Towards an automated cataloging system for open-source software: the CASICS project

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, more so for non-native 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- *Answers are potentially biased.* What people *do* know may be out of date or incorrect.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- *Publication lag.* Information is often out of date, potentially leading you down the wrong path.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.*

Why does it 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.

What is our goal?

We are developing **CASICS**, the ***Comprehensive and Automated Software Inventory Creation System***. Our goals are to develop a proof of concept:

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

What information do people want?

We performed a survey of users and developers. It revealed the information people would like to see in a catalog. The dozen most desirable bits of info are:

- *Operating system(s) supported*
- *Purpose of software*
- *Name of software*
- *License terms*
- *Domain/subject/field of application*
- *URL for home page*
- *Data formats supported*
- *How recently software was updated*
- *Software libraries needed*
- *Whether source code is available*
- *Programming languages used in implementation*
- *Type(s) of user interfaces offered*

Gathering and inferring this information from repositories automatically is the goal of CASICS.

How can software be analyzed?

A number of software features can be inferred by recognizing specific entities in the project sources.

- *File features.* File extensions (.py, .java, etc), specific build system files (e.g., setup.py), and others strongly imply certain software features.
- *Project descriptions and README files.* When available, they can used as input to text mining to infer topics and other aspects of a project.
- *Source code.* The code comments, function/class identifiers, text strings, documentation strings, imported libraries, and other elements can be used as input to **text-based classification**.

However, entity recognition can only help discover *some* features. Inferring subjects and topics needs a more powerful approach based on machine learning.

How are we doing classification?

The features used by the classifiers are **elements extracted from source code** via language-aware parsers (for function names, class names, variable names, library names, doc strings, text strings, comments) and **text extracted from document files** via converters for plain text, Markdown, HTML, and other file types.

An advance compared to prior work is using **identifier expansion** to convert typical function and variable identifiers (e.g., "readfromdb") into more meaningful strings (e.g. "read from database").

These features are used as input into **supervised, hierarchical multi-label classifiers** to label software with respect to predefined ontologies.

Which ontologies are we using?

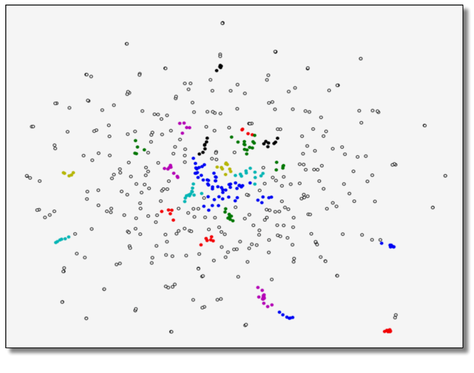
We began with SWO, the *Software Ontology* (`http://theswo.sourceforge.net`). It provides terms for topics, data formats, software licenses, and more.

However, the huge breadth of topics found in projects in GitHub has made us look for a larger subject ontology. We settled on the **Library of Congress Subject Headings (LCSH).**

In addition to using LCSH, we developed new ontologies to characterize capabilities including:

- *Interfaces*: user interfaces, programming interfaces, and network interfaces offered by a software system.
- Software *kind*: whether something is user software, library software, server software, etc.

Clustering the text extracted from project descriptions and README files using t-SNE and then running DBSCAN to identify clusters is helping us derive a leaner topic hierarchy.

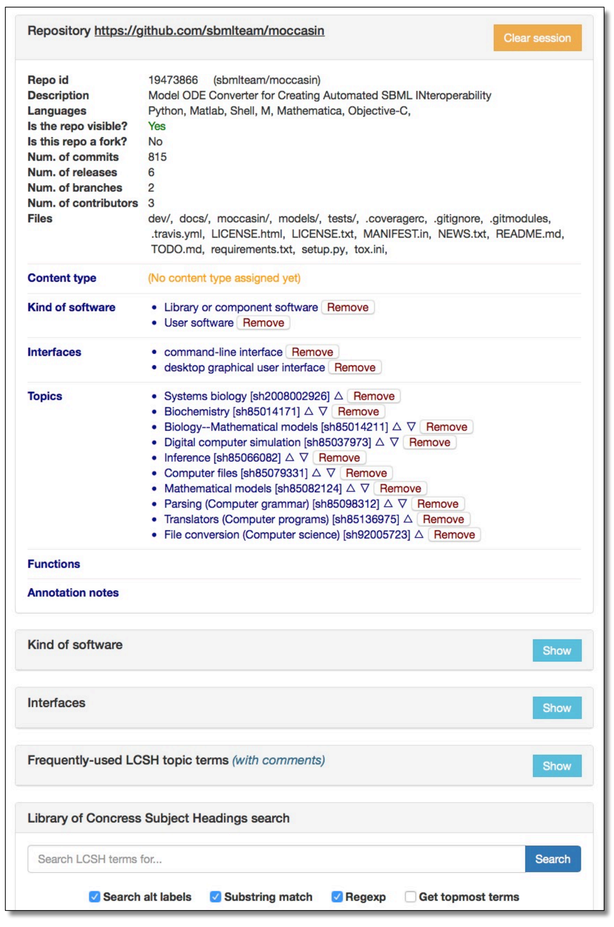


How are we labeling software projects?

We developed a repository-crawling system in Python and a MongoDB-based database, which we use as a local index and summary of over 65,000,000 public repositories in GitHub.

We created a database system for storing and navigating the graph of LCSH terms, and created an annotation system to let us pick terms.

We created a custom annotation interface using Node.js, JavaScript, and Bootstrap. We use this to select ontology terms to describe software projects.



How will we build a catalog?

We are using our manually-annotated database of software to train hierarchical multi-label classification algorithms. We are exploring two in particular (Chained Path Evaluation, and HiBLADE).

With trained classifiers in hand, we will be able to categorize and infer properties about new/unseen software projects, and use this to generate a hierarchically-organized software catalog.

Caltech This work is funded by NSF EAGER #1533792. NSF