

# Betty's (Re)Search Engine

A client-based search engine for research software stored in repositories.

Vasily Seibert  <sup>1</sup>

Andreas Rausch  <sup>1</sup>


Stefan Wittek  <sup>1</sup>

1. Institute for Software and Systems Engineering, TU Clausthal, Clausthal-Zellerfeld.

**Date Received:**

2023-02-15

**Licenses:**

This article is licensed under: 

**Keywords:**

Inggrid, Data, research data management, research software

**Data availability:**

Data can be found here:

[https://github.com/VasilySeibert/cascading\\_search\\_ecology\\_eval](https://github.com/VasilySeibert/cascading_search_ecology_eval)

**Software availability:**

Software can be found here: <http://github.com/VasilySeibert/BettysReSearchEngine>

**Abstract.** Promoting research, without providing the source code that was used to conduct the research, means a greater effort for every researcher down the line. Existing solutions that aim to make research software FAIR [1], fail to provide a wholesome solution, for they do not sufficiently consider already existing research software stored on platforms like GitHub or organizational GitLabs. We therefore present Betty's research engine, a client-based implementation of a cascading search process, that first finds research software stored on platforms like GitHub and then links them to corresponding publications or entries in third party databases. We evaluated 400 random search results from the domain of ecology and found that 345 out of 400 repositories made a reference to a corresponding publication / entry in third party database and therefore clearly indicating the potential of the cascading search. Betty's research engine is live and openly available under this URL: <http://nfdi4ing.rz-housing.tu-clausthal.de/>

## 1 Introduction

2 Findability and accessibility of research related software is crucial for every researcher who aims  
3 to (i) fully understand and (ii) reproduce software related publications as well as (iii) benchmark  
4 the own software related research to other solutions. However, finding fitting research software  
5 is not a trivial task. It is not obligatory for most authors of software related publications to  
6 provide a link to corresponding software repositories where the research software would then be  
7 stored.

8 [2] annotated a stratified sample of a collection of software mentions, extracted from the CORD  
9 19 Dataset [3] of open access publications. Their goal was to understand the status of software  
10 citation practices. Out of 295.609 software mentions in the Softcite-CORD-19 Dataset 50 % are  
11 solely names of the software without further details, 35 % provide a version, 21 % mention  
12 their publisher, and 9 % have a URL given in the Text. The authors of [4] reviewed the state  
13 of code availability in ecology using a random sample of 346 nonmolecular articles published  
14 between 2015 and 2019 under mandatory or encouraged code sharing policies. Their results  
15 show, that only 27 % of eligible articles were accompanied by code, although the percentage of  
16 ecological journals with mandatory or encouraged code sharing policies increased from 15 %  
17 (2015) to 75% (2020).

18 There are existing approaches, that aim to make research software FAIR. They can be broadly  
19 categorized as:

- 20 • Community driven approaches ( e.g. Papers with Code [5])
- 21 • Technology driven approaches ( e.g. Zenodo [6], Citation File [7])
- 22 • Standards and policies ( e.g. FAIR4RS [8])

23 Papers with Code [5] is an example of a community driven approach to make research software  
24 more findable and accessible. It offers references to papers from the field of mathematics, physics,  
25 machine learning, astronomy and computer science as well as their corresponding data- and  
26 software repositories. Generation of new content as well as quality checking is done by a user  
27 community.

28 Zenodo [6] and Citation File [7] are examples of technology driven approaches. Zenodo is a  
29 platform that allows users to upload research related digital artifacts (such as research software)  
30 and generate a DOI (Digital Object Identifier) for them (therefore making them citable). If  
31 the digital artifact is also stored on GitHub, Zenodo offers a Zenodo-Badge, a small image file  
32 that can be integrated into the README to make the Zenodo DOI more visible. The citation  
33 file format [7] is a file format that is human and machine readable. If a citation file is placed  
34 in a Github Repository it unlocks the "cite this repository" function on GitHub, which further  
35 increases visibility and the probability of getting cited.

