was used for the first time in the game Rogue (1985) to create its levels in dungeon format in a way that saved memory space. Current games utilize PCG for a variety of motives, such as reducing development time and costs, and providing variety in content generated [Togelius et al. 2013].

### 2.2. Software Architecture in Games

In the ever-evolving software ecosystem, software architecture plays a critical role in defining system quality and maintainability [Wasserman 1996]. While multiple definitions exist, the core concept of software architecture revolves around describing the system's high-level structure and the rationale behind key design decisions [Garlan 2000]. This encompasses how the system is organized and the properties attributed to each component. A well-defined architecture serves as a blueprint for designing and developing complex systems, helping ensure that key requirements, such as performance, portability, scalability, and interoperability, are met [Shaw e Clements 2006]. Conversely, poor architectural choices can hinder these goals and compromise the overall system.

In the context of games, certain patterns are widely adopted, such as the Game Loop, which is present in nearly every game [Nystrom 2014]. Games rely on multiple systems, such as input, audio, physics, and rendering. The Game Loop is the architectural

pattern for synchronizing these systems within a continuous repetition cycle, as illustrated in Figure 1a. Another architectural pattern worth mentioning is the Entity Component System, used to increase performance in games dealing with many simulated entities. Figure 1b shows a general representation of ECS. Its operation consists of allocating a continuous memory space to store the properties of objects, which are referred to as entities. Game systems iteratively traverse this memory space to access properties. As the memory space is continuous, CPU access to RAM is reduced, increasing performance considerably.




**(a) Game Loop** **(b) Entity Component System**

**Figure 1. Two architectures commonly used in digital games.**

## 3. Systematic Mapping

A Systematic Mapping (SM) is a technique within the Evidence-Based Software Engineering (EBSE) paradigm that seeks to synthesize study areas within a specific topic. The SM was conducted following the [Kitchenham e Charters 2007] guideline.

### 3.1. Definition of Research Questions

The objective of this SM is to identify the types of software architectures that have been used in games that have PCG. To refine the objective, two Research Questions (RQs) were defined:

- **RQ1**: What types of software architectures have been used in games that have PCG?
- **RQ2**: What characteristics are taken into account when designing the architecture?

### 3.2. Search Strategies

To expand the coverage of different studies, a hybrid search strategy [Mourão et al. 2020] was used. First, the search string was applied to the IEEE[1], ACM[2] and Scopus[3] databases, and then the snowball strategy (snowballing) [Wohlin 2014] was conducted. The Scopus database contains peer-reviewed publications from leading ScienceDirect (Elsevier) journals and conferences, and research articles from Springer. The materials were accessed through a CAPES-subsidized institutional subscription. The search string used was: *("game") AND ("pcg" OR "procedural content generation" OR*

[1]https://ieeexplore.ieee.org/Xplore/home.jsp
[2]https://dl.acm.org
[3]https://www.scopus.com

*"content generation") AND ("software architecture" OR "system architecture" OR "game architecture")*

Based on our RQs, a set of Inclusion Criteria (IC) and Exclusion Criteria (EC) was defined:

- **Exclusion Criteria:**
  - **IC1:** The study presents an architecture that was used in PCG;
  - **IC2:** The study is a primary study;
  - **IC2:** The study is a full paper;

- **Exclusion Criteria:**
  - **EC2:** The study is not contextualized in the area of digital games;
  - **EC3:** The study is in English or Portuguese;
  - **EC4:** Repeated studies;
  - **EC5:** Studies that were older versions of others;

Although the research was primarily conducted using peer-reviewed databases, the peer-review status of individual studies was not used as an inclusion or exclusion criterion. This decision was made to allow a broad overview of the available literature.

### 3.3. Selection Process

The study selection process was divided into three stages: (1) search in databases; (2) snowballing method; and (3) manual search, as shown in Figure 2. During **Step 1**, ICs and ECs were used to select relevant studies first based on their titles and abstracts and then based on full readings.

