FACULDADE DE TECNOLOGIA, CIÊNCIAS E EDUCAÇÃO

GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

Padrões de projeto orientado à objetos mais utilizados pela indústria de software

Sidnei Adão Luiz Izaias
Ana Paula dos Santos Braatz Vieira

RESUMO

Este estudo objetivou compreender o conhecimento sobre os padrões de projeto, bem como apresentar os mais utilizados pela indústria de software destacando exemplos de aplicação. Embora o profissional desenvolvedor de software venha a aprender técnicas bem mais elaboradas conforme sua maturidade, existe uma importância no aprendizado e iniciação no uso dos padrões de projeto. Neste estudo apresenta-se seis padrões de projetos, organizados através de seu propósito (criacional, estrutural e comportamental) e escopo (classe e objeto), destacando exemplos de uso e aplicação por meio de diagramas. Para tanto, foi utilizado como método de coleta de dados uma pesquisa quantitativa aplicado a um grupo de discentes e docentes com intuito de aprofundar sobre os conhecimentos sobre os padrões de projeto. A hipótese a ser estudada tinha base em analisar o desconhecimento dos padrões de projeto e os apresenta-los como boas práticas de solução de problemas de software. Diante de um cenário de pesquisa, ocorreu que vários desses padrões já eram de conhecimento dos entrevistados, que conforme a experiência de mercado de trabalho aumento maior são seus conhecimentos sobre o assunto. Enfim, por meio do estudo realizado e dos questionários aplicados foi possível compreender que os padrões de projeto necessitam de um aprimoramento, para que os objetivos do mercado de trabalho exigente preencha as futuras vagas que estão em ascensão.

Palavras-chave: Padrões de projeto. Catálogo de padrões. GoF.

ABSTRACT

This study aimed to understand the knowledge about design patterns, as well as to present the most used ones by the software industry highlighting examples of application. Although the professional software developer will learn much more elaborate techniques according to his maturity, there is an importance in learning

and initiating the use of design patterns. This study presents six design patterns, organized by their purpose (creative, structural and behavioral) and scope (class and object), highlighting examples of use and application through diagrams. To this end, the data collection method used was a quantitative survey applied to a group of students and professors with the intention of deepening their knowledge about project patterns. The hypothesis to be studied was based on analyzing the lack of knowledge about design patterns and presenting them as good practices for solving software problems. In the research scenario, it occurred that several of these patterns were already known to the interviewees, and that as their work market experience increases the more, they know about the subject. Finally, through the study performed and the questionnaires applied it was possible to understand that the design patterns need improvement, so that the objectives of the demanding job market fill the future vacancies that are on the rise.

Keywords: Design patterns. Pattern catalog. GoF.

Introdução

Com a alta demanda por profissionais de desenvolvimento para solucionar problemas cotidianos da indústria de software, o mercado vem buscando os profissionais mais experientes para que possam reduzir os custos diante de uma maior produtividade. Segundo levantamento realizado em agosto de 2019 pela Brasscom (Associação Brasileira das Empresas de Tecnologia da Informação e Comunicação) até 2024 serão 420 mil vagas para profissionais da área de Tecnologia de Informação (TI) no país, porém, hoje o Brasil forma cerca de 46 mil profissionais por ano (GROSSMANN, 2019).

Devido a busca por profissionais que atendam esta demanda, a indústria de software busca os que melhor qualificam-se com habilidades humanas (*softs skills*) e técnicas (*hards skills*), adequando-se as exigências de mercado. Outro levantamento realizado em maio de 2017 pela *freeCodeCamp*[1] e divulgado PRO Felipe Couto (2017) na Vulpi[2] no Brasil, aponta uma concentração de profissionais Full Stack Developer[3] representando cerca de 40,6% do mercado, desse número 92% concluíram ou estão concluindo o nível superior e cerca de 76,8% das empresas buscam profissionais com nível de conhecimento pleno ou sênior.

[1] freeCodeCamp - é uma organização sem fins lucrativos.
[2] Vulpi - plataforma especializada que conecta empresas com profissionais de tecnologia.
[3] Full Stack Developer - profissionais atuam em todo o projeto de software.

Assim, diante de um mercado exigente, apenas o curso superior não garante a solidez profissional, pelo contrário, o aprofundamento em uma determinada área como especialista e o conhecimento generalista abrangente em outras áreas da tecnologia, faz com que o desenvolvedor receba hoje o termo "dev em <T>". Segundo Karl Popper (2013) somos solucionadores de problemas e problemas podem atravessar diversas áreas do conhecimento, fazendo com que nos generalizemos, e tenhamos conhecimento nestas áreas além da nossa formação.

Para atender os requisitos de mercado, um desenvolvedor deve seguir algumas premissas para se elevar com solidez a carreira em um nível de desenvolvedor pleno ou sênior. O conhecimento de vários mecanismos e técnicas utilizadas pelo mercado de software por muitas vezes pode levar tempo e ter obstáculos para aqueles que estão começando e alguns desses mecanismos e técnicas ajudam aos novos desenvolvedores a mostrarem seus conhecimentos, por serem ágeis e de fácil aplicação, demonstrando que não é necessário reinventar técnicas de desenvolvimento ou reinventar a roda.

Algumas das técnicas de desenvolvimento que auxilia ao desenvolvedor ser mais ágil, são os padrões de soluções de problemas conhecidos como padrões de projeto, porém existem desenvolvedores que desconhecem essas técnicas, trazendo desvantagens a esses profissionais.

Desta forma, este trabalho irá apresentar alguns padrões de projeto utilizados no mercado de trabalho, destacando as vantagens e desvantagens de cada padrão. Inicialmente será apresentado de forma lúdica por meio de diagramas, o uso e conceito aplicado nos padrões de projeto.

1.1. Problema

Portanto, buscou-se reunir informações com o propósito de responder ao seguinte problema de pesquisa: Quais os efeitos da aplicação dos padrões mais utilizados pela indústria de software frente às desvantagens que o desenvolvedor tem ao não conhecer os padrões de projeto?

1.2. Objetivo

O presente trabalho tem como objetivo geral apresentar quais os efeitos da aplicação dos padrões mais utilizados pela indústria de software frente as desvantagens que o desenvolvedor tem ao não conhecer os padrões de projeto como forma de apresentar os padrões de projetos mais utilizados pela indústria de software, com a finalidade de apresentar a vantagem em colaborar para a obtenção de um design flexível, mais coeso e menos acoplado na/o estudo de aplicação dos padrões de projeto e técnicas de uso.