36 Metadata standards ( e.g. FAIR4RS [8]) and consequently policies (because someone has to  
37 enforce these standards) are designed to uniform the way research is documented and archived,  
38 with the objective of making it FAIR. Findability according to [8] can be achieved if:

- 39 1. the software is assigned a globally unique and persistent identifier,
- 40 2. the software is described with rich metadata,
- 41 3. the metadata explicitly includes the identifier of the software it describes
- 42 4. the Metadata is searchable and indexable

43 All of the described approaches rely on researchers to explicitly add additional information to  
44 their publications and therefore none of the approaches can be regarded as a wholesome solution  
45 for the problems described in [4] and [2] because of the following reasons:

- 46 • Finding existing research software is more important than the chance of finding (a fraction)  
47 of future research software.
- 48 • The effort for researchers to adapt their existing software repositories to a new database,  
49 technology, metadata standard is considerable and therefore a further obstacle.

50 For the above reasons there is a need for searching **already existing research software reposi-**  
51 **ries** (stored on GitHub [9] or GitLab [10]) automatically and then linking them to corresponding  
52 publications or databases.

53 Such a *cascading search* would (i) initiate the search process by retrieving a list of references  
54 and metadata that refer to software repositories. If a given software repository contains any  
55 identifiers that can be connected to a corresponding publication or the listing of that software

56 repository in a third-party database, then (ii) the search engine would access and retrieve that  
57 information. The result (iii) would be a list of software repositories, enriched with additional  
58 metadata and therefore sortable by the user's preferences.

59 From a technical perspective this cascading search engine faces a variety of challenges in terms  
60 of privacy, networking, individualization, reliability, scalability and the complexity that arise  
61 from the use of multiple APIs. A user must be able to:

- 62 • understand the search process and the logic behind it,
- 63 • perform the described cascading search fully automatically,
- 64 • perform the described cascading search in an acceptable amount of time,
- 65 • be assured that his/her search is private and not being tracked,
- 66 • access sources of information that are commercial or limited to members of an organisation,
- 67 • extend the cascading search with further databases.

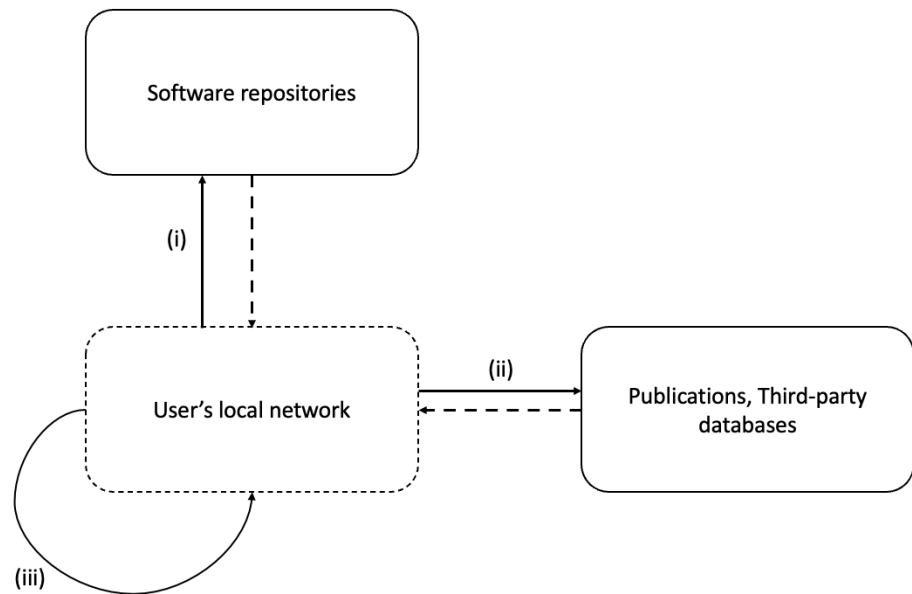
68 In this paper we reiterate the problem of not findable, accessible research software and show  
69 that existing approaches are unable to provide a wholesome solution. We motivate the need  
70 for searching existing research software and linking them to publications / database entries  
71 afterwards (this is what we call cascading search). We formulate top level requirements and  
72 elaborate how we meet them through our architectural design and our choice of technology. We  
73 evaluate our results by analyzing 400 repositories and reflect these results in a discussion. We  
74 then conclude our work.

