# Parallel Systematic Similarity Analysis of Software Repositories

Nicholas Beydler, Mark Burgess, Dao McGill, Raven Quiddaoen
Sponsors: Carlos Paradis & Rick Kazman

*github.com/sailuh/kaiaulu*

ICS 496, Fall 2024 Information and Computer Sciences Department - University of Hawai'i at Mānoa

## Introduction

Kaiāulu is an R package aimed at analyzing the dynamics of growing software development communities and the collaborative artifacts (gitlog, mailing list, files, etc.) they interact with.

## Accomplishments

- Reconfigured folder organization for project configuration files at the project level; and created a graphic of the suggested template for folder hierarchy.
- Implemented function to create folder organization on user's local device.
- Expanded on Github API endpoints; added new downloaders and refreshers.
- Created an interface to extract dependencies from source code using Scitools Understand
- Extended testing and documentation of existing tools which analyze project class files and extract Gang of Four Motifs.
- OpenHub API interfacing functions to facilitate selecting open-source projects for analysis.
- Centralized process for gathering details from project configuration files and decoupled notebooks from direct variable assignments in the configuration files.
- Implemented download and refresh of two mailing list data archives (mod mbox and pipermail).
- Developed a Python script for the parallelization of Kaiāulu functions via exec scripts.
- Extended syntax extraction capabilities by implementing functions for various elements (e.g. classes, variables, functions, etc.) leveraging srcML.
- Developed exec scripts to make parsed source code available to other tools.
- Utilized the FastText ML model to generate code embeddings for computing semantic similarities between code snippets.

## Methodology

- Project Management
  - Agile software development
  - GitHub Issues and Pull Requests for documented communication and collaboration
- Roles
  - Nicholas: Technical Lead
  - Mark: Testing Lead
  - Dao: Communications Lead
  - Raven: Meeting Scribe

## Challenges

- Technical understanding of Kaiāulu's framework
- Balancing readability, requirements, and extensibility in writing code
- Operating system restrictions (Kaiāulu has been tested on OS X and Ubuntu)
- Installation and configuration of third party packages

## Learnings

- Defining issues and scope according to specifications, and refining solutions through iterative feedback.
- Developing clear and concise documentation, providing guidance to users potentially unfamiliar with the technology.
- Efficient software configuration management using GitHub improved collaboration, version control, and overall organization during development.
- The importance of clearly defined roles in enhancing teamwork and coordination to meet milestone deadlines.

## Analyzing Projects : : CHEAT SHEET

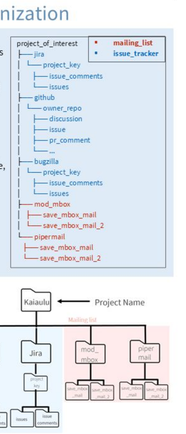
### Getting Started

Kaiāulu recommends a set of steps to analyze projects. These include the way to organize a project's folders and the way the analysis is configured for reproducibility.
Additional features facilitate project selection by different project demographics, reusing parts of Kaiāulu in other tools or server-side for parallelization.

### Folder Organization

A project analysis is organized in sub-folders per data source provenance. A project folder can be initialized by the function: `create_file_directory()`

The file organization is specified in the project configuration file. These, in turn, are used throughout the Notebooks via getter functions in Kaiāulu once the project configuration file is specified.

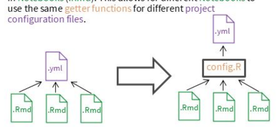






### Accessing Configs from Notebooks

The getter functions in config.R centralize the process of gathering information from project configuration files (.yml) in Notebooks (.Rmd). This allows for different Notebooks to use the same getter functions for different project configuration files.




Without the getter functions from config.R, the Notebooks would require direct variable assignments to the project config information.

With the getter functions from config.R, the Notebooks can use the getter functions to acquire the project config information.

If the project config specification evolve in the future, only the corresponding getter function implementation needs to be updated, instead of all the dependent Notebooks and exec scripts.

### Selecting Projects

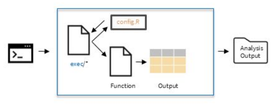
Project selection often involves tedious work in browsing different websites and repositories to identify project demographics, language, issue traceability, or bug labeling.

openhub_project_search.Rmd is a Notebook that showcases how Kaiāulu interfaces with OpenHub API to facilitate selecting open-source projects for studies. This Notebook demonstrates how to use the Kaiāulu OpenHub API interface to search for projects that meet specific criteria, streamlining the process of discovering open-source projects through OpenHub's database.