From the studies selected in the previous stage, in **Step 2**, other studies were selected using the snowballing backward and forward method. The index used to check the number of references was Google Scholar. In total, three iterations of snowballing were conducted before the search couldn't return any results, and for each selected study, the inclusion and exclusion criteria were applied. Finally, in **Step 3**, the manual search was performed by general searches on Google Scholar by combining different keywords similar to the search string. At the end of the selection process, a total of 9 studies were selected, 4 studies in the database search (Step 1), 3 additional studies based on snowballing (Step 2), and 2 studies through manual search (Step 3), as illustrated in Figure 2.

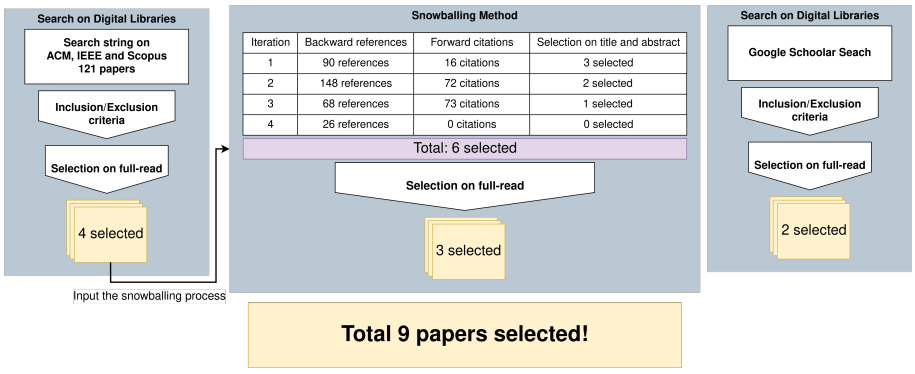



**Figure 2. Studies selection processe**

### 3.4. Data extraction and synthesis

To develop our SM, we analyzed the studies selected and identified the initial list of attributes. We then used attribute generalization and iterative refinement to derive the final map. For studies identified as relevant, we recorded them in a shared spreadsheet available on `https://docs.google.com/spreadsheets/d/1OxMWZy4dXxBQMWKP3o4riIGHjw8hDZhrhOYHivnaYcA/edit?usp=sharing` to facilitate further analysis. Our next goal was to categorize the studies to build a complete picture of the research area and answer the RQs. The Table 1 presents the final classification scheme developed and used in the SM, which was composed of seven attributes.

**Table 1. Final classification used in the SM**

| RQs | Attributes | Categories |
|---|---|---|
| - | Year | Publication Year |
| - | Type | Conference paper, Journal paper, Master's Thesis |
| RQ1 | Software Architecture Type | Orchestration (ORQ), Experience Metrics Generation (EMG), Technical (TEC) |
| RQ1 | Architectural Style | Pipeline, Monolith, Service-Based, Layered |
| RQ1 | Software Architecture Topics | product line, reference architecture, framework, design patterns, aspects, COTS, quality |
| RQ2 | Content | Maps, Quests, Enemies, URLS, Emails, Soundtrack |
| RQ2 | Requirements | Extensibility, maintainability, portability, confidentiality, usability, scalability, performance |

To obtain a consensus about studies, two researchers independently extracted and analyzed assigned study data to reach a shared understanding and classification. A third person peer-reviewed each extraction. Disagreements were resolved through discussion among the researchers. After initial extraction, a systematic peer review of each other's work ensured accuracy. The final attributes and categorizations were the result of a systematic synthesis done collaboratively following a systematic qualitative data analysis approach [Miles et al. 2014].

## 4. Results

In this section, we describe the findings of our SM in light of our research questions. Some studies were chosen to illustrate examples of each RQ's result. We consider that they are relevant and make an important contribution to the software architecture for PCG. Section 4.1 provides an overview of the studies selected. Sections 4.2 and 4.3 describe the results for the research questions $RQ_1$–$RQ_2$, respectively.

