pySol: Uma Proposta Python para Automação de Busca na SBC OpenLib

Nicolas Chagas Souza
Faculdade UnB Gama (FGA)
Universidade de Brasília (UnB)
Brasília, DF, Brasil
nicolas.souza@aluno.unb.br

André Barros de Sales
Faculdade UnB Gama (FGA)
Universidade de Brasília (UnB)
Brasília, DF, Brasil
andrebdes@unb.br

Eduardo G. Q. Palmeira
Faculdade UnB Gama (FGA)
Universidade de Brasília (UnB)
Brasília, DF, Brasil
eduardo.palmeira@unb.br

RESUMO

Conducting a systematic review is complex, but it can be simplified using computational resources. Utilizing multiple research sources is essential to cover most studies relevant to the investigated topic. In this context, SBC OpenLib (SOL), the open digital library of the Brazilian Society of Computing (SBC), is an important bibliographic source for systematic reviews in Computing-related fields, offering access to all academic and scientific content produced by the SBC. However, SOL's limitation in its automatic search feature is the lack of flexibility in exporting results, a crucial criterion for a database to be included in a systematic review's search strategy. Addressing this issue, this paper introduces pySol, a Python-based tool designed to automate searches and export results from the SOL database. PySol was developed using web scraping techniques to extract automatic search results from the SOL database. Its development followed Test-Driven Development (TDD) principles, resulting in over 280 unit tests and a code coverage of 93%. These indicators highlight the tool's reliability for systematic searches. The tool intends to support the study identification stage in systematic reviews, significantly reducing the time and effort needed to search in the SOL database. The automation of this process not only eases the execution of systematic searches in this database but also enables the export of results in BibTeX format, facilitating integration with reference managers.

KEYWORDS

Automação, Python, Revisão Sistemática de Literatura

1 INTRODUÇÃO

O progresso científico tem sido impulsionado pela utilização de diversos recursos de pesquisa com o objetivo de melhorar a qualidade de vida humana. Estudos primários são essenciais nesse cenário, pois buscam introduzir inovações e tecnologias ou detalhar aquelas já existentes em contextos específicos. De modo complementar, estudos secundários desempenham um papel crucial ao sumarizar os resultados dos estudos primários, oferecendo um panorama detalhado sobre determinado tópico e sinalizando direções para pesquisas futuras [2, 13]. Em outros termos, estudos individuais que contribuem para revisões da literatura são denominados estudos primários, enquanto as próprias revisões são consideradas estudos secundários.

Revisões da literatura podem ser realizadas de maneira informal ou sistemática. A abordagem sistemática, embora mais trabalhosa, garante um processo controlado e rigoroso, tornando as revisões sistemáticas da literatura confiáveis, auditáveis, replicáveis e imparciais em relação aos interesses dos pesquisadores [8]. Tais tipos de estudo secundário permitem a sumarização de evidências existentes sobre um tema de pesquisa ou fenômeno e ajudam a identificar lacunas na literatura, indicando direções relevantes para novos estudos primários [1, 14]. Na área de Computação e afins, a rápida evolução do estado da arte, impulsionada pela velocidade do avanço tecnológico e inovação, faz das revisões sistemáticas um método eficiente para sumarizar evidências. Elas auxiliam na tomada de decisões de profissionais e acadêmicos, integrando experiências práticas e valores humanos [16].

Conduzir uma revisão sistemática é uma atividade complexa que pode ser simplificada com o uso de recursos computacionais. Estes frequentemente automatizam partes do processo, como nas buscas automáticas realizadas em bases bibliográficas. Neste cenário, o pesquisador se limita a definir a *string* de busca (termo contendo palavras-chave) e parâmetros, enquanto o sistema computacional da base de dados desempenha um papel ativo e facilitador na execução automática da busca. Esse método de busca não apenas auxilia a identificação de estudos primários, mas também permite uma replicação confiável de consultas em bases bibliográficas [8].

Uma única base bibliográfica pode não ser suficiente para retornar todos os estudos primários relevantes para a condução de uma revisão sistemática [2, 6]. Dessa forma, considerar múltiplas fontes de pesquisa é uma estratégia necessária para abranger a maior parte dos estudos pertinentes ao tópico investigado [8]. Assim, a escolha das bases bibliográficas impacta diretamente na amplitude da identificação de evidências, essenciais para responder às questões de pesquisa da revisão e formular conclusões [6]. Uma seleção limitada de bases pode levar à perda de estudos primários relevantes, enquanto a inclusão de todas as bases bibliográficas disponíveis pode gerar um volume excessivo de estudos irrelevantes ao escopo da revisão [8]. Portanto, é crucial considerar as principais bases bibliográficas relacionadas à área de pesquisa em questão.

Em relação à área de Computação e afins, a *IEEE Xplore*[1] e a *ACM Digital Library*[2] são algumas das principais bases bibliográficas internacionais. Uma busca eficiente para revisão sistemática pode combinar elas com outros motores de busca como *Scopus*[3] e *Web of Science*[4] [8]. Neste contexto, a *SBC OpenLib* (SOL) [3], biblioteca digital aberta da Sociedade Brasileira de Computação (SBC) [5], também é uma importante base bibliográfica a ser considerada em uma revisão sistemática, por oferecer acesso a todo conteúdo acadêmico científico produzido pela SBC.

