Knowledge Islands: Visualizing Developers Knowledge Concentration

Otávio Cury
otaviocury@ufpi.edu.br
Federal University of Piauí
Teresina, Piauí, Brazil

Guilherme Avelino
gaa@ufpi.edu.br
Federal University of Piauí
Teresina, Piauí, Brazil

ABSTRACT

Current software development is often a cooperative activity, where different situations can arise that put the existence of a project at risk. One common and extensively studied issue in the software engineering literature is the concentration of a significant portion of knowledge about the source code in a few developers on a team. In this scenario, the departure of one of these key developers could make it impossible to continue the project. This work presents Knowledge Islands, a tool that visualizes the concentration of knowledge in a software repository using a state-of-the-art knowledge model. Key features of Knowledge Islands include user authentication, cloning, and asynchronous analysis of user repositories, identification of the expertise of the team's developers, calculation of the Truck Factor for all folders and source code files, and identification of the main developers and repository files. This open-source tool enables practitioners to analyze GitHub projects, determine where knowledge is concentrated within the development team, and implement measures to maintain project health. The source code of Knowledge Islands is available in a public repository[1], and there is a presentation about the tool in video[2] [3].
Software License: General Public License (GPL)

KEYWORDS

Software repository mining, knowledge concentration, code authorship

1 INTRODUCTION

Managing knowledge distribution among team members is important in software development, particularly in large and geographically dispersed projects. The increasing prevalence of remote work has exacerbated these challenges by limiting direct interactions among team members. Identifying which developers possess specialized knowledge of different source code segments in such environments becomes critical [25]. Knowledge concentration, wherein a few developers hold essential information about the codebase, poses significant risks [5, 19]. The departure of these key developers can result in severe disruptions or even project failure [2]. Consequently, understanding and mitigating knowledge concentration is vital for ensuring project continuity and stability.

Existing techniques for tracking and managing knowledge concentration in software development frequently rely on simplistic metrics, such as the number of commits, number of lines, or the identity of the last modifier, which do not adequately capture the depth of a developer's expertise [11, 26, 27]. Additionally, these techniques fall short of providing comprehensive insights into the expertise distribution among developers. This deficiency hinders project managers from making informed decisions regarding task assignments, such as bug fixing or onboarding new developers. Effective management of knowledge distribution ensures that the team does not rely excessively on any single point of failure, thereby enhancing the project's resilience to personnel changes or abandonment.

To address these limitations, in this paper, we present Knowledge Islands[4]. This open-source tool provides a more comprehensive and insightful approach to managing knowledge concentration in software projects. Knowledge Island goes beyond simplistic metrics by examining a broader range of variables and implementing advanced analytical techniques and more precise models of expertise identification. Knowledge Island provides project managers with better means to monitor and distribute knowledge within their teams by improving the accuracy of expertise identification and applying a Truck Factor algorithm. Additionally, the tool provides a hierarchical visualization of the project knowledge that helps to identify knowledge islands and manage such risks.

This work is organized as follows: Section 2 presents the main concepts implemented in Knowledge Island. Section 3 describes the tool, its key features, and its architecture. Section 4 provides a usage scenario for Knowledge Island. Section 5 discusses the tool's limitations. Section 6 reviews related tools and studies. Finally, Section 7 concludes the work and outlines future directions.

2 BACKGROUND

This section presents the main concepts and algorithms implemented in this study. Section 2.1 discusses knowledge models from the literature in the area. Section 2.1.1 explains the Degree of Expertise (DOE), the knowledge model that we use at Knowledge Islands, and Section 2.2 explains the AVL Truck Factor algorithm implemented in the tool.

2.1 Code Knowledge Models

Source code knowledge models play a crucial role in understanding expertise within software projects, especially in identifying "Knowledge Islands" – areas where knowledge is concentrated in a small number of developers. These models are vital for addressing issues like knowledge loss, developer onboarding, and risk mitigation. By accurately identifying experts, organizations can make informed decisions regarding code maintenance, bug resolution, and project management.

[1]https://github.com/OtavioCury/knowledge-islands
[2]https://youtu.be/5iWxRdx6Dp0
[3]https://doi.org/10.5281/zenodo.11410997

[4]https://github.com/OtavioCury/knowledge-islands

Current research on source code knowledge models has explored several techniques, primarily focusing on data extracted from Version Control Systems (VCS). Some studies are based mainly on information about changes such as the number of commits and who made the last change to identify expertise. Hossen et al. [18] presented an approach that identifies experts in a change request based on who last changed the files. Others count the number of changes made on source code [6, 10, 17]. Other models, such as the one proposed by Sülün et al., use the number of commits in the code artifact and related ones to recommend code reviewers [28].