### 4.1. Overview of the studies

As previously mentioned, we identified nine studies. Table 2 presents the distribution of the studies according to ID, selection strategy, reference, year, venue type, and title. Of the nine studies, 45% (4/9) were selected from a database search, which were conference and journal papers; and 23% (2/9) from a manual search, which were master's theses.

### 4.2. Architecture Types for PCG (RQ1)

This RQ aims to identify software architecture types for PCG in digital games. From the analysis of the studies, we clustered the data extracted into three categories. [$C_1$]

**Table 2. List of studies selected in the SM**

| ID | Reference | Selection Strategy | Year | Type |
|---|---|---|---|---|
| S1 | [Röpke et al. 2021] | Database Search | 2021 | Conference |
| S2 | [López Ibáñez et al. 2018] | Database Search | 2018 | Conference |
| S3 | [Pereira et al. 2022] | Database Search | 2022 | Journal |
| S4 | [Fernandes et al. 2021] | Database Search | 2021 | Conference |
| S5 | [Zhang et al. 2022] | *Snowballing* | 2022 | Conference |
| S6 | [Shu et al. 2021] | *Snowballing* | 2021 | Conference |
| S7 | [Amiri-Chimeh et al. 2024] | *Snowballing* | 2024 | Journal |
| S8 | [Olsen Peter 2014] | Manual Search | 2014 | Master's Thesis |
| S9 | [Carpenter 2011] | Manual Search | 2011 | Master's Thesis |

- **Orchestration (ORC)**: architectures designed for the integration of multiple content generators. [$C_2$] - **Experience Metrics Generation (EMG)**: Architectures for generation based on experience metrics. This type of architecture aims to improve content generators based on experience metrics extracted during game execution. This extraction is usually done by developing intelligent agents to simulate player profiles. [$C_3$] - **Technical**: Architectures to improve the generation of a type of content by introducing techniques.

Table 3 shows the software architecture type identified in the selected studies to ID (first column), Reference (second column), Software Architecture Type - SATY (third column), Software Architecture Topic - SATO (fourth column), and Architectural Style - AST (fifth column).

**Table 3. Identification of software architecture type, their topics, and architectural style**

| ID | Reference | SATY | SATO | AST |
|---|---|---|---|---|
| S1 | [Röpke et al. 2021] | ORC | Framework | Pipeline |
| S2 | [López Ibáñez et al. 2018] | TEC | Framework | Pipeline |
| S3 | [Pereira et al. 2022] | ORC, EMG | Framework | Monolith |
| S4 | [Fernandes et al. 2021] | EMG | Framework | Pipeline |
| S5 | [Zhang et al. 2022] | EMG | Framework | Pipeline |
| S6 | [Shu et al. 2021] | EMG | Framework | Pipeline |
| S7 | [Amiri-Chimeh et al. 2024] | TEC | Framework | Pipeline |
| S8 | [Olsen Peter 2014] | ORC | Framework | Service Based |
| S9 | [Carpenter 2011] | TEC | Framework | Layered |

All studies are classified as Framework proposals. 67% (6/9) of them follow an architectural style similar to a Pipeline, with a sequence of well-defined steps for generating content, one of them is [Röpke et al. 2021] (S1), which presents an architecture that allows for substituting different components with alternative implementations. For example, the Data Collection Module enables data to be gathered in various ways. The Content Generation Module can create different types of content, and the Level Controller component interfaces with the game and can be adapted for other games that use the same content.

[López Ibáñez et al. 2018] (S2) proposes a system for the procedural generation of soundtracks, adapted from emotions provoked by the player's actions. The system works by the player performing actions within the game that provoke certain emotions. An interactive dialogue with a character can provoke anger, disgust, fear, happiness, or a set of these emotions. From the set of provoked emotions, the system chooses tracks that

correspond to the emotions provoked by the action. This set of soundtracks is mixed to form a single track, which is introduced into the game by the audio system.

