Universidade de São Paulo
Instituto de Matemática, Estatística e Ciência da Computação
Bacharelado em Ciência da Computação

OSG-RA: Uma Arquitetura de Referência para Jogos Digitais *Open Source*

Fernando Yang

Monografia Final

mac 499 — Trabalho de Formatura Supervisionado

Supervisor: Dr. Pedro Henrique Dias Valle

São Paulo
2025

Agradecimentos

Primeiramente, expresso minha profunda gratidão aos meus pais, Yang Jianwei e Ye Qiaojing. Sem o apoio incondicional deles ao longo da minha vida, não teria conseguido alcançar as conquistas de hoje. Eles são muito mais do que pai e mãe; são grandes amigos que sempre me oferecem sábias orientações, sendo o alicerce e o farol que guiam a direção da minha vida.

Também gostaria de agradecer às minhas irmãs, Cristiane e Joanna, por trazerem tantas alegrias e momentos de descontração à minha jornada. Meus sinceros agradecimentos vão igualmente aos meus grandes amigos, Nicolas e Sebastião, que me acompanharam durante quase toda a carreira acadêmica, sendo excelentes companheiros e um constante exemplo a seguir.

Em especial, dirijo meus mais sinceros agradecimentos ao meu orientador, Professor Pedro Henrique Dias Valle. Sem seu ensino e inspiração, eu não teria conhecido esta área tão fascinante da Engenharia de Software, nem compreendido a importância de uma arquitetura sofisticada para jogos digitais. Além disso, ele me ofereceu uma ajuda inestimável na condução deste trabalho, guiando-me com paciência e dedicação do início ao fim desta longa jornada.

Por fim, mas não menos importante, gostaria de agradecer a todos os outros que contribuíram para esta trajetória: ao IME-USP, pela infraestrutura, recursos e ambiente acadêmico propício à realização deste trabalho; a todos os professores que, ao longo da formação, me forneceram o conhecimento e as habilidades necessárias para desenvolvê-lo; e aos colegas Francisco Henriques, Gabriel Lima, Guilherme Assis, Gustavo Umbehaun, Lorenzo Fortes, Lucas Eiji, Pedro Cortx e Renan Marcelino, pela disponibilidade e contribuições valiosas na avaliação desta arquitetura.

Resumo

Fernando Yang. **OSG-RA: Uma Arquitetura de Referência para Jogos Digitais *Open Source***. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2025.

Com a popularização de diversos *engines* para jogos digitais, o desenvolvimento desses produtos de software tornou-se mais acessível tanto para especialistas quanto para pessoas sem formação prévia em desenvolvimento de software. No entanto, a criação de jogos digitais continua sendo um campo complexo, que exige a integração de múltiplos sistemas e tecnologias. Em especial, no contexto de desenvolvimento de jogos *open source*, cujo código é aberto e pode ser modificado por uma comunidade global de desenvolvedores, a necessidade de uma arquitetura bem definida torna-se ainda mais crítica. Diante desse cenário, este trabalho teve como objetivo principal estabelecer uma arquitetura de referência que auxilie o processo de desenvolvimento de jogos digitais *open source*, denominada OSG-RA. Para isso, adotou-se o processo sistematizado de construção de arquiteturas de referência, chamado ProSA-RA. A OSG-RA foi avaliada por meio dos métodos *Architecture Tradeoff Analysis Method* (ATAM) e *Technology Acceptance Model* (TAM), a fim de verificar sua aderência aos requisitos de qualidade identificados, bem como sua eficácia e facilidade de uso no contexto de desenvolvimento de jogos. Os resultados da avaliação demonstram que a OSG-RA atende de modo satisfatório os objetivos estabelecidos, embora pontos de melhoria tenham sido identificados para prosseguimento em trabalhos futuros. Espera-se que a OSG-RA facilite a criação de jogos digitais *open source*, assegurando, simultaneamente, atributos de qualidade essenciais como confiabilidade, manutenibilidade e jogabilidade.

Palavras-chave: Arquitetura de Referência. Open Source. Jogo Digital.

Abstract

Fernando Yang. **OSG-RA: A Reference Architecture for Open Source Digital Games**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2025.

With the popularization of various game engines for digital games, the development of these software products has become more accessible to both experts and individuals without prior software development training. However, creating digital games remains a complex field, requiring the integration of multiple systems and technologies. Particularly in the context of open-source game development, where the code is open and can be modified by a global community of developers, the need for a well-defined architecture becomes even more critical. In light of this scenario, the main objective of this work was to establish a reference architecture to assist the development process of open-source digital games, named OSG-RA. To achieve this, the systematic ProSA-RA process for building reference architectures was adopted. The OSG-RA was evaluated using the Architecture Tradeoff Analysis Method (ATAM) and the Technology Acceptance Model (TAM) to verify its alignment with identified quality requirements, as well as its effectiveness and ease of use in the context of game development. The evaluation results demonstrate that the OSG-RA satisfactorily meets the established objectives, although areas for improvement have been identified for future work. It is expected that the OSG-RA will facilitate the creation of open-source digital games while simultaneously ensuring essential quality attributes such as reliability, maintainability, and gameplay.

Keywords: Reference architecture. Open Source. Digital Game.

Lista de abreviaturas

ADL	Linguagem de Descrição Arquitetural
ATAM	*Architecture Tradeoff Analysis Method*
CI/CD	Integração Contínua e Entrega Contínua
EC	Critério de Exclusão
ECS	Entidade-Componente-Sistema
IC	Critério de Inclusão
ISO/IEC	*International Organization for Standardization* / *International Electrotechnical Commission*
JED	Jogo Educacional Digital
MVC	Modelo-Visão-Controlador
OSD	*Open Source Definition*
OSG-RA	Arquitetura de Referência para Jogos Digitais Open-Source
OSI	*Open Source Initiative*
OSS	Software *Open Source*
PEOU	Facilidade de Uso Percebida
ProSA-RA	*Process-based System Architecture Reference Model*
PU	Utilidade Percebida
RA	Arquiteturas de Referência
RAModel	*Reference Architecture Model*
RQ	Questão de Pesquisa
RS	Requisitos do Sistema
SA	Arquitetura de Software
SE	Engenharia de Software
SMS	Estudo de Mapeamento Sistemático
TAM	*Technology Acceptance Model*
UML	*Unified Modeling Language*

Lista de figuras

Lista de tabelas

Sumário

Capítulo 1

Introdução

Nos últimos anos, impulsionada pela evolução tecnológica acelerada em hardware e software e pelo crescimento de suas comunidades de desenvolvimento, a indústria de jogos digitais experimentou crescimento significativo, demonstrando seu potencial no mercado mundial (Kattilakoski, 2019; Pashkov, 2021). Este crescimento é evidenciado não apenas pelo aumento no volume de vendas, mas também pela diversificação de gêneros e plataformas, que ampliam constantemente o alcance e impacto dos jogos digitais (Zhan *et al.*, 2024). No entanto, devido à natureza complexa dos jogos digitais, que exigem integração de conhecimentos multidisciplinares envolvendo computação gráfica, inteligência artificial, *design* de interação e engenharia de áudio, o desenvolvimento desse tipo de software demanda uma abordagem sistemática para criação e gestão de sistemas que atendam às necessidades de desenvolvedores e usuários (Lucchese e Ribeiro, 2009; Ampatzoglou e Stamelos, 2010; Ahmed *et al.*, 2017; Ribeiro *et al.*, 2019).

Nesse contexto, o conceito de *open-source* — marcado pela colaboração de uma ampla comunidade de desenvolvedores e pela capacidade de incorporar rapidamente as necessidades dos usuários por meio de *feedbacks* diretos — destaca-se como uma abordagem promissora para aprimorar a qualidade dos jogos desenvolvidos, oferecendo maior flexibilidade em comparação a projetos proprietários (Scacchi, 2004; Ahmed *et al.*, 2017). Contudo, a aplicação efetiva desse modelo enfrenta desafios específicos: a natureza colaborativa e distribuída do desenvolvimento pode resultar em problemas de integração de código e inconsistências arquiteturais devido à diversidade de contribuidores, além de dificuldades na garantia de segurança, estabilidade do sistema e manutenção da qualidade do código a longo prazo (Scacchi, 2004; Fogel, 2005; Garcia *et al.*, 2010; Steinmacher *et al.*, 2015; Ahmed *et al.*, 2017). Diante desses desafios, uma arquitetura de referência para jogos digitais *open-source* surge como solução para padronizar e otimizar o desenvolvimento, fornecendo um conjunto de diretrizes que facilitam a criação, manutenção e evolução dos projetos (Elisa Y Nakagawa *et al.*, 2014). Esta abordagem visa estabelecer fundamentos sólidos para o desenvolvimento colaborativo, assegurando qualidade e consistência nos projetos por meio de um modelo arquitetural bem definido.

Na área de Engenharia de Software (SE), a Arquitetura de Software (SA) é reconhecida como elemento crítico para engenharia eficiente (Garlan e Shaw, 2008). Seu foco compreende a identificação dos principais elementos de software, as relações entre es-

ses elementos e suas propriedades, analisando os *trade-offs* e auxiliando na seleção de abordagens arquiteturais adequadas (Garlan e Shaw, 2008; B. R. Oliveira *et al.*, 2022). Dessa forma, uma SA bem definida permite que sistemas atendam requisitos centrais de projeto, assegurando qualidades como desempenho, confiabilidade, manutenibilidade e segurança (Garlan e Shaw, 2008; Venters *et al.*, 2018). Nesse contexto, a SA desempenha papel crucial no desenvolvimento de software de qualidade, alinhando expectativas dos *stakeholders* com a arquitetura documentada, além de permitir identificação e solução antecipada de problemas durante o desenvolvimento, prevenindo o maior custo posterior na sua manutenção (Angelov, Trienekens *et al.*, 2008). A importância da SA se torna ainda mais evidente em projetos de grande escala e longa duração, onde decisões arquiteturais inadequadas podem comprometer todo o ciclo de vida do software.

Diferentemente de arquiteturas concretas de software, as Arquiteturas de Referência (RA) emergem como tipo especial de arquitetura com maior nível de abstração, fornecendo diretrizes importantes para especificação e construção de arquiteturas concretas de sistemas que compartilham o mesmo domínio (Angelov, Trienekens *et al.*, 2008; Cloutier *et al.*, 2010; Elisa Y Nakagawa *et al.*, 2014; Venters *et al.*, 2018). Adicionalmente, promovem a reutilização de conhecimentos especializados em *design* com compreensão fundamentada em domínios específicos, facilitando o desenvolvimento, a padronização e a evolução de sistemas de software (Cloutier *et al.*, 2010; Elisa Yumi Nakagawa, Oquendo *et al.*, 2012). Considerando sua relevância, identifica-se a aplicação de RAs em diversas áreas (Grosskurth e Godfrey, 2005; Fremantle *et al.*, 2015; Aulkemeier *et al.*, 2016), evidenciando benefícios e sua importância para o desenvolvimento de jogos digitais *open-source*. Entretanto, embora diversas arquiteturas de referência tenham sido propostas para uma ampla gama de domínios, persiste lacuna significativa no domínio específico de jogos digitais *open-source*. A ausência de RA consolidada para este contexto força comunidades e desenvolvedores a construírem novas arquiteturas a cada projeto, dificultando a reutilização sistemática de componentes e a manutenção de linha evolutiva coesa, o que impacta negativamente a produtividade e a qualidade dos resultados.

Diante desse cenário, uma arquitetura de referência para jogos digitais *open-source* pode padronizar e otimizar o desenvolvimento de jogos, fornecendo um conjunto de diretrizes que facilitam a criação, manutenção e evolução de projetos (Elisa Y Nakagawa *et al.*, 2014). A RA visa oferecer benefícios tangíveis à comunidade de desenvolvimento, incluindo: (i) aceleração do desenvolvimento e garantia de qualidade do produto final mediante reutilização de componentes arquiteturais testados e aprovados; (ii) melhoria na interoperabilidade entre módulos desenvolvidos por diferentes colaboradores, assegurando maior estabilidade do sistema; (iii) facilitação da manutenção e evolução contínua do código mediante estrutura bem definida; e (iv) estabelecimento de vocabulário comum que facilite a comunicação e integração de novos membros na comunidade, promovendo crescimento contínuo e motivação dos desenvolvedores. Estes benefícios são particularmente relevantes considerando o contexto colaborativo e distribuído típico de projetos *open-source*. Dessa forma, ao assegurar tais benefícios, torna-se possível construir uma RA com maior aceitação pela comunidade e, consequentemente, garantir produtos de maior qualidade desenvolvidos por uma comunidade ativa e engajada.

Apesar dos benefícios potenciais, muitas RAs não são adotadas ou não sobrevivem devido à falta de comunidades sustentadoras e à insuficiente consideração do atributo

de sustentabilidade durante sua construção (Venters *et al.*, 2018). Esta problemática é especialmente relevante no ecossistema *open-source*, onde a adesão da comunidade é fator determinante para o sucesso de qualquer iniciativa. Como solução para amenizar esse problema, métodos de avaliação e processos de construção sistematizada de RAs podem ser aplicados (Santos *et al.*, 2013; Elisa Y Nakagawa *et al.*, 2014). Portanto, este trabalho apresenta uma arquitetura de referência para jogos digitais *open-source*, denominada OSG-RA, construída utilizando o *Process-based System Architecture Reference Model* (ProSA-RA) para estabelecimento da análise e síntese arquitetural da OSG-RA, além de validação. A escolha do ProSA-RA deve-se à sua natureza sistemática e à ampla adoção acadêmica para o desenvolvimento de RAs robustas, sistematizando o projeto, a representação e a avaliação de arquiteturas de referência. Para aplicação efetiva do ProSA-RA, foram estabelecidos os seguintes objetivos específicos para o estudo:

- **Identificar Fontes de Informação**: Explorar fontes de informação que descrevem conhecimento necessário para compreensão do domínio de jogos digitais *open-source*, identificando requisitos, tecnologias e características relacionadas por meio de investigações em trabalhos científicos e mapeamentos sistemáticos da literatura existente;
- **Estabelecer a Análise Arquitetural**: Identificar requisitos arquiteturalmente significativos para construção da RA, avaliando relevância de cada um no sistema mediante análise de estudos acadêmicos e jogos digitais *open-source* existentes;
- **Estabelecer a Síntese Arquitetural**: Realizar análise de padrões arquiteturais existentes no mercado, identificando *trade-offs* de cada abordagem, construir a RA e modelar a arquitetura em visões arquiteturais, demonstrando propriedades, componentes e conexões em diferentes perspectivas de desenvolvimento;
- **Avaliar RA Proposta**: Realizar estudo de caso para avaliar a RA, examinando sua aceitação por especialistas da área e identificando possíveis melhorias a serem implementadas, assegurando assim sua adequação ao contexto prático.

Este documento está organizado da seguinte forma: o Capítulo 2 apresenta a fundamentação teórica relacionada ao projeto, abrangendo Jogos Digitais, *Open-Source*, Arquitetura de Referência e o processo ProSA-RA, além dos trabalhos relacionados; o Capítulo 3 descreve a etapa de coleta de informações por meio do método de mapeamento sistemático; o Capítulo 4 apresenta a análise arquitetural, extraindo os requisitos de qualidade necessários para a OSG-RA a partir dos dados coletados anteriormente; o Capítulo 5 descreve o processo de construção da OSG-RA, bem como a apresentação das suas visões arquiteturais; o Capítulo 6 contém o método de avaliação da arquitetura, juntamente com a descrição de possíveis ameaças à validade do estudo; por fim, a conclusão do trabalho e as perspectivas futuras são apresentadas no Capítulo 7.

Capítulo 2

Referencial Teórico e Trabalhos Relacionados

Este capítulo apresenta os fundamentos teóricos essenciais para a compreensão e desenvolvimento deste projeto. A elaboração de uma arquitetura de referência requer a compreensão de diversos conceitos inter-relacionados que abrangem desde os aspectos específicos do domínio de aplicação até as práticas consolidadas na engenharia de software. Para tanto, são discutidos os principais tópicos que fundamentam esta pesquisa, estabelecendo uma base sólida para o desenvolvimento da arquitetura proposta.

Adicionalmente, são apresentados e analisados trabalhos relacionados que, de forma direta ou indireta, contribuem para o entendimento do problema de pesquisa e das possíveis soluções arquiteturais. Esta revisão permite contextualizar a contribuição deste trabalho no campo acadêmico, identificando lacunas existentes e oportunidades de pesquisa no domínio de jogos digitais *open-source*.

Este capítulo está organizado na seguinte estrutura: a Seção 2.1 apresenta os principais conceitos, características e aplicações de jogos digitais; a Seção 2.2 discute o conceito de *open-source*, seus benefícios e desafios no contexto de desenvolvimento de software; a Seção 2.3 introduz os conceitos fundamentais de arquitetura de software e suas formas de representação, além de abordar as principais características e definições sobre arquiteturas de referência, bem como estratégias para sua construção sistemática; a Seção 2.4 detalha os estudos relacionados a esta pesquisa e apresenta uma análise comparativa crítica desses trabalhos; finalmente, a Seção 2.5 apresenta as considerações finais deste capítulo.

2.1 Jogos Digitais

Desde o surgimento de *Spacewar!* em 1962, os jogos digitais emergiram como um fenômeno cultural de significativa influência, moldando e sendo moldados pela sociedade em que se inserem (Newman, 2012). Para além de seu caráter lúdico e de entretenimento, esses produtos passaram a exercer relevante impacto nas esferas sociais, políticas, econômicas e tecnológicas, atingindo um mercado global que movimenta bilhões de dólares

em um intervalo de tempo relativamente curto (Araújo *et al.*, 2018; Kattilakoski, 2019; Pashkov, 2021).

Na contemporaneidade, os jogos digitais consolidaram-se como uma das principais formas de entretenimento, superando, em diversos aspectos, mídias tradicionais como a televisão e a música (Ampatzoglou e Stamelos, 2010). Seu crescimento acelerado é impulsionado tanto pela contínua evolução do hardware e do software — que possibilita a criação de consoles mais potentes e tecnologias de simulação visual e interação cada vez mais realistas — quanto pelo fortalecimento e expansão de comunidades globais de desenvolvimento (Pashkov, 2021; Zhan *et al.*, 2024). Essa combinação de avanços técnicos e suporte comunitário tem permitido a oferta de experiências cada vez mais imersivas e complexas ao público.

Diante desse cenário dinâmico e mutável, a natureza intrinsecamente instável e evolutiva dos jogos digitais tem despertado o interesse de pesquisadores de diversas áreas do conhecimento, reforçando a importância de investigações científicas que abordem seus aspectos tecnológicos, culturais e sociais (H.-C. Hsiao, 2007; Newman, 2012). A academia tem, assim, voltado seus esforços para compreender esse fenômeno midiático que se redefine constantemente, acompanhando as transformações do ecossistema tecnológico e dos hábitos de consumo.

Uma das definições pioneiras e fundamentais para a compreensão dos jogos digitais foi elaborada por Crawford *et al.*, 1984. Na sua obra, o autor identifica e descreve quatro elementos essenciais que constituem a base de todos os jogos, a saber:

- **Representação:** Um jogo é um sistema formal fechado que representa, de maneira subjetiva e deliberadamente simplificada, um subconjunto da realidade. Como um sistema fechado, ele é completo e autossuficiente, com um conjunto de regras explícitas que cobrem todas as contingências, não exigindo referências a agentes externos. Essa representação não busca uma precisão objetiva, mas sim evoca uma realidade emocional através da fantasia do jogador, diferenciando-se de uma simulação pura pela estilização e pelo foco nos aspectos que o designer julga importantes.
- **Interação:** Diferente de formas de arte estáticas, ou dinâmicas mas não interativas (como pinturas ou filmes), um jogo proporciona uma representação interativa da realidade. Esta interação, elemento crucial do meio digital, permite ao jogador explorar ativamente relações de causa e efeito dentro do sistema, criando a sua própria narrativa a partir de uma rede de ramificações. É este elemento que distingue fundamentalmente um jogo de um quebra-cabeça ou de uma história linear, conferindo-lhe um caráter dinâmico e participativo que aumenta o investimento emocional do jogador.
- **Conflito:** O conflito emerge intrinsecamente da interação no jogo. Ele surge quando o jogador, na perseguição ativa de um objetivo, depara-se com obstáculos que respondem de forma ativa e proposital aos seus atos. Este é um elemento inevitável e fundamental em qualquer jogo, pois a presença de obstáculos inteligentes e reativos leva inevitavelmente a um confronto. Este conflito pode ser direto ou indireto, violento ou não-violento, mas está sempre presente, servindo como o motor principal dos desafios enfrentados pelo jogador.

- **Segurança:** Um jogo é um artifício que proporciona as experiências psicológicas do conflito e do perigo, excluindo as suas realizações físicas na vida real. Em outras palavras, os jogos oferecem um ambiente seguro para explorar situações que, fora dali, teriam consequências severas. O jogador pode cometer erros, enfrentar riscos e sofrer derrotas sem incorrer em penalidades reais significativas. Esta dissociação entre ação e consequência no mundo real é uma característica essencial dos jogos, permitindo a experimentação e o aprendizado num contexto de risco mitigado.

Estes quatro pilares conceituais — representação, interação, conflito e segurança — formam a essência da experiência do jogo do ponto de vista do jogador e fornecem uma base teórica sólida para a análise de jogos digitais. No entanto, para materializar estes conceitos abstratos em uma experiência digital concreta e interativa, é necessária uma complexa infraestrutura de software. A implementação bem-sucedida desses elementos depende diretamente das decisões arquiteturais e das ferramentas técnicas adotadas no processo de desenvolvimento, as quais são pesquisadas e examinadas por diversos estudos existentes.

2.1.1 Desenvolvimento de Jogo Digital

O desenvolvimento de jogos digitais é uma atividade intrinsecamente complexa e multidisciplinar, que demanda a integração de componentes diversos, como gráficos 2D/3D, áudio, lógica de jogo e, de forma crescente, sistemas de inteligência artificial (Lucchese e Ribeiro, 2009; Ribeiro *et al.*, 2019). Nesse contexto, produzir um jogo é uma tarefa que consome tempo considerável e exige uma combinação de competências técnicas e criativas (Dairel *et al.*, 2025). Consequentemente, a criação de um jogo digital requer uma equipe multidisciplinar, integrando conhecimentos especializados de programação, artes visuais, design de jogos, música e sonoplastia para se obter um produto final de qualidade, alinhado com os padrões de um mercado altamente competitivo (Araújo *et al.*, 2018). Esse cenário resulta em uma equipe heterogênea, com capacidades individuais distintas: enquanto alguns integrantes possuem formação e conhecimento mais elevados, outros provêm de áreas diversas e podem não dispor do mesmo arcabouço técnico, o que impõe desafios adicionais de comunicação e integração (Dairel *et al.*, 2025). Para gerenciar essa complexidade inerente e a natureza iterativa do desenvolvimento, torna-se essencial adotar uma implementação de software que priorize a facilidade de manutenção, acompanhada de documentação detalhada. Tais práticas são fundamentais para assegurar um entendimento comum entre os membros da equipe e agilizar o processo de desenvolvimento (Ampatzoglou e Stamelos, 2010).

Além da complexidade inerente aos jogos digitais, o processo de desenvolvimento em si — elemento essencial para que o produto final atenda às qualidades requeridas — é bastante variável, sendo influenciado por fatores como orçamento, tamanho da equipe, finalidade, plataforma, público-alvo e gênero do jogo (Aires *et al.*, 2020). O processo é geralmente constituído por várias etapas, como ideação, prototipagem, desenvolvimento, testes e implantação, que antecedem o lançamento (Dairel *et al.*, 2025). Nesse fluxo, o sucesso de cada fase é condicionado pela execução adequada da etapa anterior, tendo um impacto direto e cumulativo na qualidade do produto final (Aires *et al.*, 2020). Portanto, um erro cometido nas fases iniciais pode ter um efeito de avalanche, demandando recursos extras elevados para a correção de seus efeitos.

Para dar suporte a esse empreendimento complexo e amenizar as dificuldades decorrentes, a academia e a indústria têm conduzido pesquisas dedicadas a investigar requisitos, processos e arquiteturas de software específicas para o contexto dos jogos digitais (Kanode e Haddad, 2009; Ampatzoglou e Stamelos, 2010; Aleem *et al.*, 2016a; Aleem *et al.*, 2016b; Wilson K Mizutani *et al.*, 2021). Abordagens práticas para facilitar o desenvolvimento incluem o uso de *frameworks* especializados (R. N. R. d. Oliveira *et al.*, 2018), a aplicação de técnicas de Inteligência Artificial — em especial, o Aprendizado de Máquina para tomada de decisões (Silva Teixeira *et al.*, 2020) — e a adoção de arquiteturas de software especializadas que promovem a reutilização e a manutenibilidade (Dairel *et al.*, 2025). Entretanto, ainda há uma lacuna de estudos relacionados a jogos digitais no contexto *open-source*, sendo necessária, assim, uma arquitetura de referência que direcione o desenvolvimento desses jogos, uma vez que a área de *open-source* apresenta dificuldades próprias de colaboração, além dos desafios inerentes ao desenvolvimento de jogos digitais.

2.1.2 Aplicações e Domínios de Utilização

O aspecto de entretenimento sempre constituiu uma dimensão central dos jogos digitais desde seu surgimento. Atualmente, esses jogos estão presentes na vida de diversos grupos populacionais, especialmente entre os mais jovens, graças à sua acessibilidade por meio de múltiplas plataformas — como consoles, computadores e dispositivos móveis — e à variedade de gêneros disponíveis (Rideout *et al.*, 2010; Souza e Prates, 2022). Nos últimos anos, o mercado de jogos digitais experimentou crescimento exponencial, superando em faturamento a soma das indústrias cinematográfica e musical (Araújo *et al.*, 2018). Esse desenvolvimento acelerado não apenas consolidou economicamente o setor, mas também fomentou investimentos em pesquisas que exploram aplicações dos jogos digitais além do entretenimento. Investigações acadêmicas e iniciativas industriais têm buscado aproveitar o potencial dos jogos em domínios como educação, saúde, simulação e treinamento profissional (Araújo *et al.*, 2018), consolidando-os como tecnologia de impacto social, político, econômico e tecnológico (Newman, 2012; Melo *et al.*, 2020).

Dentre as diversas aplicações não-lúdicas, destacam-se os jogos sérios (*serious games*), conceito formalizado por David Rejeski e Ben Sawyer por volta de 2002 (Bellotti *et al.*, 2010). Estes são caracterizados pela priorização de objetivos educacionais ou formativos em detrimento do mero entretenimento, visando promover processos de aprendizagem durante a experiência interativa (Bellotti *et al.*, 2010; Mouaheb *et al.*, 2012). Os jogos sérios aplicam-se a múltiplos domínios, incluindo educação, formação profissional, saúde, políticas públicas e publicidade (Bellotti *et al.*, 2010; Mouaheb *et al.*, 2012). Suas tarefas são tipicamente caracterizadas pela especificidade e aplicabilidade direta a contextos laborais ou educacionais, sendo frequentemente direcionadas a públicos-alvo bem definidos (Bellotti *et al.*, 2010). Na área médica, por exemplo, jogos têm sido utilizados como ferramentas complementares para reabilitação física e tratamento de condições mentais (Baker, 2014). Em contextos técnicos e corporativos, servem como ambientes simulados para treinamento e desenvolvimento de competências (Susi *et al.*, 2007; Laamarti *et al.*, 2014).

Um subconjunto particularmente relevante dos jogos sérios é os Jogos Educacionais Digitais (JEDs), que integram explicitamente finalidades pedagógicas à experiência lúdica. Os JEDs promovem simultaneamente o entretenimento e o aprimoramento do processo de

ensino-aprendizagem (Lautert *et al.*, 2023), contribuindo para a retenção de informações, estímulo à autonomia, desenvolvimento da criatividade, seguimento de instruções e prática da tomada de decisões (Sampaio e C. P. Pereira, 2022). Esses jogos revolucionam o ambiente educacional ao engajar os alunos enquanto facilitam a aquisição de conhecimento, apoiando o desenvolvimento cognitivo tanto no aspecto lógico quanto no crítico (Dutra *et al.*, 2021; Mayer *et al.*, 2022; D. Silva *et al.*, 2022). Visando essa importância, observa-se um crescimento significativo nas pesquisas dedicadas à integração sistemática entre ensino e diversão por meio do desenvolvimento de JEDs (Genesio *et al.*, 2024).

Apesar dessa vasta gama de aplicações possíveis de jogos digitais, os custos elevados no processo de desenvolvimento causam que, muitas vezes, os desenvolvedores sejam levados a desistir do projeto durante o seu desenvolvimento (Araújo *et al.*, 2018). Nesse contexto, a ideia de *open-source* pode surgir como uma das soluções, pelo seu caráter de colaboração de uma ampla comunidade de desenvolvedores e pela reutilização de códigos já implementados para reduzir o custo de reprodução do artefato (Scacchi, 2004; Ahmed *et al.*, 2017). Além disso, o uso de uma arquitetura de referência, a qual possa fornecer um conjunto de diretrizes que facilitam a criação, manutenção e evolução dos projetos, diminuindo os custos posteriores, também pode ser aplicado nesse contexto (Elisa Y Nakagawa *et al.*, 2014).

2.2 *Open Source*

O conceito de *open source*, ou código aberto em português, refere-se a uma filosofia de desenvolvimento de software na qual o código-fonte é disponibilizado publicamente, permitindo que terceiros visualizem, modifiquem e redistribuam o software de acordo com os termos estabelecidos pela licença aplicada ao projeto (Perens *et al.*, 1999; Wu e Lin, 2001; Scacchi, 2004). As raízes do movimento *open source* remontam ao ambiente colaborativo do Laboratório de Inteligência Artificial do MIT na década de 1970, onde programadores compartilhavam software livremente em seu ambiente de trabalho (Bretthauer, 2001). Sua formalização conceitual ocorreu na década de 1980, impulsionada pela iniciativa de Richard Stallman por meio do Projeto GNU (Perens *et al.*, 1999; Bretthauer, 2001). O termo "*Open Source*"surgiu posteriormente, em 1997, com o objetivo de superar ambiguidades associadas à expressão "*Free Software*"e destacar os benefícios práticos e a eficácia do modelo de desenvolvimento, enfatizando aspectos técnicos e comerciais em detrimento de conotações meramente gratuitas (Perens *et al.*, 1999; Bretthauer, 2001; Ferreira, 2005).

Em contraste com modelos tradicionais de desenvolvimento, nos quais o código é mantido sob restrições de propriedade intelectual e acesso mediante custo (Perens *et al.*, 1999; Wu e Lin, 2001; Bretthauer, 2001), o paradigma de código aberto enfatiza transparência, colaboração descentralizada e inovação coletiva ao longo do processo de desenvolvimento (Wu e Lin, 2001; Bretthauer, 2001; Scacchi, 2004). O software de código aberto é protegido por direitos autorais e distribuído sob licenças especificamente concebidas para assegurar que o código-fonte permaneça acessível (Bretthauer, 2001). Essa abordagem ressalta que o *open source* não se restringe a uma metodologia de desenvolvimento, mas constitui uma garantia de liberdades fundamentais ao usuário final (Perens *et al.*, 1999).

Os princípios do *open source* podem ser sintetizados em quatro liberdades fundamentais

(Perens *et al.*, 1999; Bretthauer, 2001). Tais liberdades são materializadas por meio de critérios específicos estabelecidos pela *Open Source Definition*, que servem como parâmetro para certificar se uma licença de software é genuinamente aberta (Perens *et al.*, 1999):

- Liberdade de executar o programa para qualquer finalidade;
- Liberdade de estudar e modificar o programa, pressupondo acesso ao código-fonte;
- Liberdade de redistribuir cópias, gratuitamente ou mediante taxa;
- Liberdade de distribuir versões modificadas, permitindo que a comunidade se beneficie de melhorias.

Além dos direitos garantidos pelas licenças, projetos de código aberto maduros e sustentáveis compartilham normas culturais e organizacionais características, essenciais para o êxito do desenvolvimento (E. S. Raymond, 1998; Bretthauer, 2001). Dentre essas normas, destacam-se:

- **Governança centralizada**: Embora a licença permita contribuições amplas, a manutenção do código é geralmente coordenada por um indivíduo ou grupo reduzido, responsável por incorporar correções e funcionalidades propostas pela comunidade;
- **Canais de comunicação**: Utilização de listas de discussão especializadas para desenvolvedores, destinadas a debates técnicos e análise de *patches*, e canais voltados a usuários para relato de problemas e suporte;
- **Infraestrutura de divulgação**: Emprego de sites dedicados e plataformas especializadas para anunciar novas versões do software;
- **Documentação ampliada**: Elaboração de documentação integrada ao software, complementada frequentemente por materiais publicados comercialmente em projetos consolidados.

2.2.1 Definição de Licenças Open Source

Para garantir a ideologia central desse movimento *open source*, de focar nos aspectos técnicos e de desenvolvimento colaborativo visando maior adoção por empresas e desenvolvedores, a *Open Source Initiative* (OSI) estabeleceu a *Open Source Definition* (OSD), um conjunto de critérios que uma licença de software deve atender para ser considerada *open source* (Perens *et al.*, 1999; Sabino e Kon, 2009). Conforme Perens *et al.*, esta definição serve como uma "carta de direitos" para o usuário de computador, garantindo liberdades essenciais.

A OSD é derivada das *Debian Free Software Guidelines* e foi adaptada para remover referências específicas ao Debian, tornando-se um padrão amplamente aceito (Perens *et al.*, 1999). Baseando-se na análise dos documentos fornecidos, os critérios da OSD incluem:

1. **Redistribuição Livre:** A licença não pode restringir a venda ou doação do software como componente de uma distribuição agregada contendo programas de diversas fontes. Não pode exigir *royalties* ou taxas adicionais para tal redistribuição.

2. **Código-Fonte:** O programa deve incluir o código-fonte, permitindo distribuição tanto na forma-fonte quanto compilada. Se o código-fonte não for distribuído, deve haver um meio amplamente divulgado de obtê-lo por um custo de reprodução razoável, preferencialmente via *download* gratuito pela Internet. O código-fonte deve ser a forma preferencial para modificações, sendo proibidas formas deliberadamente ofuscadas ou intermediárias (como saída de pré-processadores).

3. **Trabalhos Derivados:** A licença deve permitir modificações e obras derivadas, e deve permitir que sejam distribuídas sob os mesmos termos da licença do software original. Isto é fundamental para a evolução e adaptação do software.

4. **Integridade do Código-Fonte do Autor:** A licença pode restringir a distribuição do código-fonte modificado apenas se permitir a distribuição de "arquivos de *patch*" para modificar o programa durante a compilação. Deve permitir também explicitamente a distribuição de software compilado a partir de código modificado. Podendo exigir que trabalhos derivados usem um nome ou número de versão diferente do original.

5. **Sem Discriminação a Pessoas ou Grupos:** A licença não pode discriminar qualquer pessoa ou grupo de pessoas, garantindo igualdade de acesso.

6. **Sem Discriminação a Áreas de Empreendimento:** A licença não pode restringir o uso do programa em qualquer campo de atuação, seja comercial, de pesquisa ou outro.

7. **Distribuição da Licença:** Os direitos associados ao programa devem aplicar-se a todos os destinatários da redistribuição, sem necessidade de aceitação de licença adicional.

8. **A Licença Não Deve Ser Específica a um Produto:** Os direitos associados ao programa não devem depender dele ser parte de uma distribuição específica de software. Caso o programa seja extraído dessa distribuição e usado ou distribuído nos termos da licença do programa, todas as partes para as quais o programa é redistribuído devem ter os mesmos direitos que aqueles concedidos em conjunto com a distribuição de software original.

9. **A Licença Não Deve Restringir Outro Software:** A licença não pode impor restrições a outros softwares distribuídos junto com o software licenciado. Por exemplo, não pode exigir que todos os outros programas no mesmo meio sejam *open source*.

10. **A Licença Deve Ser Neutra às Tecnologias:** Nenhuma condição da licença deve ser estabelecida em uma tecnologia individual específica ou estilo de interface.

Com a popularização do movimento *open source*, diversas licenças foram criadas, cada uma com particularidades. Conforme a OSD se consolidou, a OSI passou a aprovar licenças que a cumprem, mantendo uma lista de mais de 70 licenças aprovadas (Sabino e Kon, 2009). A proliferação de licenças é um desafio, pois dificulta a compatibilidade e compreensão pelos usuários (Sabino e Kon, 2009). No entanto, é possível classificá-las em três categorias principais, conforme suas restrições sobre obras derivadas (Engelfriet, 2009; Sabino e Kon, 2009):

1. **Licenças Permissivas:** Conhecidas também como licenças acadêmicas, impõem o mínimo de restrições, permitindo que obras derivadas sejam distribuídas como software não livre. Exemplos incluem as licenças BSD, MIT e Apache. São ideais para maximizar a adoção e disseminação do software. Permitem integração com software proprietário, facilitando a adoção comercial.
2. **Licenças Recíprocas Parciais (Copyleft Fraco):** Exigem que modificações no software original sejam disponibilizadas sob a mesma licença, mas permitem que o software seja usado como componente em obras maiores sob licenças diferentes. Exemplos são a LGPL (*GNU Lesser General Public License*) e a MPL (*Mozilla Public License*). Equilibram a atração de contribuições com a permissão de uso em software proprietário.
3. **Licenças Recíprocas Totais (Copyleft Forte):** Determinam que qualquer obra derivada, mesmo utilizando apenas parte do código, deve ser distribuída sob a mesma licença. A GPL (*GNU General Public License*) é o exemplo mais conhecido. Visam fortalecer a comunidade de software livre, garantindo que melhorias permaneçam acessíveis a todos.

A escolha da licença impacta diretamente o crescimento da comunidade em torno do projeto. Licenças permissivas favorecem a adoção ampla, enquanto licenças recíprocas totais incentivam contribuições e impedem a apropriação privativa de melhorias (Engelfriet, 2009). Um equilíbrio entre aceitação e contribuições é crucial para o sucesso de um projeto *open source*. Além disso, a compatibilidade entre licenças é um fator crítico ao combinar componentes de diferentes origens em um mesmo projeto. Licenças permissivas são geralmente compatíveis com recíprocas, mas o inverso nem sempre é verdadeiro (Sabino e Kon, 2009). Portanto, a escolha da licença deve considerar não apenas os objetivos do projeto, mas também as licenças de software de terceiros que serão utilizadas.

2.2.2 Vantagens e Desvantagens

O desenvolvimento de software *open source* (OSS) tem ganhado destaque significativo devido ao seu modelo colaborativo e transparente. Entretanto, apesar dos benefícios trazidos pela contribuição de uma ampla comunidade de desenvolvedores, a participação de múltiplos colaboradores também apresenta diversas desvantagens, exigindo mecanismos para mitigar os efeitos negativos inerentes a esse método. (Scacchi, 2004; Ferreira, 2005; Sabino e Kon, 2009; Steinmacher *et al.*, 2015).

Entre as principais vantagens associadas ao OSS, destacam-se:

- **Custo:** A ausência de licenças proprietárias representa uma vantagem econômica significativa, permitindo a redução de custos iniciais e facilitando o acesso a tecnologias (Garcia *et al.*, 2010).
- **Liberdade e Customização:** O acesso ao código-fonte possibilita que os usuários adaptem o software às suas necessidades específicas, promovendo maior flexibilidade e independência de fornecedores (Scacchi, 2004; Ferreira, 2005; Garcia *et al.*, 2010).
- **Inovação Colaborativa:** O modelo de desenvolvimento colaborativo permite que comunidades globais de desenvolvedores trabalhem em projetos, incorporando

perspectivas diversas e facilitando a inovação contínua por meio de melhorias incrementais com ciclos de vida curtos (Scacchi, 2004; Sabino e Kon, 2009).

- **Segurança e Qualidade:** disponibilidade pública do código possibilita que uma ampla comunidade identifique e corrija vulnerabilidades rapidamente, resultando em software mais seguro e estável (Ferreira, 2005; Sabino e Kon, 2009; Garcia *et al.*, 2010).