In addition to the number of changes, other studies also use the number of interactions a developer has with a file. For example, the *Degree of Knowledge* (DOK) model, proposed by Fritz et al., considers both the developer's authorship, the *Degree of Authorship* (DOA), and their interactions with a file, the *Degree of Interest* (DOI) [15]. Although the authors demonstrated that interaction data can enhance expert identification, the computation of *DOI* requires plugins in the development environment, which complicates its usage in large studies.

2.1.1 *Degree of Expertise.*

The *Degree of Expertise* (DOE) is a knowledge model proposed by Cury et al. that uses four variables from the development history of a file to measure a developer's knowledge [13]. Differentiating itself from existing models in the literature, *DOE* combines fine-grained measures of change, authorship, recency of modification, and file characteristics, for greater precision in calculating knowledge.

This model was proposed in a study that used historical data from public and private projects. The *DOE* model demonstrated better performance in both identifying file experts and applying it in Truck Factor algorithm application [12, 13]. Knowledge Islands implements this model in an algorithm to calculate the Truck Factor of software in different levels: repositories, modules, and files. To use this linear model in the Knowledge Islands we used the coefficients empirically found in a related study [12]. Finally, The knowledge of a developer d in the version v of a file f is given by Equation 1.

$$\begin{aligned}\textbf{DOE(d, f(v))} = {} & 5.28223 + 0.23173 \cdot \ln(1 + \textbf{Adds}^{d,f(v)}) \\ & + 0.36151 \cdot (\textbf{FA}^{f}) \\ & - 0.19421 \cdot \ln(1 + \textbf{NumDays}^{d,f(v)}) \\ & - 0.28761 \cdot \ln(\textbf{Size}^{f(v)}) \qquad (1)\end{aligned}$$

where,

- **Adds**: number of lines added by developers d on file f;
- **FA**: 1 if developer d is the creator of the file f, 0 otherwise;
- **NumDays**: Number of days since the last commit of a developer d on file f;
- **Size** Number of lines of code (LOC) of the file f.

In addition to identifying key developers, Knowledge Islands also uses *DOE* to identify important files in the project. The concept is that files that were the focus of major modifications during the project's evolution have a greater significance, as supported by works in the literature[23, 24]. In our implementation, we call *importance score* the sum of the *DOE* of the contributors of a file.

2.2 Truck Factor Algorithm

Truck Factor, also called *Bus Factor*, is a measure that indicates the minimum number of developers who need to leave a software project for it to stall [20]. This metric helps practitioners identify the concentration of knowledge in their projects and has already been the focus of different studies in the software engineering literature [4, 11, 12, 14, 16, 20]. Some studies focused on proposing new ways of estimating the Truck Factor. Of these studies, Avelino's algorithm [4] stands out, highlighted in previous works with the best performance in comparison with two other algorithms [14], and used in studies that validate its results [1, 7, 9].

The Avelino's Truck Factor algorithm follows a greedy approach that relies on an authorship metric to identify the top authors of a system and iteratively estimate the impact of removing the top developers. First, in the original approach, the experts of each file in the project are identified by adopting the *Degree of Authorship* (DOA) model [15]. Then, the algorithm iteratively removes the developer who is the expert in the largest number of files and checks how many files become abandoned, i.e. files without experts after the removal. When more than half of the projects' files have no expert, the algorithm stops, returning the Truck Factor estimation of the number of experts removed. This procedure is represented in Algorithm 1.

Algorithm 1 High-level pseudo-code of the algorithm proposed by Avelino for computing the Truck Factor of a software project.

```
$E \leftarrow getExperts()$
$F \leftarrow getFiles(E)$
$tf \leftarrow 0$
while $E \neq \emptyset$ do
    $coverage \leftarrow getcoverage(F, E)$
    if $coverage \leq 0.5$ then
        break;
    end if
    $E \leftarrow removeTopAuthor()$
    $tf \leftarrow tf + 1$;
end while
return $tf$;
```

In our implementation of this algorithm in the Knowledge Islands tool, we modified the original Avelino algorithm by replacing *DOA* with *DOE* (Section 2.1.1) as the expert identification model. This same modification was made in a previous study using public and private projects, resulting in increased accuracy of Avelino's algorithm [12].