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

No entanto, a base bibliográfica SOL apresenta uma limitação no recurso de busca automática, que é a falta de versatilidade na exportação de resultados; este é um dos critérios cruciais que uma base deve possuir para ser considerada na estratégia de busca de uma revisão sistemática [6]. A falta de versatilidade de uma base dificulta a transferência de dados relativos à consulta para softwares de gerenciamento bibliográfico como Mendeley[5], Zotero[6] e StArt[7] [7]. Em situações onde uma grande quantidade de estudos é recuperada, essa limitação implica na necessidade de exportação manual e individual de cada artigo pelo pesquisador, o que pode representar um trabalho exaustivo [6]. Diante dessa questão, o objetivo deste trabalho é introduzir o pySol, uma ferramenta desenvolvida em Python que visa automatizar buscas e a exportação de resultados da base bibliográfica SBC OpenLib, apoiando a atividade de seleção de estudos durante a condução de revisões sistemáticas.

O artigo está organizado em 5 seções. A seção 2 contém uma análise do processo atual de busca e exportação dos resultados na SOL. Em seguida, na seção 3, o percurso metodológico da pesquisa e ferramentas utilizadas são definidas. A seção 4 apresenta os requisitos elicitados para a criação da ferramenta e as técnicas de codificação utilizadas. A seção 5 demonstra como a ferramenta pode ser utilizada e, por fim, a seção 6 sumariza os resultados alcançados e apresenta propostas de trabalhos futuros.

2 CONTEXTO

A presente seção apresenta o processo atual de busca e exportação de resultados na SOL, que foi avaliado no dia 15 de dezembro de 2023, por meio de uma busca utilizando o formulário de busca integrada, representado na Figura 1.

Figura 1: Formulário de busca integrada da base SOL.

Para fazer uma busca e exportar as citações no formato BibTeX, o usuário deve realizar as ações representadas na Tabela 1.

O pesquisador precisará repetir a ação 6 — selecionar um artigo e gerar a citação BibTeX — para cada resultado, o que se torna muito trabalhoso a medida que a quantidade de artigos encontrados aumenta. Como destacado na motivação desse trabalho, essa atividade trivial ocupa o tempo do pesquisador e não fornece versatilidade na exportação dos dados, dificultando a execução de revisões sistemáticas.

Ação	Descrição
1	Acessar a página de busca (Figura 1)
2	Selecionar um ou mais tipos de publicação (bases)
3	Preencher o termo desejado no campo de pesquisa
4	Clicar no botão pesquisar
5	Visualizar a tela de resultados
6	Selecionar um artigo e gerar a citação BibTeX

Tabela 1: Atividades do usuário.

3 PERCURSO METODOLÓGICO

A pesquisa visa descrever os problemas envolvidos na busca automática na SOL e apresentar uma proposta de solução, por meio da definição de uma metodologia fundamentada no raciocínio dedutivo. Quanto a natureza, é aplicada, já que objetiva gerar conhecimentos para aplicação prática e direcionados à resolução de problemas específicos. Quanto aos objetivos, é exploratória, pois visa proporcionar maior familiaridade com o problema. Quanto ao procedimento, é experimental, tem como objeto de estudo a automação de buscas na base bibliográfica SOL; consiste em selecionar as variáveis que poderiam influenciá-la e definir formas de controle e observação dos efeitos que essas variáveis produzem [9].

Para automatizar a busca de *strings* na SOL e gerar um arquivo BibTeX com os resultados, é necessário analisar o funcionamento da pesquisa — como os parâmetros da pesquisa são enviados para o servidor e em qual formato a resposta é retornada — e como são geradas as citações BibTeX.

A coleta de dados referente a essas variáveis foi realizada por meio de inspeções da página de busca, sendo possível constatar que uma pesquisa segue o seguinte fluxo: os parâmetros da pesquisa são enviados para o servidor como parâmetros na URL de uma requisição HTTP GET, que tem como resposta uma página HTML com os resultados da busca. A geração das citações nos formatos disponíveis, como o BibTeX, é realizada por um *plugin* na página de detalhes de um resultado.

Tendo em vista que os dados da pesquisa são retornados no formato HTML, pode se optar por realizar a recuperação dos dados com técnicas de *web data scraping*, processo que consiste na extração e combinação de conteúdos da web de maneira sistemática [10].

Uma das técnicas de *web scraping* consiste em navegar pelos elementos da página HTML no formato de árvore [15], atividade facilita por ferramentas disponíveis para a linguagem de programação Python. Dentre elas, destacam-se as bibliotecas: *BeautifulSoup*[8], que possibilita a interpretação de elementos HTML na forma de árvore; *requests*[9], utilizada para realizar requisições HTTP; e *re*[10], que fornece operações com expressões regulares (*regex*).

