Padrão *Strategy* aplicado à geração de múltiplas visões de dados na biblioteca ColMinerRT

Santiago Souza
Federal University of Viçosa
Florestal, Minas Gerais, Brasil
santiago.souza@ufv.br

Gláucia Braga e Silva
Federal University of Viçosa
Florestal, Minas Gerais, Brasil
glaucia@ufv.br

RESUMO

A utilização de padrões de projeto permite ao desenvolvedor aplicar estratégias vastamente testadas em um domínio específico. Este trabalho faz uso do padrão *Strategy* para melhorar a manutenibilidade e extensibilidade da arquitetura da biblioteca ColMinerRT, que tem como finalidade importar dados da ferramenta GitHub, realizar o cálculo da relevância temática e persisti-lo em um arquivo formato CSV. O padrão foi aplicado para se criar um novo mecanismo de geração dos dados a serem exportados, incluindo uma nova camada de apresentação que disponibilize múltiplas visões estruturadas dos dados. Após a aplicação do padrão, foi possível observar ganhos em termos de manutenibilidade e extensibilidade, uma vez que o código ficou mais organizado e mais desacoplado, permitindo que, no futuro, novas visões sejam adicionadas.

CCS CONCEPTS

• **Software and its engineering** → Designing software; Software design engineering.

KEYWORDS

Padrão de projeto, manutenibilidade, extensibilidade

ACM Reference Format:
Santiago Souza and Gláucia Braga e Silva. 2022. Padrão *Strategy* aplicado à geração de múltiplas visões de dados na biblioteca ColMinerRT. In *Proceedings of UNIVERSIDADE FEDERAL DE VIÇOSA campus FLORESTAL, Trabalho de conclusão de Curso (UFV 2022).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUÇÃO

A utilização dos padrões de projeto permite ao desenvolvedor aplicar estratégias vastamente testadas em um domínio específico [3]. Eles podem ser utilizados para atender às especificações de normas como a ISO/IEC 25010, que priorizam atributos de qualidade como eficiência, manutenibilidade, extensibilidade, desacoplamento e portabilidade.

Este trabalho propõe o uso do padrão de projeto *Strategy*, no contexto da biblioteca ColMinerRT, voltada ao cálculo da relevância temática de comentários postados no ambiente de *issue tracking* do GitHub e nos tópicos de discussão do StackOverflow, com o intuito de avaliar o quão relevantes os comentários são para uma determinada discussão. Com esse padrão, espera-se melhorar a arquitetura da ColMinerRT, incluindo uma nova camada de apresentação que disponibilize múltiplas visões estruturadas dos dados de relevância temática calculados e outras informações associadas que também foram coletadas das bases de dados, que possam ser exportadas em algum formato intercambiável para fins de análises futuras mais detalhadas em outras ferramentas de apoio.

Espera-se que esse refinamento do projeto arquitetural da biblioteca ColMinerRT possa trazer melhorias em termos de extensibilidade e manutenibilidade do código, em especial no que compete à funcionalidade de geração/exportação dos dados resultantes do cálculo da métrica de relevância temática. Essa funcionalidade está implementada na versão original de forma muito acoplada e de difícil manutenção. Para gerar dados mais simplificados ou mudar o formato de exportação dos dados, o usuário da biblioteca teria que alterar o código internamente, o que fere o encapsulamento e pode causar problemas de funcionamento.

O restante deste artigo está organizado da seguinte forma: na Seção 2, apresenta-se uma fundamentação acerca do padrão *Strategy* e sua influência em termos dos atributos de qualidade abordados; a Seção 3 traz os trabalhos relacionados; A aplicação do *Strategy* no contexto da biblioteca ColMinerRT é apresentada na Seção 4; a prova de conceito da arquitetura na Seção 5; e as considerações finais, na Seção 6.

2 PADRÃO DE PROJETO *STRATEGY* E QUALIDADE DE SOFTWARE

Um padrão de projeto é uma descrição de uma solução genérica, resultante de uma vasta busca realizada por especialista em sistemas existes por soluções semelhantes aplicadas a domínios distintos [7]. Após a identificação desses designs brutos, realiza-se a remoção do domínio de aplicação, mantendo apenas sua estrutura básica.

O padrão de projeto *Strategy* "define uma família de algoritmos, encapsula cada um deles e os torna intercambiáveis" [3, p. 292]. O uso desse padrão permite que a aplicação Cliente defina qual algoritmo utilizar e quando executar, mas não tem acesso direto a sua execução. Ele é aplicado, quando há a necessidade de múltiplas variantes para o mesmo algoritmo, quando uma classe possui várias operações chamadas por vários condicionais, entre outras [3, p. 293].

A Figura 1 apresenta uma versão com uma pequena adaptação de Gamma et al. [3] para o diagrama de classes do padrão de projeto *Strategy*.

As classes que representam o *Strategy* possuem as seguintes responsabilidades:

UFV 2022, Abril 01–04, 2022, Florestal, MG

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

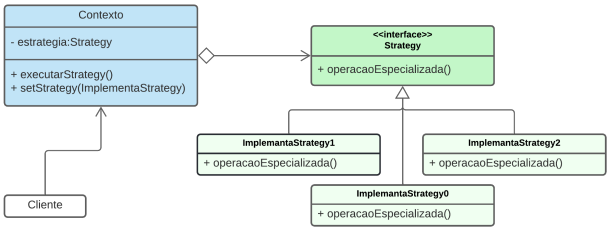

Figura 1: Diagrama geral do padrão de projeto *Strategy*

- **Cliente**: Define entre as estratégias disponíveis qual utilizar, mas não sabe como é usada.[1]
- **Contexto**: Possui a operação que executa o **ImplementaStrategy** que o **Cliente** definiu.
- **Strategy**: Interface ou classe abstrata que define uma política.
- **ImplementaStrategy**: Implementa a política definida pela *Strategy*, podendo, com isso, ser usado pelo contexto.