The work [Olsen Peter 2014] (S8) follows a service-based architecture, as they presented several independent components that interact directly with the game designer. [Olsen Peter 2014] aims to create an architecture to make PCG techniques more accessible to game designers and developers. The Modular PCG is composed of various PCG modules (the generators) that operate independently but can be used together. In this context, each module creates a part of the content, allowing for a higher granularity of options to be adjusted by the designer. [Pereira et al. 2022] work's (S3) presents a concrete architecture of a system called Overlord that orchestrates multiple procedurally generated contents. The work presents a procedural generation system composed of three generators: quests, dungeons, and enemies.

The work of [Carpenter 2011] (S9) presented an architecture with a hierarchy of access to generators structurally similar to a tree, akin to a layered architecture. This work explores the problem of generating game scenarios with virtually infinite areas in real-time, where territory is generated as it is explored, with no possibility of generating the entire world at once. To achieve this, the author developed an architecture that describes a way to organize various areas of the map in a tree structure. The shape of the tree in this case depends on the implementation; for example, some maps can be represented as a binary tree, a *quadtree*, or an *octree*.

### 4.3. Architecture Characteristics for PCG (RQ2)

This RQ aims to identify software architecture characteristics for PCG in digital games. Firstly, few studies seek to offer an architectural vision within the context of PCG, even though, separately, they are two areas that are well-founded in the literature. A second relevant observation is that they were all conducted in the academic context. A summary of the characteristics found in the architectures can be seen in Table 4.

Most studies focus on generating game maps. However, two works — [Pereira et al. 2022] (S3) and [Olsen Peter 2014] (S8) also address the generation of quests and enemies. Additionally, [Röpke et al. 2021] and [López Ibáñez et al. 2018] (S1, S2) explore diverse content types such as soundtracks, URLs, and emails. All architectures include components that feed information into the generators. In most cases, this is achieved by extracting experience metrics through gameplay simulation using player models. For instance, [Röpke et al. 2021] employs a Learner Model component, while [Pereira et al. 2022], [Fernandes et al. 2021] (S4), [Zhang et al. 2022] (S5), and [Shu et al. 2021] (S6) use player profiles or personas in combination with evolutionary algorithms. Furthermore, [Röpke et al. 2021] and [Pereira et al. 2022] incorporate player data gathered through questionnaires or automated tools before gameplay begins. In contrast, [Olsen Peter 2014] and [Carpenter 2011] (S9) emphasize designer-driven content generation.

Runtime generation is explicitly addressed in [López Ibáñez et al. 2018], [Shu et al. 2021], and [Carpenter 2011], whereas [Pereira et al. 2022] performs content generation during level selection. Architectures relying on designer input typically generate content during game development. A storage component is present in [Pereira et al. 2022] and [López Ibáñez et al. 2018], either for retaining input data or

**Table 4. Characteristics of selected architectures**

| Studies | Contents | Requirements | Input | RT Gen. | Player Rep. | Storage |
|---|---|---|---|---|---|---|
| S1 | Urls, Emails | Modularity, Personalization | Game | No | Yes | No |
| S2 | Soundtrack | Adaptability, Portability | Game | Yes | No | Yes |
| S3 | Maps, Quests, Enemies | Modifiability, Extensibility | Questionnaire | No | Yes | Yes |
| S4 | Maps, Enemies | Adaptability | Game | No | Yes | No |
| S5 | Maps | Adaptability, Extensibility | Game | No | Yes | No |
| S6 | Maps | Personalization, Performance, Playability | Game | Yes | Yes | No |
| S7 | Maps | Generality, Comprehensiveness, Controllability, Adaptability | Designer | No | No | No |
| S8 | Maps, Quests | Usability, Maintainability, Performance, Modifiability | Designer | No | Yes | No |
| S9 | Maps | Scalability, Variability, Generality, Portability, Flexibility | Designer | Yes | No | No |

**RT Gen.**: The architecture performs runtime generation;
**Player Rep.**: The architecture presents a player representation model
**Storage**: The architecture presents a content storage component;

storing generated content for later use. A distinctive feature of the [Olsen Peter 2014] architecture is the inclusion of a virtual world where multiple content generator modules can interact to resolve conflicts between generated elements.