## 75 **2 Cascading Search**

76 [4] and [2] focused on finding research software by searching for a reference in the publications.  
77 We introduce the cascading search, a search process according to which, we (i) initiate the search  
78 by retrieving a list of references and metadata that refer to software repositories (from a platform  
79 like GitHub). If a given software repository contains any identifiers that can be connected to a  
80 corresponding publication or the listing of that software repository in a third-party database, then  
81 (ii) the search engine will access and retrieve that information. The result (iii) of such a cascading  
82 search is a list of software repositories, enriched with additional metadata and therefore sortable  
83 by the user's preferences.

84 We find (i) and retrieve references to software repositories from GitHub by using the GitHub  
85 REST API. For a given search query, GitHub will find repositories that can be associated with  
86 that search query (e. g. the occurrence of that term in the name of the repository). GitHub also  
87 allows the user of the REST API to modify the search query in a way that makes adjustments  
88 to GitHub's internal search process. For example, the search query "seismology doi in:readme"  
89 would return a list of references to repositories, that can be associated with the term "seismology"  
90 and that contain the word "doi" in their README files. After retrieving a list of references,  
91 we iterate through that list to collect more information on every single repository. Among the  
92 collected information is the README file as well as the name of every other file in the main  
93 directory. We then search (ii) for any identifiers to a corresponding publication or the entry of

94 that repository in a third party database. Searching for unique identifiers is a quality determining  
 95 factor for the cascading search, since this information is most reliable for performing further  
 96 search on that repository. As the results from the evaluation will show, references turn out to  
 97 be highly heterogenous. We developed rules for identifying a *target reference* (a reference that  
 98 refers to a corresponding publication or entry in a third party database, the counterpart of a target  
 99 reference would be a *normal reference* that refers to any literature that was used to conduct or  
 100 support the research).



**Figure 1:** The cascading search (i) initiates the search process by retrieving a list of references and metadata that refer to software repositories. If a given software repository contains any identifiers that can be connected to a corresponding publication or the listing of that software repository in a third-party database, then (ii) the search engine will access and retrieve that information. The result (iii) is a list of software repositories, enriched with additional metadata and therefore sortable by the user's preferences.

101 The following rules were created with the the computational cost and number of required API  
 102 calls in mind.

- 103 • If the repository contains a citation file [7], then the reference in that file is considered a  
 104 target reference
- 105 • If the repository contains a Zenodo image file [6] then the DOI is considered a target  
 106 reference
- 107 • If the repository contains a Zenodo DOI in plain text and the data on Zenodo is similar  
 108 to the name of the repository, we consider that reference as the target reference (only  
 109 checking the first Zenodo DOI)

110 After retrieving the target reference of a repository, we search that reference on further platforms.  
 111 Current platforms include: Zenodo [6], Open Alex [11], DataCite [12] and Open citations [13].

112 The information that we retrieve from the cascading search (iii) is added to our data model.

113 With the cascading search, we enriched the repositories with additional metadata that would  
114 not be available through a search on GitHub alone. The user can now, sort the repositories  
115 according to personal preferences and interests (like the number of citations a repository holds).  
116 The repository also holds information about what reference was considered the target reference.  
117 With that, we meet the requirement of making the cascading search understandable.