Dentre as vantagens de se usar o *Strategy*, destacam-se: a) permite a troca do algoritmo entre múltiplas estratégias; b) reduz o uso de condicionais; c) gera o desacoplamento das estratégias e das demais classes; e d) facilita a inclusão de novas estratégias.

Dentre as desvantagens de seu uso, pode ocorrer: a) o aumento do número de objetos em uso; b) a exigência de que o Cliente tenha ciência das estratégias disponíveis; c) Em implementações, pode acarretar desperdício de memória caso utilize uma politica complexa [3, p. 296].

2.1 Atributos de Qualidade de Software

Este trabalho aborda melhorias na arquitetura de software, no que diz respeito à manutenibilidade e à extensibilidade de código.

Quando se fala em manutenibilidade e extensibilidade no contexto da Orientação a Objetos (O.O), é importante considerar os cinco princípios *SOLID* [1] :

- S - *Single Responsibility Principle (SRP)* ou Princípio da Responsabilidade Única, que recomenda que uma classe deveria ter uma, e apenas uma, razão para mudar;
- O - *Open Closed Principle (OCP)* ou Princípio Aberto-Fechado, que diz que objetos devem estar abertos para extensão, mas fechados para modificação;
- L - *Liskov Substitution Principle (LSP)* ou Princípio da substituição de *Liskov*, que diz que uma classe derivada deve ser substituível por sua classe base;
- I - *Interface Segregation Principle (ISP)* ou Princípio da Segregação da Interface, que recomenda que uma classe não deve ser forçada a implementar interfaces e métodos que não irá utilizar; e
- D - *Dependency Inversion Principle (DIP)* ou Princípio da Inversão de Dependência, que recomenda a dependência de abstrações e não de classes concretas (implementações).

[1]Não representado no diagrama de referência.

O padrão *Strategy* é uma alternativa interessante para a herança, já que ele decompõe códigos voláteis, encapsulando-os como objetos e os usa quando necessário. Esse padrão respeita o princípio *(OCP)* [11], uma vez que se baseia na extensibilidade e reutilização para promover a flexibilidade do código. Dessa forma, nenhuma modificação em classes existentes é requerida, em mudanças e manutenções futuras.

A manutenibilidade trata de realizar o menor esforço para modificar um software ([8] citado por [5]). Ferreira et al.[5] também reforçam as afirmações de Pigoski[10] sobre o esforço para manutenção, sendo 25% em manutenção adaptativa (capacidade de ajustar o software as mudanças do ambiente), 20% em corretiva (reparação de defeitos) e 55% em evolutiva (adição de novas funcionalidades).

Extensibilidade é a qualidade de ser extensível. No contexto de software, corresponde a uma especialização da manutenibilidade, que trata da capacidade de um software ou sistema de aumentar suas funcionalidades[4].

3 TRABALHOS RELACIONADOS

Figielska [6] propõe o uso dos padrões *Strategy* e *Template Method* no contexto de um programa de agendamento usando o mesmo algoritmo meta-heurístico. Os padrões são usados na criação de duas soluções para remover a existência de múltiplos comandos condicionais na implementação da meta-heurística *Simulated annealing*, composta de duas execuções possíveis, uma sequencial e outra paralela. O uso dos padrões melhora o princípio *(OCP)* do *SOLID* [1], aumenta a extensibilidade por facilitar a implementação de novas meta-heurísticas e remove a necessidade de manutenções nas existentes. Rana et al. [11] aplicaram os padrões *Strategy* e *Decorator*, em conjunto com dois dos princípios do *SOLID (SRP)* e *(OCP)* em um sistema orientado a objeto, para medir a flexibilidade de software.

O uso do *Strategy* também é usado em ferramentas para refatoração automática de código com intuito de aumentar a extensibilidade [2]. O trabalho usa o padrão, por meio de polimorfismo para eliminar odores no código (*code smells*) que estão relacionados ao uso extensivo de declarações condicionais complexas.

4 APLICAÇÃO DO *STRATEGY* NA BIBLIOTECA *COLMINERRT*

Esta seção apresenta os resultados da aplicação do padrão *Strategy* no refinamento da arquitetura da biblioteca ColMinerRT, no que compete à geração/exportação dos dados com as relevâncias temáticas calculadas.

4.1 A biblioteca ColMinerRT

A biblioteca ColMinerRT faz parte do projeto de modularização da ferramenta ColMiner, proposta por Neto e Braga e Silva [9]. A biblioteca ColMinerRT se concentra na funcionalidade do cálculo da relevância temática sobre as mensagens das discussões em *issue tracking* ou em fóruns de discussão do StackOverflow. As principais funcionalidades da biblioteca abrangem:

- **Importar dados**: Coleta dados no ambiente de *issue tracking* do GitHub e nos tópicos de discussão do StackOverflow.
- **Calcular relevância temática**: Consiste em gerar um valor que representa a relevância de cada mensagem em relação

ao termo em discussão. Para isso, faz uso da métrica de relevância temática [9]
- **Exportar dados com relevâncias**: Armazena os dados com as relevâncias calculadas em um arquivo CSV, por ser um formato intercambiável compatível com várias ferramentas, onde se pode utilizar o arquivo para gerar novas análises sobre os dados.

