Quantifying Synergy between Software Projects using README Files Only

Roxanne El Baff
Institute for Software Technology
German Aerospace Center (DLR)
Oberpfaffenhofen, Germany
roxanne.elbaff@dlr.de

Sivasurya Santhanam
Institute for Software Technology
German Aerospace Center (DLR)
Cologne, Germany
sivasurya.santhanam@dlr.de

Tobias Hecking
Institute for Software Technology
German Aerospace Center (DLR)
Cologne, Germany
tobias.hecking@dlr.de

***Abstract*—Software version control platforms, such as GitHub, host millions of open-source software projects. Due to their diversity, these projects are an appealing realm for discovering software trends. In our work, we seek to quantify *synergy* between software projects by connecting them via their *similar* as well as *different* software features. Our approach is based on the Literature-Based-Discovery (LBD), originally developed to uncover *implicit* knowledge in scientific literature databases by linking them through transitive connections. We tested our approach by conducting experiments on 13,264 GitHub (open-source) Python projects. Evaluation, based on human ratings of a subset of 90 project pairs, shows that our developed models are capable of identifying potential synergy between software projects by solely relying on their short descriptions (i.e. readme files).**

***Index Terms*—repository mining, natural language processing**

I. Introduction

The growing amount of open-source software projects range from small experimental software to large-scale and continuously advancing systems. Many of such projects are available on public repository hosting platforms such as GitHub that do not only provide functionalities for managing source code but also include tools for documentation and collaboration.

Thus, repository hosting platforms are not only technical means but can also be considered as an agglomeration of ideas and knowledge scattered over several projects. However, exploring this vast amount of information manually is beyond human capacity.

Several applications assisting users in exploring software repositories aim at finding *similar* repositories to one's own repository, e.g. [1]–[3], helping to identify alternative implementations, explore related projects, or identify plagiarism. However, when focusing solely on *similarities*, searchers are trapped in their search bubbles and are rarely exposed to develop something new based on repositories that *complement* their work. Thus, in this paper, we propose an approach for exploiting distributed software knowledge in repository platforms, which is inspired by Literature-Based Discovery (LBD) [4]. LBD uncovers implicit knowledge and synthesizes hypotheses from scientific literature databases by identifying complementary information sources (i.e., publications). The general idea is to seek possible transitive relationships between concepts mentioned in different publications. For example, if concepts A (e.g., a disease) and B (e.g., an enzyme) are found to be often related to a concept C (e.g., a drug) respectively, but no relationship between A and B has been reported so far, one can hypothesize that there is also a relationship between A and B which is worth to be explored.

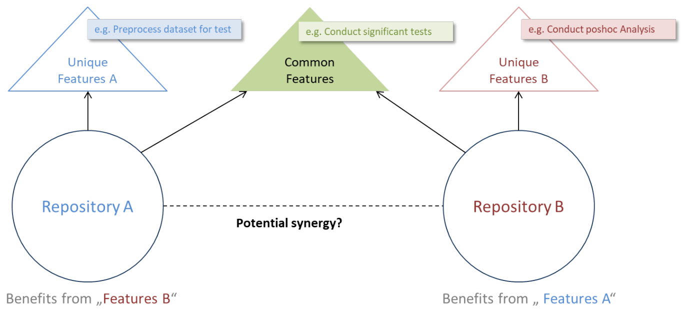

e.g. Preprocess dataset for test
Unique Features A
e.g. Conduct significant tests
Common Features
e.g. Conduct poshoc Analysis
Unique Features B
Repository A
Potential synergy?
Repository B
Benefits from „Features B“
Benefits from „ Features A“

Fig. 1. Two repositories are related by a common set of features. There is a potential synergy since both can extend the feature set of the other.

