1. Introduction

Software development teams often work with a large number of software artifacts across several different projects in a collaborative manner. However, As time passes and software projects grow and teams change, identifying the most experienced developer to fix a certain bug or develop a new feature can prove to be a difficult task. In this paper we propose ExpertiseAnalyzer – a tool that leverages the Degree-of-Authorship to identify the most experienced developers for any given file in a software project repository.

2. Related work

2.1 Degree of Knowledge

T. Fritz et al. (2010) propose a model for capturing source code familiarity that considers not only authorship data, but also interaction information. To achieve this goal, the Degree-of-Knowledge (DOK) model calculates a real value for each source code element (e.g., classes and methods) by combining the Degree-of-Authorship (DOA) and Degree-of-Interest (DOI) as follows:

$$DOK = \alpha\, FA * FA + \alpha\, DL * DL + \alpha\, AC * AC + \beta\, DOI * DOI$$

For a given developer and source code element, the DOA is defined based on three factors: if the developer created the element (FA – First Authorship), how many changes the developer made to the element after creating it (DL – Deliveries), and how many changes were made to the element by other developers (AC – Acceptances). The DOI is defined based on the amount of interaction (selections and edits) between a developer and a source code element. An edit is identified by a keystroke in an editor window, while a selection occurs when he touches the element (e.g., open a file).

To determine the weightings, the authors conducted an experiment with seven professional Java developers that included gathering data from a project's revision history and monitoring and interviewing developer to understand their interaction with code elements and their knowledge of it. The resulting DOK equation is the following:

$$DOK = 3.293 + 1.098 * FA + 0.164 * DL - 0.321 * \ln(1 + AC) + 0.19 * \ln(1 + DOI)$$

After conducting three exploratory case studies, the authors show that the DOK model provides better results if compared to existing approaches for identifying and measuring developer's expertise.

2.2 Truck Factor

The Truck Factor (also known as bus factor) refers to the minimum number of developers in a team that would have to be "hit by a truck/bus" (unexpectedly quit) to cause the project to collapse. Since it is usually understood as a measurement of how much information is concentrated in a certain number of team members, software teams may try to increase it by using a number of different approaches, such as adopting pair-programming as a way of sharing knowledge about a certain piece of software or encouraging more extensive and up-to-date documentation.

Alvelino et. al. (2015) proposes the following greedy heuristic to calculate the *TruckFactor* (TF) of 133 popular GitHub applications in [2]: to remove the author with more files in the repository until more than half of the files are orphans (i.e., no developer is assigned as its author), which the heuristic considers a sign that the project is incapacitated. To determine the authorship of a file, Alvelino et. al. rely on the Degree-of-Authorship (DOA) model as proposed by T. Fritz et al. (2011) in [1]. They show that most systems have a low *TruckFactor*, with 34% (45 systems) of them having a TF of 1 and 30% (40 systems) of them having a TF of 2. Finally, they found that systems with a large number of plug-ins cause their heuristic to overestimate the TF, such as *torvalds/linux* (TF of 130 if considering Linux's subsystem drivers but only 57 otherwise) and *caskroom/homebrew-cask* (TF of 250 if considering the files in Library/Formula, but only 2 otherwise).

2.3 Expertise Browser

Mockus and Herbsleb (2002) propose an approach based on quantifying a developer's experience through what they call experience atoms (EAs), which represent the most basic unit of experience considering the changes a person or organization makes to a software artifact. It relies on revision control data to identify developers with expertise in a certain area of a software, allowing its user to differentiate between developers who worked briefly and and developers who have extensive experience with a particular piece of code. To help a developer or organization find the experts on a software entity, the authors also propose the Expertise Browser (ExB), which is a tool that allows its users to query and visualize the people who have expertise on the given software artifact. Finally, after deploying the tool in a large software development organization, the authors show that while newer teams used the ExB to identify expertise, larger teams tended to use it for finding people who possessed a certain expertise profile.

3. ExpertiseAnalyzer

ExpertiseAnalyzer is a command-line tool written in Java that leverages the Degree-of-Authorship (DOA) as proposed by T. Fritz et al. (2010) to analyze a Git repository and identify the most experienced developers for any given file. It also determines the top-3 developers for either the entire repository or a specific branch based on the number of times a developer was identified as a "top developer" for a file. Some of the most important design decisions in this context are (i) how to treat pull requests and merges; and (ii) how to deal with different branches and its commits. *ExpertiseAnalyzer* does not consider commits that represent a response to a pull request (merge) to avoid assigning authorship of a number of commits to the developer conducting the merge process (Figure 1).