Apesar das vantagens, o modelo OSS enfrenta desafios significativos:

- **Barreiras para Novos Contribuidores:** Constitui-se de cinco categorias principais: dificuldades de interação social, devido à falta de resposta ou comunicação inadequada com membros da comunidade; exigência de conhecimento prévio do domínio e das técnicas utilizadas no projeto; dificuldade em encontrar tarefas iniciais e mentores para orientar o processo; problemas relacionados ao excesso, desatualização ou falta de clareza na documentação do projeto; e, por fim, obstáculos técnicos decorrentes da complexidade arquitetural e do código (Steinmacher *et al.*, 2015).
- **Suporte e Manutenção:** A ausência de suporte formalizado pode representar um obstáculo, especialmente para usuários menos técnicos (Garcia *et al.*, 2010). A carência de documentação adequada e a dificuldade em encontrar profissionais qualificados exacerbam esse problema (Steinmacher *et al.*, 2015).
- **Coordenação em Comunidades Distribuídas:** A natureza descentralizada do desenvolvimento introduz desafios de coordenação, nos quais contribuidores podem independentemente propor atualizações que se sobrepõem, conflitam ou geram efeitos colaterais indesejados (Scacchi, 2004; Fogel, 2005).

Como solução para mitigar os efeitos negativos, é comum a utilização de ferramentas de controle de versão nos projetos. Nos últimos anos, plataformas como GitHub, GitLab (sucessor do Gitorious) e Launchpad ganharam destaque por oferecer funcionalidades que facilitam a colaboração entre desenvolvedores, atraindo um grande número de contribuições para softwares livres (Meirelles, 2013). Ademais, é possível empregar arquiteturas de referência para orientar o desenvolvimento, dada sua capacidade de facilitar a comunicação entre a equipe, reduzir a complexidade arquitetural em comparação com processos *ad hoc* e aprimorar a documentação do projeto (Cloutier *et al.*, 2010; Elisa Y Nakagawa *et al.*, 2014).

2.3 Arquitetura de Referência

O processo de desenvolvimento de sistemas de software atual diferencia-se das práticas do passado devido aos avanços tecnológicos e às demandas observadas no mercado e na academia (Kiel, 2003; Lopes *et al.*, 2005; Bassani, 2022). Os sistemas atuais não apenas se tornaram mais complexos, incorporando mais funcionalidades e componentes, mas também frequentemente envolvem equipes de desenvolvimento distribuídas globalmente, o que introduz desafios de cooperação devido a barreiras de tempo, linguagem e cultura (Kiel, 2003; Prikladnicki e Audy, 2006). Nesse contexto, surge a motivação para a adoção de abordagens mais sistemáticas e eficientes para a estruturação e o desenvolvimento desses sistemas (Petri *et al.*, 2019).

A Arquitetura de Software (SA) constitui um pilar fundamental no desenvolvimento de sistemas, sendo responsável pela sua organização, estruturação e garantia de atributos de qualidade (Elisa Y Nakagawa *et al.*, 2014). Ela abrange metodologias de desenvolvimento, o uso de modelos, arquiteturas de referência, padrões de projeto, estilos e táticas arquiteturais (Bass *et al.*, 2021). O termo "Arquitetura de Software"surgiu pela primeira vez em 1969, mas foi a partir do início da década de 1990 que emergiu como uma disciplina distinta, com contribuições significativas da indústria e da academia (P. Kruchten *et al.*, 2006; Boehm, 2006). Segundo Bass *et al.*, a arquitetura de software de um sistema é a estrutura ou estruturas que compreendem elementos de software, as propriedades externamente visíveis desses elementos e os relacionamentos entre eles. O *Software Engineering Institute* enfatiza a forte correlação entre os requisitos de qualidade de um sistema e sua arquitetura, posicionando-a como um elemento essencial para a qualidade do software (Software Engineering Institute, 2025).

Nesse contexto, as Arquiteturas de Referência (RAs) surgem como um importante desdobramento da SA, fornecendo diretrizes especializadas para a especificação de arquiteturas concretas em um domínio de aplicação específico (Angelov, Trienekens *et al.*, 2008; Cloutier *et al.*, 2010; Elisa Y Nakagawa *et al.*, 2014). Elas exercem influência direta na qualidade e no *design* de um conjunto de arquiteturas concretas, bem como nos sistemas delas derivados (Angelov, Grefen *et al.*, 2009). Uma RA pode ser entendida como um mapeamento entre as funcionalidades, descritas em um Modelo de Referência, e os elementos de software que as implementam cooperativamente, incluindo os fluxos de dados entre eles (Bass *et al.*, 2021). Adicionalmente, uma RA aborda regras de negócio, estilos arquiteturais e as melhores práticas para o desenvolvimento de software, encapsulando decisões arquiteturais, restrições de domínio, aspectos legais, padrões e elementos de software que apoiam o desenvolvimento de sistemas para esse domínio (Elisa Y Nakagawa *et al.*, 2014). Seu principal objetivo é facilitar o entendimento de um domínio, fornecendo um vocabulário comum e esclarecendo os componentes que o compõem e as relações entre eles (Muller, 2008). Ademais, as RAs promovem a reutilização de conhecimento sobre o desenvolvimento de software em um domínio, especialmente no que tange ao projeto arquitetural (Cloutier *et al.*, 2010; Elisa Y Nakagawa *et al.*, 2014).

Como vantagem, as Arquiteturas de Referência podem ser utilizadas para fornecer (Muller, 2008): um léxico e uma taxonomia comuns, que facilitam a comunicação entre os *stakeholders* do projeto; uma visão arquitetural comum, que coordena o trabalho de diversas pessoas e equipes envolvidas; e a modularização e o contexto complementar, que auxiliam na divisão e na integração dos trabalhos subsequentes. Vale ressaltar que as arquiteturas de referência evitam a reinvenção e a revalidação de soluções para problemas recorrentes, otimizando assim o esforço de desenvolvimento.

2.3.1 Terminologia e Conceitos Básicos

Para estabelecer um vocabulário comum na área de Arquitetura de Software, são utilizados os seguintes conceitos fundamentais:

- **Arquitetura Concreta**: Sinônimo de SA, refere-se a um conjunto coerente de padrões e especificações que orientam o desenvolvimento de cada componente de um sistema de software real (Software Engineering Institute, 2025). Ela aborda

funcionalidades e qualidades relacionadas ao sistema, aos negócios e à própria arquitetura, conforme exigências dos *stakeholders* (Angelov, Trienekens *et al.*, 2008).

- **Instância Arquitetural**: Termo frequentemente utilizado como sinônimo de Arquitetura Concreta, especialmente em contextos de aplicação ou problemas específicos, derivados da utilização de uma RA (Elisa Yumi Nakagawa, 2006).
- **Tática Arquitetural**: Corresponde a uma decisão de *design* que impacta a obtenção de atributos de qualidade específicos em um sistema. As táticas oferecem métodos comprovados para melhorar ou gerenciar esses atributos, alinhando as decisões de projeto ao comportamento desejado do sistema (Bass *et al.*, 2021).
- **Estilo Arquitetural**: É uma especialização de tipos de elementos e relações, acompanhada de um conjunto de restrições sobre seu uso (Bass *et al.*, 2021). Permite a aplicação de conhecimento especializado a uma classe de sistemas, apoiando-a com ferramentas e análises específicas.
- **Padrão Arquitetural**: Descreve um problema de *design* recorrente em contextos específicos e apresenta soluções de arquitetura bem comprovadas (Frank *et al.*, 1996). É uma descrição abstrata e estilizada de boas práticas testadas em diversos sistemas (Sommerville, 2011). A diferença entre padrão e estilo reside no foco: o padrão sugere um *design* baseado no problema e no contexto, enquanto o estilo se concentra no *design* com orientação mais leve sobre sua aplicabilidade (Medvidovic e Taylor, 2010).
- **Modelo de Referência**: Refere-se à divisão de funcionalidades e ao fluxo de dados entre suas partes, sendo independente de tecnologias ou implementações específicas (Bass *et al.*, 2021; MacKenzie *et al.*, 2006). Esses modelos, que podem variar de modelos conceituais a ontologias estabelecidas, auxiliam na criação de RAs ao mapear conceitos e funcionalidades para entidades de SA (De Oliveira *et al.*, 2010; Bass *et al.*, 2021).
- **Arquitetura de Referência (RA)**: Consiste no mapeamento entre um Modelo de Referência e elementos de software que implementam as funcionalidades descritas no modelo. A RA serve como base para o projeto de Arquiteturas Concretas destinadas à construção de sistemas específicos (Cloutier *et al.*, 2010; Elisa Y Nakagawa *et al.*, 2014).

A relação entre esses conceitos é ilustrada na Figura 2.1, que mostra como o Modelo de Referência e os Padrões Arquiteturais fundamentam a definição de uma RA, que por sua vez orienta a elaboração de Arquiteturas Concretas.

2.3.2 Representações de Arquiteturas de Software

A representação adequada de uma arquitetura de software é crucial para o sucesso do desenvolvimento de sistemas (Farshidi *et al.*, 2020). Inicialmente, as SA são frequentemente representadas de forma informal por meio de diagramas do tipo *box-and-line*, nos quais os componentes representam as funcionalidades e dados do sistema, enquanto os conectores indicam a comunicação e as interações entre esses componentes (Land, 2002; Elisa Yumi

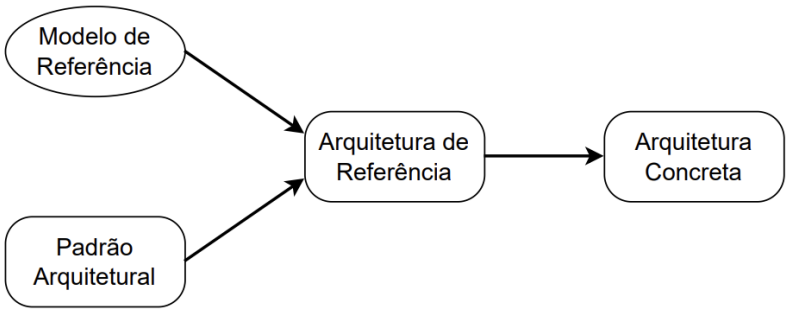

Figura 2.1: *Relação entre Modelo de Referência, Padrão Arquitetural, Arquitetura de Referência e Concreta (Adaptado de Bass et al., 2021)*

Nakagawa e Antonino, 2022). No entanto, essa abordagem, por carecer de uma linguagem formal para definição dos elementos e depender da experiência e do conhecimento do leitor para a interpretação, o que varia entre indivíduos, pode levar a interpretações ambíguas e inconsistentes do diagrama construído (Land, 2002).

Com o objetivo de superar essas limitações e permitir descrições mais precisas das SA, foram propostas diversas Linguagens de Descrição Arquitetural (ADLs), que oferecem abstrações adequadas e uma sintaxe formal para modelar sistemas complexos, assegurando detalhes suficientes para definir as propriedades desejadas do projeto (Garlan, Allen *et al.*, 1994; Medvidovic, Rosenblum e Taylor, 1999; Moriconi *et al.*, 2002; Elisa Yumi Nakagawa e Antonino, 2022). Cada ADL adota uma abordagem específica para a especificação e evolução de uma arquitetura, abordando questões de avaliação relacionadas à modelagem e explorando aspectos detalhados do sistema (Medvidovic, Rosenblum, Redmiles *et al.*, 2002). Dentre as abordagens semi-formais, a *Unified Modeling Language* (UML) destaca-se pela ampla adoção na representação de SA, devido à sua familiaridade entre desenvolvedores, facilidade de mapeamento para implementação e suporte consolidado de ferramentas especializadas (Medvidovic, Rosenblum, Redmiles *et al.*, 2002; Júnior *et al.*, 2022). Entretanto, embora as ADLs formais ofereçam maior rigor, sua natureza específica torna desafiadora a integração de modelos com outros artefatos de desenvolvimento (Medvidovic, Rosenblum, Redmiles *et al.*, 2002).

Alternativamente, a adoção de múltiplas visões para representar SA tem sido amplamente investigada, em que cada visão reflete um aspecto específico da arquitetura, correspondendo a diferentes propriedades e preocupações do sistema (P. B. Kruchten, 2002; Medvidovic, Rosenblum, Redmiles *et al.*, 2002). Por meio dessas visões, é possível observar diferentes características do sistema, dependendo do ponto de vista adotado na análise da arquitetura (Land, 2002). Dentre os modelos baseados em visões, destaca-se o **Modelo de Visualização 4+1**, proposto por P. B. Kruchten, que organiza a descrição da SA em cinco visões complementares, cada uma elaborada a partir de diferentes perspectivas e focada em um conjunto específico de preocupações. A Figura 2.2 ilustra esse modelo, demonstrando como o sistema pode ser representado sob diversas perspectivas, atendendo a diferentes *stakeholders* como usuários finais, desenvolvedores, integradores de sistemas e engenheiros.

- **Visão Lógica**: Foca nos requisitos funcionais do sistema, representando os serviços

e funcionalidades que devem ser fornecidos aos usuários finais.

- **Visão de Processo**: Aborda aspectos de concorrência, sincronização e desempenho, tratando de requisitos não funcionais como disponibilidade e tempos de resposta. Define os fluxos de execução (*threads*) e processos que realizam as operações do sistema.
- **Visão de Desenvolvimento**: Também conhecida como visão de implementação, trata da organização dos módulos de software no ambiente de desenvolvimento, incluindo a estruturação em camadas, subsistemas, bibliotecas e dependências entre componentes.
- **Visão Física**: Considera os requisitos de implantação do sistema, como confiabilidade, escalabilidade e disponibilidade. Aborda o mapeamento do software em nós físicos de hardware e infraestrutura de rede, relevante para aspectos de distribuição e deployment.
- **Visão de Cenários**: Conhecida como a visão "+1", serve como elemento unificador que ilustra e valida as outras quatro visões por meio de casos de uso e cenários específicos. Mostra como o sistema atende a diversos interesses dos *stakeholders* através de exemplos concretos de funcionamento.

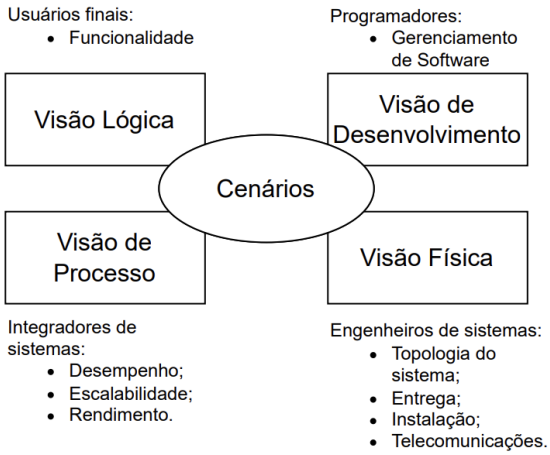

Figura 2.2: *Representação do Modelo de Visualização 4+1 (Adaptado de P. B. Kruchten, 2002)*

2.3.3 Processo para Estabelecer uma Arquitetura de Referência

A sistematização do processo para estabelecer Arquiteturas de Referência constitui uma atividade de grande relevância para garantir o êxito tanto de sua construção quanto de sua posterior utilização (Elisa Yumi Nakagawa, Martins *et al.*, 2009; Elisa Y Nakagawa *et al.*, 2014). Para apoiar a construção sistemática de arquiteturas de referência, Elisa Y Nakagawa *et al.* propuseram o processo ProSA-RA (*Process-based System Architecture Reference Model*), que estabelece um fluxo metodológico para o desenvolvimento destas arquiteturas. Este processo é composto por quatro etapas principais, conforme ilustrado na Figura 2.3, sendo elas:

1. **Passo RA-1: Investigação das Fontes de Informação:** Esta etapa envolve a

seleção das principais fontes de informação para aquisição de conhecimento sobre o domínio da arquitetura. Considerando que as arquiteturas de referência devem servir como base para arquiteturas de software concretas, essas fontes devem abranger o conhecimento mais abrangente possível sobre o domínio. Entre as fontes mais relevantes destacam-se especialistas da área, publicações científicas e arquiteturas de software já existentes relacionadas ao domínio de aplicação.

2. **Passo RA-2: Análise Arquitetural:** A partir das fontes de informação selecionadas, são identificados três conjuntos de elementos fundamentais. Primeiramente, identifica-se o conjunto de requisitos dos sistemas de software do domínio; com base nestes requisitos, descreve-se o conjunto de requisitos específicos da arquitetura de referência; finalmente, incluem-se os *concerns* arquiteturais que devem ser abordados na arquitetura de referência.

3. **Passo RA-3: Síntese Arquitetural:** Com base nos requisitos arquiteturais e conceitos identificados, elabora-se a descrição arquitetural da RA, considerando padrões e estilos arquiteturais apropriados. A descrição é construída utilizando múltiplas visões arquiteturais, tais como visões de módulos, tempo de execução, implantação e conceitual, resultando em um conjunto integrado de visões e artefatos complementares.

4. **Passo RA-4: Avaliação Arquitetural:** Na etapa final, a descrição arquitetural é submetida à verificação pelos *stakeholders* com o objetivo de identificar possíveis defeitos na arquitetura. Uma alternativa metodológica é utilizar o framework FERA (*Framework for Evaluation of Reference Architectures*) (Santos *et al.*, 2013), que avalia a completude das informações da arquitetura, a adequação da documentação e a viabilidade de instanciação da arquitetura. Os resultados obtidos são então utilizados para refinamentos iterativos na etapa de síntese arquitetural.

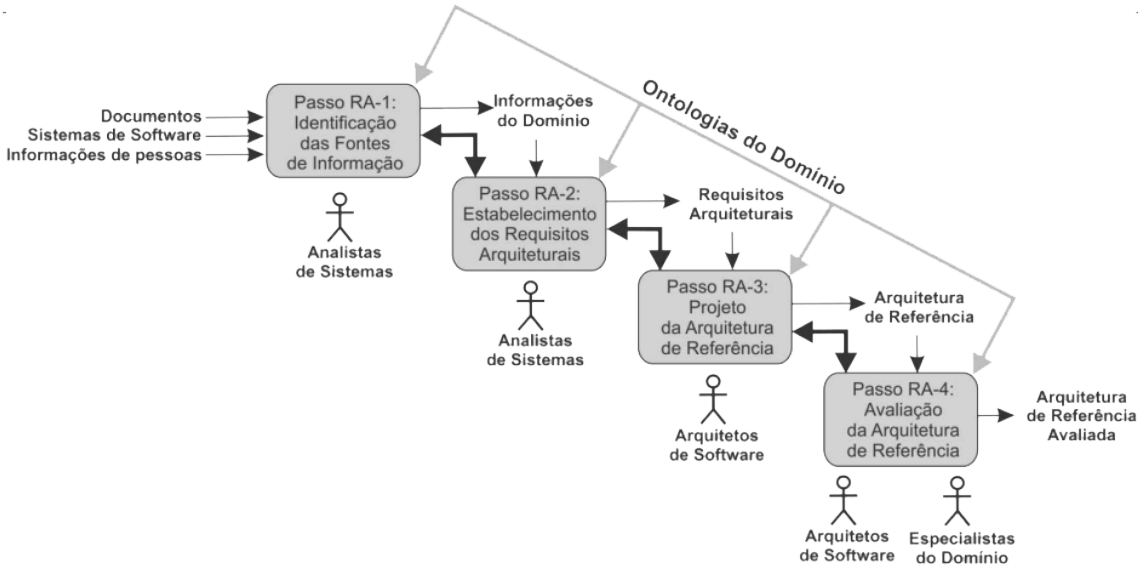

Figura 2.3: *Estrutura do processo ProSA-RA (Adaptado de Elisa Y Nakagawa et al., 2014)*

Os quatro passos definidos no ProSA-RA são sustentados pelo RAModel (*Reference*

Architecture Model), um modelo de referência para RAs que fornece informações sobre todos os possíveis elementos e seus relacionamentos que poderiam estar presentes em RAs, independentemente do domínio de aplicação ou do propósito dessas arquiteturas (Elisa Yumi Nakagawa, Oquendo *et al.*, 2012). Em resumo, o RAModel é constituído por quatro grupos de elementos fundamentais:

- **Domínio**: Elementos relacionados ao contexto de atuação humana, como legislações, padrões e processos de certificação do domínio, que influenciam sistemas e RAs associados.
- **Aplicação**: Elementos que fornecem entendimento das capacidades e limitações da RA, incluindo regras de negócio e funcionalidades dos sistemas.
- **Infraestrutura**: Elementos utilizados para construir sistemas de software baseados na RA, automatizando processos, atividades e tarefas do domínio.
- **Elementos Transversais**: Elementos distribuídos e interligados com os demais grupos, como comunicação, terminologia e decisões de domínio.

2.4 Trabalhos Relacionados

2.4.1 openKONSEQUENZ (oK) (Goering *et al.*, 2017)

Goering *et al.* relata os desafios enfrentados no gerenciamento de redes de energia elétrica por parte dos operadores do setor. A complexidade dos sistemas tem crescido continuamente devido a fatores como pressões regulatórias, demandas comerciais e a transição para redes inteligentes. Adicionalmente, a utilização de softwares proprietários e a falta de padronização na troca de dados entre sistemas resultam em custos elevados de manutenção e evolução, bem como em longos ciclos de atualização que variam entre 5 e 15 anos para versões principais.

Diante desse cenário, o consórcio *openKONSEQUENZ* propôs uma arquitetura de referência com o objetivo de reduzir a complexidade do sistema e a dependência de fornecedores específicos, além de aumentar a qualidade, usabilidade, segurança e proteção em infraestruturas críticas. A arquitetura adota predominantemente uma abordagem orientada a serviços (*Service-oriented Architecture - SOA*) e foi desenvolvida utilizando processos ágeis baseados no framework *Scrum*.

A partir da arquitetura, é possível identificar requisitos fundamentais para software de código aberto nesta área, com ênfase em: interoperabilidade, manutenibilidade, confidencialidade, integridade e disponibilidade. A arquitetura é organizada em camadas, incluindo módulos de plataforma e módulos de usuário, que se comunicam através de um barramento de serviços empresariais (*Enterprise Service Bus - ESB*) e utilizam perfis padronizados baseados no *Common Information Model* (CIM) para garantir interoperabilidade semântica.

Por fim, para avaliação da arquitetura, foi estabelecido um processo de validação técnica que incluiu:

- Implementação de projetos-piloto (como o sistema de gestão de alimentação)

- Revisões sistemáticas por comitês de arquitetura e qualidade
- Definição de diversos níveis de conformidade ("oK-ready"até "full-oK")
- Adoção em ambientes reais de operação de distribuidoras de energia

A arquitetura demonstrou potencial para reduzir significativamente o *vendor lock-in* e promover um ecossistema sustentável de desenvolvimento colaborativo no domínio de sistemas de gestão de redes elétricas.

2.4.2 SORASG (Carvalho, 2017)

O estudo realizado por Carvalho aborda os desafios inerentes ao design e desenvolvimento de Jogos Sérios (JS) educacionais, que frequentemente envolvem processos complexos, custos elevados e dificuldades de comunicação entre equipes multidisciplinares. Para enfrentar esses problemas, a pesquisa propôs duas contribuições principais, consistindo em um modelo conceitual e uma solução de arquitetura de software.

A primeira contribuição é o Modelo Baseado na Teoria da Atividade para Jogos Sérios (ATMSG). Este modelo oferece uma estrutura para analisar e projetar JS, permitindo uma compreensão detalhada de como os elementos concretos do jogo se conectam e contribuem para a realização dos objetivos pedagógicos e de entretenimento. Baseado na Teoria da Atividade, o ATMSG considera a dinâmica entre três atividades principais: a de jogo, motivada pela diversão; a de aprendizagem, motivada pela aquisição de conhecimento; e a de instrução, realizada pelo professor ou pelo próprio jogo. O modelo é acompanhado por uma taxonomia abrangente que classifica centenas de componentes comuns de JS, como ações, ferramentas e metas, auxiliando na identificação e análise sistemática desses elementos.

A segunda contribuição é a Arquitetura de Referência Orientada a Serviços para Jogos Sérios (SORASG). Esta arquitetura fornece um template para o desenvolvimento de software de JS, baseado no padrão de Arquitetura Orientada a Serviços (SOA) e desenvolvido com método de design orientado a atributos (DDA). O objetivo central é promover a reutilização de componentes comuns a diferentes JS, tais como módulos de avaliação da aprendizagem, adaptação, perfil de usuário e análises de aprendizagem, que podem ser consumidos como serviços externos. Dessa forma, os desenvolvedores podem se concentrar nas características únicas do jogo, integrando funcionalidades especializadas desenvolvidas e mantidas por terceiros, o que potencialmente reduz custos e tempo de desenvolvimento. Adicionalmente, a SORASG incentiva o uso do padrão de especificação xAPI (Experience API) para representar e armazenar dados de interação do jogador, garantindo interoperabilidade.

A eficácia e a viabilidade da SORASG foram validadas por meio de uma avaliação rigorosa. Primeiramente, foi aplicado o Método de Análise de *Trade-off* de Arquitetura (ATAM), que avaliou a arquitetura contra atributos de qualidade críticos, como capacidade de desenvolvimento distribuído, modificabilidade, desempenho e escalabilidade. Com essa análise, foi possível identificar os possíveis riscos e pontos de trade-off dos requisitos, fornecendo um entendimento profundo das consequências das decisões de design. Em segundo lugar, foi conduzida uma implementação de prova de conceito, na qual o jogo de quebra-cabeça open-source "Lix"foi modificado para integrar serviços conforme proposto

pela SORASG. Este caso prático demonstrou a integração bem-sucedida de componentes reutilizáveis em um intervalo de tempo relativamente curto.

Em conclusão, a pesquisa demonstra que a combinação do modelo teórico ATMSG – que fornece o "o quê"e "porquê"do design de JS – com a arquitetura prática SORASG – que fornece o "como"da implementação – constitui uma metodologia robusta para otimizar o desenvolvimento de Jogos Sérios. Esta abordagem promove a criação de jogos educacionais de qualidade de forma mais eficiente, reduzindo barreiras de custo e complexidade.

2.4.3 URB (Wilson Kazuo Mizutani e Kon, 2020)

Wilson Kazuo Mizutani e Kon propuseram a *Unlimited Rulebook* (URB), uma arquitetura de referência voltada ao subsistema de mecânicas econômicas em jogos digitais. A URB visa reduzir os custos de desenvolvimento associados a tais mecânicas, que são frequentemente imprevisíveis, instáveis e complexas, limitando as oportunidades de reuso de software entre diferentes jogos. Como alternativa, a URB promove o reuso de conhecimento arquitetural, integrando técnicas consolidadas como *Predicate Dispatching*, o padrão *Entity-Component-System* (ECS) e o estilo arquitetural *Adaptive Object-Model* (AOM).

A arquitetura é organizada em torno de três conceitos de domínio derivados de requisitos arquiteturais:

- **Integração de Subsistemas:** Define interfaces claras que restringem o acesso externo ao estado interno da economia, promovendo os princípios de desacoplamento e encapsulamento.
- **Modelo de Mecânicas:** Separa os tipos de recursos e as operações do sistema econômico. É necessária a distinção entre mecânicas passíveis de armazenamento, como dados, e aquelas que são dinâmicas, exigindo cálculo em tempo de execução, o que é mapeado em partes distintas da arquitetura final do jogo.
- **Desenvolvimento Iterativo:** Facilitado pela natureza modular da arquitetura. Novas funcionalidades são encapsuladas em *RulePatches*, que podem ser carregadas dinamicamente, minimizando impactos em código pre-existente e suportando design orientado a dados.

O processo de desenvolvimento da URB seguiu a metodologia iterativa ProSA-RA. As informações de domínio foram coletadas por meio de: revisão sistemática da literatura; análise de literatura cinzenta, incluindo consultas a autores da indústria; estudo de jogos e motores de código aberto; e consulta a um grupo de interesse especializado com estudantes da Universidade de São Paulo (USPGameDev). Os requisitos foram identificados e sintetizados em requisitos arquiteturais gerais, com refinamento iterativo da arquitetura durante o ciclo de desenvolvimento.

A avaliação da arquitetura foi conduzida por meio de quase-experimentos com estudantes de programação de jogos, que desenvolveram dois jogos — um com *design* livre e outro utilizando a URB. Resultados preliminares indicaram que o uso da URB não tornou a implementação significativamente mais fácil ou difícil, mas reduziu ligeiramente a quantidade de modificações arquiteturais necessárias para estender as mecânicas. Adicionalmente, provas de conceito, como uma simulação baseada em *Magic: The Gathering*, demonstraram

a eficácia da arquitetura para lidar com mecânicas complexas e sobreposição de regras em tempo de execução, embora preocupações com desempenho e curva de aprendizado tenham sido identificadas.

2.4.4 ePHoRt (Perez-Medina *et al.*, 2019)

Perez-Medina *et al.* propõe no artigo a arquitetura de referência modular ePHoRt (*e-Platform for Home Rehabilitation*) para desenvolvimento de plataformas de telerreabilitação, com foco no acompanhamento remoto de pacientes em processo de reabilitação pós-cirurgia de substituição de quadril. A arquitetura foi concebida priorizando atributos de qualidade como segurança, flexibilidade e escalabilidade, além de incorporar princípios de reusabilidade de módulos para evitar redundância no trabalho de pesquisadores e profissionais da área.

A plataforma ePHoRt consiste em uma solução baseada na Web que integra uma fase de aprendizagem com esquemas de jogos sérios para execução e avaliação de exercícios como parte de programas terapêuticos personalizados. A representação arquitetural utiliza múltiplas perspectivas por meio de diagramas UML, incluindo diagramas de classes, sequência, atividades e pacotes, complementados por descrições textuais detalhadas de cada componente. O núcleo do sistema emprega o padrão arquitetural MVC (Model-View-Controller) implementado no framework Django, responsável pelo gerenciamento dos dados do paciente, interface de usuário e processamento das informações de reabilitação.

O processo de desenvolvimento adotou a abordagem de Desenvolvimento Ágil Centrado no Usuário (User-Centered Design - UCD), enquanto a avaliação incluiu validação experimental dos algoritmos de inteligência artificial para reconhecimento de movimentos (baseado em Dynamic Time Warping - DTW) e detecção de dor (utilizando Support Vector Machines - SVM), com resultados preliminares demonstrando 100% de acurácia na discriminação entre dor e ausência de dor. Entre as principais vantagens da arquitetura destacam-se:

- Modularidade e separação de concerns através de componentes especializados
- Capacidade de extensão para outros tipos de reabilitação motora
- Integração de avaliação motora e emocional em tempo real
- Arquitetura baseada em RESTful API para interoperabilidade

A arquitetura também apresenta limitações, incluindo a dependência inicial do sensor Kinect para captura de movimentos e a necessidade de incorporar modalidades adicionais de interação, como comandos vocais, para reduzir a fadiga do paciente durante sessões prolongadas. Adicionalmente, identificam-se desafios técnicos relacionados a oclusões de articulações durante exercícios e variações morfológicas entre pacientes.

Em síntese, a ePHoRt representa uma contribuição significativa para a área de telerreabilitação, oferecendo uma arquitetura de referência bem fundamentada que integra avaliação motora e emocional avançada. O trabalho demonstra potencial para reduzir custos de infraestrutura em saúde e ampliar o acesso a serviços de reabilitação de qualidade, servindo como base para desenvolvimentos futuros na área.

2.4.5 Análise Comparativa

A comparação entre as quatro arquiteturas de referência — oK, SORASG, URB e ePHoRt — evidencia diferentes abordagens no desenvolvimento de RA, cada uma com foco em domínios e justificativas específicas, conforme ilustrado na Tabela 2.1.

Tabela 2.1: *Comparação dos trabalhos relacionados*

Funcionalidade	**Arquitetura de Referência**			
	oK	SORASG	URB	ePHoRt
Domínio	*Software Open Source* em Redes de Energia	Jogos sérios	Jogos digitais	Jogos sérios em Telerreabilitação
Justificativa	Disponibilizada	Disponibilizada	Disponibilizada	Disponibilizada
Processo de Desenvolvimento	Scrum	DDA	ProSA-RA e Desenvolvimento Iterativo	UCD
Arquitetura	SOA, camada e ESB	SOA	ECS e AOM	MVC e modular
Fontes de Informações	Detalhado	Detalhado	Detalhado	Detalhado
Atributos de Qualidade	Detalhado	Detalhado	Não foi explicitado	Detalhado
Stakeholders	Detalhado	Detalhado	Não foram mencionados	Detalhado
Visões Arquiteturais	Detalhado	Detalhado	Detalhado	Detalhado
Avaliação	Realizado	Realizado	Realizado	Realizado
Nível de Maturidade	Uso em situação real	Uso em situação real	Uso em situação real	Uso em situação real

A RA oK, voltada para o domínio de *Software Open Source* em Redes de Energia, assim como a SORASG e a ePHoRt, que se concentram em jogos sérios, e a URB, orientada para jogos digitais, possuem justificativas claras para seu desenvolvimento, refletindo demandas específicas de seus contextos de aplicação. No que diz respeito ao processo de desenvolvimento, observa-se uma variedade de metodologias adotadas: a oK utiliza Scrum, a SORASG emprega a abordagem DDA, a URB combina ProSA-RA com desenvolvimento iterativo, e a ePHoRt adota o UCD. Essa diversidade de processos impacta diretamente as etapas e a organização da construção de cada RA.

Em termos de estilo e padrão arquitetural, a oK e a SORASG compartilham a adoção da Arquitetura Orientada a Serviços (SOA), que a oK utiliza adicionalmente uma arquitetura em camadas que se comunica usando o ESB, enquanto a URB utiliza uma combinação de ECS e AOM, e a ePHoRt segue os padrões MVC com modularidade. Quanto às fontes de

informação, todas as RAs analisadas apresentam um nível de detalhamento elevado, o que facilita a compreensão de suas estruturas e decisões. No entanto, os atributos de qualidade são explicitamente detalhados na oK, SORASG e ePHoRt, mas não são claramente definidos na URB. Da mesma forma, os *stakeholders* são identificados de forma detalhada em todas as RAs, exceto na URB, onde não são mencionados.

As visões arquiteturais são documentadas em todas as arquiteturas, incluindo a URB, o que indica a importância destas para favorecer a compreensão da arquitetura de referência. Em relação à avaliação, todas as RAs passaram por processos de avaliação, indicando um compromisso com a validação de suas propostas. Ademais, todas atingiram um nível de maturidade considerável, sendo utilizadas em situações reais ou possuindo software implementado utilizando a arquitetura, o que demonstra sua aplicabilidade e eficácia em seus respectivos domínios.

A análise comparativa demonstra que, embora as RAs compartilhem objetivos de reutilização, padronização e otimização, elas diferem significativamente em suas abordagens, estilos arquiteturais e níveis de explicitação de atributos de qualidade e stakeholders. Essa diversidade reflete a adaptação de cada RA às necessidades específicas de seus domínios. Contudo, nenhuma delas é direcionada especificamente para os desafios únicos dos jogos digitais *open-source*. A ausência de uma RA específica para jogos digitais *open-source*, aliada aos benefícios de aplicar essa ideologia e às dificuldades no desenvolvimento de jogos e de softwares de código aberto, evidencia a necessidade do desenvolvimento de uma solução específica para esse domínio. Dessa forma, o desenvolvimento de uma RA é essencial para suprir essa lacuna, possibilitando a criação de jogos digitais *open-source* de alta qualidade e favorecendo o reuso de componentes de software.

2.5 Considerações Finais

Neste capítulo, foram pesquisados e detalhados os principais conhecimentos relacionados às áreas de jogos digitais, desenvolvimento *open source* e arquitetura de referência, fornecendo uma base sólida para a condução do trabalho subsequente. Adicionalmente, foram discutidas múltiplas arquiteturas de referência propostas em domínios relacionados, as quais servirão como fundamento e exemplo para a arquitetura a ser construída.

O crescimento do setor de jogos digitais, acompanhado pelo aumento da complexidade de seu desenvolvimento — que demanda conhecimentos especializados e trabalho colaborativo entre múltiplas disciplinas — evidencia a necessidade de uma abordagem sistemática para garantir a qualidade do produto final. Especialmente no contexto *open source*, embora essa prática ofereça vantagens como a customização do produto e a colaboração de uma comunidade global de desenvolvedores, ela também impõe desafios, tais como a necessidade de facilitar a integração de novos colaboradores e de coordenar esforços de desenvolvimento distribuídos. Diante desse cenário, a adoção de uma arquitetura de referência mostra-se uma estratégia capaz de regulamentar e auxiliar esse processo, assegurando a qualidade do jogo desenvolvido e facilitando a colaboração entre os diversos envolvidos.

A análise de arquiteturas de referência consolidadas, como SORASG, URB e ePHoRt, demonstra a relevância e a eficácia desse tipo de artefato para domínios de software *open-source* e jogos digitais, promovendo o desenvolvimento modular e o reúso de conhecimento

implementado. No entanto, mesmo com a comprovada eficácia dessas arquiteturas em seus respectivos domínios, identifica-se uma lacuna específica: a inexistência de uma arquitetura de referência dedicada ao desenvolvimento de jogos digitais *open-source*.

Portanto, a ausência de uma arquitetura de referência específica para esse domínio, aliada à capacidade reconhecida desses artefatos de promover a qualidade do software, evidencia a necessidade e a oportunidade de se propor uma nova arquitetura de referência para esta área de pesquisa em ascensão. Com a sua elaboração, será possível viabilizar o desenvolvimento de jogos digitais *open-source* de alta qualidade, garantindo o atendimento a requisitos essenciais. Ademais, a arquitetura fomentará o compartilhamento e o reúso de componentes, favorecendo o crescimento da comunidade de desenvolvimento. Por fim, ela poderá estabelecer uma terminologia comum e visões arquiteturais padronizadas, facilitando a comunicação e a colaboração entre equipes de desenvolvimento distribuídas.

Capítulo 3

Mapeamento Sistemático

Um Estudo de Mapeamento Sistemático (*Systematic Mapping Study* – SMS) é um método de revisão secundária da literatura que visa fornecer uma visão ampla e estruturada de uma área de pesquisa, identificando, analisando e classificando estudos publicados com base em um protocolo bem definido, a fim de evitar vieses dos pesquisadores (Kitchenham e Charters, 2007; Zavala *et al.*, 2019). Esse tipo de estudo é particularmente útil em áreas emergentes ou de amplo escopo, nas quais se faz necessário mapear o estado da arte, identificar tendências, lacunas de pesquisa e agrupar evidências de forma visual e quantitativa, destacando práticas e considerações comuns, bem como fornecer direções e subsídios para pesquisas futuras (Kitchenham e Charters, 2007; Petersen *et al.*, 2008).

O processo é composto por três etapas principais (Kitchenham e Charters, 2007; Petersen *et al.*, 2008; Vargas *et al.*, 2014; Zavala *et al.*, 2019; Gezici *et al.*, 2019):

- **Planejamento da revisão:** Envolve a identificação da necessidade da revisão, a definição das perguntas de pesquisa a serem investigadas, a elaboração do protocolo para seleção de estudos primários e extração de dados relevantes, além da especificação dos instrumentos de visualização e das ameaças à validade do estudo;

- **Condução da revisão:** O processo definido no planejamento é implementado. Envolve a busca de estudos relevantes por meio de *strings* de busca, a seleção de estudos com base nos critérios de inclusão e exclusão definidos e, por fim, a extração, classificação e síntese dos dados presentes nos estudos. É importante ressaltar que esta etapa é realizada de modo iterativo, havendo revisão dos resultados, o que pode exigir ajustes dos dados obtidos;

- **Relato do mapeamento:** Inclui a estruturação do relatório, a definição dos mecanismos de divulgação e a avaliação do processo realizado.

Cada etapa descrita anteriormente possui objetivos e resultados específicos, que serão detalhados nas seções seguintes para construir uma base de informações relevantes para a construção da RA. Vale ressaltar que, como neste estudo os dados coletados serão utilizados diretamente nos capítulos posteriores, a terceira etapa será simplificada.