In this paper, we adopt the idea of LBD to quantify a synergy score between project pairs taking into account, both, software feature similarities and differences. For example, Figure 1 illustrates a simplified scenario of two repositories with a potential synergy. Repository *A* has two features: (1)"*preprocess data for significance tests*", and (2) "*conduct significance tests*". Whereas, Repository *B* has only feature (2) as common one and an additional feature: (3) "*conduct post-hoc analysis*". As we see, these two repositories can benefit from each other because they share a common feature, and each is missing one feature from the other. Accordingly, we ask the question: How can one discover software repositories that are similar to some extent but different enough to expand each other's functionality so that they can be the basis for new developments?

Our approach solely relies on information extracted from repositories' publicly available *readme file* because it is a common practice, in high-quality open-source projects, to summarize the main project information in such files. More precisely, we use existing natural language techniques to model the software features of each project, then we quantify

DOI reference number: 10.18293/SEKE2021-162

the *synergy* between each project pair by defining a new synergy ranking method.

We conduct our experiments on more than 13K GitHub software repositories with Python as the main programming language and having a readme file written in English. We evaluate our models based on human ratings, which shows that our approach successfully identifies synergy between project pairs.

Altogether, our contribution is threefold:

- A new view on exploring synergy quantification between repositories to inspire new ideas.
- A novel approach that combines existing natural language techniques to extract relevant information from repositories' descriptions, and ranking techniques to quantify synergy between pairs of projects.
- Experimental evidence that *synergy* between software projects can be detected automatically.

Our work can be used, among others, for recommendation systems, or discovery-based systems from large software projects pile. For reproducibility, the code is publicly available: https://github.com/DLR-SC/repository-synergy.

II. Related Work

A. Recommendation of software repositories

Previous works in the repository mining community focus already on developing approaches for software recommendations by either using the metadata of GitHub repositories (e.g., stargazes, readme files, ...) such as Zhang et al. [3], or by using software artifacts (e.g., software packages, code, ...) such as McMillan et al. [1]. Others focus on categorizing software repositories and readme files, which helps to better perceive a massive pile of data and grasp the content faster (Prana et al. [5], Sharma and Thung et al. [2], ...).

Zhang et al. [3] built a recommendation system called RepoPal. They detect similar repositories using three different heuristics based on readme files and stargazing. They assume that two repositories are likely to be similar based on three measurements: 1) readme files with similar content, 2) repositories starred by users of similar interests, and 3) repositories starred together within a short period by the same user. Their recommendation system outperform CLAN (*Closely reLated ApplicatioNs*) [1]. McMillan et al. [1] developed an approach for automatically detecting similar applications for a given Java application based on packages and class hierarchy.

Our goal and methodology differ from Zhang et al. [3]. In our approach, we exploit not only software similarities but also differences which we believe can inspire for *new* directions. Moreover, our methodology is different: we exploit the implicit knowledge between software projects based only on readme files instead of relying on different metadata.

B. Cataloging software repositories

Another strand of research in repository mining essential to our work is categorizing software repositories' thematic analysis of readme files.

GitHub creates showcases where they manually catalog a set of repositories on a certain topic. Sharma et al. [2] semi-automatically expanded such showcases. Using 10K repositories with readme files, they first extract the most descriptive section in the readme file by selecting the one with the highest cosine similarity value with the repository short description on the top of the repository landing page on GitHub. They then feed all these descriptions to a Latent Dirichlet Allocation using the Genetic Algorithm model, where they manually analyze topics into meaningful categories. This work indicates that readme files are already used in existing research to deduce the software features of a software repository. Also, using a topic modeling algorithm is common for clustering readme files and hence software repositories. In this work, we use similar techniques within our approach. However, we identify sections reflecting the software features in readme files using an existing classifier, READMEClassifier, trained by Prana et al. [5]. [5] systematically classify each section of a readme file to categories reflecting its purpose (see Section III-A for more details).

III. Approach

The following section outlines our approach to quantify synergies between software repositories, which consists of a pipeline with three steps, as shown in Figure 2.