Na Figura 2, pode-se observar o diagrama de classes da biblioteca ColMinerRT. O modelo é composto pelas classes: **Cliente**, que representa a aplicação Cliente da biblioteca; **CalculoRelevanciaFachada**, uma classe de fronteira destinada a interagir com a classe **Cliente** e possibilitar a execução das operações existentes na biblioteca; classes de controle representadas em vermelho, como a **CalculoRelevanca**; e classes de modelo, representadas em azul.

Do ponto de vista do projeto arquitetural e das responsabilidades de suas classes, algumas particularidades merecem destaque:

- **CalculoRelevanciaFachada**: Responsável por simplificar e centralizar os acessos às funcionalidades da biblioteca, permitindo realizar seis operações distintas (três para cada fonte de dados): duas versões **processarDados()**, que comunica com as APIs para importar os dados de cada repositório; duas de **CalcularRelevânciaTemática**, que junto da classe especializadas realiza a medição que permite criar um valor que representa a relevância temática para cada comentário; e duas **gerarCSV()** responsável por criar o arquivo CSV seguindo uma regra para escolha dos dados exportados.
- **ControladoraAPIGitHub**: Por meio de suas duas operações acessa a APIGitHub para importar os dados deste repositório.
- **ControladoraAPI**: Por meio de suas quatro operações acessa a APIStackOverFlow, importa os dados e constrói um objeto "Topico".
- **CalculoRelevancia**: Realiza operações que resultam no valor da Relevância Temática.
- **Projeto**: Armazena os dados de um repositório do GitHub.
- **Topico**: Armazena os dados das *Issues* do GitHub ou de Tópicos do Stack Overflow.
- **Comentario**: Armazena os dados dos comentários e armazena o resultado do cálculo da Relevância Temática.

A funcionalidade alvo deste trabalho é a "Exportar dados com as relevâncias"e só considera a aplicação da biblioteca para dados do GitHub. Na versão corrente, o principal resultado é a geração de um arquivo CSV com os dados das relevâncias calculadas. Este arquivo é criado na classe **CalculoRelevanciaFachada**, por meio da operação **gerarCSVGitHub()**. O arquivo gerado apresenta uma estrutura de tabela, composta por linhas e colunas, onde na primeira linha expõe uma lista de nomes para as colunas. Estes nomes são definidos dentro da operação junto da escolha de quais atributos serão persistidos e se transformam em colunas no arquivo. A Figura 3 apresenta um exemplo de um arquivo CSV gerado pela biblioteca e aberto em um editor de planilhas para visualização.

4.2 Arquitetura com o *Strategy*

Com o intuito de prover uma arquitetura mais extensível e de fácil manutenção no que compete à geração das múltiplas visões dos dados de relevâncias temáticas calculadas, a arquitetura foi refinada. O refinamento restringe-se ao escopo da operação **gerarCSVGitHub()** da classe *CalculoRelevanciaFachada*, responsável por gerar os dados com as relevâncias temáticas. Nesse contexto, foi aplicado o padrão *Strategy*, conforme mostra a Figura 4.

Para facilitar a interpretação do diagrama, foi ocultado o conjunto de classes internas à biblioteca ColMinerRT por não terem sido modificadas. Essas classes estão representadas na cor cinza no diagrama.

Foi necessário fazer uma refatoração na classe **CalculoRelevanciaFachada**. A operação **gerarCSVGitHub()** foi substituída por **criarVisualizador()** e as demais operações mantidas inalteradas. A nova operação permite ao Cliente acessar as classes que implementam o padrão *Strategy* e por meio destas gerar o arquivo CSV, utilizando uma das visões.

As demais classes referem-se à aplicação do *Strategy* e estão descritas abaixo:

- **VisualizadorGitHub**: Ocupa a função de Contexto, armazenando a *visaoStrategy* e possui quatro operações: **gerarCSV()**, responsável por pegar o resultado da visão e armazenar no arquivo CSV. Essa operação permite escolher quais colunas e em qual ordem serão persistidas, sendo necessário informar uma lista de chaves; para saber quais chaves podem ser informadas utiliza-se a operação **colunasDisponiveis()**, que retorna uma lista de chaves, que varia a depender do comportamento da visão; **setVisao()** permite alterar a interface *VisaoStrategy* e, por consequência, alterar a visão sobre os dados persistidos; e a operação **colunasApresentar()**, limitada ao escopo da classe, remove as colunas que não estão na lista de chaves.
- **VisaoStrategy**: É uma Interface que define uma regra comum as classes, que a implementam, permitindo que a classe *VisualizadorGitHub* possa utilizar todas as classes derivadas.
- **FerramentasVisao**: Classe composta por três operações destinadas a auxiliar as implementações da *VisaoStrategy*.