3 KNOWLEDGE ISLANDS

In this section, we present Knowledge Island, a web tool designed to identify the concentration of knowledge within source code repositories. This open-source tool enables developers and project managers to asynchronously clone GitHub repositories and identify knowledge concentrations by calculating the Truck Factor of each project component. By pinpointing components with a low Truck Factor, Knowledge Island highlights areas where knowledge is concentrated in a single developer or small group, potentially posing a risk to project stability. Figure 1 provides a visual overview of the main actions and components of Knowledge Island.

The tool follows a simple client-server architecture divided into a front-end and back-end. The front-end is responsible for acquiring user input data, such as information from the repository to be cloned and analyzed, and presenting the processed results. The back-end consists of a RESTful API along with access to a relational database to store user data and their repositories.

The front-end is implemented using the React[5] Javascript library, version 18. To help with the build of web components such as tables and forms and their styling, we mainly use the component libraries React Bootstrap[6] and Material UI[7]. The main components of the front-end include a form for repository cloning and analysis, a list of cloned repositories with their analysis process status, a detail page containing repository knowledge concentration information, and pages/components for user registration and authentication.

On the other hand, the back-end is implemented in the Java programming language (version 17) and uses a PostgreSQL database for data persistence. We employ the Spring Boot[8] framework to build the API and its endpoints. The back-end of Knowledge Islands provides a set of endpoints for managing and retrieving information about GitHub repository cloning and analysis results, as presented in Table1. The first endpoint allows users to initiate the cloning and analysis process for a specified GitHub repository by providing the GitHub URL and specific branch name. The second one retrieves a list of all cloning and analysis processes started by a particular user, enabling users to track the status of their requests. The third endpoint provides access to the analysis results of a specific repository, including the knowledge concentration information and Truck Factor calculations.

Finally, to clone and manipulate repositories in the back-end, we utilize the Java library JGit[9]. This library allows the handling of Git operations programmatically. We employ a set of publicly available scripts to assist in extracting development history data, which is essential for identifying knowledge within the code. These scripts are included in our application repository[10] and play an important role in analyzing and processing the historical data needed for our analysis.

4 USAGE SCENARIO

In this section we present a usage scenario following the main features of Knowledge Islands, briefly represented in Figure 1. In this example, we will use data from the *Spring-Data-JPA*[11] project, an important repository in the Spring Ecosystem[12], and *Apache Kafka*[13], another popular project for web development.

After completing the registration and authentication process, the user is directed to the Knowledge Islands *Home* page (see Figure 2). This page features a form that initiates a process of cloning and analysis of a public GitHub repository. The form, which interacts

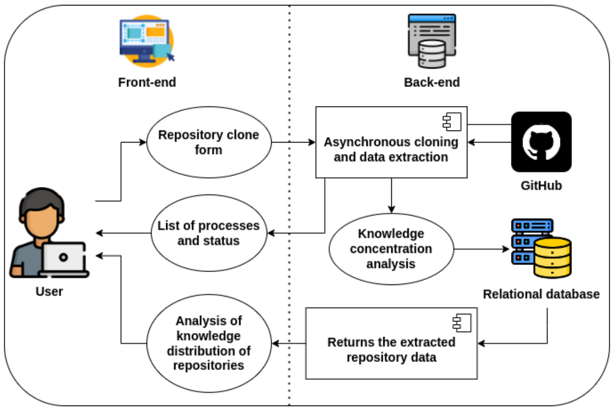

Front-end
Back-end
Repository clone form
Asynchronous cloning and data extraction
GitHub
List of processes and status
Knowledge concentration analysis
User
Relational database
Analysis of knowledge distribution of repositories
Returns the extracted repository data

Figure 1: Knowledge Islands operation diagram.

with the first endpoint described in Table 1, includes two fields: a mandatory field for entering the GitHub URL of the repository to be analyzed and an optional field for specifying the desired branch to be cloned and analyzed. The project's main branch will be cloned by default if the branch field is blank.

The analysis process begins asynchronously once the user clicks the "Start Analysis" button (see Figure 2). The initiated tasks are then listed in a table beneath the form, allowing users to manage multiple analyses simultaneously. Each row in the table provides detailed information about the ongoing tasks, including the repository URL, the date and time the analysis started, and the current stage of the process. The stages range from "Process Initialized" to "Process Finished," providing a clear and concise overview of the analysis progress for each repository. This setup ensures that users can easily track the status of their analyses and manage their workflows efficiently.

Only after a process has successfully completed ("Process Finished") can the user access the detailed analysis results via a button in the last column of the table (see Figure 2). Upon clicking the "Details" button for a repository listed on the home page (Figure 2), the user is directed to the repository analysis detail page (Figure 3). This page presents a comprehensive view of the analyzed repository.