A. Software Features Extraction

The first step extracts the repositories' descriptions of software features (Figure 2.1). To do that, we identify these sections by using the existing multi-label classifier, READMEClassifer, built by Prana et al. [5], which labels readme file sections. READMEClassifer was trained on 4k manually annotated *readme* file sections from 393 repositories with an F_1 score of 0.75 where each section was categorized into one or more of eight different categories. The *What* category is identified based on headings (e.g., *About*) or based on the text at the beginning of a README file. Sections describing a comparison to another software artifact with respect to performance, flexibility, and simplicity are categorized as *Why*. Other categories describe other metadata not related to software features. Prana et al. [5] combined *What* and *Why* into one label (*Why* sections were rare $< 3\%$), *WhatWhy*, to train their READMEClassifier.

We use this classifier to extract sections with the label *WhatWhy* for each repository, which describes the software features within it. In the next steps, we only use repositories having a readme file with *WhatWhy* section(s).

B. Software Features Modelling

Topic modeling is an unsupervised machine learning technique that automatically analyzes text data to determine cluster terms for a set of documents (in our case, readme files). Each document is assigned a weighted sum of topics. And each topic is represented by a set of *terms* and the probability of this term for a specific topic.

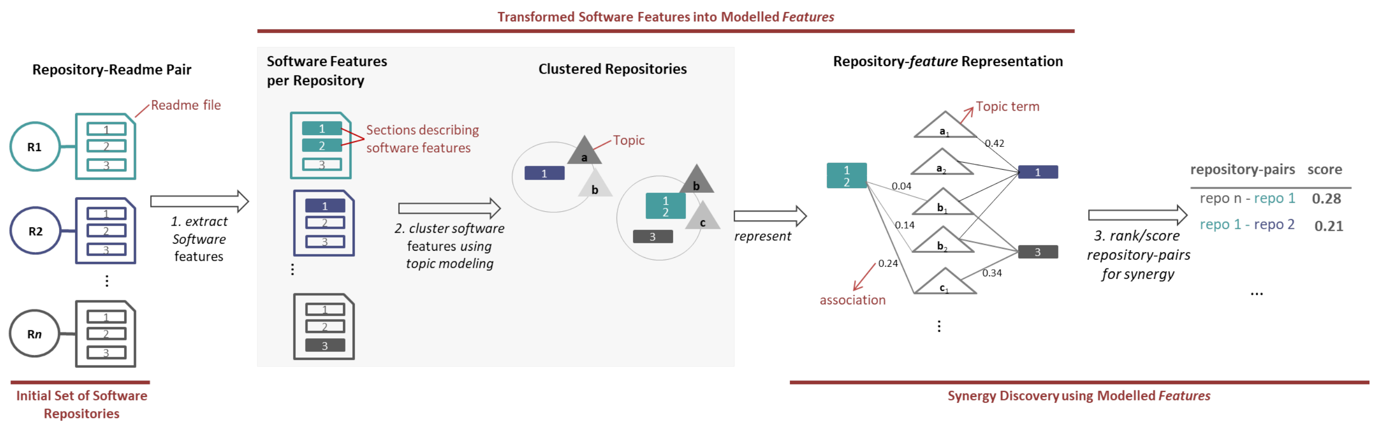
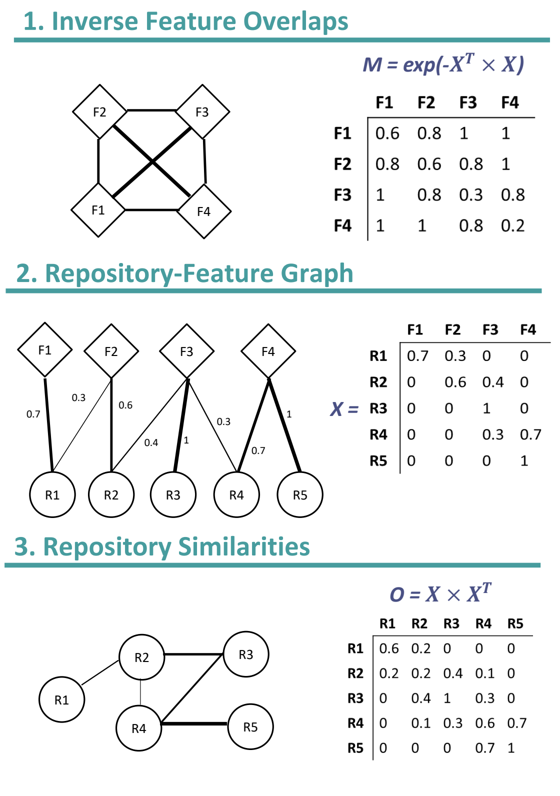

