Finding treasure in an embarassment of riches: toward a more effective software catalog

Michael Hucka mhucka@caltech.edu
Matthew J. Graham mjg@caltech.edu
John C. Doyle Caltech, Pasadena, CA

What is the problem?

There is a wealth of software now available, and more is being created. Yet, **finding software for a given purpose remains surprisingly difficult**. Few resources exist to help users discover alternatives or understand the differences between them.

What's wrong with Googling to find software?

- *Must pick good terms.* Difficult to do, worse for nonnative English speakers or field outsiders.
- *Too many results.* The relevant software can be buried and difficult to find.
- *Hard to tell differences between results.* Google results don't show software features—you must investigate each result & compare them yourself.

What's wrong with asking your colleagues?

- *Unknown unknowns.* Most people don't know about all possible options or how they compare.
- *Answers are potentially biased.* What people *do* know may be out of date or incorrect.

What's wrong with looking in the literature?

- *Publication lag.* Information is often out of date, potentially leading you down the wrong path.
- *Not all software has an associated paper or is mentioned in other people's papers.*
- *Doing thorough research is time-consuming.*

What's wrong with asking on social media?

- *Same problems as asking colleagues.*
- *Cannot predict when (or if) you get an answer.*

Does it really matter?

Lacking better info, people often don't use the best or most appropriate software, and sometimes unwittingly recreate existing tools. **Time and money are wasted, reproducibility suffers, and funding agencies get poor return on investment**.

Is there a better alternative?

A **comprehensive software index** could make it easier to find software by providing pertinent results organized with contextual information and specific details about each software resource. This would help users find software more effectively and compare alternatives more systematically.

Past cataloguing efforts have failed either because they were simplistic (thus providing incomplete, misleading or unhelpful content) or relied on humans. Humans don't scale—**automation is the only feasible way** of cataloging the vast and ever-growing number of constantly-evolving software applications, libraries and other sources.

Happily, the growing trend of putting software in repositories such as GitHub opens new avenues.

What is our goal?

We seek to *automate a continually-updated index of software* for users, particularly scientists. This will require overcoming two challenges: *effective software discovery* and *accurate software characterization*.

Our research question is: **can an ontology-based approach to software characterization produce results that humans find acceptable**?

What is our approach?

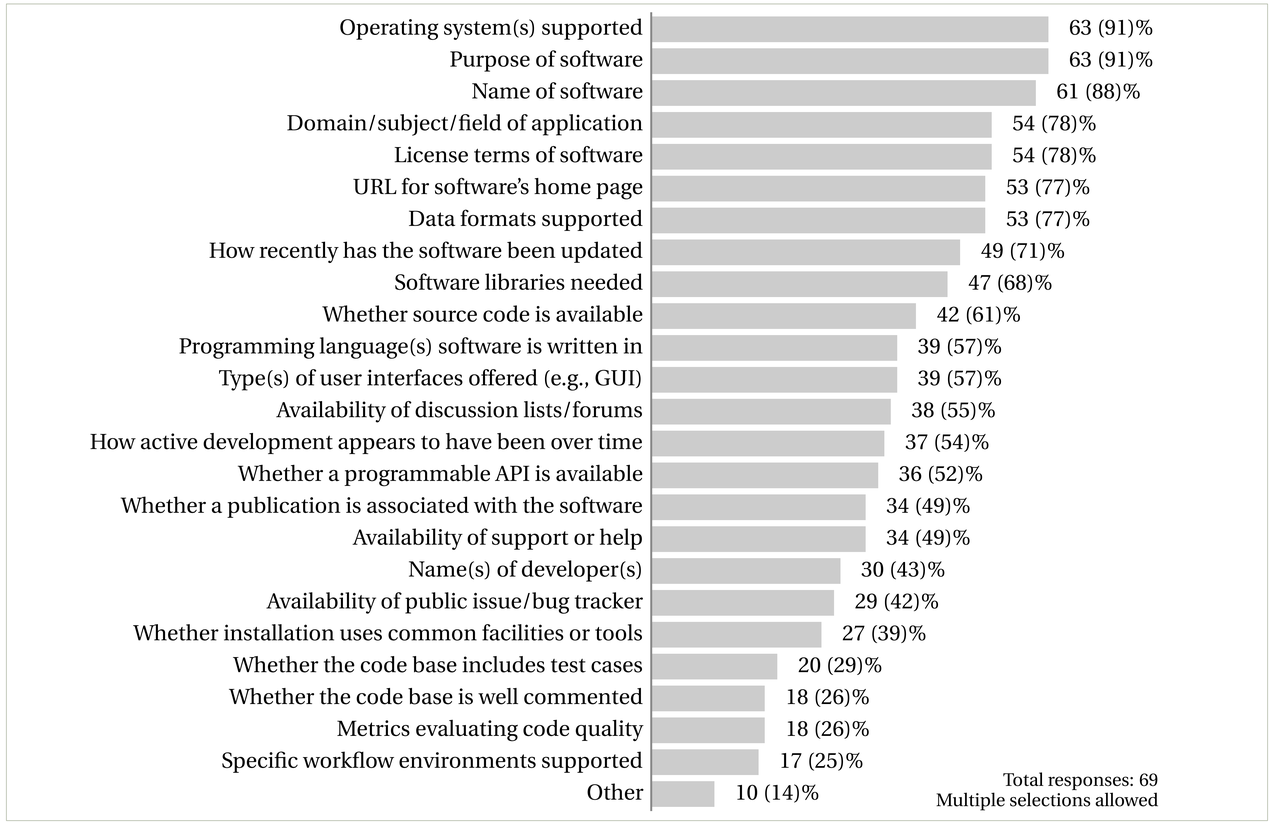
1. Infer software characteristics via ontology-based, hierarchical multi-label classification
2. Apply the methods to SourceForge and GitHub projects to characterize software
3. Leverage the ontology to improve search
4. Provide a demo interface and evaluate the results