[5] https://www.mendeley.com/
[6] https://www.zotero.org/
[7] https://www.lapes.ufscar.br/resources/tools-1/start-1
[8] https://pypi.org/project/beautifulsoup4/
[9] https://pypi.org/project/requests/
[10] https://docs.python.org/3/library/re.html

As *regex* foram utilizadas para auxiliar na seleção de elementos nas páginas, uma vez que estudos [4, 11] demonstram o menor consumo de memória com uso dessa técnica dentre as mais populares de *web scraping*: HTML DOM, XPath, seletores CSS e *regex*.

Na próxima sessão são apresentados detalhes sobre o desenvolvimento da solução.

4 PYSOL

A ferramenta pySol foi desenvolvida para apoiar as atividades envolvidas nas revisões sistemáticas, e visa fornecer métodos para automatizar a busca por artigos na SOL e a exportação de referências no formato BibTeX.

A criação do ferramenta foi dividida em duas etapas, a elicitação de requisitos, descrita na seção 4.1, e desenvolvimento, seção 4.2.

4.1 Elicitação de Requisitos

O objetivo da ferramenta é automatizar a busca integrada na SOL e gerar uma lista de artigos encontrados. A definição dos requisitos funcionais da ferramenta se deu a partir das ações do usuário (Tabela 1), e das variáveis referentes à automação da busca, definidas na seção 3:

- R.1 A ferramenta deve efetuar a pesquisa com os dados inseridos pelo usuário (ação 4 na Tabela 1);
- R.2 A ferramenta deve fornecer meios para o usuário inserir os dados da pesquisa (referente às ações 2 e 3, na Tabela 1);
- R.3 A ferramenta deve disponibilizar os resultados para o usuário (ação 5 na Tabela 1).

4.2 Desenvolvimento

A técnica *Test Driven Development (TDD)*, uma prática de desenvolvimento na qual os testes funcionais são criados antes da implementação da funcionalidade [12], foi utilizada. A técnica foi escolhida por fornecer maior manutenibilidade ao código-fonte, favorecendo refatorações, e por aumentar a confiabilidade da ferramenta, atributo essencial no contexto de automação de um processo sistemático. Um ciclo de *TDD* segue uma sequência que inicia com a codificação de testes automatizados, avança para a implementação da funcionalidade e culmina na refatoração dos códigos criados.

Cada requisito foi codificado em uma etapa de desenvolvimento dos métodos e modelagem das entidades, processos apoiados por ciclos de *TDD*. A ferramenta foi disponibilizada no repositório de um dos autores [11], onde constam, também, as condições para uso do código. Algumas das atividades desenvolvidas durante os ciclos de codificação estão destacadas a seguir. Para melhorar a compreensão da implementação, as funções nos trechos de código foram ajustadas, eliminando a necessidade de conhecimentos detalhados da sintaxe Python.

Inicialmente, foi implementado um recurso de *cache* (Trecho de Código 1), para evitar uma sobrecarga de requisições nos servidores da SOL durante o período de desenvolvimento. Uma vez que a *url* da requisição é única para cada consulta, já que contém os parâmetros de busca, ela é utilizada para gerar o nome do arquivo temporário onde a resposta da requisição é armazenado.

```python
import requests

def requisicao_get_com_cache(url):
    nome_arquivo = gerar_nome_arquivo(url)
    if arquivo_existe(nome_arquivo):
        return ler_conteudo_do_arquivo(nome_arquivo)
    resposta_http = requests.get(url)
    escrever_resposta_no_arquivo(nome_arquivo,
     resposta_http)
    return resposta_http
```

Trecho de Código 1: Lógica da implementação de *cache* nas requisições.

Considerando-se que robôs são consideravelmente afetados por mudanças no conteúdo HTML dos recursos acessados [10] — por exemplo, como ocorre em campos com geração de nome automático —, optou-se pela recuperação dos nomes dos campos a partir da interpretação da página. Os nomes dos campos do formulário de busca na SOL podem ser obtidos a partir da relação destes com as suas descrições (*label*), uma vez que cada *tag label* refere-se a um único campo, por meio da propriedade *for* (Figura 2).

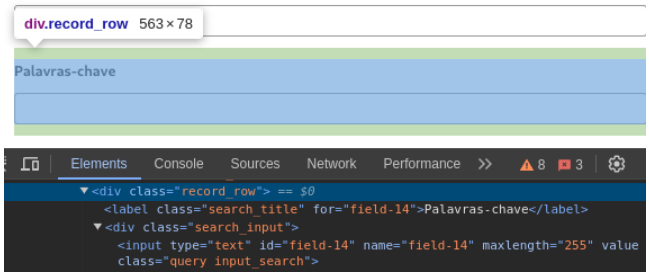

Figura 2: Campo palavras-chave e correspondência no código HTML.