Transformed Software Features into Modelled Features
Repository-Readme Pair
Readme file
R1
R2
Rn
1. extract Software features
Software Features per Repository
Sections describing software features
2. cluster software features using topic modeling
Clustered Repositories
Topic
represent
Repository-feature Representation
Topic term
association
3. rank/score repository-pairs for synergy
repository-pairs score
repo n - repo 1 0.28
repo 1 - repo 2 0.21
Initial Set of Software Repositories
Synergy Discovery using Modelled Features

Fig. 2. Proposed approach of synergy discovery consisting of three steps: 1) Software features extraction, 2) software features modeling and 3) synergy scoring using a ranking algorithm.


1. Inverse Feature Overlaps
$M = exp(-X^T \times X)$
2. Repository-Feature Graph
$X =$
3. Repository Similarities
$O = X \times X^T$

	F1	F2	F3	F4
F1	0.6	0.8	1	1
F2	0.8	0.6	0.8	1
F3	1	0.8	0.3	0.8
F4	1	1	0.8	0.2

	F1	F2	F3	F4
R1	0.7	0.3	0	0
R2	0	0.6	0.4	0
R3	0	0	1	0
R4	0	0	0.3	0.7
R5	0	0	0	1

	R1	R2	R3	R4	R5
R1	0.6	0.2	0	0	0
R2	0.2	0.2	0.4	0.1	0
R3	0	0.4	1	0.3	0
R4	0	0.1	0.3	0.6	0.7
R5	0	0	0	0.7	1

Fig. 3. The three matrices used by the ranking algorithms: 1) matrix M for inverse features overlaps, 2) matrix X for repository feature combinations and 3) matrix 0 for features overlaps of repositories.

In this step (Figure 2.2), we apply a topic modeling algorithm only on the readme file sections from Section III-A to cluster them into *topics* (i.e, set of *terms*) which yields readme-*term* association. The outcome of this step is a *numeric vector* representation for each repository. More formally, given a set of repositories *R*, we extract a set of characteristics of features *F*. A repository a can be represented by a vector $x^{(a)} \in \mathbb{R}^{|F|}$ where each of its elements $x_i^{(a)}$ denotes the association strength of the repository to *feature* (topic) i.

C. Synergy Quantification

Now that we have a vector representation for each repository, we describe here the approach for finding software project pairs with synergies using these vectors.

As mentioned previously, we base our synergy scoring on the ABC model of Literature-Based-Discovery. In the domain of software projects, A, B, and C represent software features. We first formalize the problem, and we define three requirements for our synergy scoring approach. We then suggest a random walk-based ranking function for repository pairs, which comply with these requirements.

1) Problem formalisation: From the previous step, each repository a is represented by a vector $\mathbf{x^{(a)}} \in \mathbb{R}^{|F|}$ with elements $x_{a,i}$ giving the association strength of a to feature i. All vectors are assembled as the rows of the repository-feature matrix $\mathbf{X} \in \mathbb{R}^{|R|\times|F|}$.[1] An example is shown in Figure 3.2.

2) Synergy Scoring Requirements: Synergy is a subjective notion; therefore, we suggest three main requirements for quantifying synergy between software project pairs:

- **R1 - Potential trend.** Two software projects should each bring features that were not combined by many other projects before (create a new trend). More formally, high values $m_{i,j}$ in the inverse feature overlap matrix $\mathbf{M} = exp(-\mathbf{X^T X})$ [2] (Figure 3.1) has high values for features pairs that are not frequently present in the same projects.
- **R2 - Potential complementary features.** For two software projects a and b to bear potential to create new directions when combined, it is required that one has strong associations to a subset of features for which the other has weak associations. This means for two complementary repositories a and b $\exists_{i,j \in F} : x_{a,i} \uparrow \wedge x_{b,i} \downarrow \wedge x_{a,j} \downarrow \wedge x_{b,j} \uparrow$. The dissimilarity, and thus the potential to have complementary features, of repositories is summarized in the matrix $\mathbf{X^-} = \mathbf{1} - \mathbf{X}$.