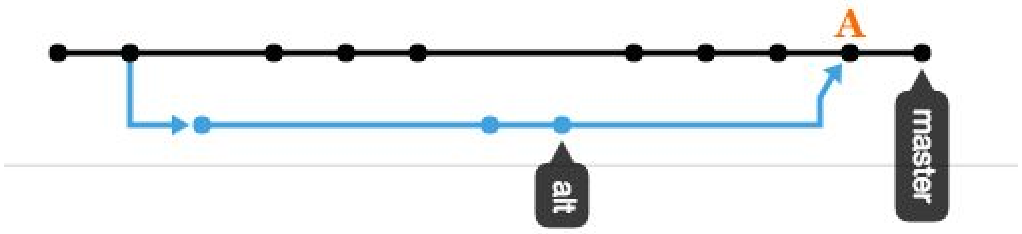

Figure 1 - Commit A is not processed because it is a response to a pull request (merge)

When dealing with the entire repository, each existing branch is considered and their commits processed, but any commits that have already been processed are ignored. In Figure 2, for example, when dealing with the entire project, commits A, B, C, and D would be processed. If D was processed when traversing the *master* branch, however, *ExpertiseAnalyzer* would not process it again when traversing the *alt* branch.

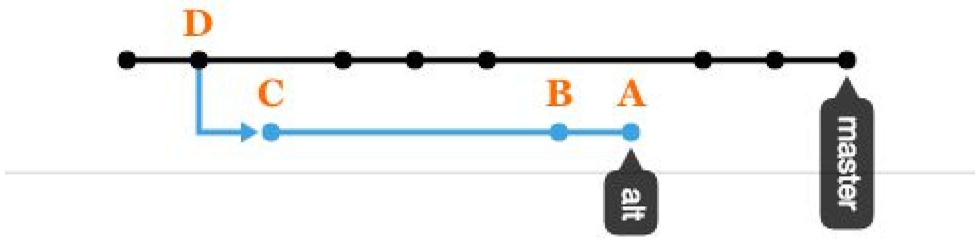

Figure 2 - Processing commits in project with two branches

As a first step to assess how our proposed approach compare to existing ones, we conduct a brief preliminary comparative study of our approach as implemented in *ExpertiseAnalyzer* and the greedy heuristic approach implemented by *TruckFactor*, since both of them are based on the Degree-of-Authorship. We run both tools on four different public software projects hosted on GitHub: *imageworks/OpenColorIO*, *imageworks/OpenShadingLanguage*, *alembic/alembic*, and *openexr/openexr*. For each of them we list the most experienced developers according to *ExpertiseAnalyzer* and the Truck Factor and the associated developers according to *TruckFactor* (Table 1).

	OpenColorIO	*OpenShading Language*	*alembic*	*openexr*
ExpertiseAnalyzer	- Jeremy Selan - Malcolm Humphreys - dbr	- Larry Gritz* - fpsunflower	- Joe Ardent - Lucas Miller - Ryan Galloway	- Piotr Stanczyk - Florian Kainz - Drew Hess
TruckFactor	TF = 2 - Jeremy Selan - malcolmhumphreys	TF = 1 Larry Gritz	TF = 1 - Lucas Miller	TF = 3 - Piotr Stanczyk - Florian Kainz - Drew Hess

Table 1 - Most experienced developers according to *ExpertiseAnalyzer* and *TruckFactor*

Overall, ExpertiseAnalyzer seems to provide very similar results and identify the same developers. From Table 1, a couple of apparent inconsistencies deserve further explanation. Firstly, while the number of developers listed by the *TruckFactor* tool varies according to the Truck Factor, *ExpertiseAnalyzer* always list up to three most experienced developers. The one exception in Table 1 is the repository *OpenShadingLanguage*, for which ExpertiseAnalyzer listed Larry Gritz as both the first and second most experienced developers. This is explained by the fact that a developer is uniquely identified by his e-mail and Larry Gritz used two different e-mails to submit changes. Secondly, for the repository *alembic*, while *ExpertiseAnalyzer* identifies Joe Ardent as the most experienced developer, the *TruckFactor* points to Lucas Miller. From our observations, this discrepancy exists because the TruckFactor considers merges as regular commits and assigns authorship to the developer who conducted the merge process. By manually inspecting *alembic*, we notice that Lucas Miller is the main developer responding to pull requests and conducting the merges. OutThing, on the other hand, ignores these commits in an attempt to achieve more realistic results.

4. References

[1] 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 Trans. Softw. Eng. Methodol.* 23, 2, Article 14 (April 2014), 42 pages.
[2] Avelino G., Valente M. T., Hora A. (2015) What is the Truck Factor of popular GitHub applications? A first assessment.
[3] Audris Mockus and James D. Herbsleb. 2002. Expertise browser: a quantitative approach to identifying expertise. In *Proceedings of the 24th International Conference on Software Engineering*(ICSE '02). ACM, New York, NY, USA, 503-512.