3.1 Fase de planejamento

Na etapa de planejamento, define-se o objetivo do SMS, formulam-se as questões que a pesquisa pretende responder e, a partir delas, cria-se a *string* de busca (Petersen *et al.*, 2008; Zavala *et al.*, 2019). A *string* é construída com base em um conjunto de palavras-chave que correspondam à essência do escopo da mapeamento e seus sinônimos, conectados por operadores booleanos, visando abranger ao máximo a literatura relevante, sendo posteriormente utilizada para a realização de buscas em diversas bases de dados (Kitchenham e Charters, 2007; Vargas *et al.*, 2014). Adicionalmente, realiza-se um teste piloto da *string* para ajustar sua sensibilidade e precisão, assegurando que recupere estudos significativos sem ruído excessivo (Zavala *et al.*, 2019). Essa etapa também inclui a elaboração dos critérios de seleção dos artigos, a fim de filtrar os estudos que não correspondem ao objetivo da pesquisa (Petersen *et al.*, 2008; Vargas *et al.*, 2014). Por fim, o protocolo construído é submetido a uma validação interna entre os pesquisadores, garantindo clareza e consistência antes do início da fase de condução (Kitchenham e Charters, 2007).

3.1.1 Questões de pesquisa

Este SMS tem como objetivo determinar as características de jogos digitais *open-source*, identificando as práticas adotadas durante o desenvolvimento desses jogos, bem como os diversos aspectos considerados. Isso levou às seguintes Questões de Pesquisa (RQs), que abrangem os tópicos mais importantes no escopo do nosso projeto e resumem os nossos resultados de pesquisa:

- **RQ_1:** Quais são os principais requisitos de qualidade considerados durante o desenvolvimento do jogo?
- **RQ_2:** Qual é o gênero do jogo?
- **RQ_3:** Qual *framework*, linguagem de programação ou motor de jogo foi utilizado no desenvolvimento do jogo?
- **RQ_4:** Qual processo de desenvolvimento foi utilizado para criar o jogo?
- **RQ_5:** Quais soluções arquiteturais (padrões, táticas ou estilos arquiteturais) foram utilizadas para o jogo?
- **RQ_6:** O jogo é de código aberto? Em caso afirmativo, qual licença é utilizada e onde o repositório pode ser encontrado?

3.1.2 *String* de Busca e Bases de Dados

Com base nas RQs definidas, foram identificadas palavras-chave e seus respectivos sinônimos para a construção da *string* de busca. Esta será aplicada em diversas bases de dados bibliográficas com o objetivo de recuperar artigos essenciais para o estudo.

Na formulação da *string*, os sinônimos foram conectados pelo operador lógico “OR” enquanto diferentes conceitos foram relacionados mediante o operador “AND”. A busca

foi realizada nas seguintes bases de dados: EI Compendex,[1] IEEE Digital Library,[2] ISI Web of Science,[3] ScienceDirect[4] e Scopus.[5]

Devido às restrições específicas de sintaxe e funcionalidade de cada base de dados, foi necessário adaptar e modificar a *string* de busca original. As *strings* efetivamente utilizadas em cada base para identificação dos estudos primários estão detalhadas na Tabela 3.1.

Tabela 3.1: *String de Busca utilizado em cada Base de Dados*

Base de Dados	Consulta de Busca
El Compendex	("Open Source"OR "free source"OR "free software"OR "libre software"OR "FLOSS"OR "FOSS") AND ("Digital Game"OR "Serious Game"OR "Electronic Game"OR "Video Game"OR "Online Game")
IEEE Digital Library	("Open Source"OR "free source"OR "free software"OR "libre software"OR "FLOSS"OR "FOSS") AND ("Digital Game"OR "Serious Game"OR "Electronic Game"OR "Video Game"OR "Online Game")
ISI Web of Science	("Open Source"OR "free source"OR "free software"OR "libre software"OR "FLOSS"OR "FOSS") AND ("Digital Game"OR "Serious Game"OR "Electronic Game"OR "Video Game"OR "Online Game")
ScienceDirect	("Open Source"OR "free source"OR "free software"OR "libre software"OR "FLOSS") AND ("Digital Game"OR "Electronic Game"OR "Video Game"OR "Online Game")
Scopus	("Open Source"OR "free source"OR "free software"OR "libre software"OR "FLOSS"OR "FOSS") AND ("Digital Game"OR "Serious Game"OR "Electronic Game"OR "Video Game"OR "Online Game")

3.1.3 Critérios de seleção

Com os estudos recuperados por meio da *string* de busca, torna-se necessário filtrar os trabalhos irrelevantes para obter evidências concretas sobre a área de pesquisa. Para tanto, foram definidos critérios de inclusão e exclusão. Um estudo prosseguirá para a etapa de análise por leitura do texto completo se, e somente se, satisfazer todos os critérios de inclusão e não atender a nenhum dos critérios de exclusão.

- Critérios de inclusão (IC)
  - IC1: O estudo apresenta um jogo digital
  - IC2: O estudo apresenta um jogo *open-source*
- Critérios de exclusão (EC)

[1] https://www.elsevier.com/products/engineering-village/databases/compendex

[2] https://www.ieee.org/publications/subscriptions/products/mdl/ieeexplore-access.html

[3] https://clarivate.com/academia-government/scientific-and-academic-research/research-discovery-and-referencing/web-of-science/

[4] https://www.sciencedirect.com/

[5] https://www.elsevier.com/products/scopus

- EC1: Estudos curtos (até 4 páginas).
- EC2: Estudos duplicados.
- EC3: O estudo não apresenta um jogo.
- EC4: O estudo não apresenta um jogo *open-source*.
- EC5: O estudo não está disponível para análise.
- EC6: O estudo não está em português ou inglês.
- EC7: O estudo é um estudo secundário.

3.2 Fase de condução

Nesta etapa, os artigos são obtidos das fontes selecionadas por meio da pesquisa utilizando a *string* de busca definida no planejamento e avaliados com base nos critérios de inclusão e exclusão estabelecidos (Vargas *et al.*, 2014; Gezici *et al.*, 2019). O processo de seleção é realizado em múltiplos estágios, começando pela análise de títulos e resumos, seguida pela leitura completa dos textos para confirmar a relevância e extrair as informações necessárias para responder às RQs (Zavala *et al.*, 2019).

Durante a extração, os dados são sistematizados com base em um esquema de classificação predefinido, que pode incluir dimensões como contexto de aplicação, artefatos utilizados durante o desenvolvimento, atributos de qualidade considerados, entre outros (Petersen *et al.*, 2008; Vargas *et al.*, 2014). Para garantir a confiabilidade, é realizado um processo de validação cruzada entre os pesquisadores, no qual cada artigo é analisado por múltiplos avaliadores, com discordâncias resolvidas por discussão ou mediante a intervenção de um terceiro revisor (Zavala *et al.*, 2019; Gezici *et al.*, 2019).

O número de artigos recuperados em cada base de dados e os resultantes de cada fase do processo de seleção são documentados de forma transparente, podendo ser utilizado um diagrama de fluxo ou tabelas para representá-los (Vargas *et al.*, 2014; Zavala *et al.*, 2019), como ilustrado na Tabela 3.2 e na Figura 3.1. Essa rastreabilidade assegura a replicabilidade do estudo e permite aos leitores avaliar a abrangência da revisão.

Tabela 3.2: *Número de estudos retornados por base de dados*

Bases de Dados	Quantidade
El Compendex	212
IEEE Digital Library	145
ISI Web of Science	245
Science@Direct	27
Scopus	412

Para auxiliar na condução do estudo, foi utilizada a ferramenta online Parsifal.[6] O

[6] https://parsif.al/

Parsifal é uma ferramenta web que permite que pesquisadores colaborem em um mesmo projeto de pesquisa, mesmo estando geograficamente distantes. Adicionalmente, oferece funcionalidades que facilitam o desenvolvimento do mapeamento sistemático, como a remoção automática de estudos duplicados e a capacidade de realizar comentários para anotar informações relevantes em cada estudo. Como resultado do processo de extração, obtiveram-se inicialmente 1041 estudos primários. Desses, 389 eram estudos duplicados e foram rejeitados automaticamente, e 567 foram excluídos com base nos critérios de seleção. Por fim, para responder às RQs, foi realizada a leitura completa dos textos, resultando na seleção de 30 estudos primários que atendem aos objetivos da pesquisa.

Figura 3.1: *Número de estudos após cada etapa do SMS*

A Tabela 3.3 apresenta a lista completa dos estudos primários selecionados para responder às RQs. A tabela é composta por três colunas: a primeira exibe o ID do estudo, facilitando futuras citações; a segunda apresenta os títulos dos estudos primários; e a terceira fornece as respectivas referências bibliográficas.

Tabela 3.3: *Lista de estudos primários selecionados*

ID	Título	Referência
S1	RaidEnv: Exploring new challenges in automated content balancing for boss raid games	Jeon *et al.*, 2023
S2	NanoAdventure: Development of a Text-Based Adventure Game in English, Spanish, and Chinese for Communicating about Nanotechnology and the Nanoscale	Hudson-Smith *et al.*, 2023
S3	An Educational Game to Learn Picking Techniques in Warehousing–WareMover	Franke *et al.*, 2024
S4	Transferring an educational board game to a multi-user mobile learning game to increase shared situational awareness	Klemke *et al.*, 2015b
S5	Development of a video game for teaching in an electrical machine laboratory	Ramirez e Rivera, 2022
S6	Implementation and evaluation of a background music reactive game	Aallouche *et al.*, 2007
S7	Lessons learned from creating a mobile version of an educational board game to increase situational awareness	Klemke *et al.*, 2015a
S8	Dungeon crawl stone soup as an evaluation domain for artificial intelligence	Dannenhauer *et al.*, 2019
S9	Games for research: a comparative study of open source game projects	Halvorsen e Raaen, 2014
S10	Experimental Feedback on Prog&Play: A Serious Game for Programming Practice	Muratet *et al.*, 2011
S11	Educating the Next Generation of Mobile Game Developers	Zyda *et al.*, 2007
S12	A Scenario-Based and Game-Based Geographical Information System (GIS) Approach for Earthquake Disaster Simulation and Crisis Mitigation	Feizizadeh *et al.*, 2023
S13	Avaler's adventure: An open source game for dysphagia therapy	Pollock *et al.*, 2017
S14	Learning analytics for a puzzle game to discover the puzzle-solving tactics of players	Vahdat *et al.*, 2016
S15	I AM A.I. Gradient Descent - an open source Digital Game for Inquiry-Based CLIL Learning	Geldhauser *et al.*, 2022
S16	Server-side verification of client behavior in online games	Bethea *et al.*, 2008
S17	Argotario: Computational argumentation meets serious games	Habernal *et al.*, 2017
S18	Madly Ambiguous: A Game for Learning about Structural Ambiguity and Why It's Hard for Computers	Gokcen *et al.*, 2018
S19	Postsynaptic simulator: an open source visual interactive simulation for teaching action potentials	Salgo *et al.*, 2021
S20	Methodology to Develop Serious Games for Primary Schools	Alaoui *et al.*, 2020
S21	Development and testing of a game-based digital intervention for working memory training in autism spectrum disorder	Wagle *et al.*, 2021
S22	Pixel Python RPG: Repurposing an Entertainment Game to an Open Educational Resource for Computer Programming Fundamentals	J. P. D. Silva *et al.*, 2021

Continua na próxima página

Tabela 3.3 (Continuação)

ID	Título	Referência
S23	From a Social POV: The Impact of Point of View on Player Behavior, Engagement, and Experience in a Serious Social Simulation Game	Schlagowski *et al.*, 2024
S24	Rulebook: An Architectural Pattern for Self-Amending Mechanics in Digital Games	Wilson Kazuo Mizutani e Kon, 2024
S25	E-polis: An Innovative and Fun Way to Gamify Sociological Research with an Educational Serious Game - Game Development Middleware Approach	Gazis e Katsiri, 2024
S26	The "Water Cooler"Game	Hollins *et al.*, 2017
S27	CT brush and CancerZap!: Two video games for computed tomography dose minimization	Alvare e Gordon, 2015
S28	DRLeague: A Novel 3D Environment for Training Reinforcement Learning Agents	Farrapo *et al.*, 2022
S29	The MentalPlus® Digital Game Might Be an Accessible Open Source Tool to Evaluate Cognitive Dysfunction in Heart Failure with Preserved Ejection Fraction in Hypertensive Patients: A Pilot Exploratory Study	V. F. A. Pereira e Valentin, 2018
S30	Building an education ecology on serious game design and development for the One Laptop Per Child and Sugar platforms: A service learning course builds a base for peer mentoring, Cooperative Education internships, and sponsored research	Jacobs, 2010

3.3 Análise de Dados

Os dados extraídos são sintetizados por meio de análises quantitativa e qualitativa, permitindo identificar tendências, lacunas e relações entre categorias (Kitchenham e Charters, 2007). Esta etapa de análise envolve resumir os dados coletados durante a fase de condução e organizar as informações para responder às RQs (Petersen *et al.*, 2008; Gezici *et al.*, 2019). Para facilitar a visualização dos dados e auxiliar no processo analítico, são construídas tabelas e figuras que representam os dados coletados (Vargas *et al.*, 2014; Zavala *et al.*, 2019). Ferramentas visuais, como diversos tipos de gráficos, são frequentemente utilizadas para representar a interseção entre diferentes facetas do esquema de classificação, oferecendo uma visão clara da distribuição das pesquisas na área.

3.3.1 Visão Geral dos Estudos Primários

A Figura 3.2 revela uma tendência de crescimento no número de estudos sobre jogos digitais *open-source* ao longo dos últimos anos. O ano de 2024 registrou o maior número de publicações, seguido por 2021, indicando que este campo de pesquisa vem ganhando atenção crescente na comunidade acadêmica. Entretanto, embora seja evidente a tendência de crescimento, o volume geral de publicações ainda é relativamente moderado, fato comprovado pelo número limitado de estudos identificados nesta área específica.

Um aspecto particularmente relevante observado na distribuição temporal é a ausência completa de publicações no ano de 2020. Esta lacuna pode ser atribuída aos impactos da pandemia global de COVID-19, que afetou significativamente as atividades de pesquisa e desenvolvimento em diversos setores, incluindo a produção acadêmica na área de jogos digitais.

Após o período de interrupção em 2020, houve uma recuperação notável no interesse pela temática, com um pico significativo em 2024. Este padrão pode refletir tanto o amadurecimento das tecnologias de desenvolvimento de jogos *open-source* quanto o reconhecimento crescente de sua importância para a indústria de jogos independentes e educacionais. Adicionalmente, é possível inferir que o aumento no número de publicações

está relacionado com a expansão das comunidades de desenvolvimento aberto e a maior adoção de práticas colaborativas no desenvolvimento de jogos. A disponibilidade de ferramentas e *frameworks open-source* mais maduros também pode ter contribuído para este crescimento, facilitando a entrada de novos pesquisadores e desenvolvedores neste campo.

Por fim, ressalta-se que, embora a trajetória seja positiva, o número absoluto de estudos ainda é modesto se comparado a outras áreas de pesquisa em jogos digitais, indicando que este permanece um campo fértil para investigações futuras e com significativo potencial para expansão.

Figura 3.2: *Número de estudos por ano de publicação*

3.3.2 RQ_1: Quais são os principais requisitos de qualidade considerados durante o desenvolvimento do jogo?

Os atributos de qualidade considerados no desenvolvimento dos jogos, presentes nos estudos analisados, são apresentados na Tabela 3.4, seguindo o padrão ISO/IEC 25010. Nesta análise, foram extraídas as qualidades principais consideradas nos estudos, priorizando-as em detrimento dos requisitos secundários, com o objetivo de identificar os atributos mais relevantes. Além disso, nem todos estudos relatam sobre qualidades que consideraram, impedindo que outras pesquisas consigam adquirir experiências destes.

Ao examinar a tabela, identifica-se que os principais atributos considerados são adaptabilidade, capacidade de aprendizagem e modificabilidade, cada um encontrado em pelo menos seis estudos. Vários estudos mencionaram que os desenvolvedores visavam que o jogo fosse fácil de aprender e jogar, bem como facilmente modificável, para permitir a inclusão de novas funcionalidades com esforço mínimo. Adicionalmente, os jogos devem funcionar em múltiplos dispositivos, inclusive naqueles com recursos limitados. Essa tendência pode ser explicada pelo fato de que uma parcela significativa dos jogos analisados é publicada academicamente e consiste em jogos educacionais ou simuladores. Consequentemente, espera-se que sejam fáceis de usar por estudantes sem experiência prévia, além de serem executados em dispositivos fornecidos pela escola. Por esse motivo, explica-se também uma certa negligência em relação ao aspecto de engajamento do usuário em sua construção, elemento central pela natureza dos jogos digitais.

As características de qualidade significativas, porém secundárias, incluem utilização de recursos, adequação reconhecível e operabilidade. Esses aspectos são importantes para permitir que os usuários utilizem os jogos em máquinas menos potentes, reduzir o esforço

necessário para identificar se o software atende às suas necessidades e facilitar a operação, fatores que também são explicados pela natureza dos estudos analisados.

Por fim, é notável uma negligência em relação aos atributos de Confiabilidade, *Security* e *Safety*. A negligência em relação à Confiabilidade pode ser devida ao fato de que os estudos são majoritariamente conduzidos em ambiente acadêmico, sem ampla utilização em situações reais. Portanto, os desenvolvedores focam prioritariamente na execução das mecânicas para avaliar os resultados, em vez de garantir a operação robusta do software como um todo. A questão da *Security* pode ser explicada pelo mesmo motivo, os desenvolvedores não precisam se preocupar com aspectos de segurança devido à falta de uso real, o que elimina preocupações com vazamento de dados pessoais para agentes externos. Por fim, a questão da *Safety* é explicada pela natureza dos jogos digitais, uma vez que a maioria não envolve o uso e controle de equipamentos externos pelo software, não havendo, portanto, preocupação com danos no mundo real além do próprio dispositivo.

Tabela 3.4: *Características de qualidade consideradas no desenvolvimento dos jogos*

Categoria	**Subcategoria**	**Estudos Acadêmicos**
Adequação Funcional	Completude Funcional	S12, S27, S29
	Correção Funcional	S27, S29
	Apropriação Funcional	S26, S28, S29
Eficiência de *Performance*	Comportamento do Tempo	S11, S12, S25
	Utilização de Recursos	S10, S11, S25, S30
Compatibilidade	Interoperabilidade	S14
Capacidade de Interação	Adequação Reconhecível	S14, S15, S27, S30
	Capacidade de Aprendizagem	S2, S13, S14, S15, S29, S30
	Operabilidade	S15, S27, S29
	Proteção de Erro do Usuário	S13
	Engajamento do Usuário	S12, S13
	Inclusividade	S13, S29
	Assistência ao Usuário	S2, S21
	Auto-descritividade	S21
Manutenibilidade	Modularidade	S1, S28, S30
	Modificabilidade	S1, S13, S17, S26, S28, S30
	Testabilidade	S22
Flexibilidade	Adaptabilidade	S1, S2, S12, S13, S15, S17, S19, S21, S22
	Escalabilidade	S25

3.3.3 RQ_2: Qual é o gênero do jogo?

Quanto ao gênero de jogo mais prevalente, observa-se no gráfico da Figura 3.3 a predominância do gênero simulação, seguido por jogos educacionais e de estratégia. Essa predominância pode ser atribuída a vários fatores inerentes ao contexto acadêmico e de

desenvolvimento *open-source*. Os jogos de simulação permitem modelar sistemas complexos de maneira controlada, oferecendo um ambiente seguro para experimentação e aprendizado. Já os jogos educacionais atendem a objetivos pedagógicos claros, alinhando-se com as finalidades de pesquisa e desenvolvimento que caracterizam muitos projetos acadêmicos. Cabe ressaltar que múltiplos projetos possuem mais de um gênero, combinando características específicas de diferentes tipos de jogos.

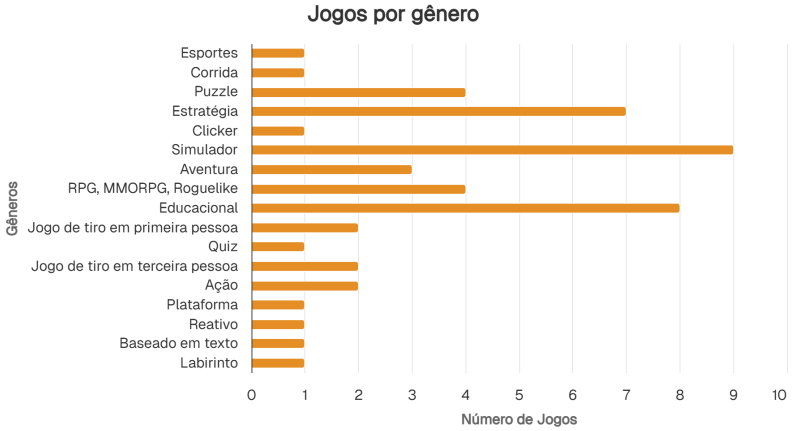

Figura 3.3: *Gêneros dos jogos pesquisados*

3.3.4 RQ_3: Qual *framework*, linguagem de programação ou motor de jogo foi utilizado no desenvolvimento do jogo?

Apesar de nem todos os projetos descreverem quais artefatos utilizaram para o desenvolvimento dos jogos, é possível observar uma tendência nas linguagens empregadas. A partir da Figura 3.4, identifica-se que JavaScript é a linguagem de programação mais utilizada no desenvolvimento de jogos digitais *open-source*, seguida por C++, Python e C. Esse resultado pode ser explicado pelo fato de essas linguagens serem amplamente difundidas no desenvolvimento de software em geral, além de serem conhecidas pela maioria dos programadores devido à sua popularidade e ao suporte da comunidade para solucionar desafios técnicos. Consequentemente, tornam-se opções mais acessíveis para o desenvolvimento de jogos. Ademais, constata-se que JavaScript é predominante em jogos implementados para navegadores web, dada sua natureza de uso no desenvolvimento web. Por outro lado, as demais linguagens são primariamente utilizadas para aplicações desktop e mobile.

Em relação aos *Game Engines* utilizados durante o desenvolvimento — artefatos comumente empregados no processo por sua capacidade de oferecer um conjunto de funcionalidades bem estruturada, evitando a necessidade de implementar soluções já disponibilizadas pelo *engine* —, podemos visualizar sua distribuição na Figura 3.4. O motor de jogo mais utilizado entre os estudos que fizeram uso dessa ferramenta, com quase 50% de ocorrência, é o Unity. Essa predominância pode ser atribuída ao seu potencial para desenvolver jogos robustos, aliado a uma comunidade grande e ativa, que fornece suporte técnico e uma variedade de *plug-ins* para mecânicas comuns. Além disso, o Unity é reconhecido por ser acessível a iniciantes, oferecendo tutoriais abrangentes e suportando a compilação

do mesmo código para múltiplas plataformas.

Enquanto isso, outros projetos optaram por motores menos conhecidos, como Twine — específico para ficções interativas — ou ChoiCo, voltado para jogos de simulação baseados em escolhas. Esses *engines* podem oferecer funcionalidades mais especializadas para domínios específicos, tornando-se opções mais adequadas em alguns contextos do que um motor de propósito geral, como o Unity. Vale ressaltar ainda que, nos projetos que não utilizaram o Unity e optaram por uma *Game Engine*, todos elegeram engines de código aberto. Isso indica uma tendência de alinhamento entre os pesquisadores e desenvolvedores da área e os princípios de abertura, transparência e personalização possibilitados por ferramentas *open-source*, as quais se adequam melhor aos objetivos de replicabilidade, adaptação e investigação técnica comuns em ambientes acadêmicos e de pesquisa.

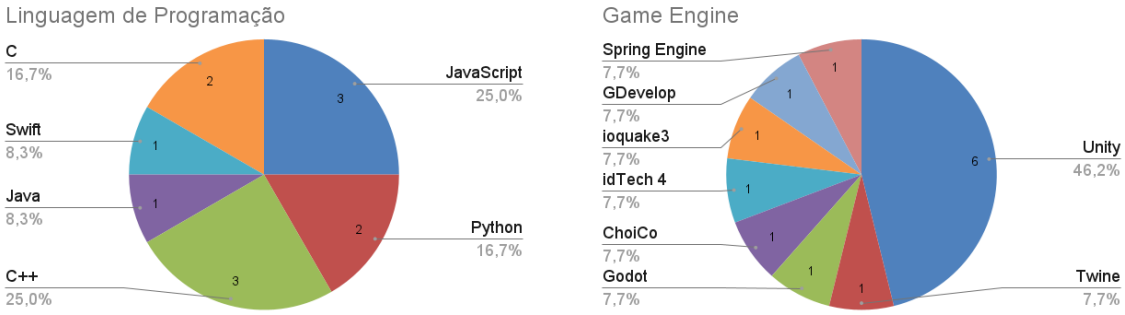

Figura 3.4: *Linguagens de programação e Game Engine utilizadas para o desenvolvimento*

3.3.5 RQ$_4$: Qual processo de desenvolvimento foi utilizado para criar o jogo?

A análise dos estudos selecionados revela que a maioria não fornece descrições detalhadas do processo de desenvolvimento adotado na criação de seus jogos educacionais. Essa lacuna documental compromete a reprodutibilidade das práticas de desenvolvimento e restringe a transferibilidade de experiências para outras iniciativas.

Contudo, alguns estudos mencionam explicitamente o uso de processos ou modelos de desenvolvimento específicos. O estudo S3 empregou o método GAMED, uma abordagem que integra princípios de gamificação e objetivos educacionais ao processo de engenharia de software. O estudo S13, embora não identifique um processo sistemático específico neste contexto, seguiu uma metodologia clássica composta por três etapas: elicitação de requisitos, implementação e avaliação - indicando, assim, um fluxo de desenvolvimento organizado.

Ademais, o estudo S20 adotou o método GLUPS, focado no desenvolvimento de jogos centrado no utilizador, e relatou a utilização do Desenvolvimento Orientado a Testes (TDD) como estratégia de implementação. A aplicação do TDD demonstra um compromisso com a qualidade, confiabilidade e manutenibilidade do código, ainda que os aspectos arquiteturais não tenham sido explorados em profundidade.

Em síntese, embora existam exemplos isolados de processos estruturados, a ausência de descrições abrangentes do processo de desenvolvimento permanece uma lacuna considerável na literatura analisada.

3.3.6 RQ_5: Quais soluções arquiteturais (padrões, táticas ou estilos arquiteturais) foram utilizadas para o jogo?

De forma análoga ao observado na questão anterior, a maioria dos estudos não fornece descrições explícitas das soluções arquitetônicas adotadas no desenvolvimento dos jogos. Essa lacuna dificulta uma compreensão mais aprofundada sobre a aplicação dos princípios de arquitetura de software no contexto de jogos digitais.

Contudo, alguns trabalhos destacam-se pelo esforço em documentar decisões arquiteturais. O estudo S13 adotou o padrão Model-View-Controller (MVC) e uma Arquitetura Orientada a Serviços (SOA), demonstrando preocupação com modularidade e escalabilidade. O estudo S16 empregou o modelo Cliente-Servidor, estilo arquitetural consagrado para aplicações distribuídas e jogos *online*. Já o estudo S24 seguiu o padrão *Rulebook*, propondo uma abordagem baseada em regras que potencialmente favorece a manutenibilidade e a transparência da lógica do jogo.

Ademais, o estudo S25 referencia princípios arquiteturais desenvolvidos pelo Instituto de Tecnologia de Massachusetts (MIT), com ênfase em clareza, robustez e modularidade. Embora não detalhados, tais referenciais sugerem uma abordagem sistemática e academicamente fundamentada. Por fim, o estudo S28 combinou uma arquitetura baseada em componentes com o padrão Observer para promover reutilização e desacoplamento entre os componentes do sistema.

Esses casos ilustram que, embora existam considerações metodológicas e arquiteturais na literatura, ainda persiste uma escassa documentação sistemática acerca do uso de padrões, táticas e estilos arquiteturais no desenvolvimento de jogos digitais *open-source*.

3.3.7 RQ_6: O jogo é de código aberto? Em caso afirmativo, qual licença é utilizada e onde o repositório pode ser encontrado?

Todos os jogos desenvolvidos nos estudos selecionados adotam a filosofia *open-source*, uma vez que estudos que não atendiam a esse critério foram removidos durante as etapas anteriores de seleção. O foco no código aberto foi essencial para investigar aspectos arquitetônicos e de desenvolvimento, práticas de licenciamento e a disponibilidade dos repositórios. Entretanto, embora os estudos afirmem que os jogos desenvolvidos são de código aberto, observa-se que nem todos disponibilizaram informações completas sobre a licença adotada ou o link para o repositório contendo o código-fonte do projeto.

Entre os estudos selecionados, a maioria fornece pelo menos uma forma de transparência, seja pela especificação da licença sob a qual o jogo é distribuído, seja pela disponibilização do código-fonte em repositórios *online*. As informações estão resumidas na Figura 3.5, da qual se observa que 19 estudos incluem um link para o repositório de código-fonte, facilitando o acesso aos detalhes de implementação e permitindo replicação

ou reutilização. Adicionalmente, 10 estudos mencionam explicitamente o tipo de licença de código aberto adotada (por exemplo, MIT, GPL, Apache). Dentre eles, 9 estudos forneceram tanto o repositório quanto a licença, garantindo clareza legal e acesso técnico.

Essa disparidade pode ser atribuída ao fato de que parte dos estudos está inserida em contextos acadêmicos onde a divulgação completa dessas informações não é obrigatória, como em trabalhos de conclusão de curso ou dissertações. Ademais, é plausível que alguns repositórios tenham acesso restrito, disponível apenas para colaboradores autorizados, o que impediria uma investigação externa mais aprofundada. Tal omissão limita a usabilidade prática dos jogos e levanta questões quanto à aderência às melhores práticas de código aberto. Portanto, embora o código aberto seja uma característica comum entre os jogos analisados, sua implementação varia significativamente em termos de completude e acessibilidade, refletindo a necessidade de padrões mais consistentes de documentação na academia.

Figura 3.5: *Estudos classificados com base na licença e repositório*

3.4 Discussão

Analisando as respostas obtidas no SMS, é possível chegar a algumas evidências. O número de estudos primários nesse campo demonstra uma tendência de crescimento ao longo dos anos, indicando um interesse crescente e uma maior atenção por parte dos pesquisadores para a área. Entretanto, observa-se que nem todos os estudos detalham adequadamente os aspectos considerados durante o desenvolvimento dos jogos propostos. Essa lacuna é evidenciada pela escassez de informações fornecidas pela maioria dos trabalhos analisados para responder às perguntas de pesquisa, permitindo uma análise apenas com base nas poucas respostas disponíveis. Tal fato aponta para uma inconsistência na condução das pesquisas: por um lado, há um aumento no interesse científico; por outro, persiste uma deficiência na disseminação sistemática do conhecimento gerado.

Adicionalmente, nota-se que nem todos os estudos adotam padrões e terminologias internacionalmente consagrados para descrever as decisões e considerações abordadas. Em vez disso, muitas descrições apresentam-se relativamente subjetivas, o que pode dificultar a interpretação e a replicação por outros pesquisadores. Isso ressalta a necessidade de que estudos futuros adotem métodos de descrição mais precisos e empreguem terminologias padronizadas. Tal prática não apenas tem o potencial de elevar a qualidade dos

softwares desenvolvidos, como também promove a acumulação e a reutilização efetiva do conhecimento no domínio de pesquisa.

3.4.1 Ameaças à Validade

Ao conduzir um estudo de mapeamento sistemático, é crucial abordar as potenciais ameaças à validade para garantir a confiabilidade e a generalização dos achados. Esta seção discute as ameaças à validade interna, externa, de construção e de conclusão associadas a este estudo, bem como as estratégias adotadas para mitigá-las.

Validade Interna: Refere-se à extensão na qual o estudo mede com precisão o que se propõe a medir, minimizando a influência de vieses ou erros no processo de pesquisa. Neste estudo, ameaças à validade interna podem surgir da seleção dos estudos primários, da extração de dados e dos processos de síntese. Para minimizar o viés de seleção, foi estabelecido um protocolo bem definido com critérios de inclusão e exclusão explícitos antes da revisão. Adicionalmente, dois revisores examinaram independentemente os estudos e realizaram revisões cruzadas para garantir a identificação precisa dos estudos primários aceitos.

Validade Externa: Diz respeito à generalização dos achados do estudo para contextos além do escopo específico desta pesquisa. Embora este estudo se concentre no desenvolvimento de jogos de código aberto, é essencial reconhecer que os seus resultados podem não ser totalmente aplicáveis a outras áreas de desenvolvimento de software ou a ambientes de desenvolvimento de jogos comerciais. A consideração exclusiva de estudos acadêmicos acessíveis publicamente também pode introduzir viés, uma vez que repositórios restritos ou privados, que poderiam conter pesquisas relevantes, não foram incluídos.

Validade de Construção: Refere-se ao grau em que o estudo mede adequadamente os conceitos teóricos que pretende investigar. Neste estudo, ameaças à validade de construção podem decorrer da má interpretação das questões de pesquisa ou de definições imprecisas de conceitos-chave. Para mitigar esses riscos, foi elaborado um protocolo de pesquisa detalhado, incluindo questões de pesquisa bem definidas, critérios consistentes para extração de dados e formulários padronizados para garantir coleta e análise uniformes. O envolvimento de múltiplos revisores nos processos de seleção e extração de dados contribuiu para manter a objetividade e reduzir o viés subjetivo.

Validade de Conclusão: Envolve o grau de credibilidade e justificabilidade das conclusões apresentadas. Para melhorar a precisão da extração de dados, foi empregado um formulário padronizado, garantindo que informações relevantes fossem consistentemente capturadas em todos os estudos. Diferentes autores revisaram independentemente as conclusões para assegurar que refletissem com precisão os dados extraídos. Embora esforços tenham sido realizados para minimizar o viés de interpretação por meio do envolvimento de múltiplos pesquisadores, reconhece-se que um risco residual de viés pode permanecer.

3.5 Considerações Finais

Esta SMS apresenta uma visão geral do estado atual do desenvolvimento de jogos digitais *open-source*, sintetizando dados de 30 estudos primários. Diversas percepções

emergiram em relação aos aspectos de qualidade, tecnologias, processos e práticas colaborativas. Em primeiro lugar, atributos de qualidade como adaptabilidade, capacidade de aprendizagem e modificabilidade foram os mais frequentemente citados. Estes são particularmente relevantes para jogos educativos e de simulação (gêneros que predominam entre os estudos analisados). A ênfase nestes atributos sugere uma forte preocupação em garantir que os jogos possam ser compreendidos, estendidos e reutilizados por diversas partes interessadas, incluindo estudantes, educadores e programadores.

No que diz respeito às tecnologias, JavaScript, C++, C e Python foram as linguagens de programação mais utilizadas, enquanto o Unity destacou-se como o *game engine* mais popular. Este resultado reflete a preferência por ferramentas acessíveis, multiplataforma e com amplo suporte comunitário. Por outro lado, alguns estudos adotaram *engines open-source* como Twine e Godot, evidenciando uma adesão à filosofia de código aberto, que privilegia a contribuição colaborativa e a disseminação do conhecimento — aspetos comuns em contextos acadêmicos e de pesquisa.

No âmbito da arquitetura de software e dos processos de desenvolvimento, os resultados revelam um cenário fragmentado. Poucos estudos adotaram processos formais ou documentaram explicitamente padrões arquiteturais. Quando presentes, foram observadas práticas como MVC, modelos cliente-servidor e arquiteturas baseadas em componentes. De modo similar, metodologias de desenvolvimento como TDD, GAMED e GLUPS foram mencionadas pontualmente. Esta lacuna salienta a necessidade de maior rigor metodológico e de documentação no desenvolvimento de jogos *open-source*, particularmente para promover a colaboração e sustentabilidade dos projetos.

Outra descoberta relevante diz respeito à diversidade na adoção de licenças e na disponibilização de repositórios. Apesar de todos os jogos analisados serem de código aberto, muitos estudos não declararam claramente o tipo de licença ou a localização do repositório. Apenas uma minoria disponibilizou ambas as informações. Esta falta de transparência coloca desafios à reutilização e à contribuição, reforçando a importância de adotar práticas de licenciamento e divulgação mais claras, alinhadas com os princípios do código aberto.

Em suma, embora a área demonstre crescimento, verifica-se ainda uma insuficiente padronização no que concerne a processos, padrões arquiteturais e documentação. Trabalhos futuros deverão concentrar-se no desenvolvimento de processos padronizados e arquiteturas de referência que orientem o desenvolvimento sustentável e colaborativo de jogos digitais *open-source*. Abordar estas questões poderá aprimorar a qualidade, a reutilização e o envolvimento comunitário neste ecossistema.

Capítulo 4

Análise Arquitetural

No passo RA-2 do ProSA-RA, utilizam-se as informações coletadas na etapa anterior, por meio da condução de um SMS, complementadas por outras fontes, como artigos científicos, livros, documentos técnicos e a experiência de especialistas da área, para realizar a elicitação dos requisitos da RA proposta. O objetivo central desta etapa é definir os requisitos arquiteturais que comporão a RA. Para isso, foi necessário extrair e analisar os requisitos de sistemas (RS) de software do domínio em questão, sintetizando-os posteriormente no terceiro passo do ProSA-RA, cujo objetivo é estabelecer, de fato, a RA para jogos digitais *open-source*.

Este capítulo será organizado da seguinte forma: a Seção 4.1 apresenta os requisitos do sistema identificados pelos especialistas da área; a Seção 4.2 apresenta a síntese dos requisitos identificados anteriormente, mapeados para as características de qualidade da ISO/IEC 25010, além da análise de relevância de cada qualidade, examinando estudos e jogos existentes; a Seção 4.3 apresenta a identificação das características de qualidades essenciais e secundárias do sistema de interesse; e, finalmente, a Seção 4.4 discute as considerações finais deste capítulo.

4.1 Identificação dos Requisitos do Sistema

Nesta etapa, identificam-se os requisitos do sistema mencionados nos estudos relacionados à área de interesse, classificados como atributos de qualidade essenciais para o sistema desejado. Incluem-se também requisitos funcionais e não funcionais identificados diretamente em jogos digitais *open-source* existentes.

Esses requisitos serão utilizados nas etapas subsequentes, quando analisados e sintetizados para a criação de requisitos arquiteturais que fundamentaram a construção da RA. A lista completa dos 52 requisitos identificados encontra-se no Apêndice A.

4.2 Síntese dos Requisitos do Sistema

Após a definição dos RS, foram identificados os requisitos não funcionais presentes neles, classificados de acordo com o padrão ISO/IEC 25010:2023 (lista completa presente na figura 4.1). A Tabela 4.1 apresenta todas as características e subcaracterísticas identificadas e as siglas de requisitos do sistema correspondentes.

Além dos requisitos apresentados no padrão ISO/IEC 25010:2023, alguns requisitos do sistema indicam a existência de aspectos específicos que devem ser considerados no contexto de jogos digitais *open source*, são eles:

- RS-41: O jogo deve considerar uma licença de código aberto clara para permitir modificações e redistribuição pela comunidade.
- RS-44: O jogo deve utilizar ferramentas para desenvolvimento colaborativo, como histórico de versões e compartilhamento de projetos na nuvem.
- RS-48: O jogo deve manter fóruns, repositórios públicos para discussão de ideias, reporte de bugs e sugestões.
- RS-49: O jogo deve reconhecer contribuidores da comunidade (ex.: créditos no jogo, *badges* em perfis públicos) para incentivar participação ativa.