At the top of this repository detail page, the users find key information such as the analyzed version data, the repository size - measured by the number of developers, commits, and files - and the overall project Truck Factor. At the bottom of the page, the user will find a list of all developers and analyzed files.

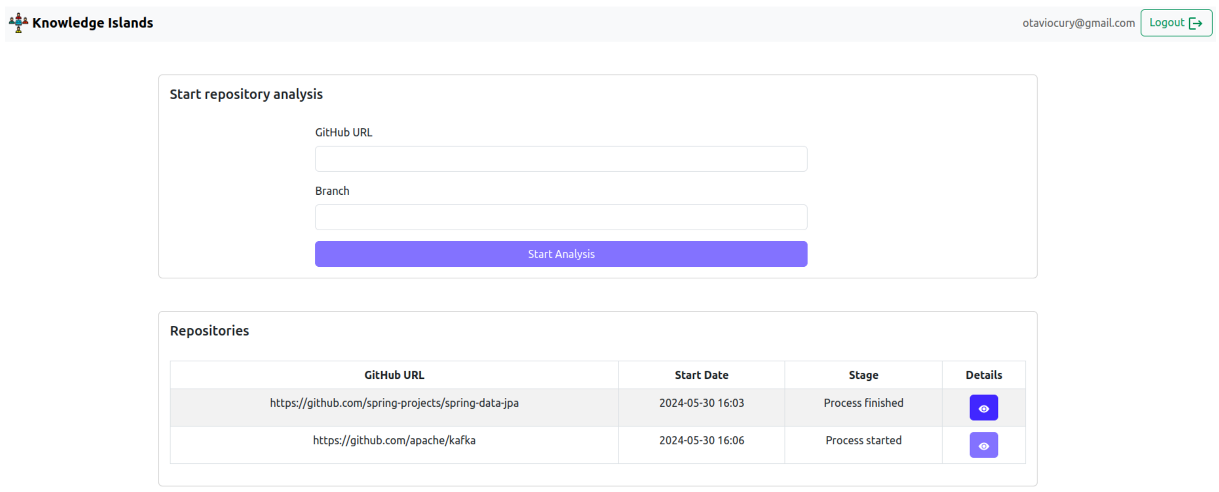
At the center of the page, a tree component displays the project's directory structure, allowing users to navigate through folders and files. Alongside each item (directory or file), a readily visible Truck Factor value indicates the level of knowledge concentration within that component (see Figure 3).

The label visually represents the Truck Factor of each item using a gradient scale of orange hues. A darker shade of orange signifies a lower Truck Factor, indicating a greater risk of knowledge loss or project disruption. This visual cue facilitates the identification of potential knowledge islands and enables proactive risk mitigation efforts.

[5]https://react.dev/
[6]https://react -bootstrap.netlify.app/
[7]https://mui.com/
[8]https://spring.io/projects/spring-boot
[9]https://www.eclipse.org/jgit/
[10]https://github.com/OtavioCury/knowledge-islands
[11]https://github.com/spring-projects/spring-data-jpa
[12]https://github.com/spring-projects
[13]https://github.com/apache/kafka

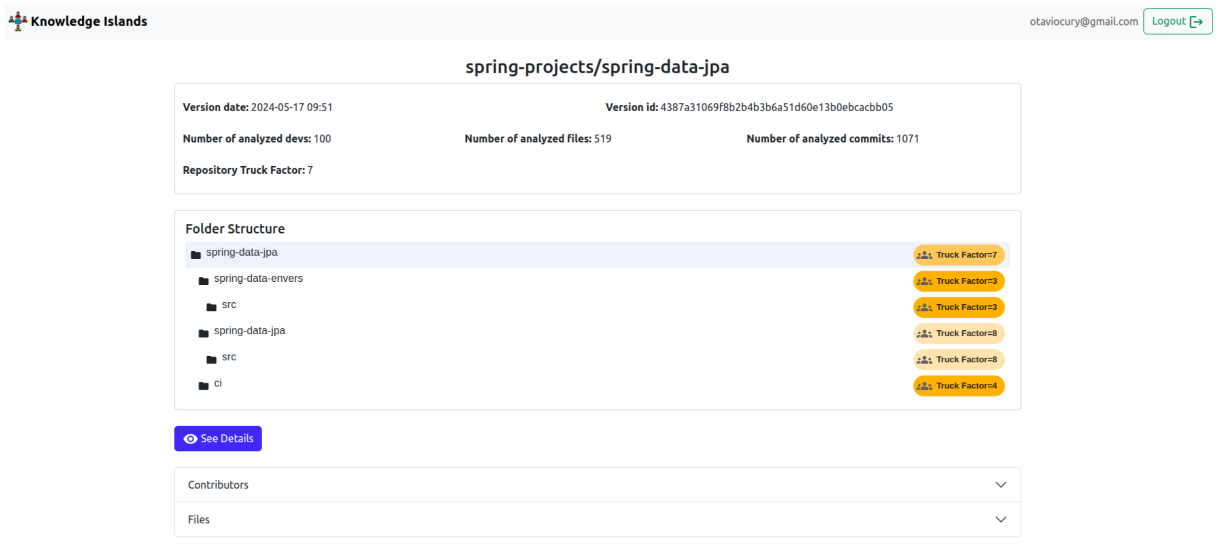
Table 1: Main endpoints of the Knowledge Islands API