What do people want?

We ran a survey in 2015 to understand how people find software and what kind of info would be useful in a catalog. Our 22-question survey went to mailing lists in astronomy and systems biology. We received 69 responses. *Demographics:* people's top 3 work responsibilities were software development (82%), software architecture (54%), and project management (46%); people had 1–45 yrs of software development experience, in small- to medium-sized teams.

What information should be captured in a catalog?

We asked "*Suppose it were possible to create a public, searchable catalog or index of software, one that would record information about software of all kinds found anywhere. What kind of info would you find most useful to include for each entry in such a catalog or index?*" The graph below summarizes the results.



How does our project differ from other related work?

Other groups have explored methods to index software, and text-based source code analysis using machine learning is not new. However, past accuracies have been modest, making the methods unsuited for reliable software cataloging, and the project goals have also been different. Crucially, past work has not explored using semantic knowledge to assist the classification process and organize the results for human use.

- Linares-Vásquez, et al. (2014). On using machine learning to automatically classify software applications into domain categories. *Empirical Software Engineering*, 19(3).
- Linstead, et al. (2008). Mining internet-scale software repositories. *Advances in Neural Information Processing Systems*, 20.
- McMillan, et al. (2012). Exemplar: A source code search engine for finding highly relevant applications. *IEEE Trans Soft. Eng.*, 38(5).
- Ugurel, et al. (2002). What's the code? Automatic classification of source code archives. In *KDD '02*.

How are we proceeding?

After acquiring and setting up server hardware, we created a basic crawling system for GitHub. We generated an index of over 25,000,000 public repositories (e.g., names, descriptions, URLs, etc.), obtained programming language info for 15,000,000 of those, README files for 2,000,000, and finally, downloaded full repository copies for 330,000.

We are currently working on the next three of several steps that will culminate in the creation of a demonstration system: CASICS, the *Comprehensive and Automated Software Inventory Creation System*.

1. Exploring hierarchical topic inference

Developing a suitable ontology will be a challenge, though starting points exist (see #2). To help, we are exploring inference methods to infer topic hierarchies.

We first applied a probabilistic generative model to 1,000,000 one-line project descriptions in GitHub, using a hierarchical Dirichlet process (a nonparametric extension of LDA in which the number of topics is learned from the data). We employed a maximal spanning tree approach to infer a hierarchy from the generative model using corpus-derived probabilities for topics. It failed to produce a robust topic hierarchy. Next: try all 25,000,000 one-line descriptions and also project README files.

2. Extending existing ontologies

In terms of existing ontologies, SWO, the *Software Ontology* (`http://theswo.sourceforge.net`), is the closest to our needs. It provides terms for topics, data formats, software licenses, and more. We are working with the SWO developers to extend it.

With colleagues at NIST, we are also exploring the use of the NIST *Dictionary of Algorithms and Data Structures* (`https://xlinux.nist.gov/dads/`).

3. Writing feature extractors

Extracting features from an input is the first step in classification. The input here is the set of files in a project. For source code, we expect an *identifier expansion* step (e.g., to expand abbreviations) will improve classification. We are implementing algorithms published by others: TRIST (Guerrouj et al, 2012) and GenTest (Lawrie et al., 2006).

Other developments are forthcoming . . .

Acknowledgments

This work is funded by NSF EAGER #1533792.

Caltech