Inicialmente foram elaborados os testes representados no Trecho de Código 2, e, em seguida, a funcionalidade foi implementada (Trecho de Código 3).

```python
import pytest
casos_de_teste = [('titulo','field-3'),
                  ('resumo','field-15'),
                  ('palavras-chave','field-14')]
@pytest.mark.parametrize("campo,expected",casos_de_teste)
def test_field_name(campo,expected):
    assert get_id_do_campo(campo) == expected
```

Trecho de Código 2: Casos de teste para os nomes dos campos

```python
def get_id_do_campo(tipo_de_campo):
    campos = dict()
    soup = get_form_de_busca_como_beautiful_soup()
    for l in soup.find_all('label'):
        id_do_campo = l.get('for')
        descricao = l.text
        campos[id_do_campo] = {'label': descricao}
    for i in soup.find_all('input'):
        id_do_campo = i.get('id')
        nome_do_campo = i.get('name')
        if id_do_campo:
            if id_do_campo not in campos:
                campos[id_do_campo] = {}
            campos[id_do_campo]['name'] = nome_do_campo
    for id_do_campo, dados_do_campo in campos.items():
        if (dados_do_campo.get('label') and
```

[11] https://gitlab.com/nszchagas/pySol

```
                sao_iguais_ignorando_case(tipo_de_campo,
    dados_do_campo.get('label'))):
            return id_do_campo
    return None
```

Trecho de Código 3: Recuperação dos identificadores dos campos.

Após garantir a recuperação dos identificadores dos campos contemplados pela ferramenta, foi possível iniciar a implementação da pesquisa a partir de parâmetros definidos pelo usuário (R.1), atividade iniciada pela criação de testes para averiguar se a quantidade de resultados está correta, a partir da comparação com os resultados encontrados de forma não automatizada em 15 de dezembro de 2023 (Trecho de Código 4).

```
import pytest
casos_de_teste = [(
# String de Busca | Campo | Tipos de Publicacao | Qtd.
'cyber','resumo','anais_de_evento,periodicos,livros',49),
('cyber AND("attacker behavior"OR"safety impact")','
    resumo','anais_de_evento,periodicos',1)]
@pytest.mark.parametrize("qs,f,t,qtd_ex",casos_de_teste)
def test_get_qt_resultados(qs,f,t,qtd_ex):
    qtd = get_qt_resultados(string_de_busca=qs,campo=f,
                            tipos_de_publicacao=t)
    assert qtd == qtd_ex
```

Trecho de Código 4: Verificação da quantidade de resultados encontrados.

Uma vez que a quantidade de resultados é apresentada em uma *div* de paginação (Figura 3) com a classe *articles_count*, é possível recuperar esse número utilizando o pacote *BeautifulSoup*, para acessar a *div* com a partir de um seletor CSS, e utilizando uma *regex* para selecionar o terceiro conjunto de dígitos no texto de paginação (Trecho de código 5).

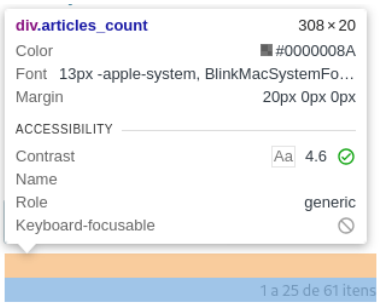

Figura 3: Elemento de paginação na visualização dos resultados.

```
from bs4 import BeautifulSoup; import re
def get_qt_resultados(string_de_busca, campo,
                      tipos_de_publicacao) -> int:
    qt = 0
    html = get_resultado_busca(string_de_busca, campo,
     tipos_de_publicacao)
    soup = BeautifulSoup(html, 'html.parser')
    for x in soup.select('.articles_count'):
        regex = r'\d+ a \d+ de (\d+) itens'
        match = re.match(regex, x.text)
        qt = int(match.group(1))
    return qt
```

Trecho de Código 5: Método para recuperar a quantidade de resultados.

Após garantir a precisão na quantidade de resultados encontrados, deu-se início à coleta de dados necessários para gerar as citações em formato BibTeX. A inspeção da página de diversos resultados, como o representado na Figura 4, indicou que tais citações são geradas por um *plugin* da SOL, cuja *url* pode ser obtida a partir da manipulação da *url* de visualização de um resultado. Sendo assim, torna-se necessário coletar apenas a *url* para a página de cada resultado (Trecho de Código 6).

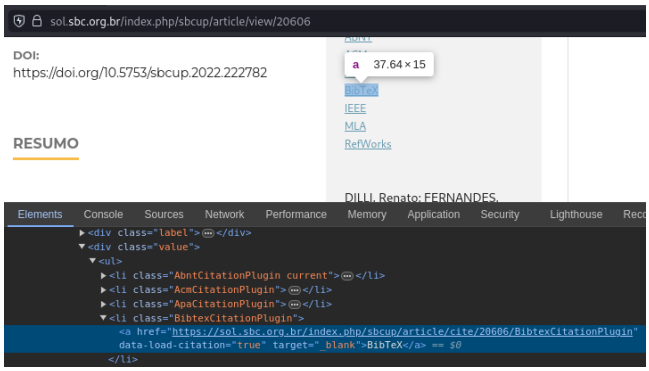

Figura 4: Elemento da página[12] responsável pela geração da citação em BibTeX.

```
from bs4 import BeautifulSoup;
def get_urls_detalhes(**kwargs):
    qt = get_qt_resultados(**kwargs); pagina = 1;
    urls = []
    while len(urls) < qt:
        html=get_pagina_resultados(**kwargs)
        soup = BeautifulSoupt(html, 'html.parser')
        for a in soup.find_all('a',
                  attrs={'class': 'record_title'}):
            urls.append(a.get('href'))
        pagina += 1
    return urls
```

Trecho de Código 6: Recuperação das *urls* dos resultados.