The mapped studies frequently addressed several quality attributes. Four of them (44%) [López Ibáñez et al. 2018, Fernandes et al. 2021, Zhang et al. 2022, Amiri-Chimeh et al. 2024] explicitly emphasized adaptability, highlighting the need for architectures that adjust to varying gameplay contexts and player profiles. Two studies (22%) [Röpke et al. 2021, Shu et al. 2021] focused on personalization, using player models to generate tailored content and enhance the gaming experience. Three studies [Shu et al. 2021, Olsen Peter 2014, Carpenter 2011] emphasized scalability or performance as essential attributes for supporting real-time content generation during gameplay. One study [Olsen Peter 2014] (S8) addressed usability for the use of PCG systems as a tool for level design, and [Shu et al. 2021] (S6) addressed playability as a way to emphasize the functional aspect of the content. Jahan [Amiri-Chimeh et al. 2024] (S7) highlighted comprehensibility, controllability, and generality as key factors that enhance the usability of PCG tools.

Four studies (44%) [Röpke et al. 2021, Pereira et al. 2022, Zhang et al. 2022, Olsen Peter 2014] addressed characteristics such as modularity, extensibility, modifiability, and maintainability, particularly in PCG contexts. These works emphasized the need to integrate and replace generator components within evolving systems flexibly. Two other studies [López Ibáñez et al. 2018, Carpenter 2011] highlighted portability, demonstrating how developers can adapt architectural solutions to

existing platforms such as Unity or Unreal Engine or embed them directly into ongoing game projects. [Carpenter 2011] also explored flexibility, generality, and variability, highlighting the system's capacity to produce diverse outputs and integrate them into a wide range of projects.

In the reviewed studies, most authors validated the identified quality attributes through the development of prototypes or proofs of concept. These validations often involved user interaction, such as playing with the system or modifying it by adding modules or generating content, followed by questionnaires before and after use. For example, [Amiri-Chimeh et al. 2024] evaluated usability by comparing the time designers spent creating maps with and without the JAHAN system and collected user feedback through questionnaires. In contrast, studies like [Olsen Peter 2014] and [López Ibáñez et al. 2018] adopted experience-based evaluations, relying on the authors' own experience to assess performance and attribute alignment.

We organized the quality attributes extracted from the selected studies into four main categories, which we highlighted using different background colors in Table 5. We used **green** to represent **software engineering attributes**, such as modularity, extensibility, modifiability, maintainability, and portability. The authors of these studies emphasized the importance of structural flexibility and long-term evolution in PCG systems [López Ibáñez et al. 2018, Pereira et al. 2022, Zhang et al. 2022, Olsen Peter 2014, Carpenter 2011]. We assigned **blue** to **performance-related attributes**, specifically scalability and performance. These studies focused on ensuring that the system could efficiently generate content, especially in real-time scenarios or under increased load [Shu et al. 2021, Olsen Peter 2014, Carpenter 2011]. We marked **red** for **human-centered attributes**, including usability, comprehension, and controllability. Researchers in these studies prioritized designer interaction, clarity, and control over the content generation process [Amiri-Chimeh et al. 2024, Olsen Peter 2014]. Lastly, we used **yellow** to group **gameplay-related attributes**, such as feasibility, adaptability, and variability. These works evaluated how well the generated content aligned with gameplay requirements, responded to player profiles, and maintained diversity in the gaming experience [Röpke et al. 2021, Shu et al. 2021, Zhang et al. 2022, Carpenter 2011]. By applying this categorization, we clarified how the selected studies approached quality in PCG architectures, linking technical, usability, and gameplay aspects.