### 118 3 Client-based search engine

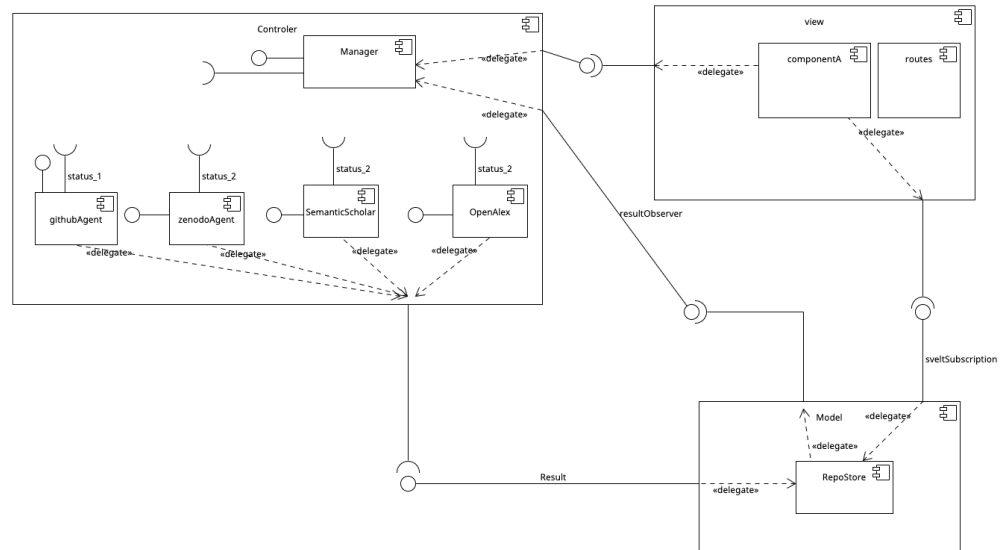
119 The cascading search faces a variety of challenges. A consequence of using multiple APIs is that  
120 multiple rate limitations have to be considered (one or more limitation rule per API). Since rate  
121 limitations are in many cases tracked through the IP Address that makes the request, a centralized  
122 backend is not possible. A centralized backend would also mean that the user could not call  
123 services and databases with restricted access (commercial or organizational).

124 To face the described challenges, Betty's research engine is written solely in languages that can  
125 run in the browser. We send the user everything needed to perform the cascading search from  
126 the local machine. For that reason, the user is required to provide own credentials, since it is the  
127 user that utilizes the different APIs. No communication is being recorded, the credentials are  
128 stored on the local machine, so the user enjoys the maximum amount of privacy that is possible.

129 The architectural design follows a MVC (Model View Controller) pattern (See Fig 2.). This  
130 means that the user interface (View) is strictly divided from the logic (Controller) and the database  
131 (Model) of the system. The user initiates the cascading search from the view. Inside the controller  
132 a *manager* holds knowledge of all available *agents*. An agent is a concrete implementation of  
133 how an API is utilized. This includes making the API call, receiving the returned information,  
134 processing it and passing that information through a defined interface to the model. By thinking  
135 in terms of agents, the complexity that arises from using multiple APIs is capsuled in a defined  
136 container. This way, the manager can orchestrate the agents by a generalized logic (and doesn't  
137 need to implement individual exceptions and error handlings). The retrieved information is  
138 stored in the database according to a defined data model. If new information is added to the  
139 model, the manager gets notified by an update mechanism. So every time a data instance is added  
140 or modified, the manager gets notified and can instantiate new agents that use that information.

141 Therefore the cascading search can be seen as a circular flow of data. The manager instantiates  
142 a GitHub agent, who performs a call to the GitHub REST API, retrieves that information and  
143 processes it before calling a model interface and adding the repositories to the database. The  
144 manager gets notified about new repositories being added, and instantiates Zenodo, DataCite,  
145 OpenCitations etc. agents that use the newly added information to perform searches on their  
146 own, before calling another model interface and adding the information.

147 Through the presented architectural design, the cascading search can now be conducted fully  
148 automatically and since there is no direct dependency between the agents, they can run in parallel.



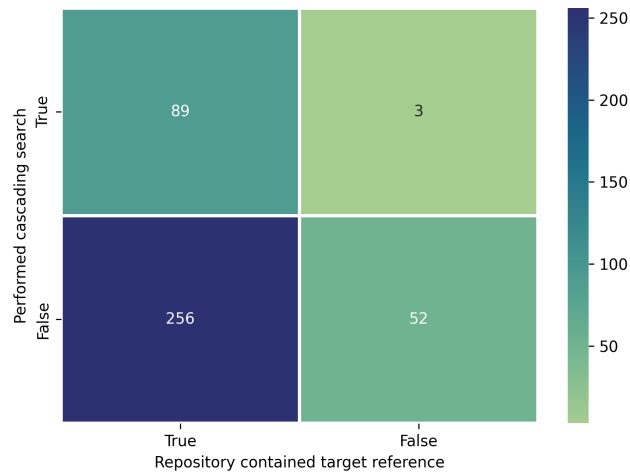
**Figure 2:** Betty's research engine is an implementation of the cascading search that faces the challenges of individual rate limitations and restricted access by a client based design. By handling the complexity that arises from the use of several APIs in agents, the cascading search can be performed in parallel processes while maintaining a sound, coherent structure.