A conclusão e verificação da automação de busca marcaram o encerramento da implementação do Requisito R.1. A primeira etapa foi concluída com a modelagem, que culminou na criação de dois pacotes: *util* (Figura 5) e *service*.

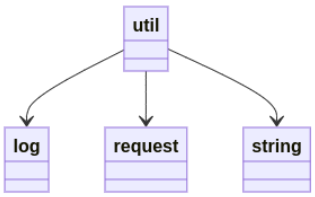

Figura 5: Diagrama de pacotes do módulo *util*.

No pacote *util* estão alguns métodos auxiliares provenientes de três pacotes: *log*, que contém métodos para centralizar a geração e formatação dos *logs*; *request*, que centraliza as requisições HTTP; e *string*, que contém métodos utilitários para a formatação de *strings*, como a remoção de espaços e quebras de linha, operações que apoiam a leitura de conteúdos HTML.

O pacote *service* contém as classes representadas na Figura 6. A classe abstrata *BaseService* visa fornecer um comportamento padrão para a *SolService*, que deve fazer a pesquisa e escrever os resultados da busca (Requisito R.3).

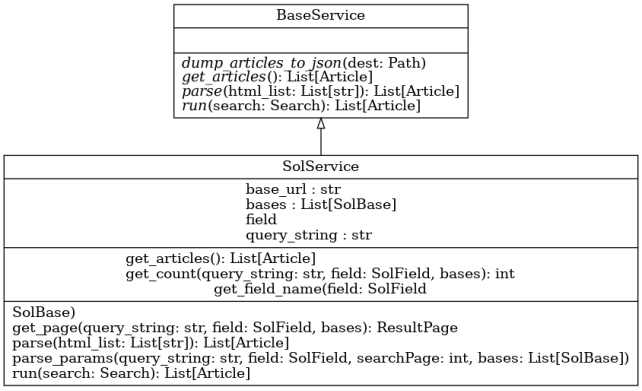

Figura 6: Classes do pacote *service*.

A segunda etapa teve por intuito permitir que o usuário forneça os parâmetros de pesquisa à ferramenta (Requisito R.2) e foi iniciada pela modelagem das entidades envolvidas no problema. A organização do código gerou a primeira versão do pacote *model* (Figura 7).

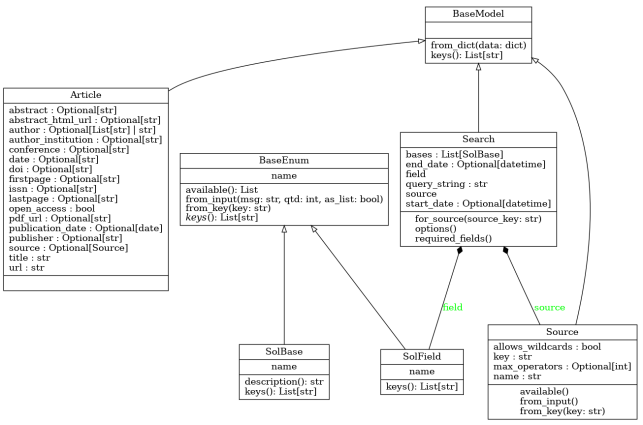

Figura 7: Diagrama de classes do pacote *model*.

A classe *Article* representa um resultado, e agrupa os dados que podem ser coletados a partir da leitura da página; a *Search* forma a interface entre o usuário e a camada *service*, mapeando os campos do formulário de busca (Figura 1) à atributos — o campo de busca foi mapeado a *field* e o tipo de publicação a *bases* — e a classe *Source* agrupa dados referentes à pesquisa na base utilizada, a SOL.

Para realizar a leitura de argumentos, foi utilizado o pacote *argparse*[13] do Python, que fornece ferramentas para a leitura de argumentos fornecidos pelo usuário durante a execução do programa (Trecho de Código 7), além de fornecer a geração automática de um serviço de ajuda.

```
import argparse; from dataclasses import fields;
class Search:
    @classmethod
    def options(cls):
        return [{'name_or_flags': f'--{atributo.name}',
                 'help': atributo.metadata['help']}
        for atributo in fields(cls)]

def main():
    parser = argparse.ArgumentParser()
    for campo in Search.options():
        parser.add_argument(**campo)
    args = parser.parse_args()
    return Search(query_string=args.query_string,
                  field=args.field,
                  start_date=args.start_date,
                  end_date=args.end_date,
                  bases=args.bases )
```

Trecho de Código 7: Utilização do *argparse*.

As configurações realizadas durante esse ciclo possibilitaram que o usuário inicie o programa fornecendo parâmetros para a pesquisa na base SOL, atendendo assim ao Requisito R.2.