1.3. Objetivo específico

Para alcançar os objetivos, são destacados os seguintes objetivos específicos:

- Apresentar padrões de projeto utilizados no mercado de trabalho.
- Apresentar as desvantagens ao não utilizar os padrões de projeto e o prejuízo que ele acarreta.
- Apresentar as vantagens e ganhos ao utilizar um padrão de projeto adequado.

2. Referencial teórico

Para a utilização dos padrões aqui apresentados será necessário o conceito do paradigma de programação.

2.1. Paradigma de programação

Segundo Tucker e Noonan (2010), paradigma é um padrão de pensamento que guia um conjunto de atividades relacionadas. Sua aplicação na programação resulta na solução de problemas que relaciona programas e linguagens.

As linguagens de programação em sua maioria utilizam o conceito básico de programação imperativa criado por John Von Neumann, tendo como o princípio formalizar sequencialmente os passos a serem cumpridos em um algoritmo, tornando este, uma linguagem de alto nível, ou seja, entendível ao programador e logo após compilado a linguagem de máquina.

Por meio da linguagem imperativa um dos paradigmas mais utilizados foi o paradigma de programação estruturada, que por sua simplicidade tornou mais fácil a inserção de novos programadores, porém, devido a novas necessidades de processamento computacional, não é mais eficiente seu uso diante do cenário atual da programação, tendo em vista novos paradigmas, como o paradigma de programação orientada a objetos (MARTIN, 2020).

2.2. Programação Orientada a Objetos (POO)

2.2.1. História

A programação orientada a objetos vem do conceito de simulação, que simula os eventos do dia a dia. Esta teoria foi criada entre 1962 e 1968 por Keith Douglas Tocher utilizando modelos matemáticos para descrever algoritmos de simulação de eventos discretos, retratando mudança de estado da aplicação por meio do tempo e seus relacionamentos.

No início surgiram várias linguagens que utilizavam estes conceitos para simular os eventos, porém Kristen Nygaard e Ole-Johan Dahl criaram baseada em *ALGOrithmic Language* (ALGO 60), a linguagem SIMULA 67, reconhecida como linguagem com paradigma de programação orientada a objetos. Um de seus conceitos era a programação orientada a problemas e não a orientada a computadores, implicando na facilidade do seu uso. A desvantagens dessa geração de linguagens que somente executavam em seus imensos computadores de origem, SIMULA 67 rodava em uma UNIVAC 1107 (MARTIN, 2020).

Na década de 1970, com a chegada de computadores pessoais, Alan Kay, pesquisador da Xerox Parc, criou a linguagem Smalltalk, considerada até hoje a linguagem que evoluiu o conceito de manipulação de objetos (TUCKER e NOONAN, 2010).

2.2.2. Definição

A programação orientada a objetos é um paradigma de composição e iteração entre diversos componentes de software. Esses componentes são chamados de objetos (JUNGTHON e GOULART, 2016).

2.2.3. Pilares

De acordo com Deitel (2017), a programação orientada a objetos possui quatro pilares básicos, que tornam fáceis o uso do processo de programação, sendo:

- **Abstração**: é o modelo de um objeto que representa os detalhes relevantes para um contexto específico do mundo real.
- **Encapsulamento**: é a habilidade de um objeto ocultar estados e/ou comportamento dos demais objetos, expondo apenas o necessário ao resto do programa. Encapsulamento é o mesmo que tornar privado atributos e métodos a outras classes, podendo somente ser acessados de fora da classe somente através de métodos públicos da classe que os permite serem manipulados. Há ainda uma restrição chamada protegido, que torna os atributos e métodos acessíveis somente pelas subclasses.
- **Herança**: é a capacidade de construir novas classes a partir de classes já existentes, estendendo-as em novas classes e acrescentando novos métodos a elas, tendo como maior benefício o reaproveitamento de código.
- **Polimorfismo**: é a capacidade que um programa tem de chamar uma instancia de classe mesmo sem conhecer seu contexto atual.

2.3. Padrões de Projeto

Padrões de projeto são soluções customizáveis para problemas, conforme a necessidade de cada aplicação em determinadas situações.

De acordo com Gamma *et al.* (2000), na programação os padrões de projeto são descritos como técnicas para soluções de problemas já conhecidos, indiferente de qualquer linguagem, aplicando-os em determinados paradigmas de programação. Um exemplo é o uso de alguns pilares do paradigma da POO em outro paradigma, como a programação estrutural para soluções de problemas no uso de classes ao invés de estruturas sequenciais, isolando o sistema em várias estruturas, tornando mais fácil sua compreensão, manutenção e isolando os componentes.

Os padrões de projeto são confundidos com algoritmo. Uma vez que os padrões de projeto abstraem e identificam um aspecto chave de uma estrutura, enquanto, o algoritmo sempre define ações para atingir metas.

Gamma *et al*. (2000) destaca que os padrões de projeto possuem quatro elementos essenciais:

- **Nome do padrão**: torna mais fácil entender por sua descrição o que um padrão

faz, distinguindo um padrão do outro e ajuda na comunicação de todos os envolvidos no desenvolvimento do software, não sendo necessária sua explicação a toda equipe.

- **Problema**: descreve o contexto da aplicação para resolução de determinados problemas, por meio de estruturas de classe e objeto. Algumas vezes lista condições para um uso correto do padrão.
- **Solução**: fornece uma solução abstrata que compõe o padrão, relacionamento, responsabilidades e colaboração, tendo como objetivo ser um gabarito para sua aplicação.
- **Consequências**: com frequência envolvem o balanceamento de espaço e tempo, tendo como o maior fator a reutilização da aplicação, tornando o maior impacto sobre a flexibilidade, extensibilidade e a portabilidade de um sistema.

2.3.1. Organizações de Padrões de Projeto

Para a organização dos padrões de projeto, Gamma *et al*. (2000) destaca que são utilizados dois critérios de classificação, o propósito e o escopo.

O **propósito** é dividido em criacional, estrutural ou comportamental, determinando o que o padrão faz:

- **Criacional**: responsável pelo mecanismo de criação de classes e objetos deixando o código mais flexível e reutilizável.
- **Estrutural**: descrevem a composição de classes e objetos em estruturas maiores, mantendo-a flexível e eficiente.
- **Comportamental**: define a maneira com que classes e objetos se comunicam distribuindo responsabilidades entre eles.