Como exemplo de uso foram criadas cinco implementações da *VisaoStrategy*, denominadas visões, sendo uma com comportamento simples e quatro mais complexas. A **VisaoPadrao** converte o objeto Projeto para o formato usado por **gerarCSV()**, permitindo gerar o mesmo arquivo que a biblioteca ColminerRT, sendo esta visão definida como a padrão, caso não seja informada outra visão ao criar o *VisualizadorGitHub*. As demais visões compartilham de uma mesma estrutura: a) Ordenar os dados com base em um atributo pertencente a uma das classes de modelo, o que tende a criar grupos, devido a equivalências de dados; e b)Ordenar estes grupos seguindo outro atributo que nestes exemplos foi a *relevanciaTematica*, da maior relevância para a menor. As outras quatro visões são: 1) **VisaoRelevanciaTematicaPorData**, que utiliza o atributo *data*[2] pertencente à classe **Comentário**. 2) **VisaoRelevanciaTematicaPorStatus**, que utiliza o atributo *status* encontrado na classe **Topico**, que também realiza um filtro para apresentar apenas comentários criadas no ano de 2022; 3) **VisaoRelevanciaTematicaPorAutor**, que utiliza o atributo Autor pertencente à

[2]Esse atributo na verdade possui data e hora, por isso foi utilizada apenas a informação de data para criar grupos por dia.