A última etapa, que consiste na disponibilização dos resultados em formato BibTeX para o usuário (Requisito R.3), foi implementada por meio dos métodos representados no Trecho de Código 8, a partir de uma requisição para o *plugin* utilizado pela SOL e a escrita dos resultados em um arquivo no formato BibTeX.

```
import bibtexparser; import requests
from bs4 import BeautifulSoup;
def get_bibtex_da_url(url):
    plugin = url.replace('view', 'cite') \
             + '/BibtexCitationPlugin'
    html = requests.get(plugin).content
    bibtex = BeautifulSoup(html, 'html.parser').text
    bibtexparser.loads(bibtex)
    return bibtex

def salva_resultados_em_bib(resultados):
    referencias = ''
    for r in resultados:
        referencias += get_bibtex_da_url(r.url) + '\n'
    with open('referencias.bib', 'w') as arquivo:
        bibtexparser.dumps(referencias, arquivo)
```

Trecho de Código 8: Métodos envolvidos na recuperação das citações.

Após a implementação dos três requisitos elicitados, a pySol contava com 284 testes unitários e uma cobertura de código de 93% (Figura 8), métricas que visam garantir manutenibilidade e confiabilidade à ferramenta.

[13] https://pypi.org/project/argparse/

```
---------- coverage: platform linux, python 3.11.2-final-0 -------
Name                           Stmts   Miss Branch BrPart  Cover
-------------------------------------------------------------------
pySol/__main__.py                 43      9     12      0    76%
pySol/model/__init__.py           28      1     18      0    98%
pySol/model/article.py            60      7     28      2    85%
pySol/model/enums.py              40      3     18      0    95%
pySol/model/search.py             98      8     52      6    91%
pySol/model/source.py             16      0      4      0   100%
pySol/service/sol_service.py     127      0     35      0   100%
pySol/util/__init__.py             3      0      0      0   100%
pySol/util/log.py                 18      0      2      1    95%
pySol/util/request.py             33      2     10      1    93%
pySol/util/string.py              17      0      0      0   100%
-------------------------------------------------------------------
TOTAL                            483     30    179     10    93%
```

Figura 8: Relatório da cobertura de código.

5 DEMONSTRAÇÃO DA FERRAMENTA

A ferramenta dispõe de um sistema de ajuda, que pode ser acessado via linha de comando (CLI), bastando executá-la com o parâmetro de ajuda (-h), conforme ilustra a Figura 9.

```
root@debian:~/pysol# python pySol -h
usage: pySol [-h] [--query_string QUERY_STRING] [--field FIELD]
             [--publication_type PUBLICATION_TYPE]
             [--start_date START_DATE] [--end_date END_DATE]
             [--bib_dest BIB_DEST]

options:
  -h, --help            show this help message and exit
  --query_string QUERY_STRING, -qs QUERY_STRING
                        Search string. [Ex:'"(test* AND software) OR
                        ("quality assurance")"']
  --field FIELD, -f FIELD
                        Field to search the query string. [Options:
                        query, título, resumo, palavras-chave]
  --publication_type PUBLICATION_TYPE, -pt PUBLICATION_TYPE
                        Publication type. Default value: ['anais',
                        'periódicos', 'livros'] (Optional)
                        [Ex:'periódicos'] [Options: anais, periódicos,
                        livros]
  --start_date START_DATE, -sd START_DATE
                        Start date for searching in ISO format.
                        (Optional) [Ex:'2013-12-01']
  --end_date END_DATE, -ed END_DATE
                        End date for searching in ISO format.
                        (Optional) [Ex:'2023-12-31']
  --bib_dest BIB_DEST, -bd BIB_DEST
                        Name of the bibtex destination. Default value:
                        search.bib (Optional)
```

Figura 9: Sistema de ajuda ao usuário.

A execução da ferramenta foi realizada em 15 de dezembro de 2023, com a busca da *string* "social engineering" no campo resumo, em publicações do tipo anais de evento e periódicos, abrangendo o período até 31 de dezembro de 2019, conforme ilustra a Figura 10.

```
root@debian:~/src# python pySol --query_string '"social engineering"' --field re
sumo --publication_type anais,periódicos --end_date=2019-12-31
[INFO 14:21:31] (pySol): Found 3 articles.
[INFO 14:21:31] (pySol): Writing 3 articles to search.bib.
```