O **escopo** é dividido em classes ou objetos, determinando qual tipo de relacionamento são aplicados:

- **Classe**: lidam com os relacionamentos entre as classes e suas subclasses, por meio de mecanismos de herança, tendo relacionamentos estáticos – em tempo de compilação.
- **Objetos**: lidam com os relacionamentos entre vários objetos, tendo relacionamentos dinâmicos – em tempo de execução.

O propósito e o escopo se relacionam de acordo com sua classificação e uso.

		Propósito		
		De criação	Estrutural	Comportamental
Escopo	Classe	Factory Method	Adapter (class)	Interpreter Template Method
	Objeto	Abstract Method Builder Prototype Singleton	Adapter (object) Bridge Composite Decorator Façade Flyweight Proxy	Chain of Responsability Command Iterator Mediator Memento Observer State Strategy Visitor

Quadro 1 – Classificação dos padrões de projeto

Fonte: GAMMA, 2000, p. 23.

2.3.2. Padrões de Criação

Os padrões de criação abstraem a instanciação, tornando assim independente a forma que os objetos são criados, aumentando a flexibilidade e reutilização de código existente.

2.3.2.1. Factory Method

O Factory Method é um padrão de projeto com proposito criacional pertencente ao escopo de classe. O objetivo deste padrão é fornecer uma interface de criação de objetos, mas permite que as subclasses decidam qual classe instanciar.

Um exemplo apresentado por Shvets (2021) descreve o uso do padrão Factory Method para criar elementos de interface de usuário (em inglês *user interface* - UI) multiplataforma sem que o código do cliente seja acoplado as classes UI concretas. Para cada sistema operacional (Windows e Linux), os elementos podem parecer diferentes, porém seu comportamento deve ser igual.

Na figura 1 a classe criadora (*Dialog*) declara o método fábrica (*createButton*) que deve retornar um objeto de uma classe produto (*Button*). As subclasses (*Windows Button e HTML Button*) da criadora geralmente fornecem a implementação desse método.

Criadores concretos (*WindowsDialog* e *WebDialog*) sobrescrevem o método fábrica (*createButton*) para mudar o tipo de produto resultante.

A interface do produto declara os métodos (*render* e *onClick*) que todos os produtos concretos devem implementar.

A aplicação seleciona o tipo de criador dependendo da configuração do sistema.

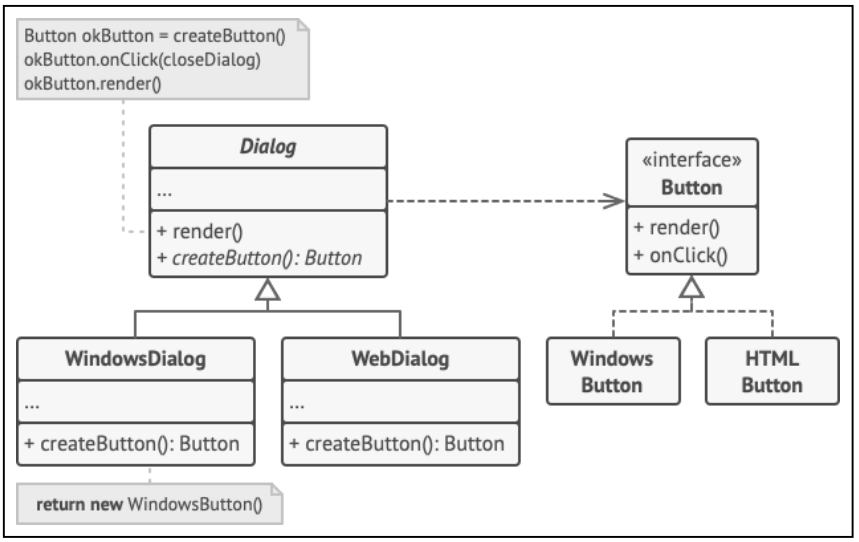

Figura 1: Exemplo de diálogo de plataforma cruzada.

Fonte: https://refactoring.guru/images/patterns/diagrams/factory-method/example.png

A aplicabilidade consiste em quando o desenvolvedor não souber os tipos e dependências exatas dos objetos de acordo com o que o código deve funcionar.

As consequências são dadas pelo fato de evitar o firme acoplamento entre o criador e os produtos concretos, porém uma desvantagem é que os clientes devem ter que fornecer subclasses da classe criadora.

2.3.2.2. Abstract Factory

O Abstract Factory é um padrão de projeto com propósito criacional pertencente ao escopo de objeto e o objetivo deste padrão é permitir a produção de famílias de objetos relacionados ou dependentes que possam trabalhar juntos.

Um exemplo apresentado por Gamma *et al.* (2000) descreve que *toolkits* que contêm componentes para interfaces gráficas funcionando em sistemas operacionais distintos como Windows e Mac. Normalmente apresentam-se várias classes representando elementos distintos como botões (*button*) e caixas de checagem (*checkbox*), sendo que cada sistema operacional tem a necessidade de implementar a classe conforme seu tipo específico, que nesse caso teríamos famílias de classes de elementos gráficos separadas por grupos de sistemas operacionais.

O padrão Abstract Factory garante que classes de mesma família trabalhem juntas, de forma que não misturem *WinButton* e *MacCheckbox*.

Na figura 2 a interface fábrica abstrata (*GUIFactory*) declara um conjunto de métodos (*createButton* e *createCheckbox*) que retorna diferentes produtos abstratos (*Button* e *Checkbox*). Estes produtos são chamados de família e estão relacionados por um tema ou conceito de alto nível.

As fábricas concretas (*WinFactory* e *MacFactory*) produzem uma família de produtos que pertencem a uma única variante garantindo que os produtos resultantes sejam compatíveis.

Cada produto distinto (*WinButton*, *WinCheckBox*, *MacButton* e *MacCheckbox*) de uma família de produtos deve ter uma interface base (*Button* e *Checkbox*).

O código cliente (*Application*) trabalha com fábricas e produtos apenas por meio de tipos abstratos (*GUIFactory*, *Button* e *Checkbox*). Isso permite que possa passar qualquer subclasse fábrica ou de produto para o código cliente sem quebrá-lo.

A aplicação seleciona o tipo de fábrica dependendo da atual configuração do ambiente e cria o *widget* no tempo de execução.