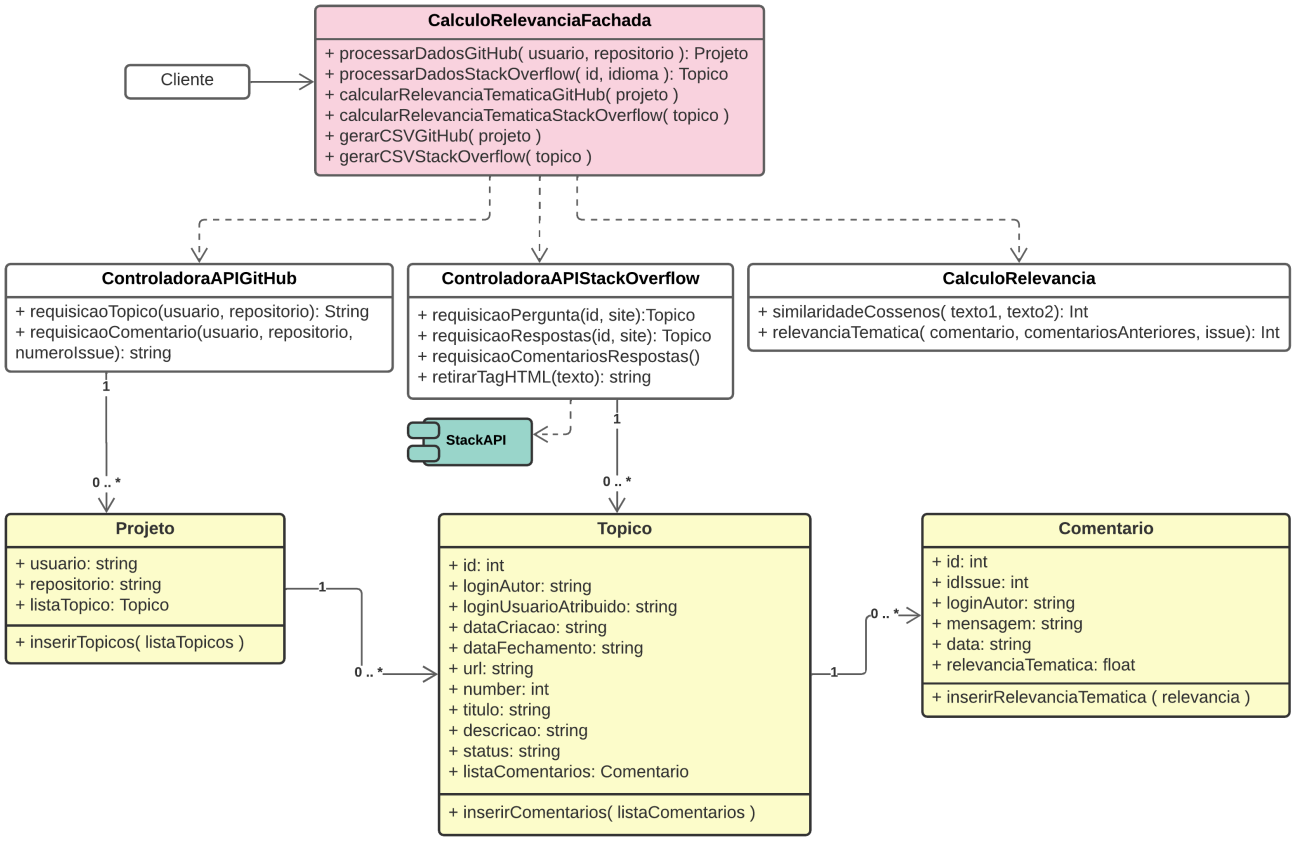

Figura 2: Diagrama de Classes da biblioteca ColMinerRT

usuarioProjeto	repositorioProj	idIssue	loginAutorIssue	loginUsuarioAt	NumberIssue	TituloIssue	dataFechament	urlIssue	descricaoIssue	CriacaoIssue	statusIssue	NumeroComentario	idIssueComentario	Comentario	DataComentario	RelevanciaTematica	AutorComentario
xmonad	xmonad	10913767(	geekosaur	CONTRIBUTOR	358	Are we running X.O.windows too often during moves/resizes?	2022-02-21T08:11:10Z	https://api.github.com/repos/xmonad/xmonad/issues/358	### Problem De A user reporting If I resize a wi This suggests to	2021-12-31T01:57:18Z	closed	1003311772	1091376760	> Should this be I think so. The p EDIT: Oh, appar	2021-12-31T08:48:28Z	15.408.134.911.929.900	slotThe
xmonad	xmonad	10913767(	geekosaur	CONTRIBUTOR	358	Are we running X.O.windows too often during moves/resizes?	2022-02-21T08:11:10Z	https://api.github.com/repos/xmonad/xmonad/issues/358	### Problem De A user reporting If I resize a wi This suggests to	2021-12-31T01:57:18Z	closed	1003337979	1091376760	@liskin pointed (	2021-12-31T10:28:03Z	3.475.714.014.525.850	geekosaur
xmonad	xmonad	10913767(	geekosaur	CONTRIBUTOR	358	Are we running X.O.windows too often during moves/resizes?	2022-02-21T08:11:10Z	https://api.github.com/repos/xmonad/xmonad/issues/358	### Problem De A user reporting If I resize a wi This suggests to	2021-12-31T01:57:18Z	closed	1019330534	1091376760	I'd almost argue There may be ot But I wouldn't go	2022-01-22T18:35:09Z	13.584.575.812.799.000	liskin
xmonad	xmonad	10913767(	geekosaur	CONTRIBUTOR	358	Are we running X.O.windows too often during moves/resizes?	2022-02-21T08:11:10Z	https://api.github.com/repos/xmonad/xmonad/issues/358	### Problem De A user reporting If I resize a wi This suggests to	2021-12-31T01:57:18Z	closed	1046580427	1091376760	I will close this fc	2022-02-21T08:11:10Z	18.686.634.375.694.000	slotThe
xmonad	xmonad	10778963(	Vermoot	NONE	355	Window decorations on floating windows		https://api.github.com/repos/xmonad/xmonad/issues/355	I realise window (As an aside, as There's an old P Is that something Thanks!	2021-12-12T20:08:34Z	open	992639122	1077896353	I suppose the ea So... help wante	2021-12-13T16:17:48Z	4.688.699.735.610.900	slotThe

Figura 3: Exemplo de arquivo CSV exportado pela ColMinerRT e aberto na ferramenta *Google Sheets*

classe **Comentário**; e 4) **VisaoRelevanciaTematicaPorComentario**, que utiliza o atributo mensagem também pertencente à classe **Comentário**.

5 PROVA DE CONCEITO DA ARQUITETURA

Esta seção apresenta as refatorações em código decorrentes da aplicação do *Strategy* na arquitetura da biblioteca ColminerRT e