Endpoint	Method	Description
/git-repository-version-process/start-git-repository-version-process	POST	Initiates the cloning and analysis of a repository. Receives JSON form with GitHub URL of the repository, and the name of the branch.
/git-repository-version-process/user/<id>	GET	Lists the repository analysis processes initiated by a user.
/git-repository-version/<id>	GET	Returns the analysis results of a repository.



GitHub URL	Start Date	Stage	Details
https://github.com/spring-projects/spring-data-jpa	2024-05-30 16:03	Process finished	
https://github.com/apache/kafka	2024-05-30 16:06	Process started	

Figure 2: Page for cloning and listing analyzed repositories.



spring-projects/spring-data-jpa

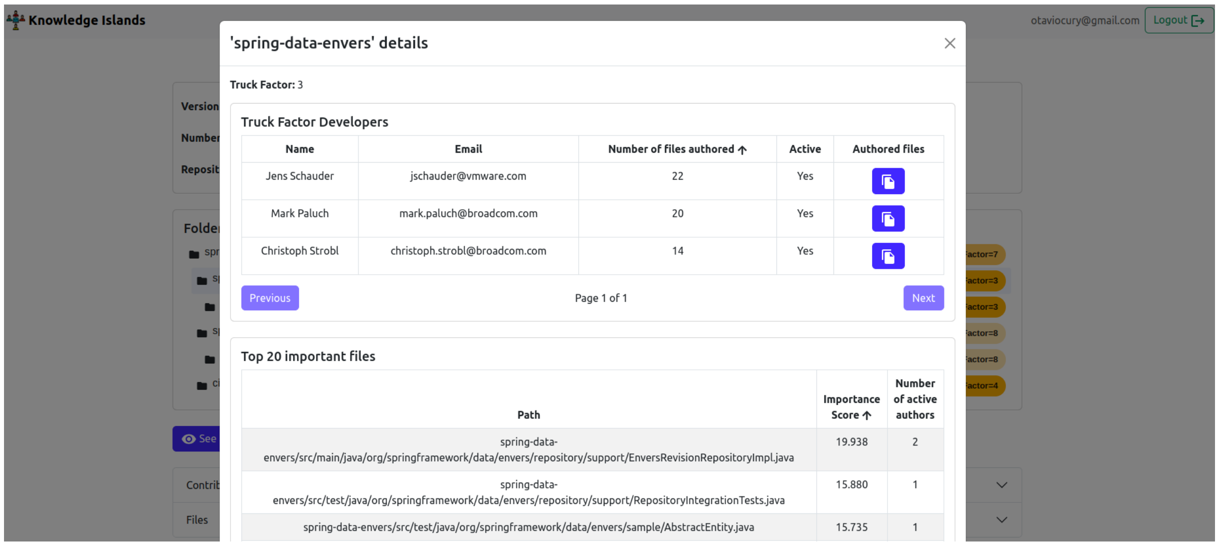
Version date: 2024-05-17 09:51
Version id: 4387a31069f8b2b4b3b6a51d60e13b0ebcacbb05
Number of analyzed devs: 100
Number of analyzed files: 519
Number of analyzed commits: 1071
Repository Truck Factor: 7

Folder Structure

spring-data-jpa	Truck Factor=7
spring-data-envers	Truck Factor=3
src	Truck Factor=3
spring-data-jpa	Truck Factor=8
src	Truck Factor=8
ci	Truck Factor=4

Figure 3: Repository analysis details page.

The users can select a specific item in the tree component and click the "See Details" button to access more in-depth information about its Truck Factor. A modal window then displays a table listing the Truck Factor developers responsible for that component, their names, email addresses, and the number of files they authored. This table is sorted in descending order by the number of files authored (see Figure 4). The user can then click the button in the last column "Authored files" to expand the row and see a list of files. In short, this feature, in addition to indicating the main developers, indicates how many and which files they maintain.