#### 149 4 Evaluation

150 [4] analyzed 346 publications, published in ecological journals. They found, that between 2015  
 151 and 2019 under mandatory or encouraged code sharing policies, only 27% of eligible articles were  
 152 accompanied by code. To evaluate Betty's re search engine, we performed a cascading search  
 153 for ecology related research software. Out of 2513 software repositories that were available  
 154 on GitHub, we randomly choose 400 for detailed analysis (all data and code that was used to  
 155 perform this analysis can be found at [https://github.com/VasilySeibert/cascading\\_](https://github.com/VasilySeibert/cascading_search_ecology_eval)  
 156 [search\\_ecology\\_eval](https://github.com/VasilySeibert/cascading_search_ecology_eval) ). We found that 345 out of 400 (86,25%) repositories referred to a  
 157 corresponding publication or the listing of the repository in a third party database.

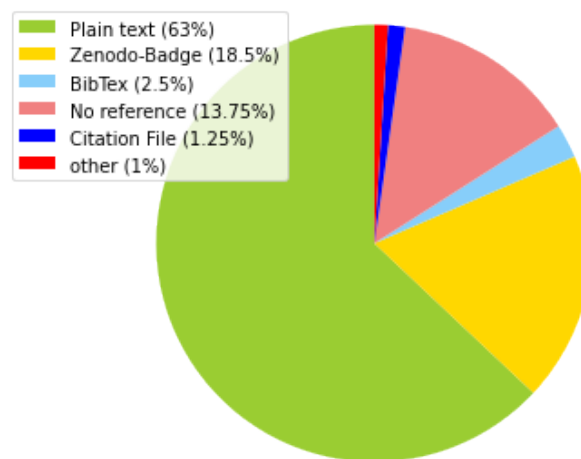
158 With the introduced rules, we were able to correctly identify 89 (22,25%) target references (true  
 159 positives). 256 (64%) repositories had a target reference, but weren't identified (false negatives).  
 160 52 (13%) repositories did not refer to corresponding research and were correctly disregarded (true  
 161 negatives). 3 (0.75%) repositories were falsely used for the cascading search (false positives).

162 Of further interest is how a repository refers to corresponding publications / entries in a third  
 163 party database. We found that the majority of references (252, 63%) were mentions in plain  
 164 text. 74 (18,5%) repositories used a Zenodo image file, 5 (1,25%) repositories had a citation  
 165 file placed in their main directory. 10 (2,5%) repositories had a BibTEX text element in their  
 166 README.



**Figure 3:** The introduced rules yield an accuracy of 35,25%. (all data and code that was used to perform this analysis can be found at

[https://github.com/VasilySeibert/cascading\\_search\\_ecology\\_eval](https://github.com/VasilySeibert/cascading_search_ecology_eval) )



**Figure 4:** We found that the majority of references (252, 63%) were mentions in plain text. 74 (18,5%) repositories used a Zenodo image file, 5 (1,25%) repositories had a citation file placed in their main directory. 10 (2,5%) repositories had a BibTEX text element in their README.

## 167 5 Discussion

168 **Does the cascading search come up for the shortcomings of current approaches?** As the  
 169 results from the evaluation chapter clearly indicate, the cascading search has the potential to  
 170 compensate for the shortcomings of the approaches which were introduced in the introduction.  
 171 Existing research software can be found on GitHub and furthermore connected to their respec-  
 172 tive publications. The cascading search utilize technological driven approaches, in fact these  
 173 approaches are most reliable when it comes to identifying target references, for their use of  
 174 structured text elements. Supported by these results, the cascading search is a valid and useful

175 alternative to community- and technological driven approaches as well as metadata standards  
176 and policies.

177 **How effective is Betty's research engine?** With an accuracy of merely 35,25%, Betty's re  
178 search engine has plenty of room for improvement. However, in relation to the 400 analyzed  
179 repositories, only 89 (22,25%) repositories made use of structured text elements Zenodo-Badges,  
180 Citation Files, BibTEX. A rule-based approach might not be well suited for this type of textual  
181 data. Since 63% of all references are written in plain, unstructured text, a data driven approach  
182 might be more promising.