Para poder identificar a importância desses aspectos e a necessidade de incluí-los na RA construída, eles serão identificados como **Adoção de Licença, Desenvolvimento colaborativo e Suporte ao *feedback*** respectivamente, sendo subcategorias da categoria ***Open Source***

Qualidade do Produto de Software								
Adequação Funcional	**Eficiência de performance**	**Compatibilidade**	**Capacidade de Interação**	**Confiabilidade**	**Security**	**Manutenibilidade**	**Portabilidade**	**Safety**
Integridade funcional Correção funcional Apropriação funcional	Comportamento do tempo Utilização de Recursos Capacidade	Coexistência Interoperabilidade	Adequação reconhecível Capacidade de aprendizado Operabilidade Proteção de erro do usuário Engajamento ao usuário Inclusividade Assistência ao usuário Auto-descritividade	Ausência de Falhas Disponibilidade Tolerância a falhas Recuperabilidade	Confidencialidade Integridade Ausência de repúdio Rastreabilidade de uso Autenticidade Resistência	Modularidade Reusabilidade Analisabilidade Modificabilidade Testabilidade	Adaptabilidade Escalabilidade Facilidade de instalação Capacidade de substituição	Limitação Operacional Identificação de Risco Falha Segura Aviso de Perigo Integração segura

Figura 4.1: *Atributos de Qualidade de Software de acordo com a ISO/IEC 25010:2023*

Tabela 4.1: *Categorização de Requisitos Não-Funcionais*

Categoria	**Subcategoria**	**ID**
Adequação Funcional	Correção Funcional	RS-1
	Apropriação Funcional	RS-1
Eficiência de *Performance*	Comportamento do Tempo	RS-10
	Utilização de Recursos	RS-13
	Capacidade	RS-13, RS-15

Continua na próxima página

Tabela 4.1 (Continuação)

Categoria	**Subcategoria**	**ID**
Compatibilidade	Coexistência	RS-27
	Interoperabilidade	RS-8, RS-27
Capacidade de Interação	Adequação Reconhecível	RS-1, RS-35
	Capacidade de Aprendizagem	RS-7, RS-20, RS-24, RS-28, RS-38
	Operabilidade	RS-4, RS-7, RS-20, RS-22, RS-24, RS-38
	Engajamento do Usuário	RS-2, RS-4, RS-12, RS-17, RS-22, RS-25, RS-27, RS-39
	Inclusividade	RS-3, RS-14, RS-37, RS-39
	Assistência ao Usuário	RS-3, RS-18, RS-24, RS-45, RS-46
	Auto-descritividade	RS-2, RS-20, RS-24, RS-28, RS-30, RS-38
Confiabilidade	Ausência de Falhas	RS-10
	Disponibilidade	RS-8, RS-10, RS-11
	Tolerância a Falhas	RS-10, RS-16
	Recuperabilidade	RS-16
Security	Confidencialidade	RS-36, RS-43
	Integridade	RS-36, RS-43
	Ausência de repúdio	RS-43
	Autenticidade	RS-36, RS-43
	Resistência	RS-10
Manutenibilidade	Modularidade	RS-19, RS-26, RS-34, RS-42
	Reusabilidade	RS-26, RS-34, RS-42
	Analisabilidade	RS-19, RS-26, RS-33, RS-47, RS-50
	Modificabilidade	RS-19, RS-26, RS-33, RS-44, RS-47, RS-50
	Testabilidade	RS-26, RS-33, RS-52
Flexibilidade	Adaptabilidade	RS-16, RS-31, RS-40
	Escalabilidade	RS-15
Safety	Limitação Operacional	RS-18
	Identificação de Riscos	RS-18
	Falha Segura	RS-18
	Aviso de Perigo	RS-18
Open Source	Adoção de Licença	RS-41
	Desenvolvimento colaborativo	RS-44
	Suporte ao feedback	RS-48

4.2.1 Identificação da Relevância dos Requisitos do Sistema

Para identificar os requisitos mais relevantes a serem incluídos na RA e diminuir o viés introduzido durante a análise de RS, analisou-se a quantidade de estudos acadêmicos e jogos *open-source* que abordam cada aspecto específico. Quanto mais estudos apontam a necessidade de um determinado requisito ou o consideram durante o desenvolvimento de jogos *open-source*, maior deve ser a ênfase atribuída a esse requisito na RA. Da mesma forma, aspectos recorrentes em jogos *open-source* já existentes tendem a ser reconhecidos como elementos comuns e essenciais nesse tipo de software.

Identificação dos Estudos Acadêmicos

Para identificar os requisitos relevantes do sistema, foram analisados 4 estudos da área de software *open-source* com mais de 150 citações e 10 estudos que relatam diretamente ou parcialmente o processo de desenvolvimento de jogos *open-source*. A Tabela 4.2 apresenta o mapeamento entre as categorias de requisitos e os estudos que fundamentam sua pertinência. Nela, um estudo é considerado pela categoria quando ressalta a sua importância ou se considera explicitamente o requisito no desenvolvimento do software. A análise revela que:

- A **manutenibilidade** apresenta-se como uma das categorias mais relevantes, com destaque especial para a subcategoria *modificabilidade*, que obteve o maior número de citações entre todas as subcategorias analisadas. Este resultado ressalta a natureza fundamental dos projetos *open-source* de serem facilmente adaptáveis para correção de falhas, implementação de novas funcionalidades e evolução contínua do software, permitindo a colaboração distribuída entre diversos contribuidores.

- A categoria **open-source** também se demonstra crucial, com especial relevância para as subcategorias *desenvolvimento colaborativo* e *suporte ao feedback*. Essas subcategorias facilitam a contribuição de múltiplos desenvolvedores e a coleta de *feedbacks* dos usuários, elementos essenciais para a identificação de falhas e necessidades de melhoria no software. Nota-se, contudo, que a subcategoria *adoção de licença*, embora importante, recebeu menor atenção nos estudos analisados, sugerindo uma possível negligência deste aspecto fundamental no ecossistema *open-source*.

- A **capacidade de interação** posiciona-se como outra categoria de grande importância, com especial ênfase no *engajamento do usuário*, aspecto central para proporcionar ludicidade e imersão no contexto de jogos digitais. As subcategorias *capacidade de aprendizagem* e *operabilidade* também emergem como fatores significativos, facilitando a experiência do jogador e permitindo um melhor aproveitamento do software.

- A significativa predominância da subcategoria *apropriação funcional* sobre a *corretude funcional* na categoria de **adequação funcional** indica que, embora os estudos priorizem a adequação do software às necessidades do usuário (sejam elas lúdicas ou educacionais), a corretude das funcionalidades implementadas pode não receber a mesma ênfase. Esta disparidade pode dever-se a uma menor priorização deste aspecto durante o relato dos projetos desenvolvidos.

- As categorias **safety** e **security** apresentam-se como as menos citadas nos estudos analisados. Esta relevância limitada pode ser atribuída à natureza específica dos jogos

digitais: a maioria não requer acesso a recursos externos críticos do ambiente de execução, diminuindo a necessidade de preocupações com *safety*. Da mesma forma, para jogos que não possuem componentes online ou não lidam com dados sensíveis dos jogadores, aspectos de *security* como a garantia de que dados locais (ex.: progresso do jogo) permaneçam inalterados e íntegros tornam-se menos proeminentes.

Tabela 4.2: *Relevância dos Requisitos do Sistema por Estudo*

Categoria	**Subcategoria**	**Estudos**
Adequação Funcional	Corretude Funcional	E. Raymond (1999) e Da Silva *et al.* (2021)
	Apropriação Funcional	E. Raymond (1999), Farrapo *et al.* (2022), Schlagowski *et al.* (2024), Geldhauser *et al.* (2022), Da Silva *et al.* (2021), Wagle *et al.* (2021), Salgo *et al.* (2021), Gokcen *et al.* (2018) e Habernal *et al.* (2017)
Eficiência de *Performance*	Comportamento do Tempo	Wu e Lin (2001)
	Utilização de Recursos	
	Capacidade	
Compatibilidade	Coexistência	Wu e Lin (2001)
	Interoperabilidade	Wu e Lin (2001)
Capacidade de Interação	Adequação Reconhecível	E. Raymond (1999) e Wagle *et al.* (2021)
	Capacidade de Aprendizagem	Farrapo *et al.* (2022), Geldhauser *et al.* (2022), Wagle *et al.* (2021) e Salgo *et al.* (2021)
	Operabilidade	E. Raymond (1999), Farrapo *et al.* (2022), Geldhauser *et al.* (2022), Wagle *et al.* (2021) e Salgo *et al.* (2021)
	Engajamento do Usuário	Farrapo *et al.* (2022), Schlagowski *et al.* (2024), Geldhauser *et al.* (2022), Da Silva *et al.* (2021), Wagle *et al.* (2021), Salgo *et al.* (2021), Gokcen *et al.* (2018) e Habernal *et al.* (2017)

Continua na próxima página

Tabela 4.2 (Continuação)

Categoria	Subcategoria	Estudos
	Inclusividade	Geldhauser *et al.* (2022), Da Silva *et al.* (2021), Wagle *et al.* (2021) e Habernal *et al.* (2017)
	Assistência ao Usuário	Wagle *et al.* (2021)
	Auto-descritividade	Da Silva *et al.* (2021)
Confiabilidade	Ausência de Falhas	Wu e Lin (2001), E. Raymond (1999), Aberdour (2007) e Fuggetta (2003)
	Disponibilidade	Gokcen *et al.* (2018) e Habernal *et al.* (2017)
	Tolerância a Falhas	E. Raymond (1999)
	Recuperabilidade	E. Raymond (1999)
Security	Confidencialidade	E. Raymond (1999) e Habernal *et al.* (2017)
	Integridade	
	Ausência de repúdio	
	Autenticidade	Habernal *et al.* (2017)
	Resistência	
Manutenibilidade	Modularidade	Aberdour (2007), Farrapo *et al.* (2022), Wilson Kazuo Mizutani e Kon (2024) e Habernal *et al.* (2017)
	Reusabilidade	Wu e Lin (2001), E. Raymond (1999) e Da Silva *et al.* (2021)
	Analisabilidade	Wu e Lin (2001), E. Raymond (1999), Aberdour (2007), Fuggetta (2003) e Wilson Kazuo Mizutani e Kon (2024)
	Modificabilidade	Wu e Lin (2001), E. Raymond (1999), Aberdour (2007), Fuggetta (2003), Farrapo *et al.* (2022), Wilson Kazuo Mizutani e Kon (2024), Da Silva *et al.* (2021), Wagle *et al.* (2021), Salgo *et al.* (2021) e Habernal *et al.* (2017)

Continua na próxima página

Tabela 4.2 (Continuação)

Categoria	Subcategoria	Estudos
	Testabilidade	E. Raymond (1999), Aberdour (2007), Fuggetta (2003), Da Silva *et al.* (2021) e Habernal *et al.* (2017)
Flexibilidade	Adaptabilidade	Wu e Lin (2001), Schlagowski *et al.* (2024), Geldhauser *et al.* (2022) e Habernal *et al.* (2017)
	Escalabilidade	E. Raymond (1999) e Farrapo *et al.* (2022)
Safety	Limitação Operacional	
	Identificação de Riscos	
	Falha Segura	
	Aviso de Perigo	
Open Source	Adoção de Licença	Wu e Lin (2001), Wilson Kazuo Mizutani e Kon (2024), Geldhauser *et al.* (2022), Da Silva *et al.* (2021) e Habernal *et al.* (2017)
	Desenvolvimento colaborativo	Wu e Lin (2001), E. Raymond (1999), Aberdour (2007), Fuggetta (2003), Da Silva *et al.* (2021), Wagle *et al.* (2021), Salgo *et al.* (2021), Gokcen *et al.* (2018) e Habernal *et al.* (2017)
	Suporte ao feedback	Wu e Lin (2001), E. Raymond (1999), Aberdour (2007), Fuggetta (2003), Schlagowski *et al.* (2024), Geldhauser *et al.* (2022), Da Silva *et al.* (2021), Wagle *et al.* (2021), Gokcen *et al.* (2018) e Habernal *et al.* (2017)

Identificação dos Jogos *Open Source*

Nesta análise, os jogos analisados foram divididos em duas categorias:

- **Jogos open-source de pequeno porte**: Desenvolvidos por poucos desenvolvedores, sem comunidade ativa de contribuidores, majoritariamente desenvolvidos no contexto acadêmico.
- **Jogos open-source de grande porte**: Possuem comunidade ativa de desenvolvimento, mantidos por um número significativo de colaboradores (critérios: $\geq$10k stars ou $\geq$300 contribuidores no GitHub).

Análise dos Jogos *Open-Source* de Pequeno Porte

Para complementar a análise da literatura científica, foram avaliados nove jogos *open-source* de pequeno porte, mapeando a presença dos requisitos não funcionais previamente identificados. A Tabela 4.3 detalha a frequência de cada subcategoria nos projetos analisados, identificados pelos índices SG-1 a SG-9. A análise comparativa revela os seguintes aspectos:

- A categoria **adequação funcional** apresenta um comportamento distinto entre suas subcategorias. Enquanto a *apropriação funcional* está presente em todos os projetos, garantindo que as funcionalidades implementadas atendam às necessidades de experiência do jogador, a *corretude funcional* não é universalmente atingida. Esta disparidade indica que, embora os jogos busquem fornecer conteúdo completo, erros persistentes podem comprometer a experiência, sugerindo uma priorização do funcional sobre o rigoroso em projetos de menor escala.
- A **eficiência de *performance*** e todas as suas subcategorias (*comportamento do tempo*, *utilização de recursos* e *capacidade*) são consideradas como universalmente satisfeitas. Este resultado é atribuído à natureza menos demandante desses projetos, que geralmente são direcionados a um único jogador e não requerem grande poder computacional, simplificando a otimização de *performance*.
- A categoria **compatibilidade** apresenta uma divisão clara. A *coexistência* é amplamente atendida em jogos que se executam em ambiente local devido à baixa demanda de recursos, enquanto a *interoperabilidade*, relativa à eficiência na troca de informações com outros sistemas, é observada em apenas um projeto. Este resultado reflete a característica predominante de jogos independentes, que frequentemente operam como sistemas isolados.
- A **capacidade de interação** emerge como uma categoria prioritária, com destaque para o *engajamento do usuário*, aspecto fundamental para jogos. A *operabilidade* também recebe atenção significativa, assegurando controles intuitivos. Em contraste, a *capacidade de aprendizagem* é relativamente negligenciada, indicando uma possível carência de tutoriais ou mecanismos que facilitem a curva de aprendizado. As subcategorias *inclusividade* e *assistência ao usuário* são as menos implementadas, sugerindo que funcionalidades avançadas de acessibilidade não são o foco principal desses projetos.

- As categorias **security** e **safety** confirmam a tendência identificada na literatura, sendo completamente negligenciadas na prática pelos jogos *open-source* de pequeno porte. Esta ausência corrobora a premissa de que, na ausência de componentes online ou manipulação de dados sensíveis, requisitos de segurança e salvaguarda tornam-se menos críticos no contexto desses jogos.
- A **manutenibilidade** demonstra alta relevância, com a *modificabilidade* sendo a subcategoria mais prevalente, reforçando a necessidade de código adaptável em projetos *open-source*. *modularidade*, *reusabilidade* e *analisabilidade* também recebem atenção considerável. Entretanto, a *testabilidade* é drasticamente negligenciada, apontando para uma grave carência de testes automatizados e práticas de garantia de qualidade robustas nestes ecossistemas.
- Na categoria **flexibilidade**, observa-se que a *adaptabilidade* é frequentemente atendida. Um fator crucial que influencia este resultado é o uso de *engines* de jogo, que fornecem ferramentas intrínsecas para compilação multiplataforma e implantação. Projetos desenvolvidos sem o suporte de tais *engines* tendem a suportar uma única plataforma, limitando sua adaptabilidade. A *escalabilidade* não é um requisito considerado, alinhando-se com o escopo limitado destes projetos.
- A categoria **open-source** exibe aderência quase total nas subcategorias *desenvolvimento colaborativo* e *suporte ao feedback*, reafirmando a natureza colaborativa inerente a estes projetos. Contudo, a *adoção de licença* não é universal, indicando que, apesar de publicamente acessíveis, uma parcela significativa dos projetos negligencia a formalização legal das licenças de uso e distribuição.

Em síntese, os jogos *open-source* de pequeno porte priorizam requisitos fundamentais para a experiência básica de jogo e a colaboração no desenvolvimento, como **funcionalidade**, ***performance***, **interação** e aspectos práticos de **manutenibilidade**. Em contrapartida, aspectos relacionados à robustez avançada como acessibilidade (**assistência ao usuário** e **inclusividade**), segurança (**security** e **safety**) e a garantia de qualidade formalizada (**testabilidade**) recebem significativamente menos atenção. Este perfil reflete as restrições de escopo, recursos e contexto operacional típicos destes projetos.

Tabela 4.3: *Requisitos em Jogos Open-Source de Pequeno Porte*

Categoria	Subcategoria	Índice
Adequação Funcional	Correção Funcional	SG-1, SG-4, SG-5, SG-6, SG-7, SG-8, SG-9
	Apropriação Funcional	SG-1, SG-2, SG-3, SG-4, SG-5, SG-6, SG-7, SG-8, SG-9
Eficiência de *Performance*	Comportamento do Tempo	SG-1, SG-2, SG-3, SG-4, SG-5, SG-6, SG-7, SG-8, SG-9
	Utilização de Recursos	SG-1, SG-2, SG-3, SG-4, SG-5, SG-6, SG-7, SG-8, SG-9

Continua na próxima página

Tabela 4.3 (Continuação)

Categoria	**Subcategoria**	**Índice**
	Capacidade	SG-1, SG-2, SG-3, SG-4, SG-5, SG-6, SG-7, SG-8, SG-9
Compatibilidade	Coexistência	SG-1, SG-2, SG-3, SG-4, SG-5
	Interoperabilidade	SG-9
Capacidade de Interação	Adequação Reconhecível	SG-3, SG-4, SG-5
	Capacidade de Aprendizagem	SG-2, SG-4, SG-7, SG-9
	Operabilidade	SG-1, SG-2, SG-3, SG-4, SG-5, SG-6, SG-8
	Engajamento do Usuário	SG-1, SG-2, SG-3, SG-4, SG-5, SG-6, SG-7, SG-9
	Inclusividade	SG-6, SG-9
	Assistência ao Usuário	SG-4, SG-9
	Auto-descritividade	SG-1, SG-2, SG-4, SG-5, SG-6, SG-7, SG-8
Confiabilidade	Ausência de Falhas	SG-1, SG-4, SG-5, SG-6, SG-7, SG-8, SG-9
	Disponibilidade	SG-2, SG-3, SG-4, SG-5, SG-6, SG-9
	Tolerância a Falhas	
	Recuperabilidade	
Security	Confidencialidade	
	Integridade	
	Ausência de repúdio	
	Autenticidade	
	Resistência	
Manutenibilidade	Modularidade	SG-1, SG-2, SG-3, SG-4, SG-9
	Reusabilidade	SG-1, SG-2, SG-4, SG-5, SG-9
	Analisabilidade	SG-2, SG-3, SG-4, SG-7, SG-8, SG-9
	Modificabilidade	SG-1, SG-2, SG-3, SG-4, SG-5, SG-7, SG-8, SG-9
	Testabilidade	SG-1
Flexibilidade	Adaptabilidade	SG-1, SG-2, SG-3, SG-4, SG-9
	Escalabilidade	
Safety	Limitação Operacional	
	Identificação de Riscos	
	Falha Segura	
	Aviso de Perigo	

Continua na próxima página

Tabela 4.3 (Continuação)

Categoria	Subcategoria	Índice
Open Source	Adoção de Licença	SG-3, SG-4, SG-5, SG-6, SG-7, SG-8
	Desenvolvimento colaborativo	SG-1, SG-2, SG-3, SG-4, SG-5, SG-6, SG-7, SG-8, SG-9
	Suporte ao feedback	SG-1, SG-2, SG-3, SG-4, SG-5, SG-6, SG-7, SG-8, SG-9

Tabela 4.4: *Lista de Jogos Open Source de pequeno porte e seus Índices*

Nome	Repositório	Índice
My Homie the Gnomie	GitLab	SG-1
Wizard P	GitHub	SG-2
Beto & Brito	GitLab	SG-3
Charge Kid	GitLab	SG-4
Backdoor Route	GitHub	SG-5
gradient descent	GitHub	SG-6
Pixel Python RPG	GitHub	SG-7
Madly Ambiguous	GitHub	SG-8
Avaler's Adventure	GitHub	SG-9

Análise dos Jogos *Open-Source* de Grande Porte

A análise de 5 jogos *open-source* de grande porte (jogos com ≥10k stars ou ≥300 contribuidores no GitHub) apresentada na Tabela 4.5 revela um perfil de maturidade significativamente distinto quando comparado aos projetos de pequeno porte, refletindo os recursos, a complexidade e as demandas de uma base de usuários ampla e de uma comunidade ativa de desenvolvimento.

- A categoria **adequação funcional** demonstra maturidade superior. Tanto a *corretude funcional* quanto a *apropriação funcional* são universalmente atendidas. Este resultado indica que projetos de grande porte, sustentados por comunidades robustas, conseguem priorizar e atingir um alto padrão de qualidade tanto na completude das funcionalidades quanto na ausência de erros, equilibrando o funcional com o rigoroso.
- Assim como nos projetos menores, a **eficiência de *performance*** e todas as suas subcategorias são plenamente satisfeitas. No entanto, o contexto é fundamentalmente diferente: aqui, a otimização é algo notável, considerando a complexidade visual, mecânica e, frequentemente, *multiplayer* desses jogos, que demandam gestão eficiente de recursos computacionais significativos.

- A categoria **compatibilidade** apresenta uma aderência mais expressiva. A *coexistência* é majoritariamente atendida, com exceção de um jogo Web (LG-3) que não executa no ambiente local, enquanto a *interoperabilidade* está presente em todos os projetos que envolvem funcionalidade on-line. Este é um forte indicativo de que jogos de grande porte evoluem para além de sistemas isolados, frequentemente requerendo e implementando interfaces de comunicação com serviços externos.
- Na **capacidade de interação**, observa-se uma cobertura mais ampla e consistente. *adequação reconhecível*, *operabilidade* e *engajamento do usuário* são universais, refletindo um profundo investimento na experiência do usuário final. A *capacidade de aprendizagem* recebe mais atenção do que nos projetos pequenos, mas ainda não é prioritária devido à complexidade de alguns dos jogos. As subcategorias *inclusividade* e *assistência ao usuário* permanecem como as menos implementadas, sugerindo que mesmo projetos grandes ainda negligenciam aspectos avançados de acessibilidade.
- A categoria **confiabilidade** mostra solidez nas subcategorias básicas de *ausência de falhas* e *disponibilidade*, essenciais para manter a base de jogadores.
- **security** emerge como uma distinção crucial. Enquanto projetos pequenos a ignoraram completamente, aqui há uma presença inicial, porém incipiente: *autenticidade* e *ausência de repúdio*, e *confidencialidade* e *integridade*. Isto está alinhado com a natureza destes projetos, que muitas vezes envolvem contas de usuário e componentes online, tornando requisitos de segurança não apenas relevantes, mas necessários.
- A **manutenibilidade** é claramente uma prioridade máxima, com *modularidade* e *modificabilidade* sendo universais. *reusabilidade* e *analisabilidade* também são altamente prevalentes. Notavelmente, a *testabilidade* recebe significativamente mais atenção do que em projetos pequenos, refletindo a adoção de práticas de integração contínua e testes automatizados como requisito para a escala e colaboração massivas nestes ecossistemas.
- Na **flexibilidade**, a *adaptabilidade* é majoritariamente atendida mesmo sem o uso dos *engines* modernos. Diferente dos projetos pequenos, a *escalabilidade* faz sua aparição, sendo um requisito considerado para alguns projetos que precisam suportar um grande número de jogadores concorrentes.
- Assim como na análise anterior, a categoria **safety** é completamente negligenciada, confirmando que requisitos de salvaguarda física não são uma preocupação no domínio de jogos digitais analisados.
- Finalmente, a categoria **open-source** exibe aderência total e irrestrita em todas as suas subcategorias, incluindo a universal *adoção de licença*. Isto corrobora a tese de que projetos de grande porte operam com um alto grau de profissionalismo e maturidade em suas operações de comunidade, entendendo a licença como um pilar fundamental para o desenvolvimento colaborativo, o uso e a distribuição.

Em síntese, os jogos *open-source* de grande porte consideram mais características de qualidade. Eles não apenas consideram os requisitos fundamentais de **funcionalidade**, ***performance*** e **interação**, mas também demonstram avanços significativos em **manutenibilidade** (especialmente em testabilidade), **compatibilidade** e dão os primeiros

passos críticos em **security**. Este perfil reflete as demandas de projetos complexos, com grandes comunidades de usuários e desenvolvedores, onde a robustez, a escalabilidade e a colaboração estruturada tornam-se requisitos não funcionais tão importantes quanto as funcionalidades do jogo em si. A principal lacuna que persiste, assim como nos projetos menores, está nas áreas de **acessibilidade** avançada (*inclusividade* e *assistência ao usuário*).

Tabela 4.5: *Requisitos em Jogos Open-Source de Grande Porte*

Categoria	**Subcategoria**	**Índice**
Adequação Funcional	Correção Funcional	LG-1, LG-2, LG-3, LG-4, LG-5
	Apropriação Funcional	LG-1, LG-2, LG-3, LG-4, LG-5
Eficiência de *Performance*	Comportamento do Tempo	LG-1, LG-2, LG-3, LG-4, LG-5
	Utilização de Recursos	LG-1, LG-2, LG-3, LG-4, LG-5
	Capacidade	LG-1, LG-2, LG-3, LG-4, LG-5
Compatibilidade	Coexistência	LG-1, LG-2, LG-3, LG-5
	Interoperabilidade	LG-1, LG-2, LG-3
Capacidade de Interação	Adequação Reconhecível	LG-1, LG-2, LG-3, LG-4, LG-5
	Capacidade de Aprendizagem	LG-1, LG-2, LG-4
	Operabilidade	LG-1, LG-2, LG-3, LG-4, LG-5
	Engajamento do Usuário	LG-1, LG-2, LG-3, LG-4, LG-5
	Inclusividade	LG-1, LG-2
	Assistência ao Usuário	LG-2
	Auto-descritividade	LG-1, LG-2, LG-3, LG-4
Confiabilidade	Ausência de Falhas	LG-1, LG-2, LG-3, LG-4, LG-5
	Disponibilidade	LG-1, LG-2, LG-3, LG-4, LG-5
	Tolerância a Falhas	
	Recuperabilidade	
Security	Confidencialidade	LG-2
	Integridade	LG-2
	Ausência de repúdio	LG-1, LG-2, LG-3
	Autenticidade	LG-1, LG-2, LG-3
	Resistência	
Manutenibilidade	Modularidade	LG-1, LG-2, LG-3, LG-4, LG-5
	Reusabilidade	LG-1, LG-2, LG-3, LG-4
	Analisabilidade	LG-1, LG-2, LG-3, LG-4

Continua na próxima página

Tabela 4.5 (Continuação)

Categoria	Subcategoria	Índice
	Modificabilidade	LG-1, LG-2, LG-3, LG-4, LG-5
	Testabilidade	LG-1, LG-2, LG-3
Flexibilidade	Adaptabilidade	LG-1, LG-2, LG-3, LG-5
	Escalabilidade	LG-1, LG-3
Safety	Limitação Operacional	
	Identificação de Riscos	
	Falha Segura	
	Aviso de Perigo	
Open Source	Adoção de Licença	LG-1, LG-2, LG-3, LG-4, LG-5
	Desenvolvimento colaborativo	LG-1, LG-2, LG-3, LG-4, LG-5
	Suporte ao feedback	LG-1, LG-2, LG-3, LG-4, LG-5

Tabela 4.6: *Lista de Jogos Open Source de grande porte e seus Índices*

Nome	Repositório	Índice
Mindustry	GitHub	LG-1
Osu!	GitHub	LG-2
OpenRA	GitHub	LG-3
2048	GitHub	LG-4
Craft	GitHub	LG-5

4.3 Conclusão de Requisitos do Sistema

Com base nas análises realizadas anteriormente, tanto da literatura acadêmica quanto da avaliação prática de jogos *open-source* existentes, os requisitos de qualidade listados abaixo são selecionados para compor a RA. A seleção foi pautada por dois critérios principais: alta frequência de menção nos estudos acadêmicos (Tabela 4.2) e alta prevalência de implementação bem-sucedida nos jogos de grande e pequeno porte analisados (Tabelas 4.3 e 4.5).

4.3.1 Requisitos Essenciais

- **Capacidade de interação**: Esta categoria é central para a experiência do jogador, com destaque para o ***engajamento do usuário***, sendo o cerne da experiência lúdica que motiva o jogador a continuar. A *capacidade de aprendizagem* é valorizada, embora sua implementação prática seja variável, indicando a necessidade de mecanismos que

facilitem a curva de aprendizado. A *operabilidade* é frequentemente implementada, assegurando controles intuitivos e precisos. E por fim, a *auto-descritividade* possui presença significativa, garantindo que a interface e os objetivos do jogo sejam claros e compreensíveis para o usuário.

- **Manutenibilidade**: Esta categoria demonstra alta relevância geral, com destaque especial para a ***modificabilidade***. A alta prevalência desta subcategoria reforça a natureza intrínseca de projetos *open-source*, que devem ser facilmente adaptáveis e evolutivos para acomodar contribuições contínuas da comunidade, correções e a adição de novas funcionalidades.

- **Open source**: As subcategorias desta categoria, *adoção de licença*, *desenvolvimento colaborativo* e *suporte ao feedback*, são amplamente reconhecidas como pilares essenciais. Elas formalizam os aspectos legais, operacionais e sociais que viabilizam o modelo de desenvolvimento aberto, permitindo a colaboração transparente, a evolução comunitária e a sustentabilidade a longo prazo do projeto.

4.3.2 Requisitos Opcionais

- **Adequação funcional**: A subcategoria *apropriação funcional* está comumente presente em projetos e estudos acadêmicos, ressaltando a importância de que as funcionalidades do jogo permitam que o jogador experimente todo o seu conteúdo pretendido. A *corretude funcional* também deve ser considerada; embora menos enfatizada na literatura, sua forte presença prática nos jogos estudados demonstra a necessidade crítica de que as funcionalidades sejam implementadas de maneira correta e confiável para garantir a experiência do usuário.

- **Eficiência de *performance***: As três subcategorias, *comportamento do tempo*, *utilização de recursos* e *capacidade*, são virtualmente universais nos projetos analisados. Esta prevalência destaca a importância fundamental de um desempenho otimizado, que garante uma experiência de jogo fluida, responsiva e acessível mesmo em hardware mais limitado, sendo um requisito básico para a aceitação do software.

- **Compatibilidade**: A subcategoria *coexistência* é amplamente atendida, especialmente em ambientes locais. Sua relevância reside na necessidade do jogo operar de forma estável junto a outros softwares no sistema do usuário, um aspecto crucial para a usabilidade prática e a adoção do jogo em diversos contextos e configurações.

- **Confiabilidade**: As subcategorias *ausência de falhas* e *disponibilidade* são majoritariamente atendidas nos projetos. Sua importância é fundamental para proporcionar uma experiência de jogo consistente e estável, prevenindo frustrações causadas por bugs, travamentos ou indisponibilidade do sistema, que podem comprometer totalmente a experiência do usuário.

- **Flexibilidade**: A subcategoria *adaptabilidade* é comumente satisfeita, porém com uma menor importância. Esse requisito assegura que o jogo possa ser instalado e executado em diferentes ambientes e plataformas, aumentando seu alcance.

4.4 Considerações Finais

Este capítulo apresentou a análise arquitetural conduzida segundo o passo RA-2 do método ProSA-RA, com o objetivo principal de elicitar e consolidar os requisitos arquiteturais que fundamentarão a RA proposta para jogos digitais *open-source*. Para tal, foi executado um processo sistemático que partiu da identificação de 52 requisitos do sistema, extraídos de estudos acadêmicos existentes e da experiência dos especialistas, e avançou para sua síntese e avaliação crítica.

A síntese dos RS, categorizados conforme o padrão ISO/IEC 25010 complementado pela categoria específica *open source*, permitiu uma visão organizada das características de qualidade relevantes para o domínio. A subsequente análise de relevância, que cruzou a frequência de menção na literatura especializada com a prevalência de implementação em uma amostra diversificada de jogos *open-source* (de pequeno e grande porte), foi crucial para priorizar e contextualizar esses requisitos. Esta análise dual revelou tanto consensos quanto lacunas entre a teoria acadêmica e a prática corrente de desenvolvimento.

Como principal resultado, a análise permitiu a distinção entre requisitos essenciais e requisitos opcionais para a RA. Os requisitos essenciais – capacidade de interação, em especialmente o engajamento do usuário, manutenibilidade, com foco em modificabilidade e os pilares da categoria *open source* – emergiram como o núcleo não negociável da arquitetura, refletindo as necessidades primárias de experiência do usuário e do modelo colaborativo de desenvolvimento. Os requisitos opcionais, que são adequação funcional, eficiência de *performance*, compatibilidade, confiabilidade e flexibilidade, embora altamente prevalentes, foram identificados como características cuja implementação pode variar conforme o escopo e os recursos específicos de cada projeto derivado da RA.

A análise também evidenciou áreas que recebem menor atenção tanto na literatura quanto na prática, nomeadamente *security*, *safety* e aspectos avançados de acessibilidade (inclusividade e assistência ao usuário). Esta constatação serve como um valioso indicativo de pontos que podem ser alvo de evolução futura tanto da RA quanto dos projetos do domínio.

Em síntese, este capítulo cumpriu seu objetivo ao fornecer uma base sólida, empiricamente fundamentada e priorizada de requisitos arquiteturais. Os resultados aqui consolidados serão diretamente empregados no capítulo subsequente, onde serão transformados nos elementos estruturais e nas decisões de design que comporão a RA propriamente dita, garantindo que ela reflita tanto as melhores práticas acadêmicas quanto as realidades e necessidades do ecossistema de desenvolvimento de jogos *open-source*.

Capítulo 5

Síntese Arquitetural

No passo RA-3 do ProSA-RA, os requisitos arquiteturais identificados anteriormente servirão de base para a construção da arquitetura de referência. Para implementar uma arquitetura que atenda adequadamente aos requisitos de qualidade, foi necessária a avaliação de padrões arquiteturais existentes, os quais serão empregados na composição da RA. Em seguida, para representar visualmente a RA com o objetivo de facilitar o entendimento dos *stakeholders*, são elaboradas diversas visões arquiteturais. Essas visões representam a arquitetura construída a partir de diferentes perspectivas, garantindo uma representação mais completa e abrangente.

Este capítulo será organizado da seguinte forma: a Seção 5.1 apresenta a análise crítica de arquiteturas existentes na indústria de jogos, identificando seus respectivos prós, contras e *trade-offs* ao utilizá-las, fornecendo uma base para a construção posterior da Arquitetura de Referência; a Seção 5.2 argumenta a seleção e justificativa da arquitetura base; a Seção 5.3 apresenta a definição e a apresentação das visões que materializam a RA proposta, utilizando o modelo de visões 4+1; e, por fim, a Seção 5.4 discute as considerações finais deste capítulo.

5.1 Análise de Padrões Arquiteturais

Esta seção avalia padrões arquiteturais consolidados na indústria de software com potencial aplicação ao desenvolvimento de jogos digitais *open-source*. Para cada arquitetura, são apresentados: (1) sua estrutura fundamental, (2) vantagens e limitações no contexto de jogos e (3) aderência aos Requisitos Arquiteturais (RA) definidos no Capítulo 1. A análise comparativa será realizada com uma matriz de avaliação (Tabela 5.1) que quantifica o atendimento aos RA.

5.1.1 Entidade-Componente-Sistema (ECS)

O padrão *Entity Component System* é utilizado predominantemente na indústria de jogos digitais e frequentemente implementado utilizando o *design* orientado a dados (Voisard *et al.*, 2025). Sua estrutura fundamenta-se em três blocos conceituais:

- **Entidade (Entity)**: Identificador único que agrega componentes e dados relacionados.
- **Componente (Component)**: Estruturas de dados que representam aspectos específicos das entidades, contém apenas os dados necessários para as suas transformações relacionadas.
- **Sistema (System)**: Classes que implementam lógica de processamento dos componentes relacionados. Cada sistema deve possuir responsabilidades únicas independentemente da sua complexidade.

Este padrão arquitetural oferece vantagens como: **Desempenho** via *arrays* contíguos (SOA) que otimizam localidade de cache e paralelismo; **Capacidade de Interação** pela manipulação dinâmica de componentes nas entidades do software, e **Manutenibilidade** pelo baixo acoplamento entre as sistemas e componentes. Como desvantagem, está presente a alta curva de aprendizagem devido ao paradigma distinto da programação orientada ao objeto (POO), além do bom entendimento das operações internas do computador (Pouhela *et al.*, 2023).

5.1.2 Arquitetura em Camadas (Layered architecture)

A Arquitetura em Camadas é um padrão que organiza o sistema em grupos distintos de subtarefas em diferentes níveis de abstração (Belle *et al.*, 2021). Seu princípio central é a separação de preocupações, em que cada camada fornece serviços para a camada superior e age como cliente para a camada inferior, promovendo um *design* modular e hierárquico (Rahmati e Tanhaei, 2021; Belle *et al.*, 2021; Kempkens *et al.*, 2000). Sua estrutura fundamenta-se em três blocos conceituais:

- **Camadas (Layers)**: Agrupamentos funcionais de componentes com responsabilidades coesas e um nível de abstração similar.
- **Contrato de Interface**: Definição formal e estável dos serviços que uma camada oferece à camada superior, podendo esconder completamente os detalhes de sua implementação interna e promovendo o baixo acoplamento (*Black Box*), revelar toda a sua estrutura interna à camada superior (*White Box*) ou revelar uma parte das informações que possui (*Gray Box*).
- **Fluxo Hierárquico**: A comunicação e o fluxo de dados e requisições ocorrem principalmente entre camadas adjacentes ou na mesma camada, evitando comunicações com outros níveis. Uma camada N só pode utilizar os serviços das camadas inferiores a ela e deve oferecer serviços para camadas superiores.

Este padrão arquitetural oferece vantagens como: **Manutenibilidade** pela localização e isolamento de mudanças dentro de uma camada específica, facilitando a evolução do sistema, e pela possibilidade de testar camadas individualmente através de *mocks* ou *stubs* que simulam camadas adjacentes; e **Flexibilidade** ao permitir múltiplas interfaces na mesma camada para usuários finais (Rahmati e Tanhaei, 2021). Como desvantagens, destacam-se a **Ineficiência** inerente à necessidade de uma requisição passar por múltiplas camadas para ser processada, incorrendo em *overhead*; e a **Dificuldade de Design** na definição clara das responsabilidades, granularidade e interfaces de cada camada, podendo levar a

um acoplamento excessivo entre elas ou a camadas muito finas/grossas que prejudicam a *performance* ou a clareza (Belle *et al.*, 2021; Kempkens *et al.*, 2000).

5.1.3 Modelo-Visão-Controlador (MVC)

O padrão *Model-View-Controller*, originado no ambiente Smalltalk-80 (Krasner, Pope *et al.*, 1988), estabelece uma separação de responsabilidades para aplicações interativas, sendo amplamente adotado no desenvolvimento de softwares que possuem múltiplas interfaces de usuário (Deacon, 2009; Frank *et al.*, 1996). Sua estrutura fundamenta-se em três componentes inter-relacionados:

- **Modelo (Model)**: Encapsula os dados centrais e regras de negócio da aplicação, permanecendo independente da representação visual ou de mecanismos de interação.
- **Visão (View)**: Responsável pela apresentação visual dos dados do Modelo ao usuário, podendo existir múltiplas visões para um mesmo modelo.
- **Controlador (Controller)**: Gerencia a interação do usuário, traduzindo inputs em ações sobre o Modelo e coordenando atualizações da Visão.

Este padrão arquitetural oferece vantagens como: **Reusabilidade** de modelos pelas diferentes visões; **Portabilidade** do sistema nas diferentes plataformas; e **Flexibilidade** de permitir a substituição da visão na hora de execução. Como desvantagens, destacam-se a **Dependência** da visão e do controlador do modelo e a **Ineficiência** do acesso a dados pela visão quando há alta demanda de interações (Frank *et al.*, 1996).