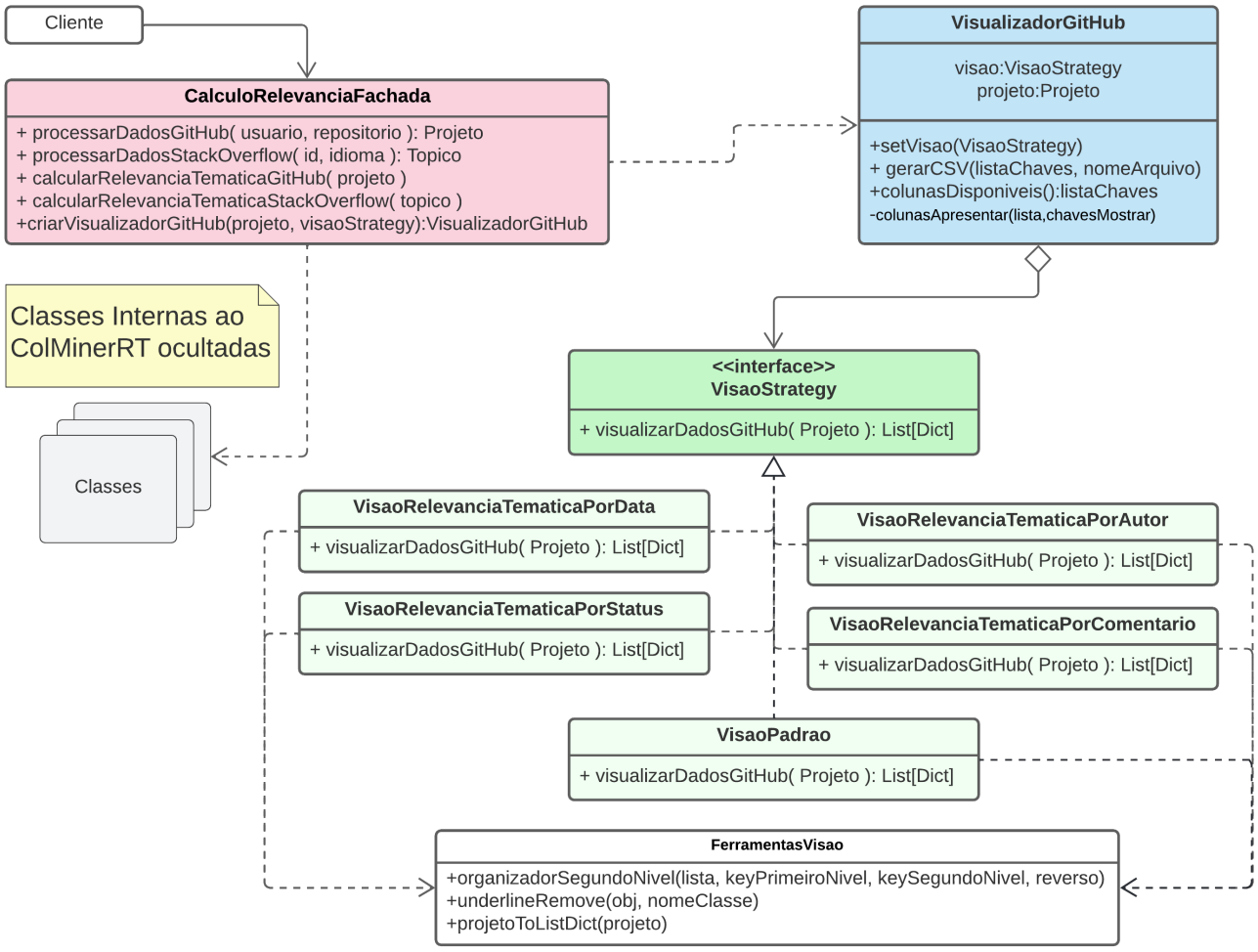

Figura 4: Diagrama de Classes da biblioteca ColMinerRT com a adição do padrão de projeto *Strategy*

também alguns exemplos de uso dos dados, a partir das visões geradas.

5.1 Refatorações em código

Aqui serão apresentadas as mudanças na implementação do código para uso da biblioteca, focando nas mudanças realizadas na aplicação **Cliente**, discutindo ganhos e perdas. Os códigos foram produzidos em Python e são apresentados aqui como recortes para os trechos modificados e ou inseridos.

A Figura 5 mostra como era a chamada do método **gerarCSVGitHub()**, responsável por gerar o arquivo CSV, na versão corrente sem o padrão. Após a aplicação do Strategy, a nova forma de criar um arquivo semelhante pode ser vista na Figura 6.

```
1    BibliotecaColMinerRTFachada.gerarCSVGitHub(projeto)
```

Figura 5: Trecho de código com a forma original de criar o arquivo CSV, na biblioteca ColminerRT

```
1    contexto = BibliotecaColMinerRTFachada.
      criarVisualizador(projeto)
2    contexto.gerarCSV()
```

Figura 6: Trecho de código com a nova forma de criar o arquivo CSV, na biblioteca ColminerRT

Pode-se observar que a nova versão é mais complexa e menos direta, mas isso acontece apenas nesta chamada. Contudo, a nova forma de criar o arquivo permite escolher quais colunas apresentar. É como se repetisse a linha dois da Figura 6, passando como parâmetro uma lista com as colunas de interesse. Na versão original da biblioteca ColminerRT, não seria possível para o Cliente fazer as mesmas operações, cabendo ao usuário a responsabilidade de remover as colunas irrelevantes, após a geração do arquivo.

Para a utilizar as visões na versão da biblioteca ColminerRT com o *Strategy*, são necessárias poucas linhas de código, como pode

ser visto na Figura 7. Neste trecho de código, a primeira linha cria o contexto usando a visão *VisaoRelevanciaTematicaPorAutor*; e na segunda, gera-se um arquivo com apenas duas colunas, que será persistido seguindo a ordem na lista passada como parâmetro. Na terceira linha, altera-se a visão em uso e, na quarta, gera-se outro arquivo, agora usando a nova visão, além de ter sido escolhidas outras colunas para persistir. Essa mesma execução não seria possível na versão original da biblioteca ColminerRT, sem haver modificação do código interno da biblioteca por parte do Cliente, que precisaria editar o método **gerarCSVGtiHub()**, e esta edição teria validade curta, pois quando necessário exportar um arquivo diferente teria que ser modificado novamente.

```
1    contexto = BibliotecaColMinerRTFachada.
      criaVisualizador(projeto,
      VisaoRelevanciaTematicaPorAutor())
2    contexto.gerarCSV(['loginAutorComentario', '
      relevanciaComentario'])
3    contexto.setVisualizador(
      VisaoRelevanciaTematicaPorData())
4    contexto.gerarCSV(['dataComentario', '
      relevanciaComentario','loginAutorTopico'])
```

Figura 7: Criando dois arquivos cada um com sua visão

Na Figura 8, é apresentado um exemplo genérico de uma classe visão, que poderia ser usada pelos usuários da biblioteca para criar, de forma personalizada, visões que atendam diferentes necessidades de visualização dos dados e que facilitem análises futuras em outras ferramentas.

```
1  class Visao(VisaoStrategy):
2
3      def visualizarDadosGitHub(self, projeto):
4          """
5          ----------
6          projeto: Projeto
7
8          Return
9          -------
10         list[Dict]
11         """
12         lista = projetoToListDict(projeto)
13
14         # Algoritmo que cria uma visao sobre os dados em
       lista
15
16         return lista
```

Figura 8: Estrutura básica de uma visão

Essa nova possibilidade de utilizar múltiplas visões simplifica significativamente a manutenibilidade. Novas visões sobre os dados não resultam em modificações em classes existentes da biblioteca, mas sim na adição de uma nova classe, que passa a fazer parte do conjunto de visões. Com tudo, a aplicação Cliente precisa ser modificada caso pretenda fazer uso das novas visões.

5.2 Uso da biblioteca ColMinerRT

Na versão original, a biblioteca ColminerRT exporta os dados em um arquivo com 18 colunas (Figura 3), deixando a cargo do usuário duplicar o arquivo e remover as colunas não desejadas. Isso resulta em desperdício de tempo por parte do usuário, principalmente quando se pretende utilizar a mesma estrutura interma do arquivo em projetos distintos ou refazer a mesma análise de forma recorrente.