[1] $\mathbf{X}$ is always row normalized.
[2] For the sake of simplicity we denote $exp(-\mathbf{X^T X})$, the element-wise application of the exponential function to the negative values of the matrix $\mathbf{X^T X}$

- **R3 - Similarities between projects.** R1 and R2 alone are not sufficient to discover the synergy between software project pairs because they can lead to matches between projects from very different domains and purposes. Thus, two projects that benefit from each other should also have some common characteristics. This can be expressed by the row-normalised repository-feature overlap matrix $\mathbf{O} \in \mathbb{R}^{|R|\times|R|}$ (Figure 3.3).

3) Synergy Scoring as Ranking functions: We define a synergy ranking function for pairs of software repositories that comply with the requirements defined above based on a restricted random walk on a heterogeneous graph of repositories and features similar to the ones shown in Figure 3, when the matrices are treated as transition probability matrices.

The transition probability matrices $\mathbf{P(X)} \sim \mathbf{X}$ and $\mathbf{P(X^T)} \sim \mathbf{X^T}$ assign probabilities to go from a repository to a feature or vice versa, respectively. The probability of moving from a feature node in the graph (Figure 3.1) to another feature node is given by $\mathbf{P(M)} \sim \mathbf{M}$.

The idea of random walk-based ranking is that a random walker starts at a repository a and randomly jumps to an associated feature i with probability $p(x_{a,i})$. In the next step, it jumps, with probability $1-d$, to another repository that is also affiliated with feature i, which accounts for commonalities between the two. With probability d, it jumps (or explores) another feature not well related to a and j by moving according to the probability $p(m_{i,j})p(x^{-}_{a,j})$. From there, it discovers another repository b with high affiliation to feature j (according to the probability $p(x_{b,j})$). The resulting matrix equation for synergy scores of all repository pairs is:

$$\mathbf{Q_{rw}} = (1-d)\mathbf{P(X)P(X^T)} + d(\mathbf{P(X)P(M)} \circ \mathbf{X^-})\mathbf{P(X^T)} \quad (1)$$

The jumping probability d can be adjusted to balance between finding similar repositories and exploring new features.

IV. Experiments

In this section, we first describe the dataset used in our experiments (Section IV-A) and then we describe our experiments.

A. Data

For our experiments, we exploit GitHub open-source repositories. We use GitHub repositories because of the availability of the data and tools to extract their metadata and readme files. We limit the extracted repositories to one main programming language, Python, to control the variability of programming language–software features dependability, which is outside the scope of our work. We use the latest dump[3] from the GHTorrent dataset (Gousios, 2013 [6]). To ensure high repository quality, we rely on the number of "Watchers" ($>$ 50) for each repository. "Watchers" are GitHub users who have asked to be notified of activity in a repository.[4] So, the high number of watchers reflect repositories with high quality or ones relevant to the community. Also, to ensure recency, we fetch repositories that are still available on GitHub, and were updated in the recent year. Lastly, we fetch the readme files using PyGithub[5] where we end up with 20,590 repositories' readme files.

TABLE I
PRE-PROCESSING OF THE 20,590 SOFTWARE REPOSITORIES FROM GITHUB.

Prepossessing	**Repositories**	**Readme Sections**
repositories with readme files	19,797	169,521
repositories with *WhatWhy* sections	14,065	28,932
repositories with English *WhatWhy* content	13,264	24,988

B. Software Features Extraction

As mentioned in Section III-A, we classify 20,590 readme files containing 169,521 sections using the READMEClassifier. Table I shows the total number of repositories and sections after each pre-processing step. Based on the READMEClassifier, only 14,065 have at least one *WhatWhy* section. We, then, filter out the non-English sections by using the *lang-detect* Python library [7]. We end up with 13,264 readme files (24,988 *WhatWhy* sections), which are used in the subsequent steps.