'spring-data-envers' details

Truck Factor: 3

Truck Factor Developers

Name	Email	Number of files authored ↑	Active	Authored files
Jens Schauder	jschauder@vmware.com	22	Yes	
Mark Paluch	mark.paluch@broadcom.com	20	Yes	
Christoph Strobl	christoph.strobl@broadcom.com	14	Yes	

Previous Page 1 of 1 Next

Top 20 important files

Path	Importance Score ↑	Number of active authors
spring-data-envers/src/main/java/org/springframework/data/envers/repository/support/EnversRevisionRepositoryImpl.java	19.938	2
spring-data-envers/src/test/java/org/springframework/data/envers/repository/support/RepositoryIntegrationTests.java	15.880	1
spring-data-envers/src/test/java/org/springframework/data/envers/sample/AbstractEntity.java	15.735	1

Figure 4: Modal displaying the Truck Factor details of a specific artifact.

The Truck Factor developer table includes a column indicating whether each developer is currently active in the project. The tool defines, by default, inactive developers as those who have not made any commits within the past year, following established criteria from the literature [3, 12]. This information enables users to identify potential knowledge gaps and address them proactively.

Still in the Truck Factor details modal, beneath the list of Truck Factor developers, users will find a list of files along with their corresponding *importance scores*—as explained in Section 2.1.1—and the number of active authors associated with each file. This feature allows users to identify files with greater significance to the project and correlate this with the number of developers knowledgeable about these files. Consequently, users can pinpoint key files at risk of being left without experts, potentially leading to project progress and maintenance complications.

5 LIMITATIONS

Knowledge Island effectively analyzes a software repository's source code knowledge concentration. However, we acknowledge limitations in the current version of the tool that will be addressed in future implementations.

Currently, the repository directory structure under analysis is presented only using the tree component, as shown in Figure 3. However, there are other ways to present this information, such as zoomable bubble plots, which can facilitate the visualization of the structure and knowledge concentration information. Offering different repository viewing options will provide users with a more flexible and intuitive experience.

Additionally, the knowledge model used in the tool is another feature that could offer more options for users. As explained in Section 2.1.1, the tool currently employs the Degree of Expertise (DOE) in the Truck Factor algorithm. Consequently, our implementation inherits all the limitations of the model discussed in the study that proposed it [12]. Therefore, the tool can incorporate other knowledge models from the literature, allowing users to conduct knowledge concentration assessments that consider different variables from the development history.

Another current limitation is that Knowledge Island does not have direct integration with GitHub, such as through the login process. The tool can use the GitHub API with OAuth 2.0 to authenticate users, thereby facilitating the process of analyzing data from their repositories.

6 RELATED WORK

Some tools have been proposed in studies examining developer expertise in source code. This section will present some of these tools and explain how Knowledge Islands differ from them.

SonarQube[14] is a well-known open-source platform designed to manage and enhance code quality by identifying poorly written code that violates coding rules or best coding practices[8]. It comes equipped with a wide array of features in its standard installation and can be further expanded through the use of both free and commercial plug-ins.

One of these extensions that complements Sonarqube's functionalities is the SoftVis3D[15] plugin, which allows you to visualize the directory and file structure of a project such as a city. Using a combination of colors and building heights, the tool indicates hot spots in the code according to different metrics such as coverage, complexity, and number of authors. The metric number of authors of a file is related to the Truck Factor concept, but different from the metrics used in Knowledge Islands, only Blame measures of the files are taken into account, not considering other variables that make the identification of expertise more robust.

In addition to plugins, there are also web tools specialized in code analysis. CodeScene[16] is a proprietary code analysis web tool that offers a variety of code quality metrics [29, 30]. Among these

[14]http://www.sonarqube.org
[15]https://softvis3d.com/
[16]https://codescene.com/

quality metrics, there are some related to the concentration of knowledge, identification of experts, and calculation of lost knowledge, simulating a Truck Factor situation. However, like the tools presented previously, CodeScene only uses LoC (number of lines of code) to identify authors [21], which represents a gap for tools that implement improvements.

There are also other less commercial tools aimed at specific studies. For example, Avelino presents a tool[17], together with a new algorithm for calculating Truck Factor [4]. Haratian et al. presented BFSig[18], another tool for calculating the Truck Factor, with the difference of taking into account the importance of software components in the calculation [16]. Almarimi et al. in the study present the tool named CsDetector[19], which, among other community smells, is capable of estimating the Truck Factor [1]. Finally, Klimov et al. introduce Bus Factor Explorer[20], a web application with an interface and an API to analyze and visualize Truck Factor information using the *Degree of Authorship* (DOA) as the knowledge model [22]. However, even though each of these tools represents an advancement in the field, they are either not web-based, which makes their use by practitioners more difficult, or they do not implement the knowledge model and file importance metrics used in this study.