Regarding the functional requirements, we extracted a list based on explicit statements found in the reviewed studies. This process involved identifying excerpts that revealed architectural intentions. For example, [Röpke et al. 2021] states in its introduction: "A solution to the aforementioned shortcomings is the personalization of learning game content towards a learner's individual characteristics." From this, we inferred a requirement to adapt content to individual players. Following this approach, Table 6 presents a consolidated list of functional requirements identified across all selected studies. Many of these requirements align directly with the quality attributes previously discussed. For instance, the requirement to "validate the gameplay of content" supports the feasibility attribute by ensuring that generated content is coherent and playable.

**Table 5. List of attributes extracted from studies**

| Studies | Attributes | Description | Artifacts |
|---|---|---|---|
| S1 | Modularity | Degree to which the system is decomposable into cohesive, loosely coupled parts | Prototype |
| S3, S5 | Extensibility | Ability to add new generators to the system | – |
| S3, S8, S9 | Modifiability | Ease of modifying system components | Prototype + Self-Assessment |
| S8 | Maintenance | Ability to keep the system functional over time | Prototype + Experience-based evaluation |
| S2, S9 | Portability | Ability to adapt the system to different environments (e.g., engines, existing games) | – |
| S9 | Scalability | Capacity to increase content generation as needed | Prototype + Benchmark |
| S6, S8 | Performance | Speed and efficiency of content generation | |
| S7, S8 | Usability | Suitability of the system for designers to use effectively | |
| S7 | Comprehension | Ease with which users understand the system | Prototype + Surveys + Benchmark |
| | Controllability | Degree of user control over the generation process | |
| S1, S6 | Feasibility | Relevance of generated content to the game context | Prototype + Benchmark |
| S2, S4, S5, S7 | Adaptability | Ability to tailor generated content to player characteristics | Prototype + Surveys |
| S9 | Variability | Capacity to produce diverse and non-repetitive content | – |

### 4.4. Threats to Validity

A key threat to validity is the limited number of studies retrieved. This limitation affects the robustness and generalizability of the findings. Some factors that may have contributed to this are: **1**) Potential flaws in the research protocol, such as the definition of inclusion/exclusion criteria, the construction of the search string, and the selection of digital libraries, which may have excluded relevant studies; **2**) The prevalence of practical implementations in the industry that are not reflected in academic literature, i.e, many games using PCG have proprietary codebases, hindering academic analysis; **3**) The still incipient demand for software architecture discussions in the specific context of PCG. Given these constraints, the results of this mapping may not fully capture the diversity of architectures applied in real-world scenarios. To mitigate this, future work should consider complementary sources, including grey literature, technical reports, and practitioner-oriented publications.

## 5. Final Considerations

Procedural Content Generation is widely used in the gaming industry to automate content creation, enhance development agility, and reduce manual workload. Despite its practical relevance, academic research on the architectural foundations that support PCG systems remains scarce. This systematic mapping study set out to identify the architectural strategies employed and the key quality attributes pursued in PCG-enabled systems. The findings show that most architectures operate as frameworks designed to orchestrate content generators, incorporate user or designer input, and embed evaluation mechanisms to ensure content quality. While map generation remains the predominant focus, several systems also support the generation of quests, enemies, and soundtracks. Typical architectural features include player profiling, content validation through intelligent agents, and adaptability to different content types.

In future developments, PCG architectures should prioritize modularity, portability, performance, scalability, and usability to ensure a balanced trade-off between

**Table 6. List of requirements extracted from studies**

| Studies | Requirements |
|---|---|
| S1 | Generate feasible content |
| | Replace data input modules |
| S2 | Adapt content to game narratives |
| S3 | Process player data and send it to the PCG system |
| S4 | Modify how content is generated |
| S6 | Use aesthetic and functional metrics for validation |
| | Generate content infinitely and consistently |
| S7 | Configure generation pipeline modules |
| | High-level control over validation and content construction |
| | Allow manual editing after generation |
| | Provide feedback on design decisions |
| S9 | Generate content of arbitrary sizes |
| | Generate content with coherent granularity |
| | Provide flexible algorithm organization templates |
| S1, S3 | Replace content generation modules |
| S1, S4 | Adapt content to players/personas |
| S2, S6 | Iteratively expand generated content |
| S2, S3, S7 | Orchestrate content facets (levels, narratives, rules) |
| S3, S7, S8 | Reuse/add generators in other game projects |
| S4, S5, S6, S7 | Validate gameplay using agents, simulation, or algorithms |