C. Software Features Modelling

In this section, we transform the 13,264 readme files, that include only *what-why* sections, into a numeric vector by using topic modeling techniques, as described in Section III-B. We use Mallet latent Dirichlet allocation (LDA) [8], [9]. We choose the optimal k (pre-set number of topics) configuration based on the coherence value, which assesses the quality of the learned topics by measuring the degree of semantic similarity between high-scoring terms in a topic.

LDA is a generative probabilistic model for a collection of discrete data such as text corpora (here, readme files). The model defines a set of topics to describe a corpus. Each document is modeled as a finite mixture over an underlying set of topics that are represented as a mixture of terms. Here, the association vectors of a *what-why* section of a repository's readme file to topics build the repository-*feature* matrix X used in the next phase.

First, we pre-process the readme files by removing the stop-words and lemmatizing the content. After that, we train Mallet LDA on the readme files (*WhatWhy* sections) where the number of topics (k) must be predefined. So, we train models with k ranging between 40[6] and 150 topics. To define the optimal k, for each value, we calculate the average of topic coherence [10] values of the inferred topics. Figure 4 plots the

[3]At the time of writing this paper: https://ghtorrent.org/downloads.html dump mysql-2019-06-01.

[4]Watchers definition is stated here: https://www.metrics-toolkit.org/github-forks-collaborators-watchers/.

[5]PyGithub library: https://github.com/PyGithub/PyGithub.

[6]We do not go below 40 because we need a handful of features for the synergy scoring later.

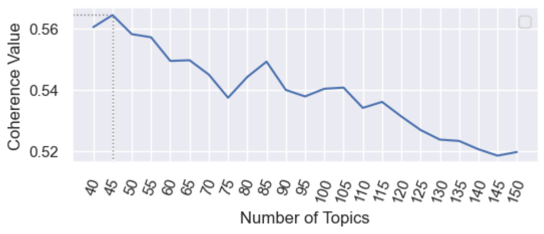


Fig. 4. Coherence values generated for each number of topics, k, trained using Mallet latent Dirichlet allocation (Mallet-LDA).

TABLE II
DISTRIBUTION OF README FILES FOR THE TOP 5 TOPICS (EACH REPRESENTED BY THE TOP KEYWORDS).

LDA	
Topic	**Readme Files Number**
model, train, dataset	856
image, target, alt	553
tool, scan, attack	526
page, html, content	484
api, application, request	454

average of the coherence values for different k. As we see, the topics are most coherent where $k = 45$. Also, Table II gives a sample of the topics' terms.

D. Synergy Quantification

Using the modeled features from LDA, we apply here the ranking algorithm, random walk, with different configurations of the jumping probability d, $d = [0.0 - 0.5]$. Using each configuration, we calculate the score of each repository-pair, sorted in descending order, in our dataset. We evaluate these models in the next Section (V).

V. EVALUATION

A first observation is that random walk-based ranking with a larger jumping probability ($d > 0.2$) results in a similar set of repositories r_b that are rated to have high synergy with any other repository r_a. Therefore, we only evaluate 1) a model with random-walk jumping probability $d = 0.0$ (focusing on similarity only), 2) with jumping probability $d = 0.2$ to see the effect of exploring similar yet more different features, and 3) random selection of repository pairs (baseline). We hypothesize that our models (1 and 2) identify synergy between repository pairs better than the random pair selection.

Due to the absence of ground truth data and comparable methods, we conducted a study where we asked programmers with good proficiency in English (the language of readme files) and high knowledge of Git and Python[7] to rate repository pairs regarding their potential for bearing synergy[8]. We created a dataset of 90 repository pairs containing 30 non-overlapping top picks of the three models mentioned above ($d = 0.0$, $d = 0.2$, random selection). The 90 pairs were divided into three different batches, where each batch contained ten pairs of each model. Each batch was rated by three different raters. As a result, we obtained 270 evaluations.