7 CONCLUSION AND FUTURE WORK

In this work, we present Knowledge Islands, a tool for visualizing the concentration of knowledge in software repositories. Utilizing state-of-the-art models and algorithms from the literature, Knowledge Islands assists developers and software managers in decision-making by providing metrics on the importance of developers and files in a repository.

In the presented usage case, using data from public repositories, we demonstrated the process of downloading, extracting, and presenting data to the user. Knowledge Islands effectively showcased the repository's Truck Factors at various granularities: project, module, and file levels. The tool also highlighted the top developers associated with each artifact, along with the top files. This usage scenario illustrates the efficiency of Knowledge Islands as a knowledge distribution analysis tool.

As future improvements, as pointed out in Section 5, we intend to facilitate integration with GitHub. By enabling authentication using GitHub credentials, users will have faster and more seamless access to their repositories' data. Additionally, we aim to enhance the tool's visual design by offering new ways to present metrics related to knowledge concentration through various graphics, such as zoomable bubble plots[21]. This will provide users with more intuitive and interactive visualizations. We plan to incorporate additional knowledge models, offering users different perspectives on code knowledge and further enriching the tool's analytical capabilities.

Finally, we plan to make the tool available to the community and collect feedback from practitioners. This information will help us identify the strengths and weaknesses of the application and determine new requirements for a knowledge analysis tool.

[17]https://github.com/aserg-ufmg/Truck-Factor
[18]https://github.com/JetBrains-Research/file-importance
[19]https://github.com/Nuri22/csDetector
[20]https://github.com/JetBrains-Research/bus-factor-explorer
[21]https://observablehq.com/@d3/zoomable-circle-packing

As a final note, we invite the community of developers and researchers to contribute to Knowledge Islands, to improve the features previously mentioned. The tool is publicly available on GitHub[22], along with documentation on its main endpoints, features, and scripts.

REFERENCES

[1] Nuri Almarimi, Ali Ouni, Moataz Chouchen, and Mohamed Wiem Mkaouer. 2021. csDetector: an open source tool for community smells detection. In *29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1560–1564.

[2] Guilherme Avelino, Eleni Constantinou, Marco Tulio Valente, and Alexander Serebrenik. 2019. On the abandonment and survival of open source projects: An empirical investigation. In *13th International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 1–12.

[3] Guilherme Avelino, Eleni Constantinou, Marco Tulio Valente, and Alexander Serebrenik. 2019. On the abandonment and survival of open source projects: An empirical investigation. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 1–12.

[4] Guilherme Avelino, Leonardo Passos, Andre Hora, and Marco Tulio Valente. 2016. A novel approach for estimating truck factors. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*. IEEE, 1–10.

[5] Guilherme Avelino, Leonardo Passos, Andre Hora, and Marco Tulio Valente. 2019. Measuring and analyzing code authorship in 1 + 118 open source projects. *Science of Computer Programming* 176 (5 2019), 14–32. https://doi.org/10.1016/j.scico.2019.03.001

[6] Christian Bird, Nachiappan Nagappan, Brendan Murphy, Harald Gall, and Premkumar Devanbu. 2011. Don't touch my code! Examining the effects of ownership on software quality. In *19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. 4–14.

[7] Fabio Calefato, Marco Aurelio Gerosa, Giuseppe Iaffaldano, Filippo Lanubile, and Igor Steinmacher. 2022. Will you come back to contribute? Investigating the inactivity of OSS core developers in GitHub. *Empirical Software Engineering* 27, 3 (2022), 1–41.

[8] G Ann Campbell and Patroklos P Papapetrou. 2013. *SonarQube in action*. Manning Publications Co.

[9] Edna Dias Canedo, Rodrigo Bonifácio, Márcio Vinicius Okimoto, Alexander Serebrenik, Gustavo Pinto, and Eduardo Monteiro. 2020. Work practices and perceptions from women core developers in oss communities. In *14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 1–11.

[10] Gerardo Canfora, Massimiliano Di Penta, Rocco Oliveto, and Sebastiano Panichella. 2012. Who is going to mentor newcomers in open source projects?. In *ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. 1–11.

[11] Valerio Cosentino, Javier Luis Cánovas Izquierdo, and Jordi Cabot. 2015. Assessing the bus factor of git repositories. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 499–503.