183 **In what use cases, can Betty's re search engine serve best?** Betty's research engine can  
184 help researcher who aim to (i) fully understand and (ii) reproduce software related publications.  
185 Unlike Papers with Code, Betty's research engine does not support any categorisation of the  
186 found research. The search results don't have any information about ongoing benchmarks or  
187 the state of the art for that matter. Enabling users to (iii) benchmark the own software related  
188 research to other solutions would require further work.

## 189 6 Conclusion

190 In this paper we reiterated the problem of not findable, accessible research software and pointed  
191 out how current approaches don't provide a satisfying solution, for they are not sufficiently  
192 consider already existing research software repositories stored on platforms like GitHub or  
193 organizational GitLabs. We proposed the cascading search, a novel approach that has the  
194 potential to compensate for the shortcomings of current approaches by searching for software  
195 repositories first and then linking them to their corresponding publications. We presented  
196 Betty's research engine, an implementation of the cascading search that faces the challenges  
197 of individual rate limitations and restricted access by a client based design. We elaborated  
198 how the cascading search can be performed in parallel processes while maintaining a sound,  
199 coherent structure. We analyzed 400 random search results from the domain of ecology and  
200 found that (a) the cascading search is a valid, useful alternative to current approaches, (b) the  
201 rule-based approaches presented in this paper yield an accuracy of 35,25% and (c) the majority  
202 of references (63%) to corresponding publications / entries in third party databases are written  
203 in plain, unstructured text. Reflecting our results we found that a rule-based approach (like the  
204 one we use now) might not be well suited for the problem at hand and a data-driven approach  
205 might be more promising for identifying references to corresponding publications. Compared  
206 to community driven approaches like Papers with Code we reflected on the importance of  
207 categorisation and benchmarking of research software and concluded that further work on Betty's  
208 research engine is necessary.



## 209 7 Acknowledgements

210 This work would not have been possible without the financial support from the German Research  
211 Foundation (DFG) - project number 442146713.

212 We are grateful to all of those with whom we had the pleasure to work during this and other  
213 related projects.

## 214 8 Roles and contributions

215 **Vasily Seibert:** Conceptualization, Writing – original draft

216 **Andreas Rausch:** Conceptualization, Writing – original draft

217 **Stefan Wittek:** Conceptualization, Writing – original draft

## 218 References

- 219 [1] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, *et al.*, “The fair guiding principles for  
220 scientific data management and stewardship,” *Scientific data*, vol. 3, no. 1, pp. 1–9, 2016.
- 221 [2] C. Du, J. Cohoon, P. Lopez, and J. Howison, “Understanding progress in software citation:  
222 A study of software citation in the cord-19 corpus,” *PeerJ Computer Science*, vol. 8, e1022,  
223 2022.
- 224 [3] L. L. Wang, K. Lo, Y. Chandrasekhar, *et al.*, “Cord-19: The covid-19 open research  
225 dataset,” *ArXiv*, 2020.
- 226 [4] A. Culina, I. van den Berg, S. Evans, and A. Sánchez-Tójar, “Low availability of code in  
227 ecology: A call for urgent action,” *PLoS Biology*, vol. 18, no. 7, e3000763, 2020.
- 228 [5] *Papers with code*. [Online]. Available: <https://paperswithcode.com/>.
- 229 [6] *Zenodo*. [Online]. Available: <https://zenodo.org/>.
- 230 [7] S. Druskat, N. C. Hong, R. Haines, and J. Baker, “Citation file format (cff)-specifications,”  
231 *Zenodo. data*, 2018.
- 232 [8] M. Barker, N. P. Chue Hong, D. S. Katz, *et al.*, “Introducing the fair principles for research  
233 software,” *Scientific Data*, vol. 9, no. 1, pp. 1–6, 2022.
- 234 [9] *Github*. [Online]. Available: <https://github.com>.
- 235 [10] *Gitlab*. [Online]. Available: <https://about.gitlab.com>.
- 236 [11] *Open alex*. [Online]. Available: <https://openalex.org>.
- 237 [12] *Data cite*. [Online]. Available: <https://datacite.org>.
- 238 [13] *Open citations*. [Online]. Available: <https://opencitations.net>.