runtime efficiency and designer control. This systematic mapping offers a structured basis for guiding the architectural evolution of PCG systems. Future studies can strengthen this field by incorporating insights from grey literature and technical documentation produced within the industry, bridging the gap between academic theory and practical application.

## References

Amiri-Chimeh, S., Haghighi, H., e Vahidi-Asl, M. (2024). Jahan: A framework for procedural generation of game maps from design specifications. *Entertainment Computing*, 50:100644.

Carpenter, E. D. (2011). Procedural generation of large scale gameworlds. Master's thesis, University of Dublin, Trinity College.

Fernandes, P. M., Jørgensen, J., e Poldervaart, N. N. T. G. (2021). Adapting procedural content generation to player personas through evolution. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 01–09.

Garlan, D. (2000). Software architecture: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, page 91–101, New York, NY, USA. Association for Computing Machinery.

Kitchenham, B. e Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report, UK. Version 2.3.

López Ibáñez, M., Álvarez, N., e Peinado, F. (2018). Towards an emotion-driven adaptive system for video game music. In Cheok, A. D., Inami, M., e Romão, T., editors, *Advances in Computer Entertainment Technology*, pages 360–367, Cham. Springer International Publishing.

Miles, M. B., Huberman, A. M., e Saldana, J. (2014). *Qualitative Data Analysis: An Expanded Sourcebook*. SAGE Publications Inc, 3rd edition.

Morelli, L. B. e Nakagawa, E. Y. (2011). A panorama of software architectures in game development. In *International Conference on Software Engineering and Knowledge Engineering*.

Mourão, E., Pimentel, J. F., Murta, L., Kalinowski, M., Mendes, E., e Wohlin, C. (2020). On the performance of hybrid search strategies for systematic literature reviews in software engineering. *Information and Software Technology*, 123:106294.

Nystrom, R. (2014). *Game Programming Patterns*. Self-published.

Olsen Peter (2014). Modular pcg. Master's thesis, Aalborg University Copenhagen.

Pereira, L. T., Viana, B. M. F., e Toledo, C. F. M. (2022). A system for orchestrating multiple procedurally generated content for different player profiles. *IEEE Transactions on Games*, pages 1–11.

Röpke, R., Drury, V., Schroeder, U., e Meyer, U. (2021). A modular architecture for personalized learning content in anti-phishing learning games. In *Software Engineering*.

Shaw, M. e Clements, P. (2006). The golden age of software architecture. *IEEE Software*, 23(2):31–39.

Shu, T., Liu, J., e Yannakakis, G. N. (2021). Experience-driven pcg via reinforcement learning: A super mario bros study.

Teoi, T., Pereira, L., e Toledo, C. (2024). Towards adapting a content orchestrator to a different game genre: Generating levels, rules, and narrative for diverse player profiles from a top-down adventure to a 2d platformer. In *Proceedings of the 23rd Brazilian Symposium on Games and Digital Entertainment*, pages 396–415, Porto Alegre, RS, Brasil. SBC.

Togelius, J., Champandard, A., Lanzi, P. L., Mateas, M., Paiva, A., Preuss, M., e Stanley, K. (2013). Procedural content generation: goals, challenges and actionable steps. *Dagstuhl Follow-Ups*, 6:61–75.

Wasserman, A. (1996). Toward a discipline of software engineering. *Software, IEEE*, 13:23 – 31.

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14, New York, NY, USA. Association for Computing Machinery.

Zhang, K., Bai, J., e Liu, J. (2022). Generating game levels of diverse behaviour engagement.