[12] Otávio Cury, Guilherme Avelino, Pedro Santos Neto, Marco Túlio Valente, and Ricardo Britto. 2024. Source code expert identification: Models and application. *Information and Software Technology* (2024), 107445.

[13] Otávio Cury, Guilherme Avelino, Pedro Santos Neto, Ricardo Britto, and Marco Túlio Valente. 2022. Identifying source code file experts. In *16th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. 125–136.

[14] Mívian Ferreira, Marco Tulio Valente, and Kecia Ferreira. 2017. A comparison of three algorithms for computing truck factors. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. IEEE, 207–217.

[15] Thomas Fritz, Gail C Murphy, Emerson Murphy-Hill, Jingwen Ou, and Emily Hill. 2014. Degree-of-knowledge: Modeling a developer's knowledge of code. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 23, 2 (2014), 1–42.

[16] Vahid Haratian, Mikhail Evtikhiev, Pouria Derakhshanfar, Eray Tüzün, and Vladimir Kovalenko. 2023. BFSig: Leveraging File Significance in Bus Factor Estimation. In *31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1926–1936.

[17] Lile Hattori and Michele Lanza. 2010. Syde: a tool for collaborative software development. In *32nd ACM/IEEE International Conference on Software Engineering-Volume 2*. 235–238.

[18] Md Kamal Hossen, Huzefa Kagdi, and Denys Poshyvanyk. 2014. Amalgamating source code authors, maintainers, and change proneness to triage change requests. In *22nd International Conference on Program Comprehension*. 130–141.

[22]https://github.com/OtavioCury/knowledge-islands

[19] E. Jabrayilzade, M. Evtikhiev, E. Tuzun, and V. Kovalenko. 2022. Bus Factor in Practice. In *2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE Computer Society, Los Alamitos, CA, USA, 97–106. https://doi.org/10.1109/ICSE-SEIP55303.2022.9793985

[20] Elgun Jabrayilzade, Mikhail Evtikhiev, Eray Tüzün, and Vladimir Kovalenko. 2022. Bus factor in practice. In *44th International Conference on Software Engineering: Software Engineering in Practice*. 97–106.

[21] Andreas Karlsson. [n. d.]. Driving Development Resilience: Analyzing Truck Factors across Proprietary and Open-Source Projects. ([n. d.]).

[22] Egor Klimov, Muhammad Umair Ahmed, Nikolai Sviridov, Pouria Derakhshanfar, Eray Tüzuü, and Vladimir Kovalenko. 2023. Bus Factor Explorer. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2018–2021.

[23] Segla Kpodjedo, Filippo Ricca, Philippe Galinier, and Giuliano Antoniol. 2008. Not all classes are created equal: toward a recommendation system for focusing testing. In *Proceedings of the 2008 international workshop on Recommendation systems for software engineering*. 6–10.

[24] H Maen, ML Collard, and J Maletic. 2010. Measuring class importance in the context of design evolution. In *Program Comprehension (ICPC), IEEE 18th International Conference on. IEEE.*

[25] Paul Ralph, Sebastian Baltes, Gianisa Adisaputri, Richard Torkar, Vladimir Kovalenko, Marcos Kalinowski, Nicole Novielli, Shin Yoo, Xavier Devroey, Xin Tan, et al. 2020. Pandemic programming: how COVID-19 affects software developers and how their organizations can help (2020). *arXiv preprint arXiv:2005.01127* (2020).

[26] Filippo Ricca, Alessandro Marchetto, and Marco Torchiano. 2011. *On the Difficulty of Computing the Truck Factor*. Vol. 6759 LNCS. 337–351. Issue ii. https://doi.org/10.1007/978-3-642-21843-9_26

[27] Peter C Rigby, Yue Cai Zhu, Samuel M Donadelli, and Audris Mockus. 2016. Quantifying and mitigating turnover-induced knowledge loss. *38th International Conference on Software Engineering (ICSE)*, 1006–1016. https://doi.org/10.1145/2884781.2884851

[28] Emre Sülün, Eray Tüzün, and Uğur Doğrusöz. 2019. Reviewer recommendation using software artifact traceability graphs. In *15th International Conference on Predictive Models and Data Analytics in Software Engineering*. 66–75.

[29] Adam Tornhill. 2015. Your code as a crime scene: use forensic techniques to arrest defects, bottlenecks, and bad design in your programs. *Your Code as a Crime Scene* (2015), 1–218.

[30] Adam Tornhill. 2018. Assessing technical debt in automated tests with codescene. In *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 122–125.