5.1.4 Microkernel

A arquitetura *Microkernel* estrutura sistemas operacionais e aplicações complexas em torno de um núcleo mínimo (*kernel*) que provê serviços essenciais, enquanto funcionalidades estendidas são delegadas a módulos externos (*plug-ins*) (Thomas, 2024; Liu *et al.*, 2021; Frank *et al.*, 1996). Sua organização fundamenta-se em três blocos conceituais:

- **Núcleo Mínimo (Microkernel)**: Implementa a funcionalidade mínima e central do sistema, gerenciando e controlando o fluxo de acesso aos recursos do sistema.
- **Plug-ins (Servidores)**: Componentes responsáveis por funcionalidades estendidas do sistema, com características de independência entre si e acoplamento apenas pela interface oferecida pelo *microkernel*.
- **Mecanismo de Comunicação**: Interface padronizada que define os métodos de comunicação entre plug-ins e *microkernel*.

Esta arquitetura oferece vantagens como: **Manutenibilidade** na adição/remoção de plug-ins sem modificar o núcleo; **Flexibilidade**, por precisar adaptar apenas o núcleo mínimo a diferentes hardwares; e **Confiabilidade** via isolamento de falhas (plug-ins falham sem comprometer o núcleo). Como desvantagens, destacam-se o **Desempenho** inferior em comparação com sistemas monolíticos e a **Complexidade de Design e Implementação** durante a análise e desenvolvimento do sistema (Frank *et al.*, 1996; Liu *et al.*, 2021).

5.1.5 Arquitetura Orientada a Eventos (EDA)

A *Event-Driven Architecture* é um padrão arquitetural assíncrono que estrutura sistemas em torno da produção, detecção, consumo e reação a eventos (Michelson, 2006), operando através de comunicação indireta e descentralizada. Neste modelo, componentes emissores (*producers*) publicam eventos sem conhecimento prévio dos consumidores (*consumers*), promovendo desacoplamento do sistema (Cabane e Farias, 2024). Sua estrutura fundamenta-se em quatro camadas essenciais:

- **Geradores de Eventos (*Event Generators*)**: Componentes responsáveis pela geração de eventos quando ocorre uma mudança de estado no sistema. Cada evento é composto por *headers* (metadados) e *body* (dados contextuais), seguindo um padrão comum para promover interoperabilidade.
- **Canais de Eventos (*Event Channels*)**: Canais que transportam eventos entre geradores, motores de processamento e assinantes, garantindo entrega assíncrona.
- **Processamento (*Event Processing*)**: Camada onde eventos são avaliados por motores de processamento especializados, que disparam ações conforme regras pré-definidas. Divide-se em três modalidades:
  - *Simples*: Reação imediata a eventos únicos.
  - *Stream*: Análise contínua de fluxos de eventos, a qual será transmitida aos assinantes da informação correspondente.
  - *Complexo (CEP)*: Realiza uma análise de correlação entre múltiplos eventos antes de decidir a ação que será executada.
- **Atividades Acionadas a Jusante (*Downstream Event-Driven Activity*)**: Ações resultantes do processamento, como invocação de serviços, notificações ou disparo de processos de negócio, executadas via *push* (acionadas pelo motor) ou *pull* (solicitadas por assinantes).

Entre as principais vantagens destacam-se: **Modularidade e Reusabilidade** via independência dos componentes do sistema; e ***Performance*** trazida por permitir a execução assíncrona. Contudo, apresenta desvantagens como **Complexidade** do sistema devido ao design da comunicação assíncrona e à circulação de grande número de eventos; e **Falta de Confiabilidade** devido à dificuldade de teste trazida pela natureza assíncrona dos componentes e à interação complexa dos eventos (Cabane e Farias, 2024; Manchana, 2021).

5.2 Seleção dos Padrões Arquiteturais

Baseando-se nas análises anteriores, juntamente com estudos conduzidos por Bi *et al.* (2018) e Haoues *et al.* (2017), além da experiência dos especialistas da área, é possível identificar os pontos positivos e negativos de cada arquitetura, mostrados na Tabela 5.1. Nela, a qualidade de cada padrão em relação aos requisitos é dividida em quatro categorias: Alto, indicando uma vantagem no requisito correspondente; Baixo, indicando uma desvantagem; Médio, indicando uma situação neutra; e "—", indicando que a arquitetura não leva em consideração tal requisito. Por fim, observando a tabela, decide-se pelo uso dos padrões

ECS e MVC, juntamente com Microkernel, para construir a RA para jogos digitais *open source*, criando uma nova arquitetura baseada nos conceitos centrais desses, que preserva ao máximo os pontos positivos ao mesmo tempo que ameniza os pontos negativos.

Tabela 5.1: *Aderência de Padrões Arquiteturais aos Requisitos do Sistema*

Requisito	ECS	Layered	MVC	Microkern	EDA
Adequação Funcional	-	-	-	-	-
Eficiência de *Performance*	Alto	Baixo	Médio	Baixo	Alto
Compatibilidade	Médio	Baixo	Baixo	Médio	Médio
Capacidade de Interação	Alto	-	Alto	-	-
Confiabilidade	Médio	Alto	Médio	Alto	Baixo
Manutenibilidade	Alto	Alto	Alto	Alto	Médio
Flexibilidade	-	Alto	Médio	Alto	-
Open Source	Médio	Médio	Alto	Alto	Médio

5.3 Representação de Visões Arquiteturais

Para representar de forma visual a arquitetura de referência construída, serão adotados o modelo de visão 4+1 e a visão modular. Essas abordagens permitem organizar a descrição da arquitetura de software por meio de múltiplos pontos de vista complementares uns aos outros, cada um abordando um conjunto específico de preocupações dos *stakeholders*. Na solução proposta, os três principais tipos de *stakeholders* são considerados, sendo eles: jogadores, desenvolvedores (programadores e mantenedores) do jogo digital *open-source*.

5.3.1 Visão Geral

A Visão Geral apresenta uma perspectiva de alto nível da RA, construída com base nos padrões arquiteturais MVC, ECS e Microkernel, ilustrando os componentes fundamentais e suas interações. A Figura 5.1 representa visualmente esta visão, com a seguinte organização:

- **Visão**: Representa as diferentes plataformas de execução (Desktop, Mobile, Web). É atualizada pelo Modelo do sistema.
- **Controlador**: Gerencia os diversos dispositivos de entrada do sistema. É responsável por coletar e enviar os dados de entrada do jogador, que podem alterar o estado do Modelo.
- **Modelo**: É dividido em dois componentes principais:
  - **Núcleo**: Contém as funcionalidades básicas do sistema, incluindo um Motor de Jogo que controla o fluxo e a execução do jogo, além de diferentes Gerenciadores para operar os sistemas do jogo.

– **Plugins**: Representam os componentes extensíveis do jogo, compostos por sistemas especializados e seus respectivos componentes. Comunicam-se com o Núcleo por meio de interfaces pré-definidas.

Com este modelo, garantem-se as características de qualidade essenciais para o funcionamento de um jogo digital *open source*, fornecendo uma arquitetura de referência extensível, adaptável a implementações concretas de jogos com estilos e conteúdos variados. A equipe de desenvolvimento pode incluir diversos *plugins* conforme a finalidade do jogo, como serviços de multijogador, integração com servidores de banco de dados, entre outros.

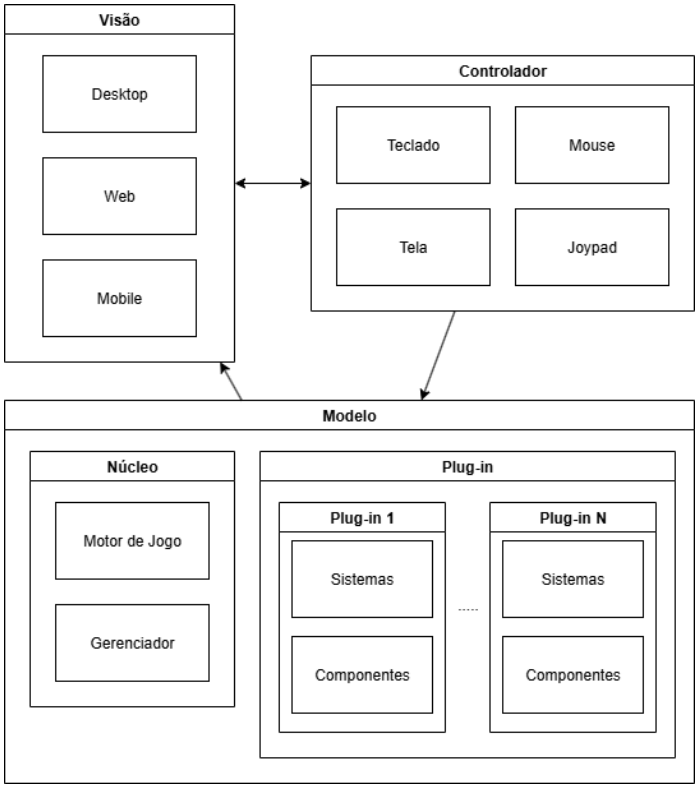

Figura 5.1: *Visão Geral da OSG-RA*

5.3.2 Visão Lógica

A Visão Lógica descreve principalmente os requisitos funcionais do sistema, ou seja, os serviços fornecidos aos usuários ou *stakeholders*. Para isso, o sistema é decomposto em um conjunto de abstrações-chave, derivadas principalmente do domínio do problema. Tais abstrações são objetos ou classes que incorporam os princípios de orientação a objetos, como abstração, encapsulamento e herança. Além de auxiliar na análise funcional, essa decomposição identifica mecanismos e elementos de *design* comuns a todo o sistema (P. B. Kruchten, 2002).

Para representar essa visão, foi elaborado um Diagrama de Classes, ilustrado na Figura 5.2. Esse diagrama apresenta o conjunto de classes principais do sistema e seus respectivos relacionamentos lógicos, conforme descrito a seguir:

- **GameEngine**: Tem a responsabilidade de coordenar o sistema como um todo, inicializando e finalizando os componentes centrais, além de executar o método *update()* em intervalos determinados pelo *deltaTime* para atualizar o estado do sistema.
- **Managers**: Responsável por gerenciar os diferentes partes do sistema, como entidades, componentes e sistemas, além da alocação e libertação da memória de forma eficiente. É parte central do padrão arquitetural ECS.
- **Logger**: Componente que registra todas as informações essenciais da execução do sistema, com o objetivo de facilitar os testes, a manutenção e a evolução do sistema. É um componente essencial para auxiliar o desenvolvimento do projeto.
- **RenderSystem**: Representa a Visão do padrão arquitetural MVC, responsável por atualizar a interface do usuário a cada ciclo de atualização do sistema.
- **InputSystem**: Representa o Controlador do padrão arquitetural MVC, responsável por coletar as entradas do jogador e distribuí-las aos componentes que necessitam desses dados.

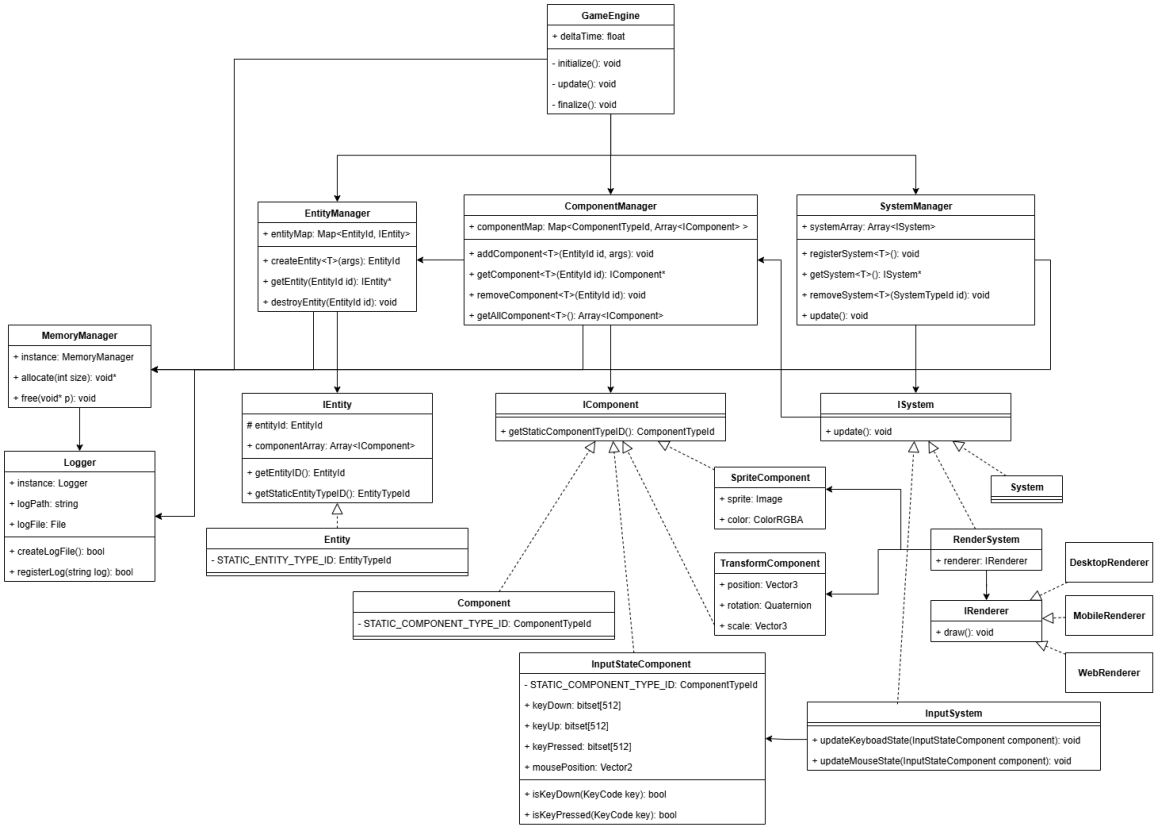

Figura 5.2: *Visão Lógica da OSG-RA*

5.3.3 Visão de Processos

A Visão de Processos considera principalmente os requisitos não funcionais do sistema, como desempenho e disponibilidade (P. B. Kruchten, 2002). Esta visão aborda aspectos de

concorrência e sincronização, descrevendo o software particionado em um conjunto de tarefas independentes, *threads* de controle separadas que podem ser agendadas individualmente em diferentes nós de processamento (P. B. Kruchten, 2002). Sua funcionalidade é esclarecer aspectos dinâmicos da arquitetura em ambientes distribuídos ou de alto desempenho.

Para representar essa visão, foi utilizado o Diagrama de Sequência, conforme mostrado nas Figuras 5.3 e 5.4. Nesse contexto, os diagramas ilustram o fluxo de inicialização e atualização do sistema, bem como o processo de captura de entrada e renderização, respectivamente.

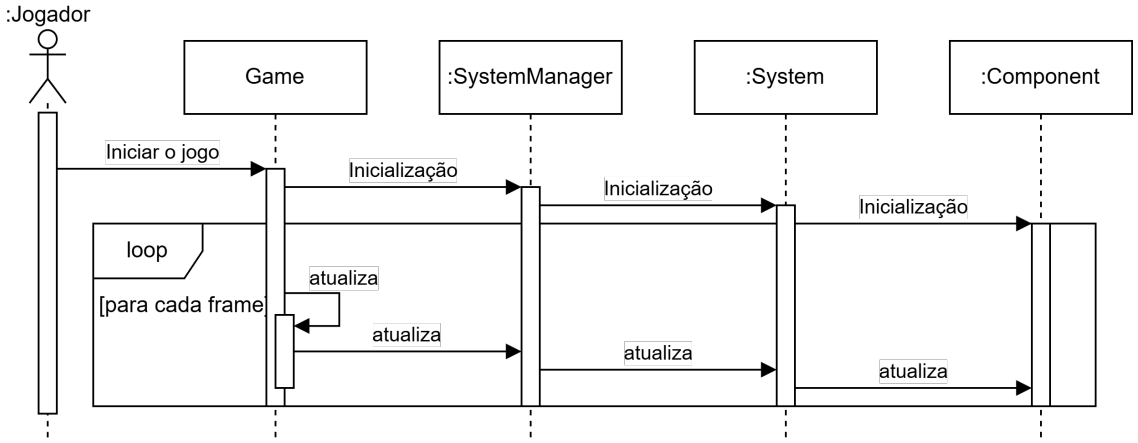

Figura 5.3: *Visão de Processos da OSG-RA da inicialização e atualização do sistema*

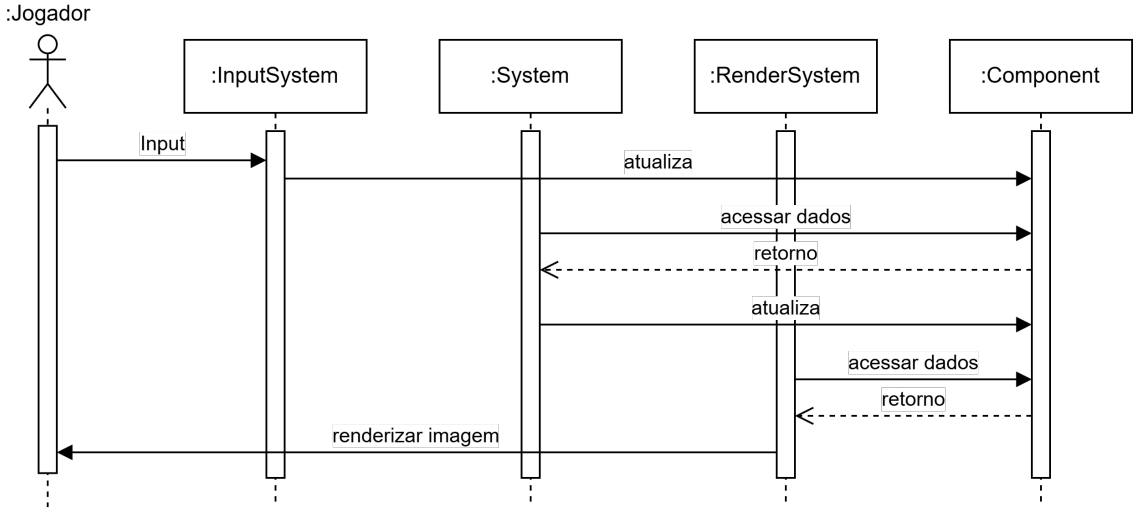

Figura 5.4: *Visão de Processos da OSG-RA da captura de entrada e renderização do sistema*

5.3.4 Visão de Desenvolvimento

A Visão de Desenvolvimento (ou Visão de Implementação) foca na organização dos módulos de software reais no ambiente de desenvolvimento. O software é empacotado em pequenos pacotes, bibliotecas de programas ou subsistemas, organizados em uma hierarquia de camadas que podem ser desenvolvidos por um ou mais desenvolvedores (P. B. Kruchten, 2002). Esta visão considera requisitos internos relacionados à facilidade de desenvolvimento, gerenciamento de software, reutilização e restrições impostas pelo conjunto de ferramentas ou linguagem de programação (P. B. Kruchten, 2002).

Para representar essa visão, foi construído um Diagrama de Pacotes representado na Figura 5.5.

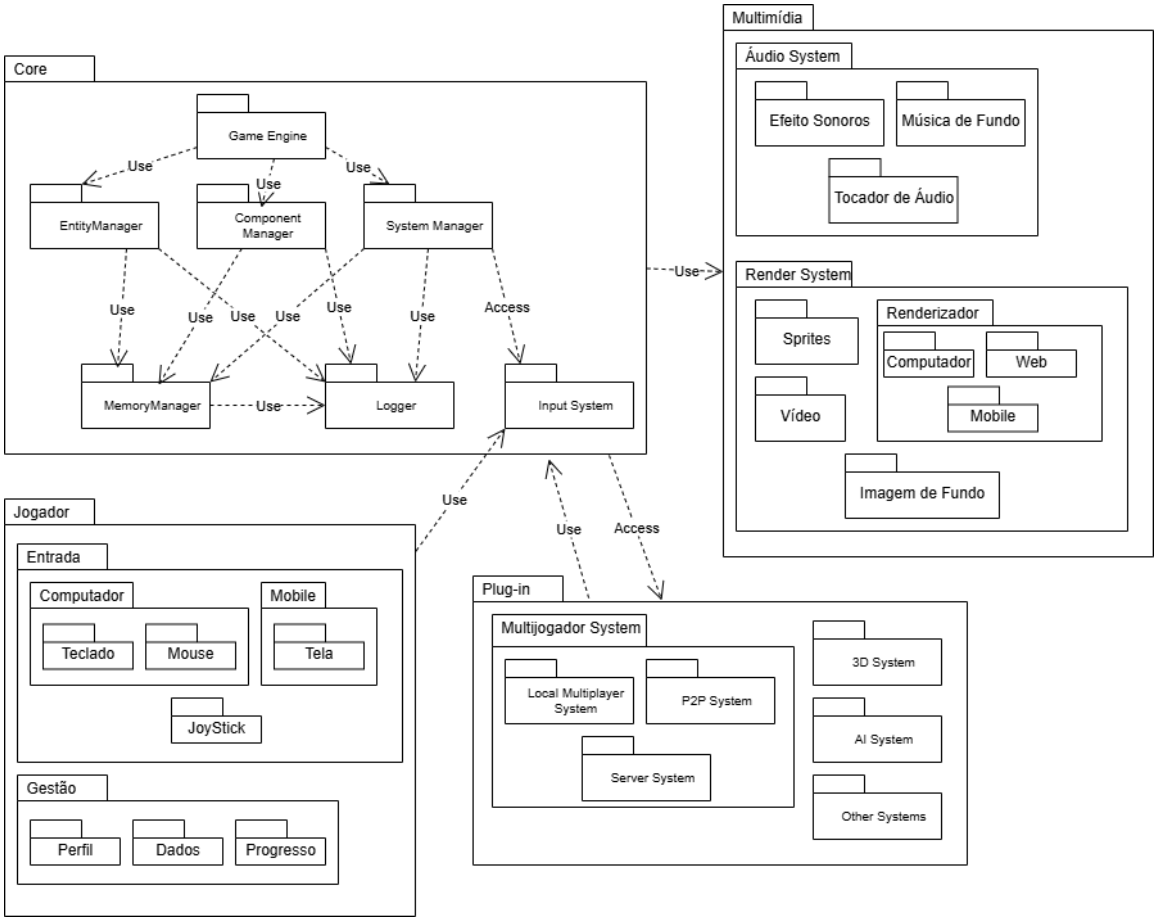

Figura 5.5: *Visão de Desenvolvimento da OSG-RA*

O pacote **Core** é o núcleo central do sistema, contendo os componentes essenciais e sendo responsável por coordenar as atividades e o fluxo de dados, além de gerenciar a interface de conexão com os outros pacotes. O pacote **Jogador** contém as funcionalidades relacionadas à experiência do usuário, incluindo mecanismos para capturar seus *inputs* e os dados gerados durante a execução do jogo, como configurações e progressos.

O pacote **Multimídia** é responsável por garantir a imersão do jogador durante sua experiência, gerenciando os elementos visuais e sonoros presentes no jogo. Utilizando o

subpacote de **Áudio**, o jogo pode reproduzir efeitos sonoros adequados à narrativa, além de música de fundo para criar um ambiente mais envolvente. Da mesma forma, o pacote de **Renderização** ajuda a apresentar o jogo utilizando recursos gráficos visualmente atrativos, impactando diretamente na experiência do usuário.

Por fim, o pacote de **Plug-in** representa todas as funcionalidades que podem ser adicionadas ou removidas de acordo com a necessidade e objetivo do jogo em desenvolvimento, o que oferece maior liberdade e organização à equipe de desenvolvedores do projeto. Todos os *plugins* do sistema comunicam-se com o *Core* através da interface definida pelo *System Manager* e *Component Manager*, e devem estar organizados de forma independente, permitindo seu desenvolvimento por equipes distintas.

5.3.5 Visão Física

A Visão Física (ou Visão de Implantação) considera os requisitos não funcionais do sistema, como disponibilidade, confiabilidade, desempenho e escalabilidade. Nesta visão, os elementos identificados nas visões anteriores (redes, processos, tarefas e objetos) devem ser mapeados para diversos nós de forma altamente flexível, com impacto mínimo no código-fonte (P. B. Kruchten, 2002). Em outras palavras, esta visão descreve a infraestrutura de hardware necessária para executar o sistema, incluindo servidores, redes, dispositivos e topologias de implantação, mapeando os componentes de software em recursos físicos e considerando restrições de desempenho, confiabilidade e distribuição geográfica.

Para representar essa visão, um Diagrama de Implantação foi construído, conforme ilustrado na Figura 5.6. Neste diagrama, os dispositivos clientes representam as diferentes plataformas onde o jogo pode ser executado, como navegadores web, computadores ou dispositivos móveis. Cada dispositivo cliente possui o componente **Jogo Digital**, adquirido por meio do **Repositório** do projeto, seja compilando o código-fonte ou baixando diretamente uma versão pré-construída pela equipe de desenvolvimento. Com esse componente, é possível processar e executar as funcionalidades do jogo no dispositivo onde está instalado, funcionando de forma offline e armazenando os dados localmente, sem a necessidade de um servidor externo.

Entretanto, caso haja necessidade, a equipe de desenvolvimento pode optar por utilizar um servidor externo com os seguintes componentes: **Servidor de Jogo Digital** (responsável pelo processamento central do jogo *open source*), **Servidor de Rede** (para gerenciar a comunicação e conexão entre os dispositivos clientes e o servidor) e **Banco de Dados** (para armazenar os dados do jogador e o estado do jogo). Para isso, é necessário que a equipe de desenvolvimento implemente e adicione o *plugin* adequado às necessidades do sistema. A arquitetura de referência proposta não se restringe ao uso de um servidor central, podendo também utilizar outras topologias, como servidores *peer-to-peer* (P2P) e entre outras.

5.3.6 Visão de Caso de Uso

A Visão de Caso de Uso utiliza um pequeno subconjunto de cenários importantes (instâncias de casos de uso) para demonstrar e validar que os elementos das quatro visões anteriores trabalham de forma integrada e coerente (P. B. Kruchten, 2002). Esses cenários têm como finalidade auxiliar os arquitetos e *designers* a identificar elementos arquiteturais

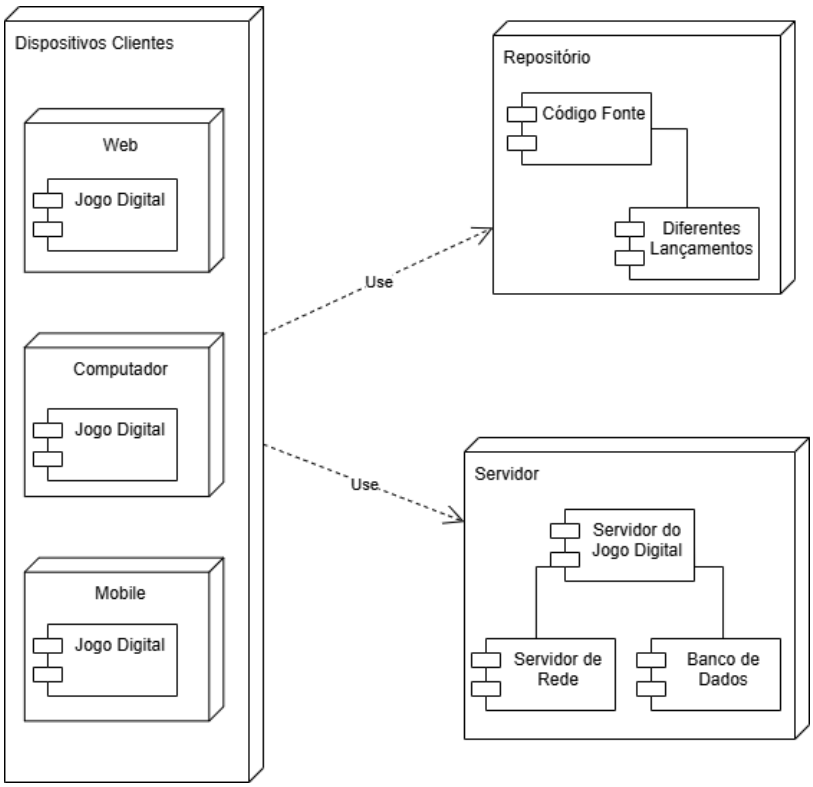

Figura 5.6: *Visão Física da OSG-RA*

durante o projeto, além de validar e ilustrar a arquitetura proposta, assegurando que ela atenda aos requisitos dos *stakeholders* (P. B. Kruchten, 2002). Para representar visualmente essa visão, foi elaborado um Diagrama de Caso de Uso, conforme apresentado na Figura 5.7.

Nesta Visão de Caso de Uso é apresentada diversas interações dos *stakeholders* do sistema (Jogador e Desenvolvedor) com o jogo digital *open source* desenvolvido. Como Jogador, o usuário precisa ter a capacidade de executar o jogo desenvolvido. Para isso, é necessário ter acesso à última versão estável do jogo, o que, por sua vez, depende da capacidade do Desenvolvedor publicar novas versões. Outra necessidade do Jogador é poder fornecer *feedback* sobre o jogo, reportando problemas e sugerindo melhorias para a equipe de desenvolvimento — necessidade essa que também é relevante para os próprios Desenvolvedores.

Com os *feedbacks* coletados, é possível direcionar o desenvolvimento futuro do jogo, característica que facilita a contribuição de desenvolvedores ao código-fonte do projeto *open source*. Por fim, é essencial que o Desenvolvedor tenha a capacidade de inspecionar e gerenciar o código existente do projeto, permitindo a realização de testes, manutenções e a evolução contínua do sistema.

5.4 Considerações Finais

Este capítulo apresentou o processo de síntese arquitetural correspondente ao passo RA-3 do método ProSA-RA, cujo objetivo central foi transformar os requisitos arquiteturais

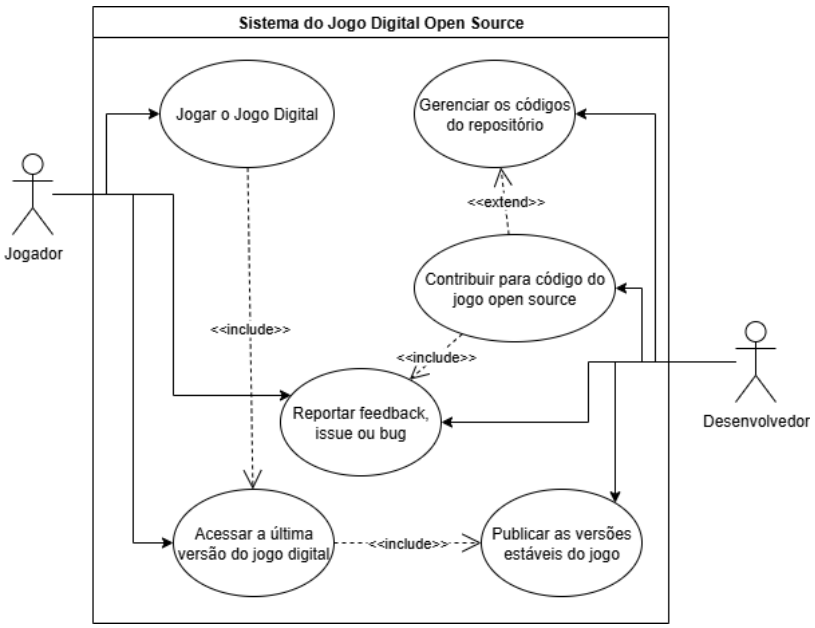

Figura 5.7: *Visão de Caso de Uso da OSG-RA*

previamente eliciados em uma RA para jogos digitais *open-source*. A construção da RA, denominada OSG-RA, seguiu um processo estruturado que iniciou com a análise crítica de padrões arquiteturais consagrados, avançou para a seleção e combinação fundamentada de seus princípios mais adequados e culminou na representação com múltiplas visões da solução proposta.

A análise comparativa de padrões como ECS, Camadas, MVC, Microkernel e EDA, avaliando suas vantagens, desvantagens e aderência aos requisitos específicos do domínio, forneceu a base objetiva para a decisão arquitetural. A matriz comparativa resultante evidenciou que nenhum padrão isolado atende de forma ótima a todos os requisitos, mas que uma combinação estratégica poderia potencializar os pontos fortes de cada um. Consequentemente, a OSG-RA foi concebida como uma arquitetura híbrida, fundamentada nos conceitos centrais do MVC, para separação clara de responsabilidades e interface de usuário, do ECS, para desempenho, flexibilidade e manutenibilidade da lógica de jogo e do Microkernel, para extensibilidade e isolamento de funcionalidades via *plugins*.

Finalmente, a OSG-RA proposta foi materializada e representada por meio do modelo de visões 4+1. A visão geral integrou os padrões selecionados em um modelo coeso. As visões lógica, de processos, de desenvolvimento e física detalharam, respectivamente, a estrutura estática, o comportamento dinâmico, a organização modular e a implantação do sistema. Por fim, a visão de casos de uso complementa as anteriores, validando a arquitetura por meio de cenários de uso pertinentes aos *stakeholders* identificados: jogadores e desenvolvedores.

Em síntese, este capítulo cumpriu seu objetivo de analisar, sintetizar e propor uma base arquitetural sólida para jogos digitais *open-source*. A OSG-RA resultante foi concebida para equilibrar as demandas de desempenho e experiência do jogador com os imperativos de colaboração, extensibilidade e manutenibilidade inerentes ao desenvolvimento aberto. As visões construídas fornecem uma representação abrangente e visual da arquitetura, viabilizando a sua posterior validação com especialistas.

Capítulo 6

Avaliação Arquitetural

Este capítulo apresenta a avaliação da OSG-RA, mostrando os respectivos resultados obtidos. Para a avaliação, foi adotado o método *Architecture Tradeoff Analysis Method* (ATAM), que visa avaliar se a RA estabelecida atende aos requisitos de qualidade identificados, além de identificar os pontos de *trade-off* entre esses atributos, facilitando a comunicação entre os *stakeholders*, esclarecendo e refinando os requisitos e fornecendo uma estrutura para o processo contínuo de análise e construção do sistema (Kazman *et al.*, 1998).

Outro método de avaliação considerado foi o *Technology Acceptance Model* (TAM), que visa identificar os principais determinantes que impactam a aceitação do usuário quanto ao uso da tecnologia. Esse método pode ser avaliado por meio de um formulário de múltipla escolha, cujas respostas estão em uma escala *Likert*, que varia entre "Concordo Totalmente" e "Discordo Totalmente", além de campos disponíveis para comentários que justifiquem as escolhas, os quais podem ser utilizados para orientar a identificação de problemas de aceitação e soluções para melhorar os aspectos da RA (Davis *et al.*, 1989a; Davis *et al.*, 1989b).

A estrutura deste capítulo está organizada da seguinte forma: A Seção 6.1 apresenta uma introdução abrangente ao método ATAM, descrevendo suas diferentes etapas de avaliação, juntamente com a descrição dos grupos de pesquisadores envolvidos no processo. Na Seção 6.2 são detalhados os preparativos realizados antes do início efetivo da avaliação. A Seção 6.3 contém a apresentação da arquitetura construída, a identificação e a representação sistemática das abordagens arquiteturais adotadas, bem como a análise de riscos e a exploração de possíveis *trade-offs* entre diferentes atributos de qualidade, conduzidas pelos especialistas da área. Na Seção 6.4, são descritas as atividades de análise e a priorização dos cenários de qualidade realizadas pelos *stakeholders* convidados, as quais avaliam a RA com base na experiência e na necessidade específica de cada um. Nessa seção, também são apresentados os resultados consolidados durante o processo de avaliação. A Seção 6.5 descreve a avaliação de aceitação da tecnologia realizada, identificando os possíveis pontos de melhoria sob esse aspecto específico, o que agrega uma dimensão comportamental e de percepção de uso ao estudo. Por fim, a Seção 6.6 apresenta uma consideração final dos conteúdos desse capítulo.

6.1 *Architecture Tradeoff Analysis Method* (ATAM)

Para realizar a avaliação pelo método ATAM, foi necessária a colaboração de três grupos principais, cada um com funções e responsabilidades específicas na avaliação (Bass *et al.*, 2021). São eles:

- **Tomadores de decisão**: compostos pelos pesquisadores que desenvolveram a arquitetura. Estes possuem autoridade para representar o projeto, sendo responsáveis por definir suas diretrizes e por implementar mudanças na arquitetura, quando necessário.
- **Equipe de avaliação**: composta por três a cinco profissionais externos ao projeto, selecionados com base em seu conhecimento na área analisada, garantindo uma avaliação qualificada. São responsáveis por uma avaliação técnica preliminar para identificar pontos de melhoria na arquitetura construída, bem como no processo de avaliação.
- ***Stakeholders***: grupo de pessoas envolvidas e interessadas no projeto de pesquisa, que avaliam a arquitetura com base em seus pontos de vista, identificando os *trade-offs* de acordo com suas necessidades reais.

A aplicação do método ATAM estrutura-se em quatro fases principais: (1) coleta de cenários e requisitos; (2) análise de visões arquiteturais e realização de cenários; (3) construção e avaliação de modelos; e (4) identificação de *trade-offs*. Essas fases são desdobradas em nove etapas distintas, conforme ilustrado na Figura 6.1.

Inicialmente, definem-se o conjunto inicial de cenários e requisitos do sistema, bem como uma proposta da OSG-RA. Posteriormente, cada atributo de qualidade foi analisado individualmente quanto ao seu impacto no projeto. Em uma etapa crítica, identificam-se as compensações entre os diferentes atributos, o que possibilita ajustes na arquitetura, nos modelos ou nos requisitos. Esse ciclo iterativo promove um refinamento progressivo, assegurando que a arquitetura desenvolvida atenda de forma eficaz às necessidades do sistema (Kazman *et al.*, 1998).

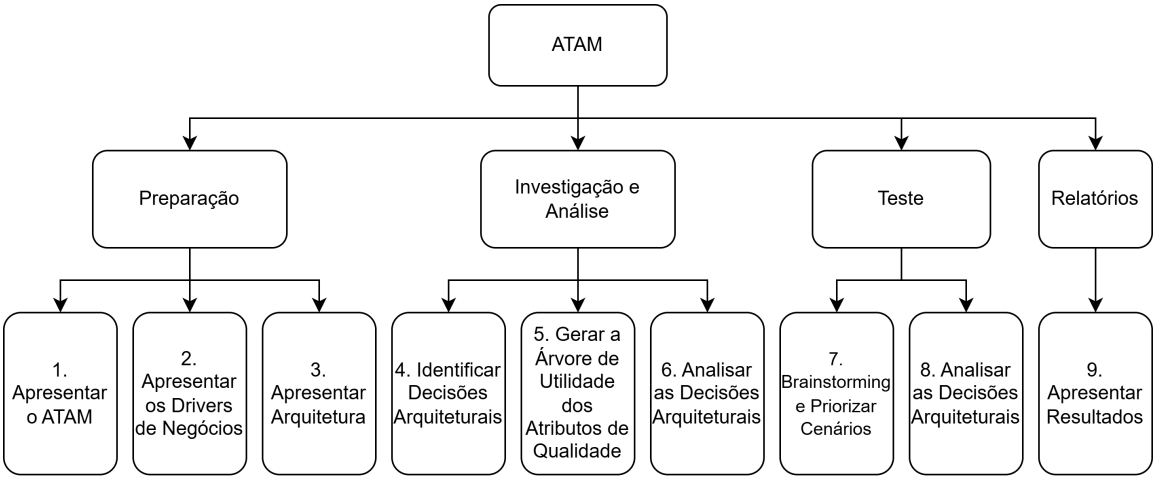

Figura 6.1: *Estrutura das etapas de ATAM (Adaptado de Kazman et al., 1998)*

6.2 Fase 0: Preparação da Avaliação ATAM

Como preparação para a avaliação, nesta fase, os tomadores de decisão precisam selecionar uma equipe de avaliação composta por especialistas experientes na área, adequada para analisar a RA. Posteriormente, a equipe de avaliação e os tomadores de decisão realizam reuniões para definir e discutir os detalhes e a logística do processo avaliativo (Bass *et al.*, 2021).