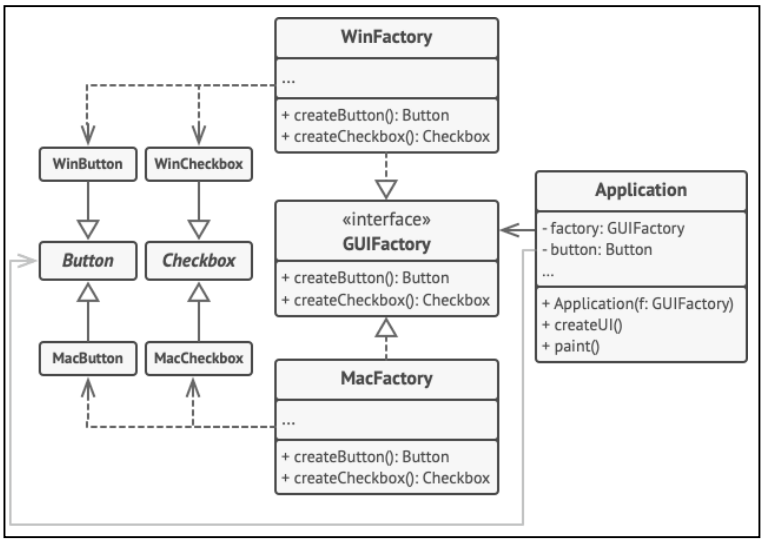

Figura 2: Exemplo das classes UI multiplataforma.

Fonte: https://refactoring.guru/images/patterns/diagrams/abstract-factory/example.png

A aplicabilidade se dá quando há necessidade de trabalhar com diversas famílias de produtos relacionados, não havendo a necessidade de dependência de classes concretas dos produtos.

As consequências são dadas pelo fato do isolamento das classes concretas, fácil troca de famílias de produtos, além de assegurar que os objetos pertencem a uma mesma família, porém o código se torna mais complexo no decorrer de novas interfaces e classes introduzidas.

Muitos projetos implementam inicialmente o padrão de projeto Factory

Method, por ter uma complexidade menor e ser mais customizável, evoluindo para Abstract Factory.

2.3.3. Padrões de Estrutura

Os padrões de estrutura se preocupam como os objetos são compostos para formar estruturas maiores mantendo essas estruturas flexíveis e eficientes.

2.3.3.1. Adapter

O Adapter é um padrão de projeto com proposito estrutural pertencente ao escopo de classe e o objetivo é permitir que objetos com interfaces incompatíveis possam colaborar entre si.

Um exemplo apresentado por Shvets (2021) descreve o uso do padrão Adapter para solucionar o conflito entre pinos quadrados e buraco redondos, tendo como solução para o problema o adaptador fingir ser um pino redondo para adequação do pino quadrado.

Na figura 3 existem duas classes com interfaces compatíveis (*RoundHole* e *RoundPeg*) ambas para pinos redondos, porém existe uma classe incompatível (*SquarePeg*) com pino quadrado. Para a utilização, uma classe adaptadora permite que os pinos quadrados encaixem em buracos redondos. Está classe estende a classe *RoundPeg* para que o objeto seja adaptado e se comporte como pinos redondos.

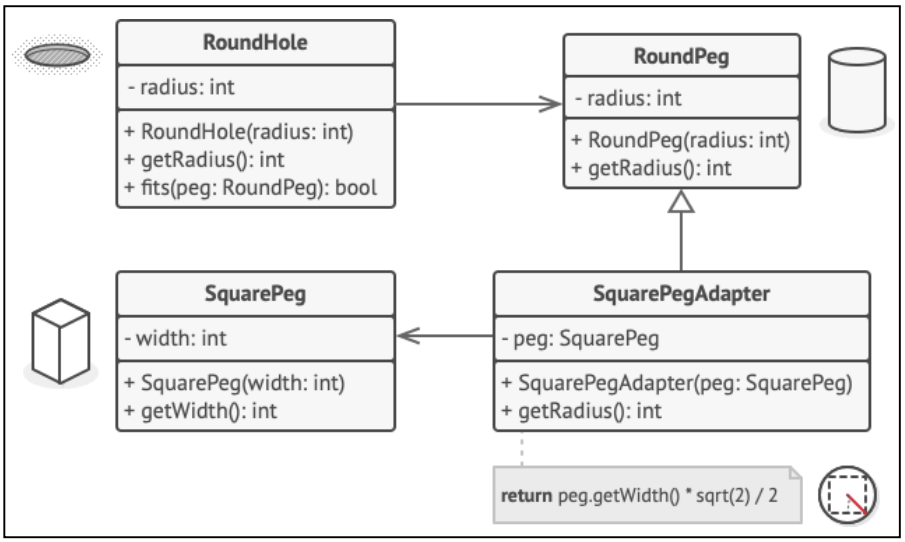

Figura 3: Adaptando pinos quadrados para buracos redondos.

Fonte: https://refactoring.guru/images/patterns/diagrams/adapter/example.png

A aplicabilidade se dá a reutilização de uma classe já existente, porém com incompatibilidade entre a classe que irá implementá-la.

As consequências são de a eliminação da limitação das interfaces, ocasionando no reaproveitamento de objetos.

2.3.3.2. Decorator

O Decorator é um padrão de projeto com propósito estrutural pertencente ao escopo de objeto. O objetivo é permitir que adicione novos comportamentos aos objetos existentes sem a necessidade de alterar o código.

Um exemplo apresentado por Shvets (2021) descreve o uso do padrão Decorator para comprimir e encriptar dados independentes do código que os usa. Antes de serem escritos em disco, o Decorator codifica e comprime os dados para que possam ser salvos e logo antes de serem lidos do disco o Decorator é novamente responsável por descomprimi-los e decodificá-los.

Na figura 4 a interface componente (*DataSource*) define operações que podem ser alteradas por Decorator (*DataSourceDecorator*). Os componentes concretos (*FileDataSource*) fornecem uma implementação padrão para as operações. O Decorator implementa a mesma interface que os outros componentes (*DataSource*). Por meio de um campo para armazenar os componentes envolvidos (*EncryptionDecorator* e *CompressionDecorator*) o Decorator delega todos os trabalhos para os componentes.