As shown in Table III, we asked our raters to evaluate each repository pair by reading their readme files and then answering two questions. In question 1, we ask if there is a synergy between the pair. The possible answers ranged from *None* to *Strong*. Also, the raters briefly explained the rationale of their choices (question 2). The formulation of questions and the rating guidelines were refined in a pilot study prior to the main evaluation.

Table V shows that the majority agreement (2 out of 3) between our human raters is very high when synergy intensity is considered. However, the full agreement between all 3 raters is often not achieved, which indicates that there is a degree of subjectivity in human judgment.

VI. RESULTS

Table VI shows the count of the synergy evaluations for repository pairs selected by the different models. We observe that repository pairs picked randomly have only 23% (21 annotations) of annotations indicating synergy, whereas the repository pairs generated by our models have higher synergy reports of 66% and 58% for models 1 and 2, respectively. The difference to the baseline is significant with regard to the non-parametric Kruskal-Wallis Test [11] (not normally distributed data), $p < 0.001$. This was further confirmed in a post-hoc analysis using the Mann-Whitney test [12] with Bonferroni correction that showed that the repository pairs generated by model 1 and model 2 are significantly rated with higher synergy than the random pairs at $p < 0.001$ and effect sizes, r, of 0.30 and 0.39 respectively. However, the difference between models 1 and 2 is not significant. Table IV shows examples of repository pairs rated high by our models.

While overall, model 1 ($d = 0.0$), focusing on similarities of project pairs, have a higher agreement with human ratings, a closer look at the highest rated pairs by humans from the entire evaluation set shows a different picture. We define the discovery rate, $dr(p_{top})$, as the intersection of repository pairs belonging to the top p ($p \in [0, 1]$) fraction of repository pairs ranked by human annotators (top_{human}) and the algorithm (top_{algo}) relative to the number of top pairs, that is: $dr = \frac{|top_{human} \cap top_{algo}|}{p_{top} * n}$, where n is the number of all pairs in the evaluation dataset. For model 1, $dr(p_{top} < 2) = 0.0$. While for model 2 ($d = 0.2$), $dr(p_{top} = 0.1) = 0.1$ and $dr(p_{top} = 0.15) = 0.08$. This indicates that model 2 incorporates differences of features more strongly. Contrarily, model 1 scores higher synergy for pairs that have more redundancies, which makes them more obvious to the raters.

VII. CONCLUSION

This paper explored a novel approach for discovering synergies between software projects that may inspire innovations. To this end, we adapted the idea of Literature-Based Discovery (LBD), which aims at uncovering implicit knowledge by

[7]Raters studied for their bachelor/masters degree in English and have > 3 years experience in Python.

[8]Evaluation web application: https://synergy-annotation.herokuapp.com/reposynergy with username: *seke2021* and password: *seke2021*.

TABLE III
THE QUESTIONS THAT OUR RATERS HAD TO ANSWER AFTER READING THE *readme* FILES OF TWO REPOSITORIES.

#	Questions	Answers
1.	I see that there is synergy between the 2 repositories	a. None – No complementary or common features
		b. Weak – More common features than complementary
		c. Somewhat – Some features can be merged
		d. Strong – Clear complementary features that lead to a new project
2.	Explain your choice(s) (Keep it short)	*Free text*

TABLE IV
EXAMPLES OF REPOSITORY PAIRS MAJORLY ANNOTATED AS HAVING STRONG SYNERGY (CLEAR COMPLEMENTARY FEATURES THAT LEAD TO A NEW PROJECT), RANKED BY LDA -RW FOR $d = 0.0$ AND $d = 0.2$.