Como forma de mostrar as vantagens da utilização da nova versão da biblioteca ColminerRT com uso do *Strategy*, serão apresentados exemplos de uso dos dados das visões, sempre discutindo os ganhos obtidos em relação à versão anterior. Para isso, serão utilizados dados exportados em duas das visões propostas na seção 4.2, com geração de gráficos na ferramenta Google Sheets[3]. Para este fim, foram utilizados dados públicos do repositório Xmonad[4], disponível no GitHub. Os exemplos apresentados possuem a finalidade exclusiva de mostrar a utilização da biblioteca e não a de analisar os dados do repositório em si.

Um exemplo simples dos ganhos na utilização das visões pode ser visto na Tabela 1, que apresenta a visão "Relevância Temática por Autor". Nessa visão, são apresentadas apenas duas informações: o autor e as relevâncias de todos os comentários postados, ordenadas de forma decrescente. Os valores do campo *loginAutorComentario* são agrupados por autor e a primeira linha de cada grupo traz a maior relevância temática para os comentários daquele autor. As demais linhas seguem em ordem decrescente. Para ter acesso aos mesmos dados na versão original, seria necessário fazer uma seleção/recorte, a partir das 18 colunas geradas no CSV padrão (Figura 3), tarefa muito mais trabalhosa e humano-dependente.

loginAutorComentario	relevanciaComentario
2shrestha22	0,03693260312
2shrestha22	0,01150358369
2shrestha22	0,0001022704029
ashincoder	0,008293026522
ashincoder	0,006687178855
AusCyberman	0,1548215249
AusCyberman	0,0436499175
AusCyberman	0,008763645465
burunduk3	0,06015617353
burunduk3	0,02458671272
burunduk3	0,01145639106
dmwit	2,95E-05
doums	0,07944822765
doums	0,01073306427
doums	0,002770651712
drewbarbs	0,06769878044
duplode	0,1262261712
ekmett	0,02227402122
elkowar	0,02227402122
exorcist365	0,01746620751
fabiodl	0,07697864798
fabiodl	0,07697864798
geekosaur	0,4992112488
geekosaur	0,4982220862

Tabela 1: Exemplo que arquivo .csv transformado em tabela, utilizou a visão Relevância por Autor

[3]https://www.google.com/sheets/about/
[4]https://github.com/xmonad/xmonad/

Os dados gerados na visão poderiam ser facilmente visualizados em algum ferramenta de geração de gráficos ou até mesmo utilizados no cálculo de alguma métrica ou análise estatística, a critério do usuário. A Figura 9 apresenta um gráfico gerado com os dados dessa visão. Nele, pode-se visualizar, por exemplo, quais autores possuem comentários relevantes e os que possuem o maior número de comentários. Essas informações podem ser usadas por um gerente de projetos, em conjunto com outras análises, para mensurar, por exemplo, o envolvimento e a qualidade das participações de membros em um projeto.

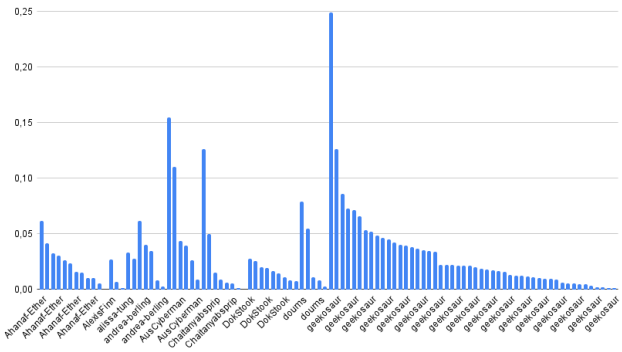

Figura 9: Relevância temática no por Autor Comentário, usando a mesma fonte de informações da Tabela1

Um segundo exemplo de um arquivo CSV gerado, usando a visão "Relevância Temática por Status das *Issues*", pode ser visualizado na Tabela 2. Nesse recorte de dados, tem-se três colunas: *idTopico statusTopico* e *relevanciaComentario*.

idTopico	statusTopico	relevanciaComentario
1144832458	closed	0,1688622452
1098238192	closed	0,1081061795
1098238192	closed	0,0877012043
1098238192	closed	0,0612397314
1093918803	closed	0,0444178372
1119535474	closed	0,0083132683
838934012	closed	0
1077896353	open	0,2422956823
1073522144	open	0,1575319991
1130498842	open	0,1545851503
993927243	open	0,1405014142
1022508325	open	0,1306193974
992751744	open	0,0485278502
1156244675	open	0,0034077172

Tabela 2: Recorte de um arquivo .csv para a visão Relevâncias por Status das *Issues*

Para ilustrar o uso desses dados, foi gerado um gráfico (Figura 10), com *status de Issues* referentes ao ano de 2022 e as relevâncias dos respectivos comentários postados. Essa visão representa outro exemplo de arquivo CSV que a biblioteca ColminerRT original não é capaz de gerar, devido ao arquivo persistido não passar pela reorganização gerada pela visão e pelo filtro criado. Em uma observação rápida do gráfico, percebe-se que os comentários mais relevantes do período pertencem a *Issues* que ainda se encontram abertas. Os dados dessa visão poderiam ser cruzados com outros dados das *issues*, em análises futuras, como por exemplo, tempo de resolução com o intuito de avaliar, por exemplo, se a relevância impacta no tempo de resolução. Onde este parâmetro é composto do resultado da subtração de data de fechamento da *issue* pela data de sua criação. Para *issues* que se encontram em aberto, a data de fechamento ainda não existe, por isso pode ser usada em substituição a data do último comentário.