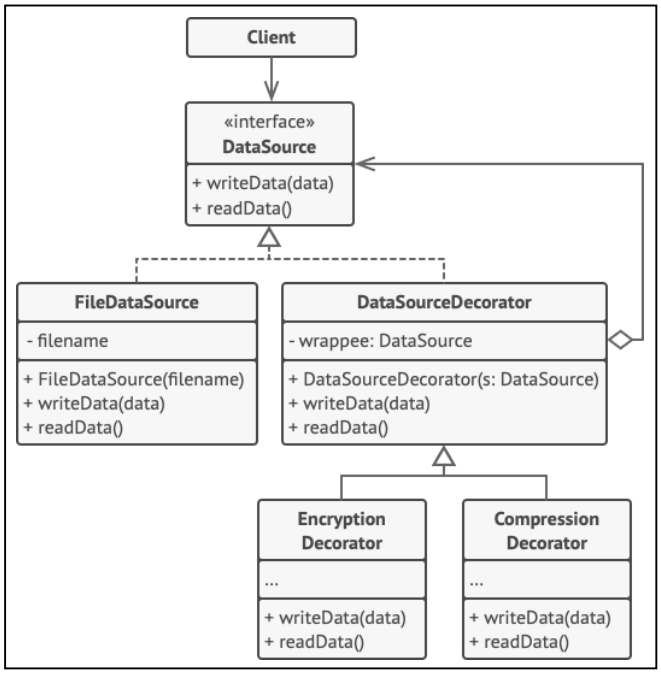

Figura 4: Exemplo da encriptação e compressão com decoradores.

Fonte: https://refactoring.guru/images/patterns/diagrams/decorator/ example.png

A aplicabilidade adiciona responsabilidade a objetos sem afetar outros objetos.

As consequências são estender o comportamento de um objeto sem fazer

uma nova subclasse e adicionar ou remover responsabilidades de um objeto.

O Adapter muda a interface de um objeto existente e o Decorator melhora um objeto sem mudar sua interface além de suportar composição recursiva.

2.3.4. Padrões de Comportamento

Os padrões de comportamento se preocupam com a atribuição de responsabilidade entre objetos e seus algoritmos.

2.3.4.1. Template Method

O Template Method é um padrão de projeto com propósito comportamental pertencente ao escopo de classe e o objetivo é definir um modelo padrão de algoritmo na superclasse, mas permite que as subclasses sobrescrevam etapas especificas do algoritmo permanecendo com a estrutura padrão.

Um exemplo apresentado por Shvets (2021) descreve o uso do padrão Template Method como um modelo padrão de um jogo de estratégia simples que usa várias ramificações de inteligência artificial (IA), em sua aplicação todas as raças do jogo possuem o mesmo tipo de unidades e construções, podendo reutilizar a mesma estrutura de inteligência artificial para várias raças do jogo.

Na figura 5 a classe abstrata (*GameAI*) define um método padrão (*turn*) contendo o modelo de algoritmo composto por chamadas. Subclasses concretas (*OrcsAI* e *MonstersAI*) que implementam estas operações, mas deixam o método padrão em si intacto. As classes concretas devem implementar todos os métodos da classe abstrata, mas não podem sobrescrever o método padrão (*turn*).

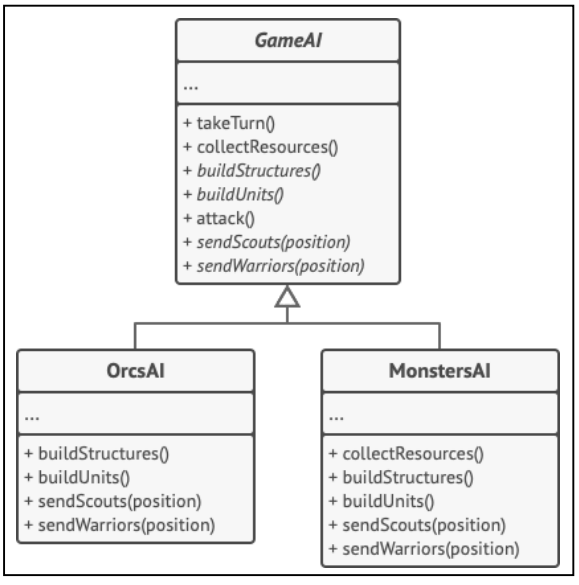

Figura 5: Classes IA de um jogo simples.

Fonte: https://refactoring.guru/images/patterns/diagrams/template-method/example.png

A aplicabilidade adiciona autonomia aos clientes, permitindo-os estender apenas pequenas particularidades de um algoritmo, mas não toda a estrutura.

As consequências evitam duplicação de código e permite fácil alteração de algoritmo.

O Factory Method é uma especialização do Template Method, podendo servir como uma etapa inicial para a aplicação de um Template Method maior.

2.3.4.2. Command

O Command é um padrão de projeto com propósito comportamental pertencente ao escopo de objeto. O objetivo é encapsular um comando em um objeto, permitindo a parametrização em diferentes solicitações, filas ou registros de clientes.

Um exemplo apresentado por Shvets (2021) descreve o uso do padrão Command como um registro do histórico de operações executadas e torna possível sua reversão se caso necessário.

Na figura 6 a classe abstrata (*Command*) definirá a interface comum para todos os comandos concretos (*CopyCommand, CutCommand, PasteCommand e UndoCommand*). O método de execução abstrato (*execute*), força todos os comandos concretos a fornecerem suas próprias implementações, devendo retornar verdadeiro ou falso dependendo do estado do editor.

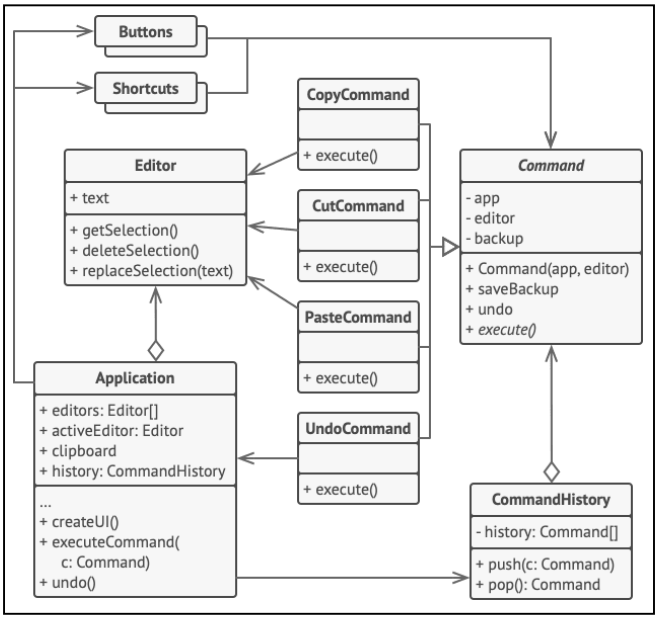

Figura 6: Operações não executáveis em um editor de texto.

Fonte: https://refactoring.guru/images/patterns/diagrams/command/ example.png

A aplicabilidade é desacoplar o objeto que envia a solicitação do objeto que recebe, fazendo com que as solicitações possam ser feitas e desfeitas.