Figura 10: Execução da busca utilizando a ferramenta pySol.

```
@inproceedings{sbsi, author = {Gliner Alencar and Marcelo Lima and André Firmo},
 title = {O Efeito da Conscientização de Usuários no Meio Corporativo no Combate
 à Engenharia Social e Phishing}, booktitle = {Anais do IX Simpósio Brasileiro d
e Sistemas de Informação}, location = {João Pessoa}, year = {2013}, keywords = {
}, issn = {0000-0000}, pages = {254--259}, publisher = {SBC}, address = {Porto A
legre, RS, Brasil}, doi = {10.5753/sbsi.2013.5694}, url = {https://sol.sbc.org.b
r/index.php/sbsi/article/view/5694} }
@inproceedings{sbseg_estendido, author = {Cristoffer Leite and João Gondim and P
riscilla Barreto and Eduardo Alchieri}, title = {Waste Flooding: Ferramenta para
 Retaliação de Phishing}, booktitle = {Anais Estendidos do XIX Simpósio Brasilei
ro de Segurança da Informação e de Sistemas Computacionais}, location = {São Pau
lo}, year = {2019}, keywords = {}, issn = {0000-0000}, pages = {27--34}, publish
er = {SBC}, address = {Porto Alegre, RS, Brasil}, doi = {10.5753/sbseg_estendido
.2019.14001}, url = {https://sol.sbc.org.br/index.php/sbseg_estendido/article/vi
ew/14001} }
@inproceedings{sbrc, author = {Lucas Ayres and Italo Valcy S. Brito and Rodrigo
Gomes e Souza}, title = {Utilizando Aprendizado de Máquina para Detecção Automát
ica de URLs Maliciosas Brasileiras}, booktitle = {Anais do XXXVII Simpósio Brasi
leiro de Redes de Computadores e Sistemas Distribuídos}, location = {Gramado}, y
ear = {2019}, keywords = {}, issn = {2177-9384}, pages = {972--985}, publisher =
 {SBC}, address = {Porto Alegre, RS, Brasil}, doi = {10.5753/sbrc.2019.7416}, ur
l = {https://sol.sbc.org.br/index.php/sbrc/article/view/7416} }
```

Figura 11: Conteúdo do arquivo search.bib.

A ferramenta encontrou três resultados e os armazenou no arquivo *search.bib*, cujo conteúdo está representado na Figura 11.

Em comparação, para executar a mesma busca pela plataforma SOL é necessário preencher os campos do formulário de busca (Figura 1) e seguir os passos listados na Tabela 1.

Os resultados obtidos estão representados na Figura 12 e correspondem aos resultados gravados no arquivo gerado pela ferramenta, conforme mostrado na Figura 11.

Figura 12: Resultados da busca.

Para exportar os resultados, sem utilizar a ferramenta, o pesquisador deve clicar em cada resultado, escolher o tipo de citação como BibTeX e a citação será exibida em uma caixa de texto, conforme exemplificado para o primeiro resultado [14] na Figura 13.

6 CONSIDERAÇÕES FINAIS

Este trabalho teve como objetivo apresentar a proposta de uma ferramenta para automatizar a busca e exportação de referências na base bibliográfica SOL, oferecendo versatilidade na exportação de resultados dessa base. A ferramenta pySol foi criada, utilizando recursos de *web scraping*, para extrair resultados de buscas automáticas na SOL, a partir da pesquisa de uma *string* de busca em um

[14] https://sol.sbc.org.br/index.php/sbsi/article/view/5694

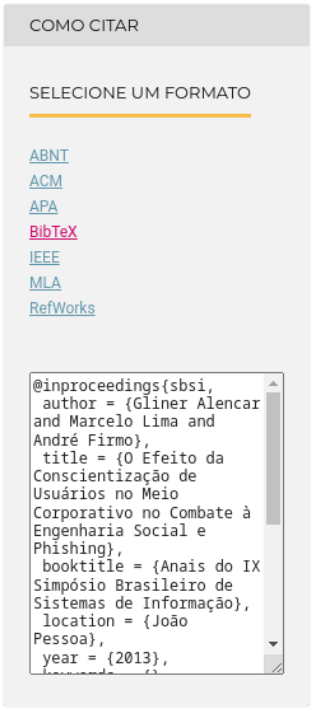

Figura 13: Exemplo de citação para o primeiro resultado.

campo – entre resumo, palavras-chave e título – e exportá-los em formato BibTeX.

As principais contribuições deste trabalho incluem a documentação do processo de desenvolvimento da ferramenta pySol e a própria ferramenta[15], que objetiva otimizar as revisões sistemáticas feitas na SOL. A codificação, apoiada em práticas de TDD, resultou em mais de 280 testes unitários e em uma cobertura de código de 93% — indicadores que apoiam a confiabilidade da ferramenta para buscas sistemática.

A eficácia de uma revisão sistemática está intrinsecamente atrelada à disponibilidade de uma variedade de recursos [8]. Neste contexto, a ferramenta pySol visa apoiar a etapa de identificação de estudos em revisões sistemáticas, reduzindo significativamente o tempo e esforço necessários para busca na base SOL. A automação desse processo não apenas facilita a realização de buscas sistemáticas, mas também viabiliza a exportação de resultados em formato BibTeX, favorecendo a integração com gerenciadores de referências e aumentando o rigor no processo de identificação de estudos.

A ferramenta também pode auxiliar na auditoria e reprodutibilidade de buscas automáticas realizadas na base SOL. Isso permite que outros pesquisadores ou pareceristas, interessados em replicar as buscas automáticas, ou verificar o protocolo da revisão sistemática, possam fazê-lo com maior facilidade e precisão nessa base.

Dentre as limitações da ferramenta, encontram-se a restrição da pesquisa aos campos supracitados, a ausência de interface gráfica e de testes automatizados a nível de sistema e de integração.