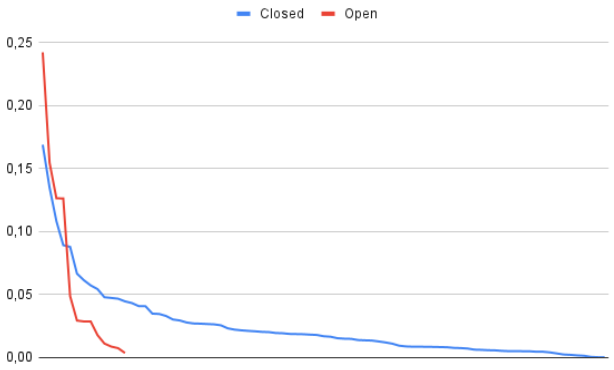

Figura 10: Relevância temática no período de 2022 por *status* das *Issues*

6 CONCLUSÃO

Este trabalho apresentou os resultados da aplicação do padrão *Strategy* em uma funcionalidade da biblioteca ColMinerRT. Essa revisão no projeto da arquitetura trouxe ganhos importantes em termo de manutenibilidade e extensibilidade do código, em especial, por propiciar o desacoplando da classe *bibliotecaColMinerRTFachada*. Houve melhoria no contexto do mecanismo de geração dos dados em um formato intercambiável, permitindo agora a geração de múltiplas visões, escolhidas pelo usuário, ao invés de uma tabela única com todos os dados. Com o uso das visões, criou-se um mecanismo para adição de micro funcionalidades, de fácil utilização por parte do usuário, que pode vir a ter um leque de visões mais amplo. Por fim, as mudanças aplicadas facilitam o uso posterior dos arquivos gerados em ferramentas específicas de geração de gráficos e análises estatísticas dos dados.

Como trabalhos futuros, vislumbra-se: investigar o uso de outros padrões de projeto que melhorem a coesão da classe **CalculoRelevanciaFachada**, de forma a respeitar o princípio da responsabilidade única *(SRP)* do *SOLID*; criar novas visões para os dados gerados; e adicionar novos mecanismos para exportação dos dados em formatos intercambiáveis, além do CSV.

REFERÊNCIAS

[1] Uncle Bob. 2022. The Principles of OOD. http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod

[2] Aikaterini Christopoulou, E.A. Giakoumakis, Vassilis E. Zafeiris, and Vasiliki Soukara. 2012. Automated refactoring to the Strategy design pattern. *Information and Software Technology* 54, 11 (2012), 1202–1214. https://doi.org/10.1016/j.infsof.2012.05.004

[3] Ralph Johnson John Vlissides Erich Gamma, Richard Helm. 2000. *Padrões de Projeto: Soluções reutilizáveis de software orientado a objetos*. Bookman Companhia Editora, Porto Alegre, RS, BR. Trad. Luiz A. Meirelles Salgado..

[4] Mohamed Fayad and Marshall P. Cline. 1996. Aspects of Software Adaptability. *Commun. ACM* 39, 10 (oct 1996), 58–59. https://doi.org/10.1145/236156.236170

[5] Isaias Ferreira, Antônio Maria de Resende, and Heitor Costa. 2012. Analysis of the Impact of the Application of Design Patterns on the Maintainability of an Object Oriented System. In *Anais do XI Simpósio Brasileiro de Qualidade de Software* (Fortaleza). SBC, Porto Alegre, RS, Brasil, 341–348. https://doi.org/10.5753/sbqs.2012.15327

[6] Ewa Figielska. 2017. Using Template Method and Strategy Design Patterns in the Python Implementation of a Metaheuristic Algorithm for Solving Scheduling Problems. *Zeszyty Naukowe Warszawskiej Wyższej Szkoły Informatyki* (2017).

[7] Anthony Lauder and Stuart Kent. 1998. Precise visual specification of design patterns. In *ECOOP'98 — Object-Oriented Programming*, Eric Jul (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 114–134.

[8] W. Li and S. Henry. 1993. Maintenance metrics for the object oriented paradigm. In *[1993] Proceedings First International Software Metrics Symposium*. 52–60. https://doi.org/10.1109/METRIC.1993.263801

[9] Luiz Eugênio Coelho Neto and Gláucia Braga e Silva. 2018. ColMiner: A Tool to Support Communications Management in an Issue Tracking Environment. In *Proceedings of the XIV Brazilian Symposium on Information Systems* (Caxias do Sul, Brazil) *(SBSI'18)*. Association for Computing Machinery, New York, NY, USA, Article 50, 8 pages. https://doi.org/10.1145/3229345.3229398

[10] Thomas M Pigoski. 1996. *Practical software maintenance: best practices for managing your software investment*. Wiley Publishing.

[11] Muhammad Ehsan Rana and Eddy Khonica. 2021. Impact of Design Principles and Patterns on Software Flexibility: An Experimental Evaluation Using Flexible Point (FXP). *Journal of Computer Science* 17, 7 (Jul. 2021), 624–638. https://doi.org/10.3844/jcssp.2021.624.638