As consequências são a criação de comandos simples e complexos, podendo conter outros comandos, implementar a execução adiada e implementar fazer, desfazer e refazer.

3. Materiais e métodos

Para o levantamento dos dados que foram utilizados para a realização do trabalho e possível obtenção de resultados, a pesquisa bibliográfica foi embasada em fundamentos teóricos, utilizando artigos científicos e publicações literárias. Os textos contidos foram consultados de forma física e virtual.

Além da pesquisa bibliográfica, foi utilizado também a pesquisa quantitativa, que houve aplicação aos docentes e discentes do curso de Ciências da Computação ministrado pelo instituto de educação Faculdade de Tecnologia, Ciência e Educação (FATECE).

Foi aplicado um questionário composto de 8 perguntas, e suas respostas apresentadas de acordo com seus resultados. O questionário aplicado obteve uma amostra de 10 voluntários. Aos participantes foi apresentado um termo de responsabilidade que garante que todas as informações obtidas serão aplicadas somente a esta pesquisa.

A perguntas da pesquisa são as seguintes:

Período da graduação:
Trabalha/trabalhou como desenvolvedor de software? - Sim - Não
Conhece quais paradigmas de programação mencionado? - Programação estruturada - Paradigma Orientado a Objetos - Paradigma Funcional
Conhece o termo padrões de projeto ou Design Patterns?

- Sim - Não
Conhece algum tipo de padrão de projeto abaixo? - Padrões criacionais - Padrões estruturais - Padrões comportamentais
Dos padrões criacionais, conhece algum tipo abaixo? - *Abstract Factory* - *Builder* - *Factory Method* - *Prototype* - *Singleton*
Dos padrões estruturais, conhece algum tipo abaixo? - *Adapter* - *Bridge* - *Composite* - *Decorator* - *Facade* - *Flyweight* - *Proxy*
Dos padrões comportamentais, conhece algum tipo abaixo? - *Chain of Responsibility* - *Command* - *Iterator* - *Mediator* - *Memento* - *Observer* - *State* - *Strategy* - *Template Method* - *Visitor*

Por fim, foram analisados os resultados das pesquisas com o intuito de entender se realmente a comunidade acadêmica, docentes e discentes, desconhecem os padrões de projeto.

4. Resultados e Discussão

A pesquisa quantitativa aplicada teve os seguintes resultados:

Foi observado que os entrevistados já formados, cerca de 20%, possuem o conhecimento da maioria dos padrões de projeto, porém não foi possível saber se este conhecimento se dá a alguma especialização acadêmica ou experiencia profissional.

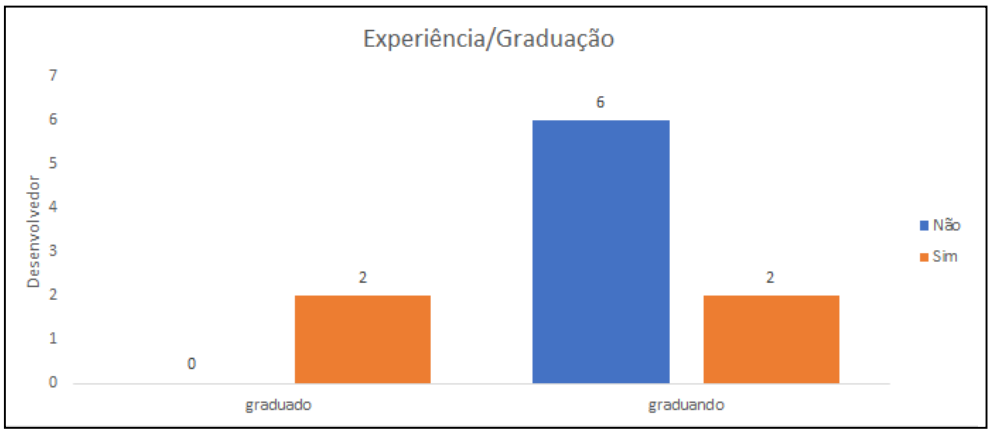

Gráfico 1: Experiência/Graduação

Fonte: Elaborado pelo autor

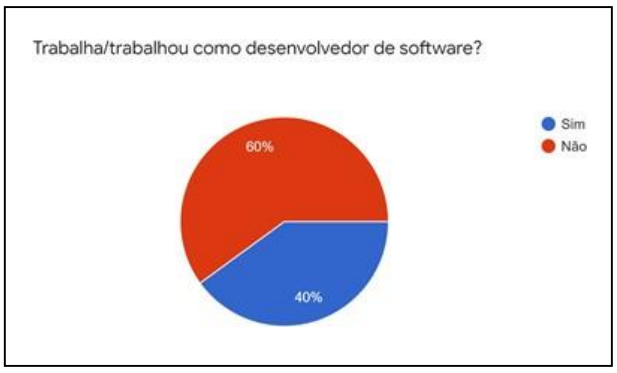

Gráfico 2: Trabalha/trabalhou como desenvolvedor de software?

Fonte: Elaborado pelo autor

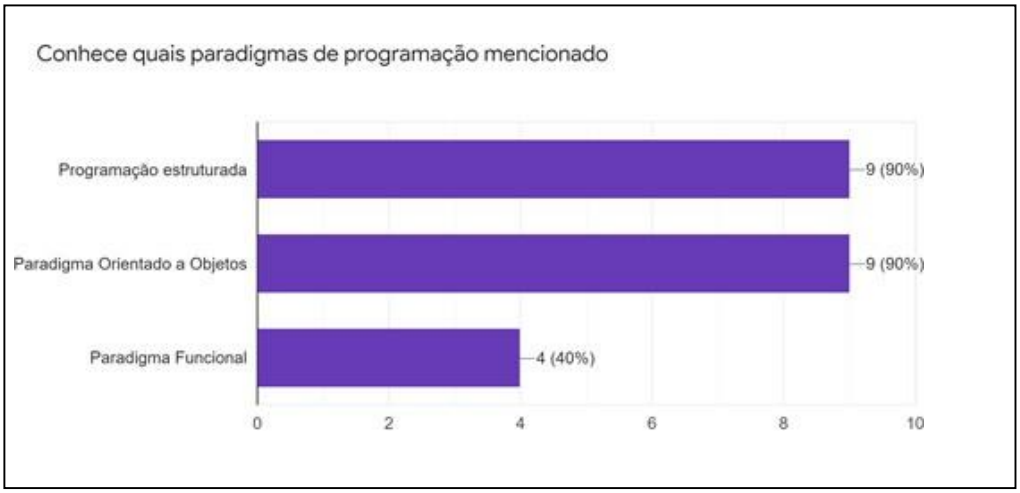

Gráfico 3: Conhece quais paradigmas de programação mencionado?