Para trabalhos futuros, planeja-se implementar a busca nos demais campos, extrair informações mais abrangentes sobre os resultados, disponibilizar um pacote da ferramenta proposta no repositório *PyPI*[16], criar uma versão gráfica da ferramenta na forma de *plugin* para a SOL e automatizar os testes de integração. Essas ações visam melhorar e expandir continuamente a utilidade da ferramenta.

REFERÊNCIAS

[1] Jorge Biolchini, Paula Gomes Mian, Ana Candida Cruz Natali, and Guilherme Horta Travassos. 2005. *Systematic Review in Software Engineering.* Technical Report. Universidade Federal do Rio de Janeiro (COPPE/UFRJ), Rio de Janeiro, Brasil. RT - ES 679/05.

[2] Pearl Brereton, Barbara A. Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil. 2007. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of Systems and Software* 80, 4 (2007), 571–583. https://doi.org/10.1016/j.jss.2006.07.009

[3] Biblioteca Digital da Sociedade Brasileira de Computação. 2023. Sobre a SBC OpenLib. https://sol.sbc.org.br/index.php/indice/about.

[4] Irfan Darmawan, Muhamad Maulana, Rohmat Gunawan, and Nur Widiyasono. 2022. Evaluating Web Scraping Performance Using XPath, CSS Selector, Regular Expression, and HTML DOM With Multiprocessing Technical Applications. *JOIV: International Journal on Informatics Visualization* 6, 4 (Dec. 2022), 904. https://doi.org/10.30630/joiv.6.4.1525

[5] Sociedade Brasileira de Computação. 2022. Sobre a SBC. https://www.sbc.org.br/institucional-3/sobre.

[6] Oscar Dieste, Anna Grimán, and Natalia Juristo. 2009. Developing search strategies for detecting relevant experiments. *Empirical Software Engineering* 14, 5 (2009), 513–539. https://doi.org/10.1007/s10664-008-9091-7

[7] Sandra Fabbri, Cleiton Silva, Elis Hernandes, Fábio Octaviano, André Di Thommazo, and Anderson Belgamo. 2016. Improvements in the StArt Tool to Better Support the Systematic Review Process. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering* (Limerick, Ireland) *(EASE '16)*. Association for Computing Machinery, New York, NY, USA, Article 21, 5 pages. https://doi.org/10.1145/2915970.2916013

[8] Katia Romero Felizardo, Elisa Yumi Nakagawa, Sandra Camargo Pinto Ferraz Fabbri, and Fabiano Cutigi Ferrari. 2017. *Revisão sistemática da literatura em engenharia de software: Teoria e prática* (1 ed.). Elsevier, Rio de Janeiro.

[9] Tatiana Engel Gerhardt and Denise Tolfo Silveira (Eds.). 2009. *Métodos de Pesquisa.* Editora da UFRGS, Porto Alegre. 120 pages. Coordenado pela Universidade Aberta do Brasil – UAB/UFRGS e pelo Curso de Graduação Tecnológica – Planejamento e Gestão para o Desenvolvimento Rural da SEAD/UFRGS.

[10] Daniel Glez-Peña, Anália Lourenço, Hugo López-Fernández, Miguel Reboiro-Jato, and Florentino Fdez-Riverola. 2013. Web scraping technologies in an API world. *Briefings in Bioinformatics* 15, 5 (04 2013), 788–797. https://doi.org/10.1093/bib/bbt026

[11] Rohmat Gunawan, Alam Rahmatulloh, Irfan Darmawan, and Firman Firdaus. 2019/03. Comparison of Web Scraping Techniques : Regular Expression, HTML DOM and Xpath. In *Proceedings of the 2018 International Conference on Industrial Enterprise and System Engineering (IcoIESE 2018)*. Atlantis Press, 283–287. https://doi.org/10.2991/icoiese-18.2019.50

[12] Atul Gupta and Pankaj Jalote. 2007. An Experimental Evaluation of the Effectiveness and Efficiency of the Test Driven Development. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*. IEEE. https://doi.org/10.1109/esem.2007.41

[13] Barbara Kitchenham and Stuart Charters. 2007. *Guidelines for performing Systematic Literature Reviews in Software Engineering – Version 2.3.* Technical Report. Keele University and University of Durham, Keele/Staffs and Durham, UK. EBSE-2007-01.

[14] Barbara Kitchenham, O. Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman. 2009. Systematic literature reviews in software engineering – A systematic literature review. *Information and Software Technology* 51, 1 (2009), 7–15. https://doi.org/10.1016/j.infsof.2008.09.009

[15] Harshit Nigam and Prantik Biswas. 2021. Web Scraping: From Tools to Related Legislation and Implementation Using Python. 149–164. https://doi.org/10.1007/978-981-15-9651-3_13

[16] He Zhang and Muhammad Ali Babar. 2013. Systematic reviews in software engineering: An empirical investigation. *Information and Software Technology* 55, 7 (2013), 1341–1354. https://doi.org/10.1016/j.infsof.2012.09.008

[15] https://gitlab.com/nszchagas/pySol

[16] https://pypi.org/