Jumping Probability	Repository 1	Repository 2
$d = 0.00$	Flexible and scalable Django authorization backend for unified per object permission management	Core common behaviors for Django models, e.g. Timestamps, Publishing, Authoring, Editing and more.
$d = 0.2$	Python scripts and documentation for generating topographically accurate Minecraft maps from historical map scans	Blender python addon to increase workflow for creating minecraft renders and animations

TABLE V
MAJORITY AND FULL AGREEMENT BETWEEN HUMAN SCORING FOR REPOSITORY PAIRS SELECTED BY DIFFERENT MODELS.

	Synergy Intensity		Synergy vs. No Synergy
	Majority	Full	Full
LDA - Random Walk (d=0.0)	80%	20%	50%
LDA - Random Walk (d=0.2)	83%	20%	57%
random baseline	100%	50%	57%

TABLE VI
COUNTS OF THE RATED SYNERGIES FOR THE 90 REPOSITORY PAIRS IN OUR DATASET, FOR EACH ALGORITHM (LDA RANDOM WALK WITH $d = 0.0$, WITH $d = 0.2$ AND *random* BASELINE). EACH PAIR WAS RATED BY THREE RATERS.

	Synergy w Intensity				Has Synergy	
Algorithm	0	1	2	3	No	Yes
LDA - Random Walk (d=0.0)	31	12	25	22	31	59
LDA - Random Walk (d=0.2)	38	15	25	12	38	52
Baseline *random*	69	06	11	04	69	21

exploring similarities and differences of knowledge artifacts, to the software domain. Based on human rating evaluation for identifying synergy between pairs of software projects showed that it is possible to quantify synergy using projects' readme files only. Our results indicate that models focusing on similarities to identify synergy are slightly higher rated by humans.

However, in the original spirit of Literature-Based Discovery for novelty identification, it is acceptable that not every new knowledge combination leads to a useful finding. The same applies to systems that use our approach. If the aim is to build a discovery system that assists in identifying *new* directions for novel developments, one would put more emphasis on differences in software features. The developed methodology is, however, flexible enough to be configured to identify repository pairs that serve similar purposes.

REFERENCES

[1] C. McMillan, M. Grechanik, and D. Poshyvanyk, "Detecting similar software applications," in *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 2012, pp. 364–374.

[2] A. Sharma, F. Thung, P. S. Kochhar, A. Sulistya, and D. Lo, "Cataloging GitHub Repositories," in *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering - EASE'17*. Karlskrona, Sweden: ACM Press, 2017, pp. 314–319. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3084226.3084287

[3] Y. Zhang, D. Lo, P. S. Kochhar, X. Xia, Q. Li, and J. Sun, "Detecting similar repositories on GitHub," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. Klagenfurt, Austria: IEEE, Feb. 2017, pp. 13–23. [Online]. Available: http://ieeexplore.ieee.org/document/7884605/

[4] D. R. Swanson, "Undiscovered public knowledge," *The Library Quarterly*, vol. 56, no. 2, pp. 103–118, 1986.

[5] G. A. A. Prana, C. Treude, F. Thung, T. Atapattu, and D. Lo, "Categorizing the Content of GitHub README Files," *Empirical Software Engineering*, vol. 24, no. 3, pp. 1296–1327, Jun. 2019. [Online]. Available: http://link.springer.com/10.1007/s10664-018-9660-3

[6] G. Gousios, "The ghtorrent dataset and tool suite," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, ser. MSR '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 233–236. [Online]. Available: http://dl.acm.org/citation.cfm?id=2487085.2487132

[7] S. Nakatani, "Language detection library for java," 2010. [Online]. Available: https://github.com/shuyo/language-detection

[8] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[9] A. K. McCallum, "Mallet: A machine learning for language toolkit," 2002, http://mallet.cs.umass.edu.

[10] M. Röder, A. Both, and A. Hinneburg, "Exploring the space of topic coherence measures," in *Proceedings of the eighth ACM international conference on Web search and data mining*, 2015, pp. 399–408.

[11] W. H. Kruskal and W. A. Wallis, "Use of ranks in one-criterion variance analysis," *Journal of the American statistical Association*, vol. 47, no. 260, pp. 583–621, 1952.

[12] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.