Inicialmente, foram identificados os possíveis participantes para a análise e os convites foram enviados por e-mail para confirmar sua disponibilidade. Como resultado, três professores, todos com formação em nível de doutorado e experiência nas áreas de requisitos de software, *open source* ou jogos digitais, aceitaram participar da avaliação. Em seguida, foi agendado em conjunto um horário compatível com a disponibilidade de todos os participantes e tomadores de decisão, e então o propósito e o processo da avaliação foram explicados.

6.3 Fase 1: Avaliação Inicial

Nesta primeira fase, a equipe de avaliação e os tomadores de decisão reuniram-se por videoconferência para dar início ao processo de coleta e análise de informações. Este processo foi dividido em seis etapas, descritas nas subseções a seguir.

6.3.1 Etapa 1: Preparação do ATAM

Nesta etapa, foi realizada uma apresentação para introduzir o método ATAM à equipe de avaliação e aos *stakeholders* que participam ativamente da avaliação. Durante a sessão, foram apresentadas as diferentes fases do ATAM, conforme descrito na Seção 6.1, bem como o processo metodológico que seria seguido durante a avaliação.

6.3.2 Etapa 2: Apresentação do *Business Driver*

Depois da apresentação do método de avaliação a ser utilizado, reuniram-se as informações apresentadas no Capítulo 2, proporcionando uma visão geral do contexto em que a OSG-RA foi desenvolvida. Dessa forma, apresentaram-se os principais conceitos relacionados à área de pesquisa, a fim de contextualizar tanto a equipe de avaliação quanto os *stakeholders*.

6.3.3 Etapa 3: Apresentação da Arquitetura

Nesta etapa, após a contextualização, inicia-se a apresentação da OSG-RA. Um dos membros da equipe de tomadores de decisão realizou uma apresentação da OSG-RA, com ênfase em como ela atende às diretrizes de negócio estabelecidas. A apresentação abordou os 10 Requisitos Arquiteturais identificados para a RA, conforme descrito na Seção 4.2, discutindo como foram extraídos e avaliados, além de relatar os requisitos essenciais e secundários identificados. Adicionalmente, foram apresentados os principais objetivos definidos para a RA, os *stakeholders* envolvidos e os principais interesses. Por

fim, apresentaram-se as cinco visões arquiteturais e a visão geral da modelagem da RA para representação visual do artefato, conforme descrito na Seção 5.3.

Durante a avaliação, a equipe de avaliação e os *stakeholders* solicitaram esclarecimentos sobre os aspectos abordados pela RA e suas respectivas representações, além de questionarem as fontes de informação das quais os tomadores de decisão extraíram as evidências para embasar a construção da arquitetura. As perguntas e observações realizadas foram registradas para que possam ser utilizadas posteriormente como auxílio na análise da OSG-RA, a qual será descrita com maior detalhe nas próximas seções.

6.3.4 Etapa 4: Identificação das Abordagens Arquiteturais

Após a apresentação da etapa anterior, nesta fase, a equipe de avaliação identificou as abordagens arquiteturais consideradas para a OSG-RA. Tais abordagens foram identificadas com base nas informações apresentadas na subseção anterior, sendo registradas e analisadas pela equipe para fornecer embasamento às etapas subsequentes da avaliação.

6.3.5 Etapa 5: Apresentação da Árvore de Atributos de Qualidade

Nesta etapa, os atributos de qualidade a serem considerados no sistema implementado são detalhados por meio de uma Árvore de Atributos de Qualidade, instrumento utilizado para identificar e priorizar os atributos mais importantes para o sistema (Bass *et al.*, 2021).

Os Requisitos Arquiteturais (ou atributos de qualidade) considerados pela OSG-RA foram previamente identificados na Seção 4.2 sob forma de tabela, validados e priorizados por meio da análise de estudos acadêmicos e jogos existentes. Com base nessa identificação, foi construída a Árvore de Atributos de Utilidade, conforme ilustrado na Figura 6.2. Essa estrutura hierárquica auxilia na organização e representação visual das prioridades dos atributos de qualidade considerados pela arquitetura, facilitando a análise e a tomada de decisões sobre a RA.

A Árvore de Atributos de Qualidade, apresentada na Figura 6.2, possui um único nó raiz **Qualidade**, que representa o objetivo principal da RA construída: garantir a qualidade do jogo digital *open-source* concreta implementado a partir dela. Os nós imediatamente abaixo da raiz representam os atributos de qualidade desejados para o sistema, seguindo o padrão ISO/IEC 25010 – ou seja, os atributos priorizados para atender aos objetivos da RA. Abaixo desses atributos de qualidade, encontram-se os sub-requisitos específicos desejados para cada atributo, descrevendo-se, em seguida, como cada item deve ser implementado no jogo.

A partir dos atributos de qualidade descritos na árvore, foram analisados e construídos os cenários de caso de uso do sistema, observando-se a ocorrência dos atributos de qualidade em diferentes situações de aplicação e sua importância no contexto do comportamento e dos resultados esperados do jogo digital *open-source*. Com base nessa análise, criaram-se oito cenários distintos, um para cada atributo, selecionando o sub-requisito mais significativo para o sistema, conforme apresentado a seguir.

Para especificar um requisito de atributo de qualidade por meio de um cenário de uso, é possível formalizá-lo em seis partes inter-relacionadas, que descrevem e representam os

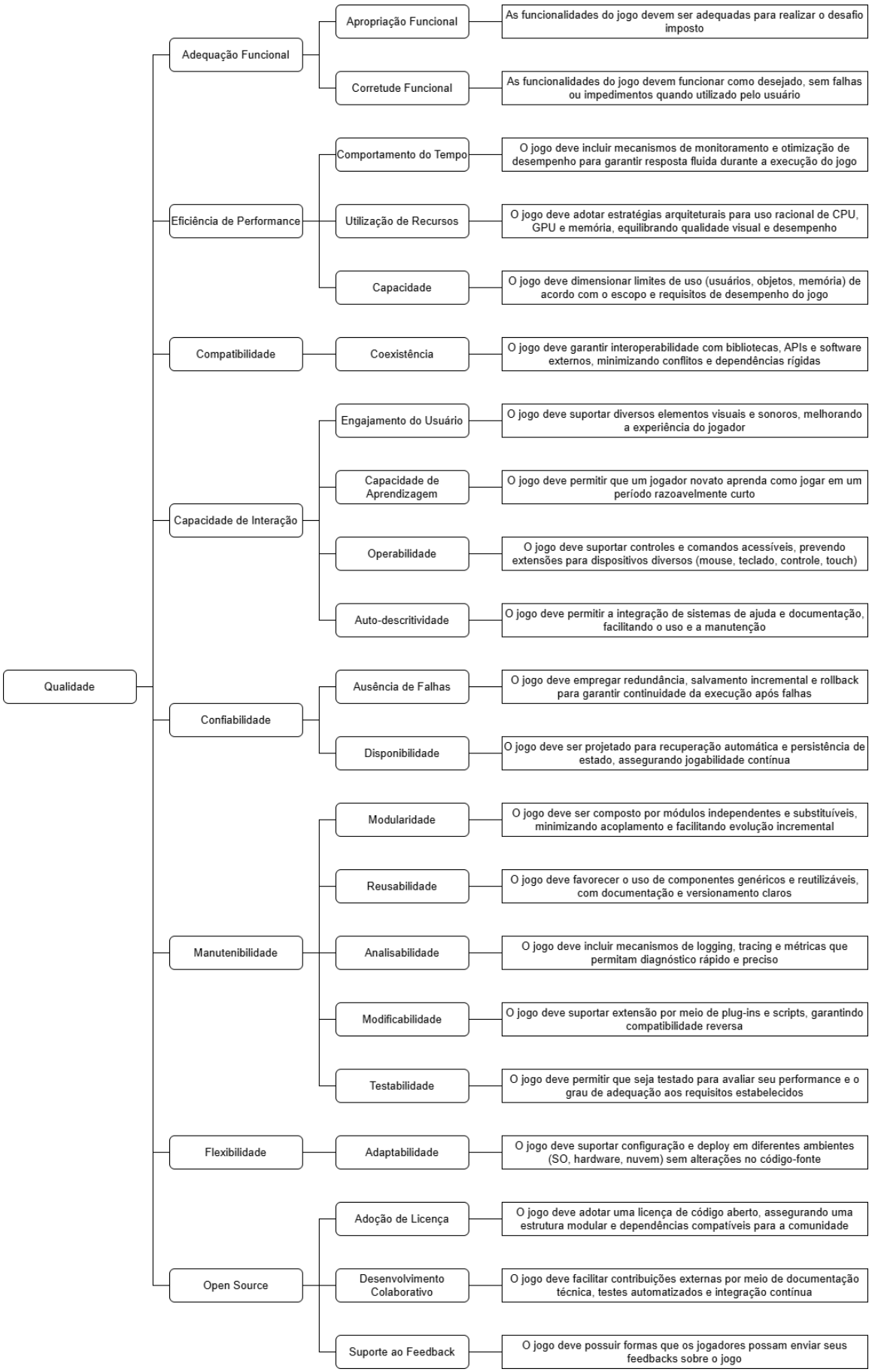

Figura 6.2: *Árvore de Atributos de Qualidade.*

diferentes componentes envolvidos, conforme ilustrado na Figura 6.3. A primeira parte, **Fonte de Estímulo**, refere-se a um agente que gera o estímulo para o sistema, podendo ser um ser humano, como um jogador, ou um elemento automatizado, como o próprio sistema ou outro software. A segunda parte, **Estímulo**, consiste em uma condição ou evento que, ao atingir o sistema, exige uma resposta, representando uma ação específica realizada pela fonte de estímulo. Em seguida, o **Ambiente** define o contexto ou estado em que o sistema se encontra no momento em que o estímulo é recebido, como operação normal, sobrecarga ou manutenção. O **Artefato** corresponde ao elemento que foi estimulado, podendo ser um conjunto de sistemas, o sistema como um todo ou apenas parte dele, como um módulo ou componente específico. A **Resposta** é a atividade executada pelo sistema como consequência da chegada do estímulo, incluindo comportamentos, processamentos ou saídas geradas. Por fim, a **Medida de Resposta** estabelece que, quando a resposta ocorre, ela deve ser mensurável por meio de indicadores quantitativos ou qualitativos, permitindo que o requisito seja testado e validado adequadamente (Bass *et al.*, 2021).

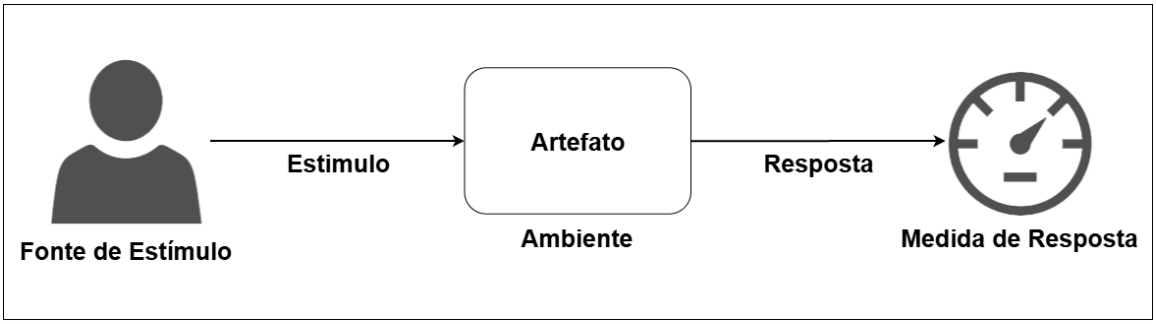

Figura 6.3: *Componentes de um Cenário de Atributo de Qualidade (Adaptado de Bass et al., 2021)*

O primeiro cenário, Apropriação Funcional da Adequação Funcional (Figura 6.4), avalia se os mecanismos implementados pela RA garantem a realização das tarefas desejadas pelos usuários de maneira intuitiva e simplificada.

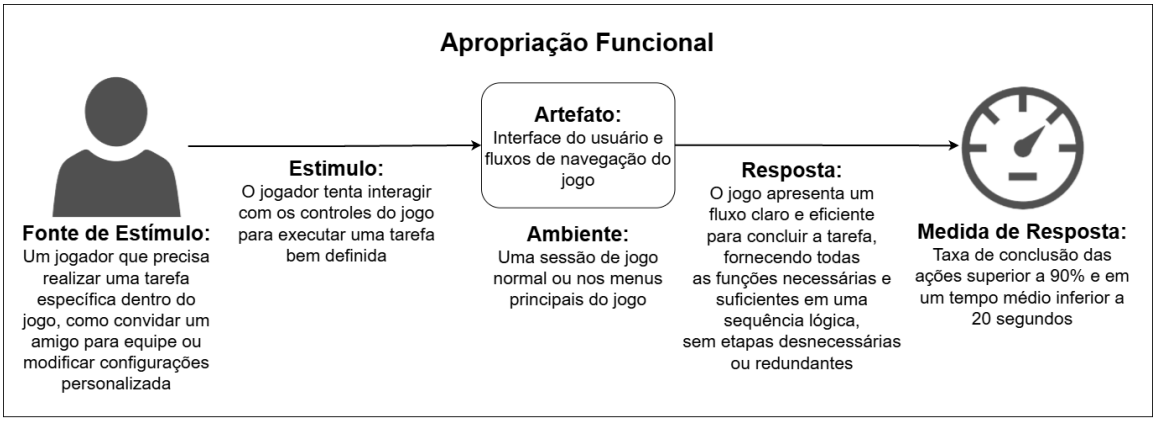

Figura 6.4: *Um cenário geral para Adequação Funcional*

O segundo cenário, Comportamento de Tempo da Eficiência de *Performance* (Figura 6.5), visa avaliar a eficácia da RA em assegurar que o jogo concreto possa ser executado com fluidez, sem grandes variações de *framerate*, mesmo em cenas complexas, quando o jogo está configurado no padrão recomendado.

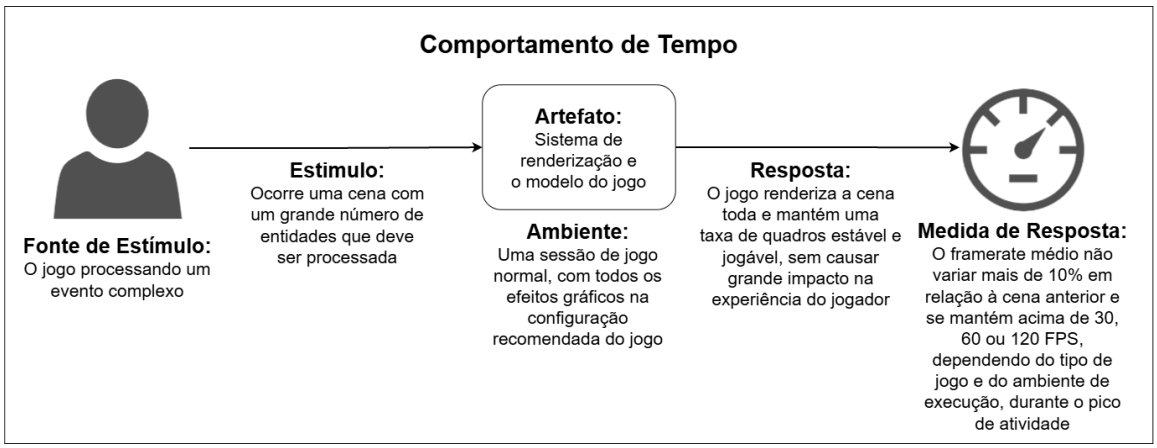

Figura 6.5: *Um cenário geral para Eficiência de Performance*

O terceiro cenário, Coexistência da Compatibilidade (Figura 6.6), destaca a importância de que o jogo digital *open-source* implementado seja capaz de alocar o uso dos componentes da máquina de forma eficaz, garantindo sua execução em paralelo com outros softwares.

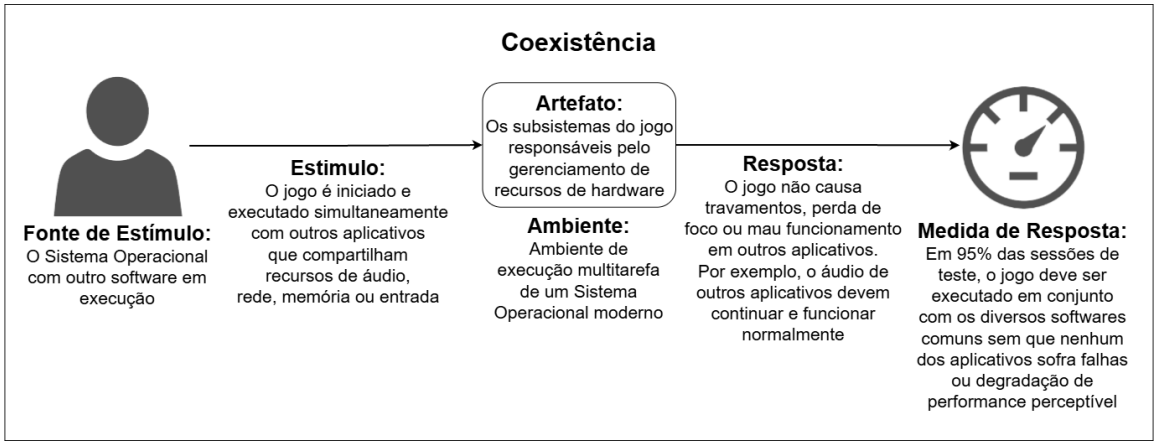

Figura 6.6: *Um cenário geral para Compatibilidade*

O quarto cenário, Engajamento do Usuário da Capacidade de Interação (Figura 6.7), assegura que a RA proporcione uma experiência agradável ao usuário durante o jogo, oferecendo *feedbacks* sonoros, visuais ou outras formas de recompensa que mantenham sua atenção e motivação.

O quinto cenário, Ausência de Falhas da Confiabilidade (Figura 6.8), garante que o jogo digital *open-source* possa ser jogado do início ao fim sem a ocorrência de falhas críticas que impeçam o progresso, além de assegurar que as mecânicas e funcionalidades ocorram conforme o esperado, sem desvios.

O sexto cenário, Modificabilidade da Manutenibilidade (Figura 6.9), reforça que a RA facilita a modificação e a manutenção do jogo concreto, permitindo a inclusão eficiente de novas funcionalidades.

O sétimo cenário, Adaptabilidade da Flexibilidade (Figura 6.10), avalia como a RA apoia a adaptabilidade do sistema desenvolvido, permitindo que o jogo digital construído

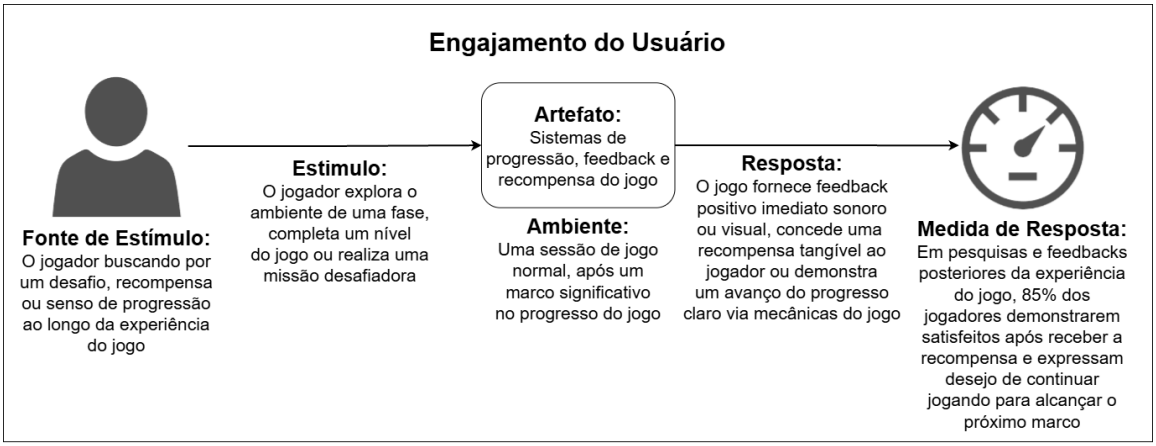

Figura 6.7: *Um cenário geral para Capacidade de Interação*

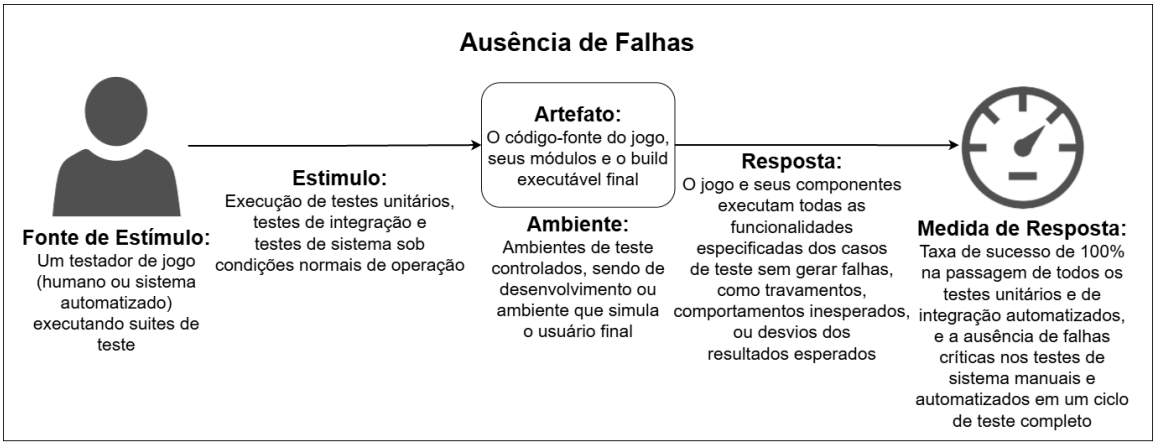

Figura 6.8: *Um cenário geral para Confiabilidade*

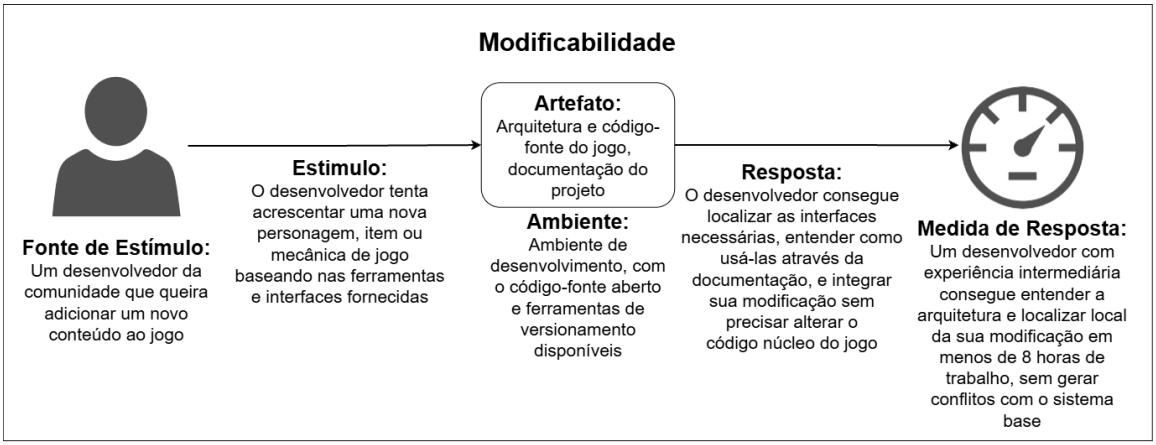

Figura 6.9: *Um cenário geral para Manutenibilidade*

seja executado em múltiplas plataformas de especificações distintas sem a necessidade de alteração do código-fonte.

Por fim, o oitavo cenário, Adoção de Licença do *Open Source* (Figura 6.11), concentra-se em garantir que o usuário possa encontrar facilmente a licença adotada pelo jogo, a qual

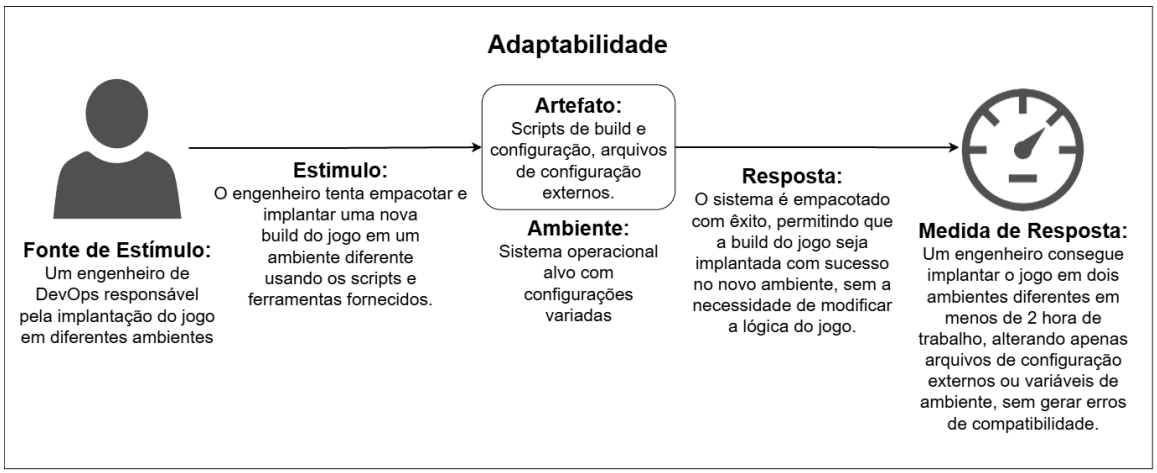

Figura 6.10: *Um cenário geral para Flexibilidade*

deve estar localizada em um local de fácil acesso no repositório, facilitando a contribuição e assegurando os direitos dos usuários.

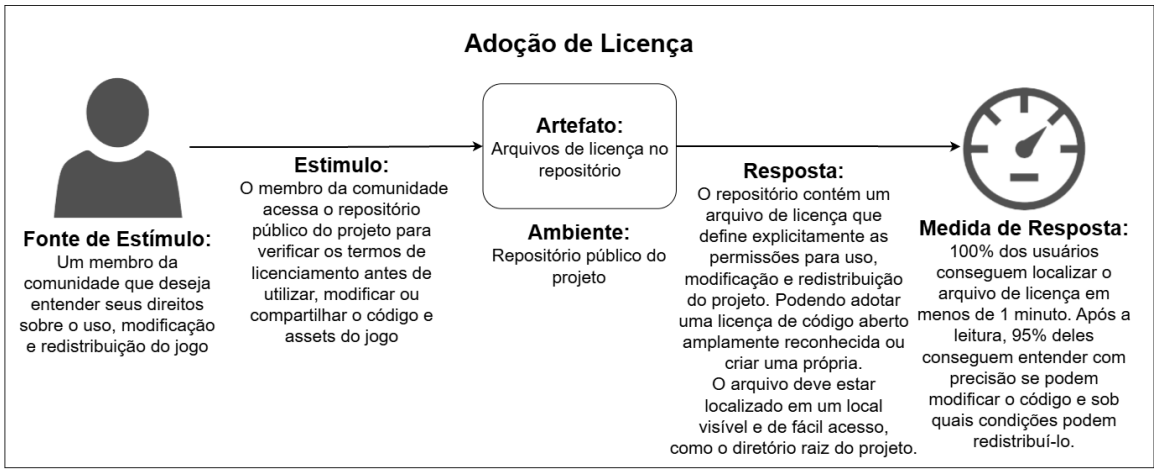

Figura 6.11: *Um cenário geral para Open Source*

6.3.6 Etapa 6: Análise das Abordagens Arquiteturais

Nesta etapa, a equipe de avaliação analisou individualmente os atributos de qualidade considerados na OSG-RA, com base nos cenários de caso de uso e na árvore de atributos de qualidade previamente identificados. Com base nas respostas dos avaliadores, foram identificados pontos de destaque, riscos, *trade-offs* e oportunidades de melhoria da arquitetura, que servirão de suporte para um aprimoramento posterior.

A avaliação da OSG-RA ocorreu imediatamente após as apresentações descritas nas etapas anteriores. Durante esse processo, os três professores avaliadores, identificados como PA1, PA2 e PA3 (para preservar suas identidades, conforme estabelecido no Termo de Consentimento Livre e Esclarecido fornecido antes do início da avaliação), preencheram um formulário específico. Nesse formulário, os avaliadores atribuíram uma pontuação de 1 a 5 a cada atributo de qualidade, com base em uma escala de concordância que variava de

"Não Atendido" a "Totalmente Atendido", avaliando o nível em que a arquitetura satisfaz os requisitos correspondentes (os resultados das pontuações são apresentados na Tabela 6.1). Adicionalmente, registraram observações pertinentes à avaliação dos atributos, como riscos potenciais, pontos fortes, pontos de sensibilidade e *trade-offs* associados.

Tabela 6.1: *Pontuações atribuídas aos atributos de qualidade pela equipe de avaliação*

Atributo de Qualidade	**PA1**	**PA2**	**PA3**	**Média**
Adequação Funcional	4	4	5	4.33
Eficiência de *Performance*	5	5	5	5
Compatibilidade	3	5	1	3
Capacidade de Interação	5	4	4	4.33
Confiabilidade	4	4	4	4
Manutenibilidade	5	5	5	5
Flexibilidade	5	5	5	5
Open Source	5	5	4	4.67

Com base nas pontuações atribuídas pela equipe de avaliação e nos respectivos comentários sobre cada atributo, pode-se chegar às seguintes conclusões em relação a cada atributo de qualidade:

- **Adequação Funcional:** A avaliação relacionada a este atributo de qualidade revela que o requisito foi atendido de forma satisfatória. No entanto, o PA1 destacou a necessidade de inclusão de mecanismos de auxílio ao usuário, como *tooltips* ou tutoriais integrados, pois, embora a OSG-RA seja intuitiva, usuários leigos podem encontrar dificuldades em interações complexas. Por outro lado, o PA2 identificou que, apesar da arquitetura não avaliar todas as subcaracterísticas de qualidade definidas pela ISO (deixando de avaliar a Completude Funcional), os itens avaliados são suficientes para o contexto analisado.
- **Eficiência de *Performance*:** Todos os avaliadores consideraram este atributo plenamente atendido. Entretanto, o PA1 destacou a importância do gerenciamento dinâmico de recursos, como escalonamento de servidores e alocação sob demanda, para garantir desempenho estável mesmo sob picos de uso. O PA3 observou que, como o sistema adota separação entre o sistema multimídia, o *core* do jogo e a possível instalação de *plugins*, o acoplamento é baixo entre diferentes componentes, o que facilita a otimização independente de cada parte e melhora a *performance* do sistema como um todo.
- **Compatibilidade:** Este atributo apresentou a maior divergência entre avaliadores: o PA2 atribuiu nota máxima, o PA3 atribuiu nota mínima e o PA1 adotou pontuação intermediária. O PA1 identificou que a proposta de ser multiplataforma pode gerar preocupações sobre a viabilidade de desenvolver software de qualidade sem uso de linguagens nativas, o que poderia exigir desenvolvimento e manutenção de múltiplos projetos. O PA2 argumentou que, como não há necessariamente cenários que demandem avaliação de interoperabilidade, a arquitetura satisfaz o requisito de

Compatibilidade. Já o PA3 considerou que a arquitetura não apresenta interface ou camada específica para interação com o sistema operacional (para gerenciamento de recursos como memória e processamento), concluindo ser impossível avaliar este atributo apenas com a arquitetura disponível.

- **Capacidade de Interação:** A OSG-RA foi considerada capaz de satisfazer este atributo. O PA2 sugeriu a inclusão do sub-requisito de Acessibilidade na RA, visto que aspectos desse atributo estão parcialmente descritos na subcaracterística de Operabilidade.
- **Confiabilidade:** Houve consenso entre os avaliadores sobre a necessidade de melhorias neste atributo. O PA1 apontou que, embora haja menção à realização de testes, não foi especificado como estes serão conduzidos, especialmente considerando que em projetos *open source* a abordagem de testes pode ser variável. O PA2 argumentou pela necessidade de considerar o sub-requisito de Recuperabilidade, por sua relevância em jogos digitais. O PA3 questionou a ausência de um *handler* ou mecanismo específico para tratamento de falhas, apesar da existência de módulos bem isolados.
- **Manutenibilidade:** Todos os avaliadores concordaram que este atributo foi devidamente considerado pela Arquitetura de Referência, sem observações adicionais.
- **Flexibilidade:** Todos os avaliadores consideraram este atributo bem tratado pela RA. O PA2 ressaltou que, embora nem todas as subcaracterísticas de Flexibilidade sugeridas pela ISO tenham sido consideradas, a subcaracterística escolhida mostrou-se efetiva para cobrir os aspectos principais em jogos digitais. O PA3 avaliou que a RA atende ao requisito, pois prevê uso de diferentes dispositivos físicos e a possibilidade de adicionar ou remover servidores conforme necessário.
- ***Open Source*:** Este requisito foi considerado efetivo na RA. Entretanto, o PA3 indicou a necessidade de documentação mais completa, com descrições claras dos componentes, suas funções e interações, aspecto essencial para garantir colaboração em contextos de *open source*. Adicionalmente, sugeriu a possibilidade de incluir a funcionalidade de *feedback* do usuário como um *plugin*.

De modo geral, a OSG-RA foi considerada útil, de fácil aprendizado e com potencial para aumentar a produtividade e a qualidade dos jogos digitais desenvolvidos. O PA1 identificou que a arquitetura demonstra preocupação com diversos atributos de qualidade, garantindo a confiabilidade do sistema proposto. No entanto, ressaltou a necessidade de identificar ferramentas que possibilitem o controle de documentação, testes e servidores para que todos os atributos sejam plenamente atendidos. O PA2 argumentou que a arquitetura adota uma abordagem conceitual de baixa complexidade, o que favorece sua adoção inclusive por profissionais e estudantes com pouca experiência em desenvolvimento de software. Além disso, afirmou que ela aborda diversas características de qualidade e sua utilização pode contribuir não apenas para a produção de jogos de código aberto, mas também para a observância de critérios de qualidade na criação de jogos em geral. Entretanto, identificou a necessidade de incluir outros cenários e exemplos de jogos concretos que adotem a arquitetura, de modo a apoiar melhor seu entendimento, especialmente considerando o perfil de desenvolvedores iniciantes. Por fim, o PA3 indicou que os critérios de qualidade foram construídos de forma bem definida e que a visão lógica da arquitetura é clara e bem

elaborada. No entanto, também apontou que a documentação poderia ser mais completa, com descrições mais claras dos componentes, suas funções e suas interações.

6.4 Fase 2: Avaliação Final

Após a análise da equipe de avaliação, realizou-se uma segunda avaliação em conjunto com os *stakeholders* interessados no projeto. Para isso, os tomadores de decisão apresentaram as etapas 1 a 6, de modo que os *stakeholders* compreendam o método de avaliação adotado e os papéis que devem desempenhar, além de se manterem atualizados sobre o projeto após as modificações realizadas com base nas observações da equipe de avaliação. Dessa forma, eles são devidamente informados sobre os resultados obtidos até o momento, o que permite a continuidade do processo de avaliação nas etapas subsequentes (Bass *et al.*, 2021). Com isso, as etapas posteriores podem ser conduzidas.

6.4.1 Etapa 7: *Brainstorming* e Priorização de Cenários

Na presente etapa, os tomadores de decisão solicitaram aos *stakeholders* que identificassem os cenários considerados mais críticos para o sucesso da OSG-RA, com base em suas funções e experiências individuais. Para tanto, utilizou-se a árvore de utilidade (Figura 6.2) e os cenários dela derivados para detalhar os atributos de qualidade considerados pela arquitetura. O objetivo desta análise foi capturar uma visão ampla e prática da comunidade de *stakeholders*, complementando a avaliação teórica dos atributos de qualidade com perspectivas baseadas em experiência real (Bass *et al.*, 2021).

Para essa avaliação, adotou-se uma abordagem de votação ponderada: cada *stakeholder* recebeu um número de votos equivalente a 30% do total de cenários avaliados, permitindo que distribuíssem seus votos conforme julgassem mais adequado (Bass *et al.*, 2021). Considerando os 8 cenários de qualidade previamente definidos (Subseção 6.3.5), cada participante pôde distribuir 3 votos entre os cenários que julgou mais relevantes. A avaliação contou com a participação de 8 especialistas, totalizando 24 votos. A Figura 6.12 sumariza a distribuição de votos por cenário.

A análise dos resultados revela que os cenários de **Manutenibilidade** e **Capacidade de Interação** emergiram como os mais votados, com 6 e 4 votos, respectivamente. Esse resultado reforça a percepção dos *stakeholders* de que a capacidade de manter e evoluir o sistema, bem como suportar diversas formas de interação com o usuário, garantindo o aspecto lúdico do jogo digital, é fator crítico para o sucesso da arquitetura em projetos concretos. Os participantes destacaram que a facilidade de adição de novos recursos é fundamental em arquiteturas de jogos, enquanto a capacidade de interação foi considerada um pilar essencial, dado que jogos digitais dependem intrinsecamente da diversidade de mecanismos interativos.

Entre os três cenários inicialmente identificados como mais importantes pelos tomadores de decisão, o cenário de ***Open Source*** demonstrou menor prioridade, obtendo apenas 2 votos. Os *stakeholders* reconheceram que, embora a arquitetura ofereça uma base sólida para implementação e manutenção de projetos, sua contribuição para cenários específicos de *open source* não está suficientemente clara. Contudo, observou-se que a

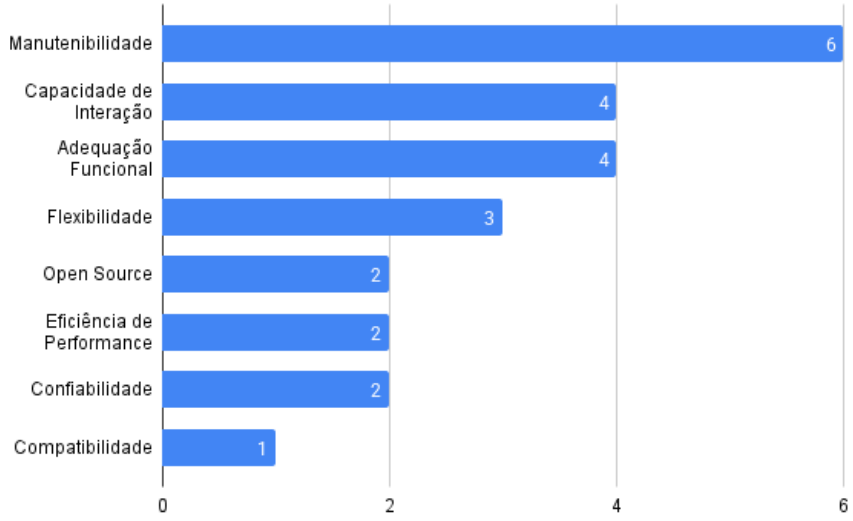

Figura 6.12: *Priorização dos cenários*

estrutura modular foi apontada como facilitadora da colaboração entre desenvolvedores, reduzindo conflitos em projetos colaborativos. Adicionalmente, a flexibilidade arquitetural foi reconhecida como um fator positivo para o trabalho em equipes distribuídas, ainda que a documentação represente um desafio adicional.

Em relação aos cenários inicialmente classificados como secundários, **Adequação Funcional** e **Flexibilidade** destacaram-se como segundo e terceiro atributos mais relevantes, respectivamente. Isso reflete a preocupação dos *stakeholders* com a capacidade da arquitetura de atender aos requisitos funcionais e adaptar-se a mudanças do ambiente e da situação de execução, aspectos considerados vitais para a longevidade do projeto.