Fonte: Elaborado pelo autor

A pesquisa apontou que 40% dos entrevistados trabalham como desenvolvedores de software e conhecem os 3 paradigmas existentes.

Gráfico 4: Conhece o termo padrões de projeto ou Design Patterns?

Fonte: Elaborado pelo autor

A questão sobre o conhecimento do termo padrões de projeto apontou que 80% dos entrevistados dizem conhecer, porém não é possível saber se somente o conhecimento do termo implica na sua aplicação, uma vez que a prática da aplicação pode levar um tempo para ocorrer devido à complexidade da maioria dos padrões.

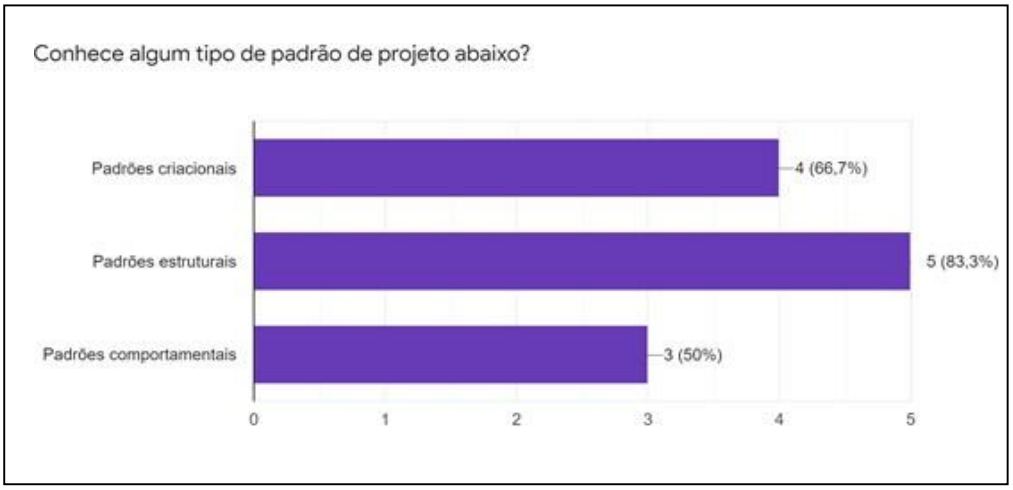

Gráfico 5: Conhece algum tipo de padrão de projeto abaixo?

Fonte: Elaborado pelo autor

Ao serem questionados sobre conhecimento do tipo de padrões deprojeto, 60% dos participantes informaram conhecer pelo menos 1 tipo e a partirdeste ponto começam a se destacar a senioridade dos entrevistados, pois, são os que informam conhecer todos os padrões e a partir deste ponto o índice do conhecimento sobre o assunto começa a reduzir.

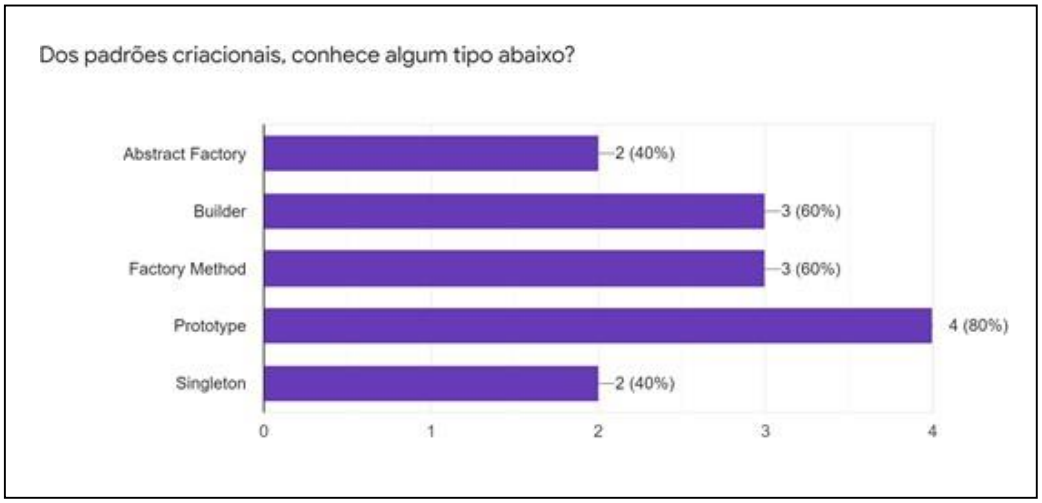

Gráfico 6: Dos padrões criacionais, conhece algum tipo abaixo?

Fonte: Elaborado pelo autor

Em relação ao tipo de padrões de projeto criacionais, cerca da metade dos participantes dizem não conhecer nenhum padrão e 20% informam conhecer todos os padrões, sendo estes, os apontados como sênior.

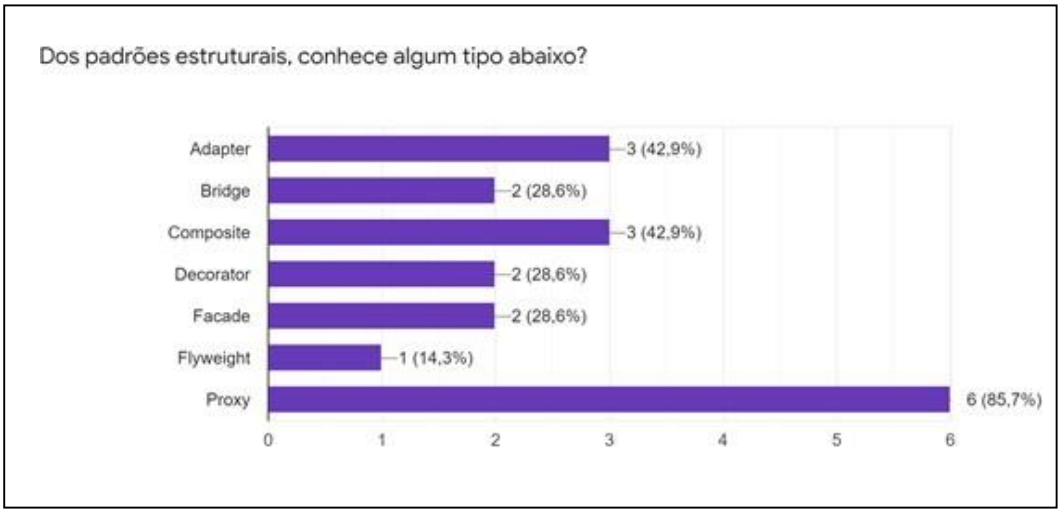

Gráfico 7: Dos padrões estruturais, conhece algum tipo abaixo?