Once projects for analysis are decided upon, they can be documented as project configuration files, in turn accessed in Notebooks by getter functions.

### Command Line Interface

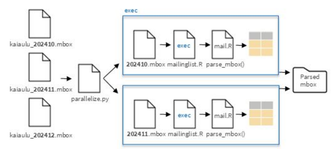
Exec scripts serve as interface between Kaiāulu functions and external tools, allowing a subset of capabilities to be accessed via the command line in other languages.




Exec scripts also utilize getter functions to access project configuration files, and Kaiāulu's API to externalize functionality. Exec scripts expect a combination of both project configuration files as input and optional arguments.

### Parallelization

Through exec scripts, Kaiāulu functions can be called in parallel, enabling concurrent analysis of large projects.




The parallelize.py script can be used with other Kaiāulu exec scripts depending on the analysis needs.

Kaiāulu



## File Similarity by Code Representation : : CHEAT SHEET

### About

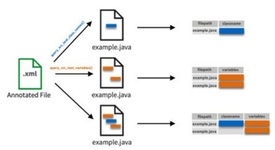
Files can be represented differently depending on how we subset their content, such as functions, class names, variables, or documentation. These representations shape the way files are similar to one another, as selected code snippets emphasize different semantic relationships between files. This cheat sheet describes how to use **Kaiāulu** to represent source code differently for semantic similarity.

### Source Code Representation

The Syntax Extractor in Kaiāulu is used to extract selected content from source code using srcML. The srcML tool is used to annotate source code with descriptive tags that help in identifying code elements.


`annotate_src_text()`: generates an annotated XML file.

XPath queries are used to query the annotated file, and output a table of filenames and the source code element being queried (variable names, function names, file class documentation, etc.)




Query functions: Kaiāulu provides an API for extracting different portions of source code files as structured tables. These representations include variable names, function names, documentation, and more, each capturing specific file elements. Using these tables, we can analyze and transform code into selected representations for further processing.

`query_src_text_class_names()`: outputs a table of with a class names column, and a file paths column.

`query_src_text_namespace()`: outputs a table with a namespaces, and a file paths column.

Relevant Notebook: syntax_extractor.Rmd

### Exec Scripts

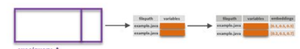
Exec scripts act as interfaces between Kaiāulu functions and external tools, exposing Kaiāulu's capabilities to external sources through the command line. Users can execute scripts for specific queries (e.g., extracting function names or documentation) from the terminal.


src_content_parser.R: serves as the main entry point for querying. It takes a flag specifying the query type (e.g., --class-names, --variables) and calls the associated query function. The annotated file is passed as a parameter, determining the contents to be queried.

query_* scripts: implement the logic for specific types of queries, such as extracting class names or variables. Each query focuses on a particular aspect of the code representation.

### Embeddings

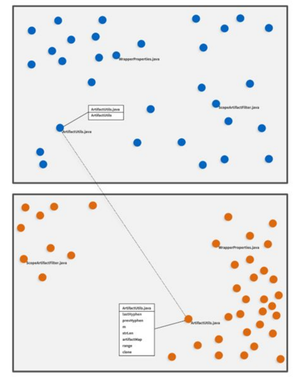
Word embeddings represent text semantics by mapping words or sentences to a numerical representation. The numerical vector representation can be used for semantic similarity comparisons which would be lost if direct word comparison was utilized.




Beyond the selection of a file representation, different models to convert content to embeddings can also be selected. For example, Java-specific FastText embeddings may be better for source code representations, while a file representation containing only code documentation may benefit from natural language embeddings instead of source code. Other language models may also be more appropriate if the source code is not Java. Kaiāulu parameterizes the pipeline.

### UMAP

UMAP transforms high-dimensional embeddings by reducing dimensionality and fitting on a plane, which allows for visualization of the relationships between files.



The diagrams above show how the representation of code elements, such as class name and variables, impacts the relative positions of files in a UMAP projection. Each representation emphasizes different aspects of the code, resulting in varied groupings and relationships.
This variability demonstrates the importance of selecting the appropriate representation for the specific insights required. By choosing different code elements, similarity analysis can be tailored to answer diverse research questions.

Kaiāulu