Por outro lado, cenários como **Eficiência de *Performance***, **Compatibilidade** e **Confiabilidade** receberam menor ênfase quantitativa, alinhando-se com a avaliação preliminar dos tomadores de decisão. Entretanto, um dos *stakeholders* ressalta que a eficiência de desempenho permanece crucial para aplicações em tempo real, como jogos digitais, e que a confiabilidade é um atributo valorizado pelos usuários finais, especialmente considerando a complexidade inerente às simulações interativas.

Em síntese, embora existam divergências pontuais entre a priorização dos *stakeholders* e a classificação inicial dos tomadores de decisão, a análise dos votos e comentários revela um alinhamento consistente na maioria dos casos. Os atributos de **Manutenibilidade** e **Capacidade de Interação** confirmaram-se como pilares centrais para o sucesso em ambientes de desenvolvimento colaborativo. Apesar de o ***Open Source*** não ter sido identificado como aspecto crítico, sua natureza intrínseca à proposta arquitetural indica a necessidade de maior atenção e aprimoramento para garantir seu pleno atendimento.

6.4.2 Etapa 8: Análise das Abordagens Arquiteturais com os Cenários

Nesta etapa, procedeu-se à replicação das atividades realizadas na Etapa 6, agora incorporando as avaliações e comentários fornecidos pelos *stakeholders*. Os resultados quantitativos das pontuações atribuídas são sumarizados na Tabela 6.2, com os *stakeholders* identificados como S1, S2, e assim sucessivamente, enquanto as análises qualitativas foram integradas à avaliação dos cenários arquiteturais.

A Tabela 6.2 evidencia padrões distintos nas percepções dos *stakeholders* quanto aos atributos de qualidade avaliados. Os atributos de **Capacidade de Interação**, **Adequação Funcional** e **Compatibilidade** emergiram como os mais bem avaliados, refletindo o reconhecimento da adequação da arquitetura aos requisitos fundamentais de desenvolvimento de jogos. Em contrapartida, os atributos de **Eficiência de *Performance*** e **Confiabilidade** obtiveram avaliações mais modestas, indicando preocupações específicas que demandam atenção no processo de refinamento arquitetural. Além disso, observa-se uma divergência significativa entre as avaliações da equipe especializada e dos *stakeholders*, reflexo das distintas perspectivas e experiências desses grupos. Enquanto a equipe de avaliação tendeu a analisar a arquitetura sob uma ótica técnica mais aprofundada, os *stakeholders* demonstraram maior sensibilidade a aspectos práticos de implementação e usabilidade.

Por fim, os comentários qualitativos dos *stakeholders* revelam apreciação pela abordagem de combinação de arquiteturas consolidadas no domínio do desenvolvimento de jogos, reconhecendo o potencial sinérgico desta estratégia. Contudo, manifestam preocupação quanto à efetiva incorporação dos pontos fortes das arquiteturas base e à possível herança de suas limitações. Adicionalmente, destacam desafios relacionados à compreensão integral da proposta arquitetural, sugerindo que a complexidade resultante da integração de múltiplos padrões pode comprometer a clareza conceitual.

Tabela 6.2: *Pontuações atribuídas aos atributos de qualidade pelos stakeholders*

Atributo de Qualidade	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**Média**
Adequação Funcional	5	5	4	5	5	5	5	4	4.75
Eficiência de *Performance*	4	5	3	5	4	4	4	4	4.125
Compatibilidade	5	5	5	5	5	4	4	5	4.75
Capacidade de Interação	5	5	4	5	5	5	5	5	4.875
Confiabilidade	5	5	3	5	4	3	4	5	4.25
Manutenibilidade	5	5	3	5	5	3	4	4	4.25
Flexibilidade	4	5	5	5	5	4	4	5	4.625
Open Source	5	5	5	5	4	4	5	5	4.75

O primeiro cenário, apresentado na Tabela 6.3, trata da Apropriação Funcional da Adequação Funcional, visando garantir que as funcionalidades do jogo sejam implementadas de forma completa e eficiente. A análise do cenário revela que a arquitetura deve priorizar mecanismos de auxílio ao usuário para garantir intuitividade.

Tabela 6.3: *Análise do Cenário de Apropriação Funcional*

Resumo do Cenário	O jogador tenta interagir com controles do jogo para executar tarefa bem definida, como convidar amigo ou modificar configurações.
Objetivos de Negócio	Garantir experiência intuitiva e eficiente para jogadores de diferentes níveis de experiência.
Atributo de Qualidade	Adequação Funcional
Análise Arquitetural	A arquitetura deve incluir mecanismos de auxílio ao usuário, como *tooltips* ou tutoriais integrados.
Riscos	Fluxos complexos ou confusos podem dificultar a execução da ação pelo usuário.
Trade-offs	

O segundo cenário, na Tabela 6.4, avalia o Comportamento de Tempo da Eficiência de *Performance*, essencial para jogos que exigem resposta em tempo real. O cenário de Comportamento de Tempo evidenciou a necessidade de mecanismos robustos de gerenciamento de recursos, para garantir a capacidade da arquitetura em manter *performance* adequada em cenários complexos. Além disso, o *trade-off* identificado entre modularidade e *performance* representa um desafio crítico que deve ser abordado no refinamento arquitetural.

Tabela 6.4: *Análise do Cenário de Comportamento de Tempo*

Resumo do Cenário	O jogo processa cena com grande número de entidades que devem ser renderizadas mantendo taxa de *frame rates* estável.
Objetivos de Negócio	Manter experiência de jogo fluida e responsiva mesmo em cenários complexos.
Atributo de Qualidade	Eficiência de *Performance*
Análise Arquitetural	A arquitetura deve possuir mecanismos de gerenciamento dinâmico de recursos para garantir desempenho estável mesmo sob picos de uso.
Riscos	A arquitetura não pressupõe balanceamento de carga entre servidores quando há uso destes.
Trade-offs	A divisão modular dos *plug-ins*, por sua separação definida pela arquitetura, apesar de facilitar a colaboração e aumentar a manutenibilidade, pode prejudicar a *performance* do jogo.

O terceiro cenário, na Tabela 6.5, aborda a Coexistência da Compatibilidade, crucial para jogos executados em ambientes multitarefa. A análise revela a necessidade de especificar claramente como a arquitetura lidará com recursos compartilhados. O *trade-off* entre compatibilidade multiplataforma e custo de manutenção emerge como consideração importante para implementação.

Tabela 6.5: *Análise do Cenário de Coexistência*

Resumo do Cenário	O jogo é executado simultaneamente com outros aplicativos, compartilhando recursos de áudio, rede e memória, e ambos devem executar normalmente.
Objetivos de Negócio	Garantir que o jogo não cause conflitos com outros softwares no sistema.
Atributo de Qualidade	Compatibilidade
Análise Arquitetural	A arquitetura deve apresentar uma interface ou camada específica de interação com o SO, além de como lidar com recursos de diferentes SOs.
Riscos	Falta de interface específica com o SO pode causar conflitos de recursos com outros aplicativos.
Trade-offs	Para o software funcionar com qualidade em diferentes SOs, pode ser necessário utilizar linguagens nativas da plataforma, o que pode aumentar o custo de manutenção.

O quarto cenário, na Tabela 6.6, trata do Engajamento do Usuário da Capacidade de Interação, fundamental para retenção de jogadores. A análise sugere a necessidade de incorporação explícita de diretrizes de acessibilidade para ampliar o engajamento de diferentes perfis de usuários.

Tabela 6.6: *Análise do Cenário de Engajamento do Usuário*

Resumo do Cenário	O jogador completa nível desafiador e recebe *feedback* visual/sonoro com recompensas tangíveis.
Objetivos de Negócio	Aumentar engajamento e satisfação do jogador, incentivando continuidade no jogo.
Atributo de Qualidade	Capacidade de Interação
Análise Arquitetural	A arquitetura pode esclarecer a adoção da qualidade de Acessibilidade.
Riscos	
Trade-offs	

O quinto cenário, na Tabela 6.7, avalia a ausência de falhas de confiabilidade, crítica para a qualidade do software. A análise revela a necessidade de mecanismos explícitos de tratamento de falhas e de recuperabilidade. A natureza do padrão ECS, embora eficiente, introduz riscos à integridade dos dados devido à mutabilidade dos componentes. A relação diretamente proporcional entre complexidade do sistema e probabilidade de falhas emerge como princípio fundamental a ser considerado no projeto arquitetural.

Tabela 6.7: *Análise do Cenário de Ausência de Falhas*

Resumo do Cenário	Testes unitários, de integração e de sistema são executados sob condições normais de operação e devem passar com êxito, garantindo a confiabilidade do sistema.
Objetivos de Negócio	Garantir estabilidade e confiabilidade do jogo em diferentes cenários de uso.
Atributo de Qualidade	Confiabilidade
Análise Arquitetural	A arquitetura deve detalhar como os testes são realizados, deve incluir descrição dos mecanismos para tratamento de falhas, assim como necessidade de incorporar atenção ao requisito de Recuperabilidade.
Riscos	Mutabilidade dos dados dos componentes, pela natureza do ECS, pode representar um risco à integridade do programa.
Trade-offs	Com o aumento da complexidade do sistema, a possibilidade de ocorrerem erros aumenta, o que reduz a confiabilidade do sistema.

O sexto cenário, na Tabela 6.8, aborda a Modificabilidade da Manutenibilidade, essencial para projetos *open source*. A análise identifica a necessidade de documentação detalhada das interfaces de *plugins* para facilitar a contribuição da comunidade. O *trade-off* entre manutenibilidade e complexidade é particularmente relevante neste contexto, onde a capacidade de extensão deve ser equilibrada com a usabilidade para desenvolvedores menos experientes.

Tabela 6.8: *Análise do Cenário de Modificabilidade*

Resumo do Cenário	Desenvolvedor tenta adicionar uma nova personagem ou mecânica usando ferramentas e interfaces fornecidas.
Objetivos de Negócio	Facilitar a extensão do jogo pela comunidade sem modificar o núcleo do sistema.
Atributo de Qualidade	Manutenibilidade
Análise Arquitetural	A arquitetura deve detalhar a forma de integração de recursos adicionais por meio de *plugins*, descrevendo melhor a sua interface.
Riscos	A compreensão da arquitetura pode ser bastante desafiadora para novos desenvolvedores.
Trade-offs	A manutenibilidade nesse projeto é prejudicada devido ao aumento da complexidade, que torna mais difícil ter uma visão global clara do funcionamento do sistema.

O sétimo cenário, na Tabela 6.9, trata da Adaptabilidade da Flexibilidade, importante para uso do software em diferentes ambientes. A análise revela que a relação entre com-

plexidade e flexibilidade replica o padrão observado em outros atributos, reforçando a necessidade de equilíbrio entre capacidade de adaptação e simplicidade de implementação.

Tabela 6.9: *Análise do Cenário de Adaptabilidade*

Resumo do Cenário	Engenheiro de DevOps implanta *build* do jogo em ambiente diferente usando *scripts* fornecidos.
Objetivos de Negócio	Garantir portabilidade e facilidade de *deploy* em diferentes configurações.
Atributo de Qualidade	Flexibilidade
Análise Arquitetural	Nenhuma observação adicional foi encontrada para este atributo.
Riscos	
Trade-offs	Assim como a manutenibilidade, o aumento da complexidade pode prejudicar a flexibilidade.

O oitavo cenário, conforme ilustrado na Tabela 6.10, avalia a Adoção de Licença do *Open Source*, destacando a necessidade de documentação clara dos termos de uso para a comunidade *open source*. A análise enfatiza a importância da documentação e processos bem definidos para o sucesso de projetos colaborativos, além de identificar a necessidade de integrar mecanismos de coleta de *feedback* e estabelecer práticas claras de versionamento e CI/CD.

Tabela 6.10: *Análise do Cenário de Adoção de Licença*

Resumo do Cenário	Um membro da comunidade acessa o repositório público do projeto para verificar os termos de licenciamento antes de utilizar, modificar ou compartilhar o código e *assets* do jogo.
Objetivos de Negócio	Permitir que usuários entendam claramente seus direitos sobre uso, modificação e redistribuição do projeto.
Atributo de Qualidade	*Open Source*
Análise Arquitetural	A arquitetura deve ressaltar a importância de uma documentação detalhada, como API, versionamento, *changelog*, além de descrever o processo de CI/CD. A arquitetura pode sugerir também o uso de um *plug-in* para coletar o *feedback* dos usuários.
Riscos	
Trade-offs	

6.4.3 Etapa 9: Apresentação dos resultados

Por fim, nesta etapa, todas as informações coletadas e adquiridas anteriormente são resumidas e apresentadas. Com base na análise das abordagens arquiteturais e dos cenários

concretos realizados na etapa anterior, conclui-se que a especificação da arquitetura de referência para jogos digitais *open-source* apresenta diversos riscos, os quais são descritos a seguir.

De modo geral, um dos principais riscos está relacionado à complexidade de equilibrar requisitos de *performance* e manutenibilidade sem comprometer a experiência do jogador. Nesse contexto, um desafio importante é a integração da arquitetura ECS com o padrão de Microkernel, onde há o risco de que o excesso de modularidade e abstração possa prejudicar o desempenho ao introduzir *overhead* desnecessário. Por outro lado, a ausência desses elementos pode comprometer a capacidade de extensão pela comunidade e a evolução do projeto.

Além disso, a confiabilidade, manutenibilidade e flexibilidade enfrentam riscos associados à complexidade arquitetural, uma vez que o aumento desta interfere no sistema como um todo, exigindo um custo maior para garantir que essas qualidades sejam satisfeitas. Ademais, a mutabilidade dos dados nos componentes do ECS e a falta de mecanismos explícitos de tratamento de falhas representam ameaças à estabilidade do sistema.

No contexto *open source*, a documentação inadequada e a complexidade de compreensão da arquitetura para novos contribuidores representam riscos significativos para o engajamento da comunidade. A ausência de especificações claras sobre processos de CI/CD e versionamento também pode comprometer a colaboração efetiva e a sustentabilidade do projeto.

Por fim, há riscos associados à coexistência de software em diferentes ambientes e à adaptabilidade para *deploy* em múltiplas plataformas. Caso a arquitetura não ofereça interfaces bem definidas com sistemas operacionais, a portabilidade do jogo pode ser prejudicada, impactando diretamente sua abrangência e adoção. Assim, esses desafios na arquitetura de referência devem ser abordados com soluções que garantam o equilíbrio entre os atributos considerados e os objetivos que a arquitetura possui.

6.5 *Technology Acceptance Model* (TAM)

O ***Technology Acceptance Model***, originalmente proposto por Davis *et al.*, é um modelo teórico amplamente utilizado para compreender e prever a aceitação de sistemas e tecnologias por parte dos usuários. O modelo postula que dois construtos centrais, a Utilidade Percebida (*Perceived Usefulness* - PU) e a Facilidade de Uso Percebida (*Perceived Ease of Use* - PEOU), são determinantes primários da atitude do usuário em relação ao uso da tecnologia, o que, por sua vez, influencia a intenção de uso e o comportamento real de adoção. A Utilidade Percebida refere-se ao grau em que o usuário acredita que o uso do sistema melhorará seu desempenho no trabalho, enquanto a Facilidade de Uso Percebida diz respeito à crença de que utilizar o sistema será livre de esforço (Davis *et al.*, 1989a; Davis *et al.*, 1989b).

Para operacionalizar esses construtos, o TAM recorre a instrumentos de pesquisa compostos por escalas *Likert*, com itens que avaliam as percepções dos usuários em relação às dimensões de PU e PEOU (Davis *et al.*, 1989a). Essas escalas foram validadas empiricamente em diversos contextos e são adaptáveis a diferentes tecnologias e cenários

de uso (C. H. Hsiao e Yang, 2011).

Neste estudo, adotou-se o método TAM para avaliação da OSG-RA, adaptando as escalas de PU e PEOU ao contexto de jogos digitais *open-source*. Foram utilizados seis itens para cada construto, totalizando doze perguntas, apresentadas na Tabela 6.11. As respostas são baseadas em uma escala de concordância de cinco pontos, variando de "Concordo totalmente" a "Discordo totalmente", com a possibilidade dos avaliadores acrescentarem uma justificativa para a escolha. Essa abordagem permite não apenas medir a aceitação do sistema em questão, mas também identificar barreiras e facilitadores relacionados à sua adoção.

Tabela 6.11: *Questionário TAM para avaliação de RA*

ID	Tópico	Pergunta
P1	Utilidade Percebida	A arquitetura me permitiria desenvolver jogos de forma mais rápida.
P2	Utilidade Percebida	A arquitetura melhoraria a qualidade dos jogos desenvolvidos.
P3	Utilidade Percebida	A arquitetura aumentaria minha produtividade no desenvolvimento.
P4	Utilidade Percebida	A arquitetura aumentaria minha eficácia no desenvolvimento de jogos.
P5	Utilidade Percebida	A arquitetura tornaria mais fácil desenvolver jogos.
P6	Utilidade Percebida	Eu consideraria esta arquitetura útil no desenvolvimento de jogos.
P7	Facilidade de Uso Percebida	Aprender a usar esta arquitetura seria fácil para mim.
P8	Facilidade de Uso Percebida	Seria fácil fazer com que esta arquitetura fizesse o que eu desejo.
P9	Facilidade de Uso Percebida	Minha interação com esta arquitetura seria clara e compreensível.
P10	Facilidade de Uso Percebida	Eu consideraria esta arquitetura flexível para interagir.
P11	Facilidade de Uso Percebida	Seria fácil para me tornar habilidoso no uso desta arquitetura.
P12	Facilidade de Uso Percebida	Eu consideraria esta arquitetura fácil de usar.

Após a realização da avaliação pelo método ATAM, todos os participantes, incluindo tanto a equipe de avaliação quanto os *stakeholders*, são convidados a responder ao questionário TAM apresentado na Tabela 6.11. As respostas devem ser elaboradas com base em suas concepções e experiências individuais junto com as visões e cenários apresentados durante a avaliação. Os resultados quantitativos derivados das escalas *Likert* são consolidados e sumarizados na Tabela 6.12, fornecendo uma medida padronizada da aceitação da tecnologia.

Tabela 6.12: *Resultado da TAM respondido pelos avaliadores*

ID	PA1	PA2	PA3	S1	S2	S3	S4	S5	S6	S7	S8	Média
P1	4	3	4	5	5	1	5	5	2	4	4	3.82
P2	5	5	5	5	5	4	5	5	3	5	4	4.64
P3	5	4	4	5	5	2	5	5	4	4	4	4.27
P4	4	5	4	5	5	2	5	5	4	4	4	4.27
P5	5	4	5	5	5	1	4	5	2	4	4	4
P6	4	5	5	5	5	3	5	5	5	5	4	4.64
P7	5	5	4	5	5	1	4	4	2	4	4	3.9
P8	2	5	5	5	5	3	5	5	4	4	5	4.36
P9	5	5	5	5	5	3	4	5	3	4	5	4.45
P10	5	5	5	5	5	5	4	4	4	5	5	4.72
P11	4	5	5	4	5	3	3	5	2	5	5	4.18
P12	5	5	5	5	5	1	4	5	2	4	4	4.09

6.5.1 Análise da Utilidade Percebida (PU)

A análise dos resultados referentes à Utilidade Percebida revela uma avaliação geralmente positiva da arquitetura proposta, com nota média de 4,273 nesse construto. Este resultado indica que os avaliadores reconhecem o potencial da arquitetura para aprimorar tanto a qualidade dos jogos desenvolvidos quanto o processo de desenvolvimento em si. Os itens P2 e P6 obtiveram as maiores médias neste construto, demonstrando consenso entre os participantes quanto à capacidade da arquitetura de melhorar a qualidade dos jogos e sua utilidade geral no contexto de desenvolvimento. Em contrapartida, os itens P1 e P5 apresentaram as avaliações mais modestas, sugerindo percepções mais reservadas quanto à capacidade da arquitetura de acelerar ou simplificar significativamente o processo de desenvolvimento de jogos digitais *open-source*.

A análise qualitativa dos comentários revela nuances importantes nessas percepções. Em relação ao potencial de aceleração do desenvolvimento (P1), observa-se uma divisão nas perspectivas dos avaliadores. PA1 e PA3 enfatizam a importância crítica de documentação abrangente como pré-requisito para que a arquitetura possa efetivamente contribuir para ganhos de velocidade. S3 e S6, no entanto, manifestam preocupação quanto à curva de aprendizado associada à arquitetura, argumentando que o conhecimento prévio necessário introduziria um *overhead* significativo que poderia comprometer ganhos imediatos de produtividade.

Quanto à melhoria da qualidade dos jogos (P2), PA2 indica que a arquitetura é percebida como facilitadora da consideração sistemática de atributos de qualidade que frequentemente são negligenciados em abordagens convencionais. A fundamentação da arquitetura em princípios estabelecidos e literatura abrangente é citada pelo PA3 como fator que contribui para essa percepção positiva. Adicionalmente, S3 destaca que os padrões arquiteturais adotados promovem benefícios em dimensões como flexibilidade e manutenibilidade, que impactam a qualidade final do produto.

No que concerne aos aspectos de produtividade (P3) e eficácia (P4) no desenvolvimento, S3 sugere que os benefícios podem ser mais pronunciados em fases posteriores do ciclo de vida do software, particularmente nas atividades de manutenção e evolução. Enquanto PA1 indica que a divisão arquitetural é reconhecida como potencial facilitadora de melhorias no processo de desenvolvimento, porém com a ressalva de que tais benefícios estão condicionados à qualidade e atualidade da documentação associada.

Um aspecto relevante emergente da análise é a percepção de que a adequação da arquitetura varia conforme o contexto de uso. Para projetos de grande porte e equipes numerosas, a abordagem holística e a segmentação em componentes são vistas como vantajosas. Contudo, para projetos de escopo reduzido ou equipes menores, avalia-se que uma arquitetura simplificada poderia ser mais apropriada, dado o *overhead* cognitivo associado à compreensão e implementação da arquitetura proposta.

A utilidade geral da arquitetura (P6) é reconhecida pela maioria dos participantes, embora com a ressalva recorrente sobre a necessidade de documentação adequada para realizar seu potencial. S6 argumenta que a abordagem arquitetural é considerada promissora para o contexto de jogos digitais *open-source*, sugerindo que seus benefícios podem ser relevantes nesse domínio.

Esta análise evidencia que, embora a arquitetura seja percebida como útil, sua efetiva utilização está condicionada a fatores complementares, notadamente a qualidade da documentação e a adequação ao contexto específico de projeto. A percepção de utilidade parece estar mais associada a benefícios de médio e longo prazo, como qualidade e manutenibilidade, do que a ganhos imediatos de produtividade.

6.5.2 Análise da Facilidade de Uso Percebida (PEOU)

A análise do construto de Facilidade de Uso Percebida revela uma avaliação moderadamente positiva, com média de 4,28, indicando que a arquitetura atende satisfatoriamente a este aspecto, embora existam oportunidades de melhoria identificadas pelos avaliadores. Entre os itens que compõem este construto, o P10 obteve a maior avaliação, demonstrando reconhecimento unânime quanto à flexibilidade da arquitetura para desenvolvimento de jogos digitais *open-source*. Em contraste, os itens P7, P11 e P12 apresentaram as menores médias, sugerindo desafios significativos relacionados à compreensão e utilização prática da arquitetura proposta.

A análise qualitativa dos comentários permite identificar padrões específicos nessas percepções. Em relação à facilidade de aprendizado (P7), observa-se preocupação recorrente com a complexidade inerente à arquitetura. S4 e S6 indicam que o uso combinado de múltiplos padrões arquiteturais introduz uma carga cognitiva considerável, embora PA3 reconheça que essa complexidade pode ser mitigada através de documentação abrangente e detalhada. A compreensibilidade da arquitetura é reconhecida, porém, com ressalvas sobre o esforço necessário para sua plena assimilação.

Quanto à facilidade de realização de tarefas específicas (P8), identifica-se uma ambiguidade na interpretação do item do questionário por PA1. Esta observação sugere a necessidade de refinamento no instrumento de coleta para melhor adequação ao contexto arquitetural. Não obstante, a arquitetura é considerada pelo PA3 ser apropriada para

o domínio de jogos digitais *open-source*, com reconhecimento de sua adaptabilidade a diferentes necessidades projetuais.

Os itens relacionados à facilidade de interação (P9) e à flexibilidade de uso (P10) não geraram comentários específicos, indicando possível concordância com as avaliações quantitativas ou ausência de preocupações significativas nestes aspectos particulares.

No que concerne ao domínio progressivo da arquitetura (P11), PA1 revela que o tempo necessário para se tornar proficiente está diretamente relacionado à qualidade e completude da documentação disponível. Avaliadores com diferentes níveis de experiência concordam que, embora exista uma curva de aprendizado inicial, a aquisição de familiaridade com a arquitetura permite sua utilização efetiva. S4 ressalta, no entanto, a importância da experiência prática para avaliação mais precisa deste aspecto.

A avaliação geral da facilidade de uso (P12) corrobora as percepções anteriores, com ênfase na complexidade arquitetural como fator limitante. A sofisticação inerente à proposta, embora reconhecida como necessária para atender aos requisitos de qualidade, é identificada como barreira potencial à adoção, especialmente para desenvolvedores com menor experiência ou em projetos de menor escala.

Esta análise evidencia uma tensão fundamental entre a sofisticação arquitetural necessária para atender aos requisitos de qualidade e a facilidade de uso desejada pelos potenciais desenvolvedores. A flexibilidade da arquitetura é amplamente reconhecida, porém a complexidade decorrente da integração de múltiplos padrões arquiteturais representa um desafio significativo para sua adoção imediata. A disponibilidade de documentação compreensiva e recursos de aprendizado estruturados emerge como fator crítico para superar essas barreiras iniciais e realizar o potencial completo da arquitetura proposta.

6.6 Considerações Finais

Este capítulo apresentou o processo de avaliação arquitetural da OSG-RA, empregando os métodos *Architecture Tradeoff Analysis Method* (ATAM) e *Technology Acceptance Model* (TAM). O objetivo central foi submeter a arquitetura de referência proposta a uma análise crítica e sistemática, visando verificar sua adequação aos requisitos de qualidade eliciados, identificar pontos de melhoria, riscos e *trade-offs*, além de avaliar sua potencial aceitação pelos *stakeholders* do domínio.

A aplicação do ATAM, conduzida em duas fases com uma equipe de avaliadores especialistas e posteriormente com *stakeholders* representativos, permitiu uma análise profunda dos atributos de qualidade. A construção e priorização de cenários de qualidade, seguida da análise detalhada de como a arquitetura os atende, gerou *insights* fundamentais. Os resultados, sumarizados nas Tabelas 6.1 e 6.2, indicam que a OSG-RA atende de forma satisfatória à maioria dos atributos essenciais. No entanto, a análise também revelou riscos significativos, particularmente relacionados à tensão entre a complexidade arquitetural, inerente à combinação de padrões, e atributos como desempenho, confiabilidade e facilidade de aprendizado. Os *trade-offs* identificados, especialmente entre modularidade e *performance*, e entre flexibilidade e custo de manutenção, são decisões críticas que devem ser considerados pela RA.

Complementarmente, a avaliação pelo TAM ofereceu uma perspectiva valiosa sobre a utilidade e a facilidade de uso percebidas da arquitetura. Os resultados apresentados na Tabela 6.12 sugerem uma boa aceitação geral, com os avaliadores reconhecendo o potencial da OSG-RA para melhorar a qualidade e a estruturação do desenvolvimento de jogos *open-source*. A categoria de PEOU obteve uma avaliação ligeiramente mais reservada, ecoando as preocupações do ATAM sobre a curva de aprendizado associada à arquitetura. A importância crucial de uma documentação clara, completa e acessível emergiu como um consenso entre ambos os métodos, sendo apontada como condição essencial para a efetiva adoção e utilidade prática da RA.

Em síntese, a avaliação conduzida valida a OSG-RA como uma proposta arquitetural fundamentada e relevante para o domínio, ao mesmo tempo em que a aprimora ao destacar suas fragilidades e áreas que demandam atenção especial em implementações futuras. Os riscos e *trade-offs* mapeados não invalidam a arquitetura, mas fornecem um guia crucial para seus usuários, permitindo que decisões informadas sejam tomadas durante o projeto de sistemas concretos. Os resultados positivos de aceitação indicam que a comunidade de *stakeholders* vê valor na proposta, desde que acompanhada do suporte documental e das ferramentas adequadas. Esta avaliação, portanto, cumpre seu papel ao encerrar o ciclo de projeto da arquitetura de referência, fornecendo evidências de sua viabilidade e utilidade, e estabelecendo uma base sólida para sua eventual instanciação, uso e evolução futura no ecossistema de desenvolvimento de jogos digitais *open-source*.

Capítulo 7

Conclusão

7.1 Ameaças à Validade

Esta seção tem como objetivo identificar e discutir as ameaças à validade interna, externa, de construção e de conclusão associadas ao projeto desenvolvido, que podem impactar os resultados e conclusões deste trabalho, bem como as estratégias adotadas para mitigá-las. Considerando que as ameaças relacionadas ao mapeamento sistemático já foram discutidas anteriormente na Subseção 3.4.1, esta seção focará nas ameaças introduzidas em outras etapas do desenvolvimento do trabalho.

Validade Interna: Uma ameaça principal desta categoria está relacionada à subjetividade na identificação dos Requisitos de Software e dos Requisitos da Arquitetura, que envolveu a interpretação dos estudos e a experiência prática dos pesquisadores. Essa subjetividade pode ter influenciado a seleção e priorização de requisitos. Para minimizar os vieses relacionados, o processo de extração e categorização dos requisitos foi conduzido de forma supervisionada, com discussões e análises qualitativas entre pesquisadores e especialistas, garantindo consistência e alinhamento com os objetivos do trabalho.

Validade Externa: Como este estudo tem foco principal na construção de uma arquitetura de referência para jogos digitais *open-source*, os resultados obtidos podem não ser totalmente aplicáveis a outras áreas de desenvolvimento, que podem exigir especificações próprias e não necessariamente precisam garantir aspectos de manutenibilidade, como requerido no contexto *open source*. Além disso, a participação de um número restrito de especialistas na avaliação pode comprometer a representatividade dos resultados. No entanto, a diversidade de conhecimentos dos especialistas envolvidos garantiu uma análise mais aprofundada e representativa da RA.

Validade de Construção: Neste trabalho, as ameaças à validade de construção podem decorrer do conhecimento inadequado dos requisitos da área de pesquisa e dos métodos utilizados, bem como da possível inadequação dos métodos de avaliação empregados. Para mitigar esses riscos, foi realizada uma pesquisa profunda sobre conceitos e necessidades específicas da área de estudo, conforme sumarizado no Capítulo 2. Os métodos de avaliação adotados, ATAM e TAM, embora não sejam específicos para o contexto de jogos digitais

open source, foram adaptados para se adequar a esta área de pesquisa, criando oito cenários de uso e adaptando as perguntas do formulário.

Validade de Conclusão: Para aumentar o grau de credibilidade da arquitetura de referência desenvolvida, foram adotados dois métodos sistemáticos amplamente utilizados, adaptados para a área de jogos digitais *open source*, para realizar a avaliação dos resultados obtidos. Para minimizar o viés introduzido durante a pesquisa, a avaliação foi realizada com a participação de três profissionais da área, proporcionando maior profundidade às análises realizadas. Adicionalmente, incluiu-se um grupo de *stakeholders* para avaliar a arquitetura sob suas perspectivas pessoais, garantindo maior abrangência e aplicabilidade.

7.2 Contribuição Científica

A principal contribuição fornecida por este trabalho é a construção de uma arquitetura de referência para o desenvolvimento de jogos digitais *open-source*. A OSG-RA estabelece diretrizes para a implementação de jogos concretos, disponibilizando documentos de requisitos, padrões arquiteturais e taxonomias para apoiar o processo de desenvolvimento de forma mais estruturada e eficiente, garantindo uma melhor qualidade do produto final. Espera-se também que a OSG-RA auxilie no desenvolvimento de jogos digitais *open-source* não apenas nos aspectos técnicos, mas também ao facilitar que desenvolvedores iniciantes possam iniciar ou participar de um projeto, preparando-os para contribuir não apenas em jogos *open-source*, mas para a comunidade *open-source* como um todo, uma vez que as experiências são compartilhadas nessas áreas.

Além da arquitetura estabelecida, os Requisitos do Sistema, Requisitos Arquiteturais, *stakeholders* e diferentes interesses identificados ao longo deste trabalho podem ser reutilizados em outros projetos de pesquisa. Considerando que a maioria dos requisitos identificados é genérica e aplicável a diversos contextos, estes podem ser utilizados em outros estudos que envolvam jogos digitais ou softwares *open-source*. Ademais, embora o conteúdo desenvolvido não seja universalmente válido, conforme identificado na validade externa, o processo sistemático adotado neste trabalho, comprovadamente eficiente por seu uso no desenvolvimento da arquitetura proposta, pode ser reutilizado por outros projetos para construir outras arquiteturas de referência, servindo como um guia para pesquisas posteriores.

As avaliações realizadas por meio dos métodos ATAM e TAM, embora tenham necessitado de modificações para atender às especificidades da área de pesquisa, demonstraram-se benéficas, comprovando a abrangência de aplicação desses métodos na avaliação de software. Os métodos adotados permitiram a identificação de possíveis refinamentos e ajustes na arquitetura com a ajuda de uma equipe de avaliação composta por especialistas da área, a fim de aumentar a confiança e a eficiência da RA no desenvolvimento de jogos digitais *open-source* de alta qualidade. Além disso, a avaliação realizada pelos *stakeholders* permitiu um maior alinhamento das decisões arquiteturais com os interesses reais do ambiente de uso, bem como a incorporação de observações sob diferentes perspectivas, contribuindo para o reconhecimento de ajustes previamente não identificados na RA.

Em suma, este trabalho não só oferece uma arquitetura de referência validada para impulsionar o desenvolvimento de jogos digitais *open-source*, como também disponibiliza

um conjunto de artefatos reutilizáveis e um processo metodológico que podem ser adotados e adaptados pela comunidade. O trabalho realizado estabelece, portanto, uma base sólida para futuras pesquisas e iniciativas práticas, com o potencial de elevar o padrão de qualidade e a colaboração no ecossistema de desenvolvimento de jogos em código aberto. A evolução contínua da RA, alimentada pelo feedback da comunidade e pela aplicação em projetos reais, representa o próximo passo natural para consolidar e expandir seu impacto.

7.3 Trabalhos Futuros

Apesar dos desafios enfrentados, como a adequação dos métodos de avaliação à especificidade da área de jogos *open-source*, foi possível desenvolver e validar a OSG-RA por meio de um processo sistemático. Visando dar continuidade a esta pesquisa, propõe-se a evolução da OSG-RA com base nos *feedbacks* e nas lacunas identificadas pelos avaliadores durante a aplicação dos métodos ATAM e TAM. Um dos focos primários será o aprimoramento da documentação, tornando as descrições dos componentes, suas funções e interações mais claras e abrangentes, um aspecto considerado essencial para o contexto colaborativo de *open source*.

Como parte da melhoria contínua da RA, sugere-se a condução de um estudo de caso para verificar sua eficácia na prática, por meio do desenvolvimento de um jogo digital concreto baseado nela. Além disso, pesquisas futuras podem envolver a reimplementação ou o desenvolvimento de novos jogos utilizando a RA, possibilitando uma avaliação comparativa que quantifique ganhos em termos de produtividade, manutenibilidade e qualidade. A inclusão de cenários adicionais e exemplos concretos na documentação, como sugerido pelos avaliadores, apoiará significativamente essa iniciativa, especialmente para desenvolvedores iniciantes.

Por fim, espera-se que a RA contribua para a maturidade do desenvolvimento de jogos digitais *open-source*, promovendo maior qualidade e favorecendo aspectos como reúso, modularidade e confiabilidade. Nesse contexto, planeja-se realizar uma análise quantitativa do impacto proporcionado pelo uso da RA, coletando métricas sobre a adoção de boas práticas e a redução do esforço de desenvolvimento. Com essa abordagem, será possível obter mais evidências concretas dos benefícios da utilização da RA, consolidando-a como uma ferramenta valiosa para a comunidade.

Apêndice A

Requisitos do sistema

Esse Apêndice apresenta os 52 Requisitos do Sistema (RS) identificados durante a pesquisa, fundamentais para a criação de uma arquitetura de referência para jogos digitais *open-source*.

RS-1: O jogo precisa atender aos padrões e expectativas do jogador.

RS-2: O jogo precisa ter uma organização visual para melhorar a experiência do jogador.

RS-3: O jogo precisa ser capaz de se ajustar dinamicamente às necessidades individuais dos usuários.

RS-4: O jogo precisa ser visual e interativamente atraente para envolver os jogadores.

RS-5: O jogo precisa incentivar e facilitar o trabalho em equipe.

RS-6: O jogo precisa incluir elementos que promovam uma competição saudável.

RS-7: O jogo deve garantir que os usuários possam compreender e interagir facilmente com o conteúdo do jogo.

RS-8: O jogo deve facilitar a comunicação e a interação contínuas entre jogadores e recursos on-line relevantes.

RS-9: O jogo deve oferecer desafios envolventes e significativos aos jogadores.

RS-10: O jogo deve funcionar perfeitamente, sem atrasos ou travamentos.

RS-11: O jogo deve estar acessível aos usuários sempre que precisarem.

RS-12: O jogo deve ser divertido e envolvente.

RS-13: O jogo deve utilizar recursos computacionais de forma eficiente.

RS-14: O jogo deve ser adequado e apropriado atendendo à todas as normas e regulamentos impostos, tanto a legislação ou regra de negócios.

RS-15: O jogo deve ser capaz de acomodar um número crescente de usuários.

RS-16: O jogo deve funcionar de forma confiável e consistente sob diversas condições, sem travar ou apresentar bugs significativos.

RS-17: O jogo precisa ter uma boa experiência.

RS-18: O jogo deve permitir experimentação flexível e segura por parte dos jogadores.

RS-19: O jogo deve ser projetado para incorporar facilmente novos conteúdos, recursos ou correções.

RS-20: O jogo precisa ser fácil de aprender a jogar.

RS-21: O enredo do jogo precisa garantir a imersão do jogador no ambiente do jogo.

RS-22: O jogo deve envolver os jogadores, permitindo-lhes interagir com o conteúdo, fazer escolhas e realizar feedback.

RS-23: Os problemas apresentados no jogo precisam ser intuitivos.

RS-24: O jogo precisa ser fácil de jogar.

RS-25: Os elementos do jogo precisam ser lúdicos.

RS-26: O jogo precisa ter uma facilidade de manutenção.

RS-27: O jogo deve incorporar diversas formas de mídia (texto, imagens, áudio, vídeo) para aprimorar o envolvimento.

RS-28: A interface do jogo deve ser intuitiva e fornecer caminhos claros para os jogadores se moverem pelo jogo e acessar diferentes recursos.

RS-29: O jogo deve apresentar informações de forma precisa, evitando interpretações subjetivas.

RS-30: O jogo precisa ser claro quanto às regras e funcionalidades.

RS-31: O jogo deve ser projetado para rodar em diversas plataformas (desktop, mobile, Web) sem modificações significativas.

RS-32: O nível de dificuldade ou complexidade do jogo deve aumentar gradualmente à medida que o jogador avança.

RS-33: O jogo deve fornecer documentação clara e controle de versão de todos os códigos e ativos.

RS-34: O jogo deve ser estruturado para facilitar a reutilização de código, ativos em vários módulos ou versões.

RS-35: O jogo deve satisfazer as expectativas dos jogadores.

RS-36: O jogo deve manter a integridade dos dados do jogador e evitar acesso não autorizado.

RS-37: O conteúdo do jogo deve ser apropriado para a faixa etária alvo e respeitar as sensibilidades culturais.

RS-38: O jogo deve ser fácil de entender, navegar e operar, especialmente em termos de regras e interface do usuário.

RS-39: O jogo deve oferecer uma experiência confortável e agradável aos jogadores.

RS-40: O jogo precisa ter uma viabilidade técnica.