Fonte: Elaborado pelo autor

Em relação ao tipo de padrões de projeto estruturais, cerca de 30% dos participantes não conhecem nenhum padrão e 10% informa conhecer todos os padrões, novamente, os apontados como sênior.

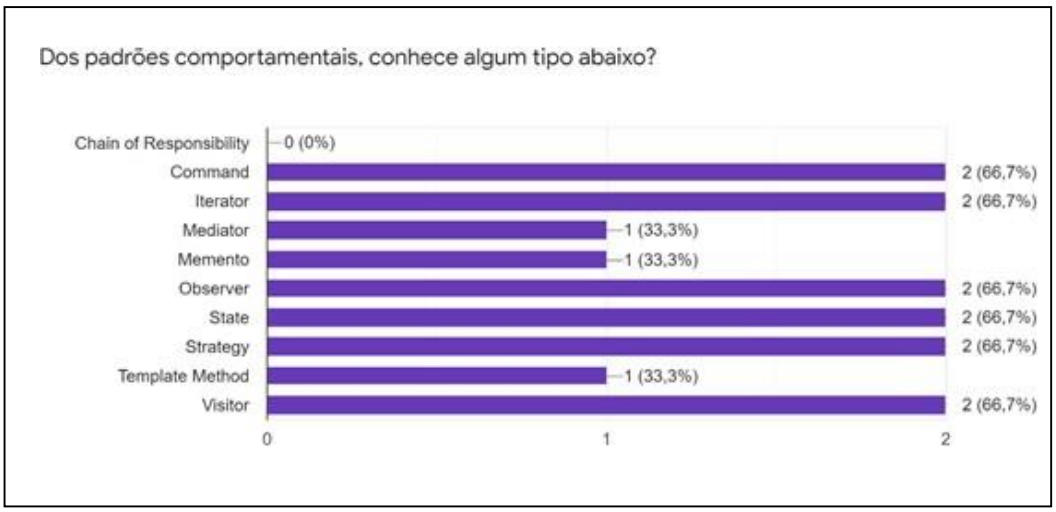

Gráfico 7: Dos padrões comportamentais, conhece algum tipo abaixo?

Fonte: Elaborado pelo autor

Em relação ao tipo de padrões de projeto comportamentais, cerca de 70% dos participantes não conhecem nenhum padrão e 10% informa conhecer todos os padrões.

Verificando os resultados da pesquisa, foi possível observar que a aplicação deste questionário não foi tão abrangente quanto a necessidade real de verificar o conhecimento dos entrevistados. Para isso seria necessário aplicar

uma avaliação do conhecimento dos padrões, relacionando o escopo, diagrama ou algoritmo, problemas, soluções e consequências ao padrão específico.

Considerações Finais

O desenvolvimento do presente trabalho possibilitou uma análise do conhecimento sobre os padrões de projeto, uma reflexão acerca dos benefícios dos recursos dos padrões mais usados pela indústria de software, além disso, também permitiu descrevê-los e apresentar de forma didática seus conceitos e aplicações.

De um modo geral, a pesquisa mostrou que o assunto abordado é de conhecimento de alguns entrevistados que buscaram meios para estarem atualizados, mas nem todos possuem um conhecimento de todos os padrões.

Os entrevistados demonstraram de acordo com a sua experiência conhecer mais sobre o assunto. Diante, das respostas ficou evidente isso, fazendo parte do amadurecimento dos desenvolvedores.

Na pesquisa, foi possível identificar o perfil dos entrevistados que apresentaram maior conhecimento e a visão deles em relação aos padrões apresentados.

Dada à importância do tema, torna se necessário o desenvolvimento de uma pesquisa com uma população de entrevistados maior, aplicando testes de conhecimento com uma estrutura mais complexa para que realmente possa identificar um conhecimento mais específico do assunto.

Neste sentido, os padrões de projeto possuem uma grande importância para os profissionais de desenvolvimento e para a indústria de software e de certa forma ajuda o profissional iniciante a aumentar suas chances no mercado de trabalho, motivando-o com a aplicação de melhores práticas de desenvolvimento de software.

Referências

DEITEL, P **Java**: Como Programar. 10. ed. São Paulo: Pearson Education do Brasil, 2017.

COUTO, F. (Brasil). **Qual o perfil dos desenvolvedores?** 2017. Disponívelem: https://blog.vulpi.com.br/perfil-dos-desenvolvedores/. Acesso em: 25 ago. 2021.

GAMMA, E; HELM, R; JOHNSON, R; VLISSIDES, J. **Padrões de projeto**: soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2000.

JUNGTHON, G; GOULART, C. M. **Paradigmas de Programação**. 2016. 8 f. TCC (Doutorado) - Curso de Sistemas de Informação, Faculdade de Informática de Taquara, Taquara, 2016.

GROSSMANN, L. O. (Brasil). **TI precisa de 420 mil novosprofissionais até 2024.** 2019. Disponível em: https://brasscom.org.br/ti-precisa- de-420-mil-novos-profissionais-ate-2024/. Acesso em: 25 ago. 2021.

MARTIN, R. C. **Arquitetura Limpa**: o guia do artesão para estrutura e design de software. Rio de Janeiro: Alta Books, 2020.

POPPER, K. **A lógica da pesquisa científica**. 2. ed. São Paulo: Cultrix, 2013.

QUINCY LARSON (Estados Unidos). **Freecodecamp**. The 2018 New Coder Survey: 31,000 people told us how they′re learning to code and getting jobs as⁚. 31,000 people told us how they're learning to code and getting jobs as…. 2017. Disponível em: https://www.freecodecamp.org/news/we-asked-20-000-people-who-they-are-and-how-theyre-learning-to-code-fff5d668969/. Acesso em: 25 ago. 2021.

SHVETS, A. **Mergulho nos Padrões de Projeto**. 1.17. ed. Ucrânia: Refactoring.Guru, 2021.

TUCKER, A. B.; NOONAN, R. E. **Linguagens de programação**: Princípios e paradigmas. 2. ed. São Paulo: AMGH, 2008.