RS-41: O jogo deve adotar uma licença de código aberto clara para permitir modificações e redistribuição pela comunidade.

RS-42: O jogo deve ser estruturado em módulos independentes para facilitar a reutilização e colaboração.

RS-43: O jogo deve possuir sistema de autenticação de jogadores, salvamento de progresso na nuvem.

RS-44: O jogo deve utilizar derramentas para desenvolvimento colaborativo, como histórico de versões e compartilhamento de projetos na nuvem.

RS-45: O jogo deve permitir permitir que jogadores redefinam teclas, ajustem sensibilidade ou usem dispositivos adaptativos.

RS-46: O jogo deve garantir que elementos de interface sejam descritos por áudio e compatíveis com tecnologias assistivas, como leitores de tela.

RS-47: O jogo deve fornecer guias detalhados para novos desenvolvedores, incluindo padrões de código, fluxo de trabalho e diretrizes para pull requests.

RS-48: O jogo deve manter fóruns, repositórios públicos para discussão de ideias, reporte de bugs e sugestões.

RS-49: O jogo deve reconhecer contribuidores da comunidade (ex: créditos no jogo, badges em perfis públicos) para incentivar participação ativa.

RS-50: O jogo deve coletar informações dos jogadores (pelo feedback ou coleta de dados do jogo) para identificar direções de desenvolvimento.

RS-51: O jogo deve fornecer ferramentas para que a comunidade crie mods, novos níveis ou assets, ampliando a vida útil do jogo.

RS-52: O jogo deve automatizar testes e builds para facilitar as contribuições da comunidade.

Referências

[Aallouche *et al.* 2007] Khalid Aallouche *et al.* "Implementation and evaluation of a background music reactive game". In: *Proceedings of the 4th Australasian conference on Interactive entertainment*. 2007, pp. 1–6 (citado na pg. 31).

[Aberdour 2007] Mark Aberdour. "Achieving quality in open-source software". *IEEE software* 24.1 (2007), pp. 58–64 (citado nas pgs. 46, 47).

[Ahmed *et al.* 2017] Faheem Ahmed, Muhammad Zia, Hasan Mahmood e Shayma Al Kobaisi. "Open source computer game application: an empirical analysis of quality concerns". *Entertainment Computing* 21 (2017), pp. 1–10 (citado nas pgs. 1, 9).

[Aires *et al.* 2020] Samanta F Aires, Jorge FR Barbosa e Charles AG Madeira. "Desenvolvendo jogos educacionais digitais inovadores e instigantes com o framework playeduc". *Proc. SBGames 2020–XIX Simpósio Brasileiro de Jogos e Entretenimento Digital* (2020) (citado na pg. 7).

[Alaoui *et al.* 2020] Younes Alaoui, Lotfi El Achaak e Mohammed Bouhorma. "Methodology to develop serious games for primary schools". In: *The Proceedings of the Third International Conference on Smart City Applications*. Springer. 2020, pp. 195–205 (citado na pg. 31).

[Aleem *et al.* 2016a] Saiqa Aleem, Luiz Fernando Capretz e Faheem Ahmed. "A digital game maturity model (dgmm)". *Entertainment Computing* 17 (2016), pp. 55–73 (citado na pg. 8).

[Aleem *et al.* 2016b] Saiqa Aleem, Luiz Fernando Capretz e Faheem Ahmed. "Game development software engineering process life cycle: a systematic review". *Journal of Software Engineering Research and Development* 4.1 (2016), p. 6 (citado na pg. 8).

[Alvare e Gordon 2015] Graham Alvare e Richard Gordon. "Ct brush and cancerzap!: two video games for computed tomography dose minimization". *Theoretical Biology and Medical Modelling* 12 (2015), pp. 1–23 (citado na pg. 32).

[Ampatzoglou e Stamelos 2010] Apostolos Ampatzoglou e Ioannis Stamelos. "Software engineering research for computer games: a systematic review". *Information and Software Technology* 52.9 (2010), pp. 888–901 (citado nas pgs. 1, 6–8).

[Angelov, Grefen *et al.* 2009] Samuil Angelov, Paul Grefen e Danny Greefhorst. "A classification of software reference architectures: analyzing their success and effectiveness". In: *2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture*. IEEE. 2009, pp. 141–150 (citado na pg. 14).

[Angelov, Trienekens *et al.* 2008] Samuil Angelov, Jos JM Trienekens e Paul Grefen. "Towards a method for the evaluation of reference architectures: experiences from a case". In: *European Conference on Software Architecture*. Springer. 2008, pp. 225–240 (citado nas pgs. 2, 14, 15).

[Araújo *et al.* 2018] Wendell Oliveira de Araújo, Eduardo Henrique da Silva Aranha e Charles Andrye Galvão Madeira. "Geração procedural de conteúdo aplicada a jogos digitais educacionais". *Jornada de Atualização em Informática na Educação* 7.1 (2018), pp. 1–20 (citado nas pgs. 6–9).

[Aulkemeier *et al.* 2016] Fabian Aulkemeier, Milan Schramm, Maria-Eugenia Iacob e Jos Van Hillegersberg. "A service-oriented e-commerce reference architecture". *Journal of theoretical and applied electronic commerce research* 11.1 (2016), pp. 26–45 (citado na pg. 2).

[Baker 2014] Christopher J Baker. "Video games: their effect on society and how we must modernize our pedagogy for students of the digital age" (2014) (citado na pg. 8).

[Bass *et al.* 2021] Len Bass, Paul Clements e Rick Kazman. *Software architecture in practice*. Addison-Wesley Professional, 2021 (citado nas pgs. 14–16, 70–72, 74, 80).

[Bassani 2022] Patricia Scherer Bassani. "Estudos de software e o protagonismo do código". In: *Simpósio Brasileiro de Sistemas de Informação (SBSI)*. SBC. 2022, pp. 245–252 (citado na pg. 13).

[Belle *et al.* 2021] Alvine B Belle *et al.* "Systematically reviewing the layered architectural pattern principles and their use to reconstruct software architectures". *arXiv preprint arXiv:2112.01644* (2021) (citado nas pgs. 58, 59).

[Bellotti *et al.* 2010] Francesco Bellotti, Riccardo Berta e Alessandro De Gloria. "Designing effective serious games: opportunities and challenges for research". *International Journal of Emerging Technologies in Learning (iJET)* 5.2010 (2010) (citado na pg. 8).

[Bethea *et al.* 2008] Darrell Bethea, Robert A Cochran e Michael K Reiter. "Server-side verification of client behavior in online games". *ACM Transactions on Information and System Security (TISSEC)* 14.4 (2008), pp. 1–27 (citado na pg. 31).

[Bi *et al.* 2018] Tingting Bi, Peng Liang e Antony Tang. "Architecture patterns, quality attributes, and design contexts: how developers design with them". In: *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE. 2018, pp. 49–58 (citado na pg. 60).

[Boehm 2006] Barry Boehm. "A view of 20th and 21st century software engineering". In: *Proceedings of the 28th international conference on Software engineering*. 2006, pp. 12–29 (citado na pg. 14).

[Bretthauer 2001] David Bretthauer. "Open source software: a history" (2001) (citado nas pgs. 9, 10).

[Cabane e Farias 2024] Hebert Cabane e Kleinner Farias. "On the impact of event-driven architecture on performance: an exploratory study". *Future Generation Computer Systems* 153 (2024), pp. 52–69 (citado na pg. 60).

[Carvalho 2017] Maira B Carvalho. "Serious games for learning: a model and a reference architecture for efficient game development" (2017) (citado nas pgs. xi, 20).

[Cloutier *et al.* 2010] Robert Cloutier *et al.* "The concept of reference architectures". *Systems Engineering* 13.1 (2010), pp. 14–27 (citado nas pgs. 2, 13–15).

[Crawford *et al.* 1984] Chris Crawford *et al.* "The art of computer game design" (1984) (citado na pg. 6).

[Da Silva *et al.* 2021] Josivan Pereira Da Silva, Paulo Henrique Gonçalves Pimentel, Luciano Gonçalves Pimentel e Ismar Frango Silveira. "Pixel python rpg: repurposing an entertainment game to an open educational resource for computer programming fundamentals". In: *2021 XVI Latin American Conference on Learning Technologies (LACLO)*. IEEE. 2021, pp. 326–333 (citado nas pgs. 45–47).

[Dairel *et al.* 2025] João Gabriel de Matos Dairel *et al.* "Uma proposta de arquitetura computacional para autoria de jogos digitais educacionais com suporte a técnicas de analíticas de aprendizagem" (2025) (citado nas pgs. 7, 8).

[Dannenhauer *et al.* 2019] Dustin Dannenhauer, Michael W Floyd, Jonathan Decker e David W Aha. "Dungeon crawl stone soup as an evaluation domain for artificial intelligence". *arXiv preprint arXiv:1902.01769* (2019) (citado na pg. 31).

[Davis *et al.* 1989a] Fred D Davis *et al.* "Technology acceptance model: tam". *Al-Suqri, MN, Al-Aufi, AS: Information Seeking Behavior and Technology Adoption* 205.219 (1989), p. 5 (citado nas pgs. 69, 87).

[Davis *et al.* 1989b] Fred D Davis, Richard P Bagozzi e Paul R Warshaw. "Technology acceptance model". *J Manag Sci* 35.8 (1989), pp. 982–1003 (citado nas pgs. 69, 87).

[De Oliveira *et al.* 2010] Lucas Bueno Ruas De Oliveira, Katia Romero Felizardo, Daniel Feitosa e Elisa Yumi Nakagawa. "Reference models and reference architectures based on service-oriented architecture: a systematic review". In: *European Conference on Software Architecture*. Springer. 2010, pp. 360–367 (citado na pg. 15).

[Deacon 2009] John Deacon. "Model-view-controller (mvc) architecture". *Online][Citado em: 10 de março de 2006.] http://www.jdl.co.uk/briefings/MVC.pdf* 28 (2009), p. 61 (citado na pg. 59).

[Dutra *et al.* 2021] Taynara Cerigueli Dutra *et al.* "Métodos de avaliaçao de ihc no contexto de jogos sérios educacionais: um mapeamento sistemático". In: *Simpósio Brasileiro de Informática na Educação (SBIE)*. SBC. 2021, pp. 564–575 (citado na pg. 9).

[Engelfriet 2009] Arnoud Engelfriet. "Choosing an open source license". *IEEE software* 27.1 (2009), pp. 48–49 (citado nas pgs. 11, 12).

[Farrapo *et al.* 2022] Hyuan P Farrapo, FF Rômulo Filho, José GR Maia e Paulo BS Serafim. "Drleague: a novel 3d environment for training reinforcement learning agents". In: *2022 21st Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*. IEEE. 2022, pp. 1–6 (citado nas pgs. 32, 45–47).

[Farshidi *et al.* 2020] Siamak Farshidi, Slinger Jansen e Jan Martijn van der Werf. "Capturing software architecture knowledge for pattern-driven design". *Journal of Systems and Software* 169 (2020), p. 110714 (citado na pg. 15).

[Feizizadeh *et al.* 2023] Bakhtiar Feizizadeh, Seyed Javad Adabikhosh e Soodabe Panahi. "A scenario-based and game-based geographical information system (gis) approach for earthquake disaster simulation and crisis mitigation". *Sustainability* 15.14 (2023), p. 11131 (citado na pg. 31).

[Ferreira 2005] AJL Ferreira. "Open source software". *Departamento de Engenharia Informática. Universidade de Coimbra* (2005) (citado nas pgs. 9, 12, 13).

[Fogel 2005] Karl Fogel. *Producing open source software: How to run a successful free software project*. "O'Reilly Media, Inc.", 2005 (citado nas pgs. 1, 13).

[Frank *et al.* 1996] Buschmann Frank, Meunier Regine, Rohnert Hans, Sommerlad Peter e Stal Michael. "Pattern-oriented software architecture: a system of patterns". *Wiley, ISBN 0* 471.95889 (1996), p. 7 (citado nas pgs. 15, 59).

[Franke *et al.* 2024] Sven Franke, Sandro Hermes e Moritz Roidl. "An educational game to learn picking techniques in warehousing–waremover". *Simulation & Gaming* (2024), p. 10468781241258606 (citado na pg. 31).

[Fremantle *et al.* 2015] Paul Fremantle *et al.* "A reference architecture for the internet of things". *WSO2 White paper* (2015), pp. 02–04 (citado na pg. 2).

[Fuggetta 2003] Alfonso Fuggetta. “Open source software—-an evaluation”. *Journal of Systems and software* 66.1 (2003), pp. 77–90 (citado nas pgs. 46, 47).

[Garcia *et al.* 2010] Mauro Neves Garcia, Silvana Mara Braga dos Santos, Raquel da Silva Pereira e George Bedineli Rossi. “Software livre em relação ao software proprietário: aspectos favoráveis e desfavoráveis percebidos por especialistas”. *Gestão & regionalidade* 26.78 (2010), pp. 106–120 (citado nas pgs. 1, 12, 13).

[Garlan, Allen *et al.* 1994] David Garlan, Robert Allen e John Ockerbloom. “Exploiting style in architectural design environments”. *ACM SIGSOFT software engineering notes* 19.5 (1994), pp. 175–188 (citado na pg. 16).

[Garlan e Shaw 2008] David Garlan e Mary Shaw. *Software Architecture*. 2008 (citado nas pgs. 1, 2).

[Gazis e Katsiri 2024] Alexandros Gazis e Eleftheria Katsiri. “E-polis: an innovative and fun way to gamify sociological research with an educational serious game–game development middleware approach”. *International Journal of Education and Information Technologies* 18 (2024), pp. 20–32 (citado na pg. 32).

[Geldhauser *et al.* 2022] Carina Geldhauser, Andreas Daniel Matt e Christian Stussak. “I am ai gradient descent–an open-source digital game for inquiry-based clil learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 11. 2022, pp. 12751–12757 (citado nas pgs. 31, 45–47).

[Genesio *et al.* 2024] Nikolas Oliver Sales Genesio, Alessandreia Marta de Oliveira, Edmar Welington Oliveira e Pedro Henrique Dias Valle. “Panorama de estudos sobre jogos educacionais digitais em educação em computação”. In: *Workshop sobre Educaçao em Computaçao (WEI)*. SBC. 2024, pp. 737–749 (citado na pg. 9).

[Gezici *et al.* 2019] Bahar Gezici *et al.* “Quality and success in open source software: a systematic mapping”. In: *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE. 2019, pp. 363–370 (citado nas pgs. 27, 30, 32).

[Goering *et al.* 2017] Andre Goering *et al.* “Reference architecture for open, maintainable and secure software for the operation of energy networks”. *CIRED 24* 2017.1 (2017), pp. 1410–1413 (citado nas pgs. xi, 19).

[Gokcen *et al.* 2018] Ajda Gokcen, Ethan Hill e Michael White. “Madly ambiguous: a game for learning about structural ambiguity and why it’s hard for computers”. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*. 2018, pp. 51–55 (citado nas pgs. 31, 45–47).

[Grosskurth e Godfrey 2005] Alan Grosskurth e Michael W Godfrey. “A reference architecture for web browsers”. In: *21st IEEE International Conference on Software Maintenance (ICSM’05)*. IEEE. 2005, pp. 661–664 (citado na pg. 2).

[Habernal *et al.* 2017] Ivan Habernal *et al.* "Argotario: computational argumentation meets serious games". *arXiv preprint arXiv:1707.06002* (2017) (citado nas pgs. 31, 45–47).

[Halvorsen e Raaen 2014] Stig Magnus Halvorsen e Kjetil Raaen. "Games for research: a comparative study of open source game projects". In: *Euro-Par 2013: Parallel Processing Workshops: BigDataCloud, DIHC, FedICI, HeteroPar, HiBB, LSDVE, MHPC, OMHI, PADABS, PROPER, Resilience, ROME, and UCHPC 2013, Aachen, Germany, August 26-27, 2013. Revised Selected Papers 19*. Springer. 2014, pp. 353–362 (citado na pg. 31).

[Haoues *et al.* 2017] Mariem Haoues, Asma Sellami, Hanêne Ben-Abdallah e Laila Cheikhi. "A guideline for software architecture selection based on iso 25010 quality related characteristics". *International Journal of System Assurance Engineering and Management* 8.Suppl 2 (2017), pp. 886–909 (citado na pg. 60).

[Hollins *et al.* 2017] Paul Hollins, Sarah Humphreys, Li Yuan, Gareth Sleightholme e Michael Kickmeier-Rust. "The "water cooler" game". In: *Proceedings of the 11th European Conference on Games Based Learning, ECGBL*. 2017, pp. 262–271 (citado na pg. 32).

[C. H. Hsiao e Yang 2011] Chun Hua Hsiao e Chyan Yang. "The intellectual development of the technology acceptance model: a co-citation analysis". *International Journal of Information Management* 31.2 (2011), pp. 128–136 (citado na pg. 88).

[H.-C. Hsiao 2007] Hui-Chun Hsiao. "A brief review of digital games and learning". In: *2007 first IEEE international workshop on digital game and intelligent toy enhanced learning (DIGITEL'07)*. IEEE. 2007, pp. 124–129 (citado na pg. 6).

[Hudson-Smith *et al.* 2023] Natalie V Hudson-Smith *et al.* "Nanoadventure: development of a text-based adventure game in english, spanish, and chinese for communicating about nanotechnology and the nanoscale". *Journal of chemical education* 100.6 (2023), pp. 2269–2280 (citado na pg. 31).

[Jacobs 2010] Stephen Jacobs. "Building an education ecology on serious game design and development for the one laptop per child and sugar platforms: a service learning course builds a base for peer mentoring, cooperative education internships and sponsored research". In: *2010 2nd International IEEE Consumer Electronics Society's Games Innovations Conference*. IEEE. 2010, pp. 1–6 (citado na pg. 32).

[Jeon *et al.* 2023] Hyeon-Chang Jeon *et al.* "Raidenv: exploring new challenges in automated content balancing for boss raid games". *IEEE Transactions on Games* (2023) (citado na pg. 31).

[Júnior *et al.* 2022] Ed Wilson Júnior, Kleinner Farias e Bruno da Silva. "On the use of uml in the brazilian industry: a survey". *Journal of Software Engineering Research and Development* 10 (2022), pp. 10–1 (citado na pg. 16).

[Kanode e Haddad 2009] Christopher M Kanode e Hisham M Haddad. "Software engineering challenges in game development". In: *2009 Sixth International Conference on Information Technology: New Generations*. IEEE. 2009, pp. 260–265 (citado na pg. 8).

[Kattilakoski 2019] Joel Kattilakoski. "The growth of video game industry: current status and future prospects" (2019) (citado nas pgs. 1, 6).

[Kazman *et al.* 1998] Rick Kazman *et al.* "The architecture tradeoff analysis method". In: *Proceedings. fourth ieee international conference on engineering of complex computer systems (cat. no. 98ex193)*. IEEE. 1998, pp. 68–78 (citado nas pgs. 69, 70).

[Kempkens *et al.* 2000] Ralf Kempkens, Peter Rösch, Louise Scott e Joerg Zettel. "A multi-layer multi-view architecture for software engineering environments". *Information and Software Technology* 42.2 (2000), pp. 141–149 (citado nas pgs. 58, 59).

[Kiel 2003] Lori Kiel. "Experiences in distributed development: a case study". In: *Proceedings of international workshop on global software development at ICSE, Oregon, USA*. 2003 (citado na pg. 13).

[Kitchenham e Charters 2007] BA Kitchenham e S Charters. *Guidelines for performing systematic literature reviews in software engineering*. en. KeroCite.ItemAlsoKnownAs: 2129771:7RP54LK8 2129771:G45N5C6G 2405685:ETPE564Y 2486141:KVCIQUQU. EBSE Technical Report, 2007 (citado nas pgs. 27, 28, 32).

[Klemke *et al.* 2015a] Roland Klemke, Shalini Kurapati, Heide Lukosch e Marcus Specht. "Lessons learned from creating a mobile version of an educational board game to increase situational awareness". In: *Design for Teaching and Learning in a Networked World: 10th European Conference on Technology Enhanced Learning, EC-TEL 2015, Toledo, Spain, September 15-18, 2015, Proceedings 10*. Springer. 2015, pp. 183–196 (citado na pg. 31).

[Klemke *et al.* 2015b] Roland Klemke, Shalini Kurapati, Heide Lukosch e Marcus Specht. "Transferring an educational board game to a multi-user mobile learning game to increase shared situational awareness". In: *Learning and Collaboration Technologies: Second International Conference, LCT 2015, Held as Part of HCI International 2015, Los Angeles, CA, USA, August 2–7, 2015, Proceedings 1*. Springer. 2015, pp. 583–594 (citado na pg. 31).

[Krasner, Pope *et al.* 1988] Glenn E Krasner, Stephen T Pope *et al.* "A description of the model-view-controller user interface paradigm in the smalltalk-80 system". *Journal of object oriented programming* 1.3 (1988), pp. 26–49 (citado na pg. 59).

[P. Kruchten *et al.* 2006] Philippe Kruchten, Henk Obbink e Judith Stafford. "The past, present, and future for software architecture". *IEEE software* 23.2 (2006), pp. 22–30 (citado na pg. 14).

[P. B. Kruchten 2002] Philippe B Kruchten. "The 4+ 1 view model of architecture". *IEEE software* 12.6 (2002), pp. 42–50 (citado nas pgs. 16, 17, 62–67).

[Laamarti *et al.* 2014] Fedwa Laamarti, Mohamad Eid e Abdulmotaleb El Saddik. "An overview of serious games". *International Journal of Computer Games Technology* 2014.1 (2014), p. 358152 (citado na pg. 8).

[Land 2002] Rikard Land. "A brief survey of software architecture". *Mälardalen Real-Time Research Center (MRTC) Report* (2002), pp. 1–15 (citado nas pgs. 15, 16).

[Lautert *et al.* 2023] Cíntia Lautert, Cláudia Clarice Klein Pires e Muriel Lago. "Jogos digitais como estratégia pedagógica para a compreensão dos conceitos matemáticos: um estudo de caso no ensino fundamental-anos finais". In: *Workshop de Informática na Escola (WIE)*. SBC. 2023, pp. 1262–1272 (citado na pg. 9).

[Liu *et al.* 2021] Buquan Liu, Chunguang Wu e Hao Guo. "A survey of operating system microkernel". In: *2021 International Conference on Intelligent Computing, Automation and Applications (ICAA)*. IEEE. 2021, pp. 743–748 (citado na pg. 59).

[Lopes *et al.* 2005] Leandro Teixeira Lopes *et al.* "Um modelo de processo de engenharia de requisitos para ambientes de desenvolvimento distribuído de software" (2005) (citado na pg. 13).

[Lucchese e Ribeiro 2009] Fabiano Lucchese e Bruno Ribeiro. "Conceituação de jogos digitais". *São Paulo* (2009), p. 7 (citado nas pgs. 1, 7).

[MacKenzie *et al.* 2006] C Matthew MacKenzie *et al.* "Reference model for service oriented architecture 1.0". *OASIS standard* 12.S18 (2006), pp. 1–31 (citado na pg. 15).

[Manchana 2021] Ramakrishna Manchana. "Event-driven architecture: building responsive and scalable systems for modern industries". *International Journal of Science and Research (IJSR)* 10.1 (2021), pp. 1706–1716 (citado na pg. 60).

[Mayer *et al.* 2022] Roberto Mayer, Paulo Varela, Michel Albonico, Adair Rohling e Vilmar Steffen. "Experiências de um jogo educacional digital para auxiliar no processo de ensino-aprendizagem de transformações químicas para o ensino médio". In: *Workshop de Informática na Escola (WIE)*. SBC. 2022, pp. 59–67 (citado na pg. 9).

[Medvidovic, Rosenblum, Redmiles *et al.* 2002] Nenad Medvidovic, David S Rosenblum, David F Redmiles e Jason E Robbins. "Modeling software architectures in the unified modeling language". *ACM Transactions on Software Engineering and Methodology (TOSEM)* 11.1 (2002), pp. 2–57 (citado na pg. 16).

[Medvidovic, Rosenblum e Taylor 1999] Nenad Medvidovic, David S Rosenblum e Richard N Taylor. "A language and environment for architecture-based software development and evolution". In: *Proceedings of the 21st international conference on Software engineering*. 1999, pp. 44–53 (citado na pg. 16).

[Medvidovic e Taylor 2010] Nenad Medvidovic e Richard N Taylor. "Software architecture: foundations, theory, and practice". In: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*. 2010, pp. 471–472 (citado na pg. 15).

[Meirelles 2013] Paulo Roberto Miranda Meirelles. "Monitoramento de métricas de código-fonte em projetos de software livre". *São Paulo* (2013), p. 19 (citado na pg. 13).

[Melo *et al.* 2020] Rafaela Melo, Fernanda Pires, Polianny Lima, Marcela Pessoa e David Braga Fernandes de Oliveira. "Metodologias para a criação de jogos educacionais: um mapeamento sistemático da literatura". *Simpósio Brasileiro de Informática na Educação (SBIE)* (2020), pp. 572–581 (citado na pg. 8).

[Michelson 2006] Brenda M Michelson. "Event-driven architecture overview". *Patricia Seybold Group* 2.12 (2006), pp. 10–1571 (citado na pg. 60).

[Wilson K Mizutani *et al.* 2021] Wilson K Mizutani, Vinícius K Daros e Fabio Kon. "Software architecture for digital game mechanics: a systematic literature review". *Entertainment Computing* 38 (2021), p. 100421 (citado na pg. 8).

[Wilson Kazuo Mizutani e Kon 2020] Wilson Kazuo Mizutani e Fabio Kon. "Unlimited rulebook: a reference architecture for economy mechanics in digital games". In: *2020 IEEE International Conference on Software Architecture (ICSA)*. IEEE. 2020, pp. 58–68 (citado nas pgs. xi, 21).

[Wilson Kazuo Mizutani e Kon 2024] Wilson Kazuo Mizutani e Fabio Kon. "Rulebook: an architectural pattern for self-amending mechanics in digital games". *IEEE Transactions on Games* 16.3 (2024), pp. 711–721 (citado nas pgs. 32, 46, 47).

[Moriconi *et al.* 2002] Mark Moriconi, Xiaolei Qian e Robert A Riemenschneider. "Correct architecture refinement". *IEEE transactions on software engineering* 21.4 (2002), pp. 356–372 (citado na pg. 16).

[Mouaheb *et al.* 2012] Houda Mouaheb, Ahmed Fahli, Mohammed Moussetad e Said Eljamali. "The serious game: what educational benefits?" *Procedia-Social and Behavioral Sciences* 46 (2012), pp. 5502–5508 (citado na pg. 8).

[Muller 2008] Gerrit Muller. "A reference architecture primer". *Eindhoven Univ. of Techn., Eindhoven, White paper* (2008), p. 24 (citado na pg. 14).

[Muratet *et al.* 2011] Mathieu Muratet, Patrice Torguet, Fabienne Viallet e Jean-Pierre Jessel. "Experimental feedback on prog&play: a serious game for programming practice". In: *Computer graphics forum*. Vol. 30. 1. Wiley Online Library. 2011, pp. 61–73 (citado na pg. 31).

[Elisa Y Nakagawa *et al.* 2014] Elisa Y Nakagawa, Milena Guessi, José C Maldonado, Daniel Feitosa e Flavio Oquendo. "Consolidating a process for the design, representation, and evaluation of reference architectures". In: *2014 IEEE/IFIP Conference on Software Architecture*. IEEE. 2014, pp. 143–152 (citado nas pgs. 1–3, 9, 13–15, 17, 18).

[Elisa Yumi Nakagawa 2006] Elisa Yumi Nakagawa. "Uma contribuição ao projeto arquitetural de ambientes de engenharia de software". Tese de dout. Universidade de São Paulo, 2006 (citado na pg. 15).

[Elisa Yumi Nakagawa e Antonino 2022] Elisa Yumi Nakagawa e Pablo Oliveira Antonino. "An overview of reference architectures". *Reference Architectures for Critical Domains: Industrial Uses and Impacts* (2022), pp. 5–15 (citado nas pgs. 15, 16).

[Elisa Yumi Nakagawa, Martins *et al.* 2009] Elisa Yumi Nakagawa, Rafael Messias Martins, K Felizardo e Jose Carlos Maldonado. "Towards a process to design aspect-oriented reference architectures". In: *Proceedings of the XXXV Latin American Informatics Conference (CLEI 2009)*. 2009, pp. 1–10 (citado na pg. 17).

[Elisa Yumi Nakagawa, Oquendo *et al.* 2012] Elisa Yumi Nakagawa, Flavio Oquendo e Martin Becker. "Ramodel: a reference model for reference architectures". In: *2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*. IEEE. 2012, pp. 297–301 (citado nas pgs. 2, 19).

[Newman 2012] James Newman. *Videogames*. Routledge, 2012 (citado nas pgs. 5, 6, 8).

[B. R. Oliveira *et al.* 2022] Brauner RN Oliveira *et al.* "An overview of software architecture education". In: *Congresso Ibero-Americano em Engenharia de Software (CIbSE)*. SBC. 2022, pp. 76–90 (citado na pg. 2).

[R. N. R. d. Oliveira *et al.* 2018] Rháleff Nascimento Rodrigues de Oliveira, Rodrigo Pennella Cardoso, Juliana Cristina Braga Braga e Rafaela Vilela da Rocha. "Frameworks para desenvolvimento de jogos educacionais: uma revisão e comparação de pesquisas recentes". In: *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*. Vol. 29. 1. 2018, p. 854 (citado na pg. 8).

[Pashkov 2021] Savelii Pashkov. "Video game industry market analysis: approaches that resulted in industry success and high demand" (2021) (citado nas pgs. 1, 6).

[V. F. A. Pereira e Valentin 2018] Valeria Fontenelle Angelim Pereira e Livia Stocco Sanches Valentin. "The mentalplus® digital game might be an accessible open source tool to evaluate cognitive dysfunction in heart failure with preserved ejection fraction in hypertensive patients: a pilot exploratory study". *International journal of hypertension* 2018.1 (2018), p. 6028534 (citado na pg. 32).

[Perens *et al.* 1999] Bruce Perens *et al.* "The open source definition". *Open sources: voices from the open source revolution* 1 (1999), pp. 171–188 (citado nas pgs. 9, 10).

[Perez-Medina *et al.* 2019] Jorge-Luis Perez-Medina *et al.* "Ephort: towards a reference architecture for tele-rehabilitation systems". *IEEE Access* 7 (2019), pp. 97159–97176 (citado nas pgs. xi, 22).

[Petersen *et al.* 2008] Kai Petersen, Robert Feldt, Shahid Mujtaba e Michael Mattsson. "Systematic mapping studies in software engineering". In: *12th international conference on evaluation and assessment in software engineering (EASE)*. BCS Learning & Development. 2008 (citado nas pgs. 27, 28, 30, 32).

[Petri *et al.* 2019] Giani Petri, Christiane Gresse Von Wangenheim e Adriano Ferreti Borgatto. "Meega+: um modelo para a avaliação de jogos educacionais para o ensino de computação". *Revista Brasileira de Informática na Educação* 27.03 (2019), pp. 52–81 (citado na pg. 13).

[Pollock *et al.* 2017] Catherine R Pollock *et al.* "Avaler's adventure: an open source game for dysphagia therapy". In: *Proceedings of the 26th International Conference on Software Engineering and Data Engineering, SEDE*. 2017 (citado na pg. 31).

[Pouhela *et al.* 2023] Franc Pouhela, Dennis Krummacker e Hans D Schotten. "Entity component system architecture for scalable, modular, and power-efficient iot-brokers". In: *2023 IEEE 21st International Conference on Industrial Informatics (INDIN)*. IEEE. 2023, pp. 1–6 (citado na pg. 58).

[Prikladnicki e Audy 2006] Rafael Prikladnicki e Jorge Luis Audy. "Construção do conhecimento e complexidade na área de engenharia de software". In: *Anais do II Workshop Um Olhar Sociotécnico sobre a Engenharia de Software (WOSES 2006)*. 2006, pp. 51–63 (citado na pg. 13).

[Rahmati e Tanhaei 2021] Zahed Rahmati e Mohammad Tanhaei. "Ensuring software maintainability at software architecture level using architectural patterns". *AUT Journal of Mathematics and Computing* 2.1 (2021), pp. 81–102 (citado na pg. 58).

[Ramirez e Rivera 2022] John Ramirez e Sergio Rivera. "Development of a video game for teaching in an electrical machine laboratory". *Métodos numéricos para cálculo y diseño en ingeniería: Revista internacional* 38.2 (2022), pp. 1–9 (citado na pg. 31).

[E. Raymond 1999] Eric Raymond. "The cathedral and the bazaar". *Knowledge, Technology & Policy* 12.3 (1999), pp. 23–49 (citado nas pgs. 45–47).

[E. S. Raymond 1998] Eric S Raymond. "Homesteading the noosphere" (1998) (citado na pg. 10).

[Ribeiro *et al.* 2019] Bruno Ribeiro, Fabiano Lucchese, Maycon Rocha e Vera Figueiredo. "Inteligência artificial em jogos digitais". *UNICAMP, sd Disponível em: http://www.dca.fee.unicamp.br/~martino/disciplinas/ia369/trabalhos/t4g3. Consultado em* 25 (2019) (citado nas pgs. 1, 7).

[Rideout *et al.* 2010] Victoria J Rideout, Ulla G Foehr e Donald F Roberts. "Generation m 2: media in the lives of 8-to 18-year-olds." *Henry J. Kaiser Family Foundation* (2010) (citado na pg. 8).

[Sabino e Kon 2009] Vanessa Sabino e Fabio Kon. "Licenças de software livre: história e características" (2009) (citado nas pgs. 10–13).

[Salgo *et al.* 2021] Alexander Salgo, Galen O'Shea, Kim Hellemans e Jim Davies. "Postsynaptic simulator: an open-source visual interactive simulation for teaching action potentials." *International Association for Development of the Information Society* (2021) (citado nas pgs. 31, 45–47).

[Sampaio e C. P. Pereira 2022] Leandro Pereira Sampaio e Claudia Pinto Pereira. "Jogo digital educativo para auxílio a crianças com autismo". In: *Simpósio Brasileiro de Informática na Educação (SBIE)*. SBC. 2022, pp. 597–608 (citado na pg. 9).

[Santos *et al.* 2013] José Filipe Marreiros Santos, Milena Guessi, Matthias Galster, Daniel Feitosa e Elisa Yumi Nakagawa. "A checklist for evaluation of reference architectures of embedded systems (s)." In: *SEKE*. Vol. 13. 2013, pp. 1–4 (citado nas pgs. 3, 18).

[Scacchi 2004] Walt Scacchi. "Free and open source development practices in the game community". *IEEE software* 21.1 (2004), pp. 59–66 (citado nas pgs. 1, 9, 12, 13).

[Schlagowski *et al.* 2024] Ruben Schlagowski *et al.* "From a social pov: the impact of point of view on player behavior, engagement, and experience in a serious social simulation game". In: *Proceedings of the 19th International Conference on the Foundations of Digital Games*. 2024, pp. 1–12 (citado nas pgs. 32, 45, 47).

[D. Silva *et al.* 2022] Douglas Silva, Fernanda Pires, Rafaela Melo e Marcela Pessoa. "Glboard: um sistema para auxiliar na captura e análise de dados em jogos educacionais". In: *Simpósio Brasileiro de Jogos e Entretenimento Digital (SBGames)*. SBC. 2022, pp. 959–968 (citado na pg. 9).

[J. P. D. Silva *et al.* 2021] Josivan Pereira Da Silva, Paulo Henrique Gonçalves Pimentel, Luciano Gonçalves Pimentel e Ismar Frango Silveira. "Pixel python rpg: repurposing an entertainment game to an open educational resource for computer programming fundamentals". In: *2021 XVI Latin American Conference on Learning Technologies (LACLO)*. 2021, pp. 326–333. doi: 10.1109/LACLO54177.2021.00041 (citado na pg. 31).

[Silva Teixeira *et al.* 2020] Thiago da Silva Teixeira, Helyane Bronoski Borges, Simone Nasser Matos, Vinicius Schultz Garcia da Luz e Tamara Liz Schwab Ribeiro. "Pegagente: modelagem de agentes por aprendizado de reforço em jogos educacionais". *RENOTE* 18.2 (2020), pp. 225–234 (citado na pg. 8).

[Software Engineering Institute 2025] Software Engineering Institute. *Software Engineering Institute Home Page*. Acessado em: 17 de novembro de 2025. 2025. url: https://www.sei.cmu.edu/ (citado na pg. 14).

[Sommerville 2011] Ian Sommerville. "Software engineering 9th edition". *ISBN-10* 137035152 (2011), p. 18 (citado na pg. 15).

[Souza e Prates 2022] Joana GR de Souza e Raquel O Prates. "Modelo para o desenvolvimento de jogos educacionais digitais por educadores do ensino fundamental brasileiro". In: *Simpósio Brasileiro sobre Fatores Humanos em Sistemas Computacionais (IHC)*. SBC. 2022, pp. 192–200 (citado na pg. 8).

[Steinmacher *et al.* 2015] Igor Steinmacher, Marco Aurelio Graciotto Silva, Marco Aurelio Gerosa e David F Redmiles. "A systematic literature review on the barriers faced by newcomers to open source software projects". *Information and Software Technology* 59 (2015), pp. 67–85 (citado nas pgs. 1, 12, 13).

[Susi *et al.* 2007] Tarja Susi, Mikael Johannesson e Per Backlund. "Serious games: an overview" (2007) (citado na pg. 8).

[Thomas 2024] Richard Thomas. "Microkernel architecture" (2024) (citado na pg. 59).

[Vahdat *et al.* 2016] Mehrnoosh Vahdat *et al.* "Learning analytics for a puzzle game to discover the puzzle-solving tactics of players". In: *Adaptive and Adaptable Learning: 11th European Conference on Technology Enhanced Learning, EC-TEL 2016, Lyon, France, September 13-16, 2016, Proceedings 11*. Springer. 2016, pp. 673–677 (citado na pg. 31).

[Vargas *et al.* 2014] Juan A Vargas, Lilia García-Mundo, Marcela Genero e Mario Piattini. "A systematic mapping study on serious game quality". In: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. 2014, pp. 1–10 (citado nas pgs. 27, 28, 30, 32).

[Venters *et al.* 2018] Colin C Venters *et al.* "Software sustainability: research and practice from a software architecture viewpoint". *Journal of Systems and Software* 138 (2018), pp. 174–188 (citado nas pgs. 2, 3).

[Voisard *et al.* 2025] Laurent Voisard, Henrique De Freitas Serra, Cristiano Politowski, Fabio Petrillo e Yann-Gael Guéhéneuc. "A mapping study of the entity component system pattern". In: *2025 IEEE/ACM 9th International Workshop on Games and Software Engineering (GAS)*. IEEE. 2025, pp. 33–40 (citado na pg. 57).

[Wagle *et al.* 2021] Surbhit Wagle *et al.* "Development and testing of a game-based digital intervention for working memory training in autism spectrum disorder". *Scientific reports* 11.1 (2021), p. 13800 (citado nas pgs. 31, 45–47).

[Wu e Lin 2001] Ming-Wei Wu e Ying-Dar Lin. "Open source software development: an overview". *Computer* 34.6 (2001), pp. 33–38 (citado nas pgs. 9, 45–47).

[Zavala *et al.* 2019] Edith Zavala, Xavier Franch e Jordi Marco. "Adaptive monitoring: a systematic mapping". *Information and software technology* 105 (2019), pp. 161–189 (citado nas pgs. 27, 28, 30, 32).

[Zhan *et al.* 2024] Yarong Zhan, Tengfei Wang e Xuecheng Bi. "Creative production in the digital age: a network analysis of the digital game industry in china". *Geoforum* 157 (2024), p. 104158 (citado nas pgs. 1, 6).

[Zyda *et al.* 2007] Michael Zyda *et al.* "Educating the next generation of mobile game developers". *IEEE Computer Graphics and Applications* 27.2 (2007), pp. 96–95 (citado na pg. 31).