Nalanda

a socio-technical graph platform for building software analytics tools at enterprise scale

Maddila, Chandra; Shanbhogue, Suhas; Agrawal, Apoorva; Zimmermann, Thomas; Bansal, Chetan; Forsgren, Nicole; Agrawal, Divyanshu; Herzig, Kim; Van Deursen, Arie

DOI
10.1145/3540250.3558949

Publication date
2022

Document Version
Final published version

Published in
ESEC/FSE 2022 - Proceedings of the 30th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering

Nalanda: A Socio-technical Graph Platform for Building Software Analytics Tools at Enterprise Scale

Chandra Maddila
chandu.maddila@gmail.com
Microsoft Research
USA

Suhas Shanbhogue
suhas0711@gmail.com
Microsoft Research
India

Apoorva Agrawal
t-aagraw@microsoft.com
Microsoft Research
India

Thomas Zimmermann
tzimmer@microsoft.com
Microsoft Research
USA

Chetan Bansal
chetanb@microsoft.com
Microsoft Research
USA

Nicole Forsgren
niforsgr@microsoft.com
Microsoft Research
USA

Divyanshu Agrawal
divagrawal@microsoft.com
Microsoft Research
India

Kim Herzig
kimh@microsoft.com
Microsoft
USA

Arie van Deursen
Arie.vanDeursen@tudelft.nl
Delft University of Technology
The Netherlands

ABSTRACT

Software development is information-dense knowledge work that requires collaboration with other developers and awareness of artifacts such as work items, pull requests, and file changes. With the speed of development increasing, information overload, and information discovery are challenges for people developing and maintaining these systems. Finding information about *similar* code changes and *experts* is difficult for software engineers, especially when they work in large software systems or have just recently joined a project. In this paper, we build a large-scale data platform named Nalanda platform to address the challenges of information overload and discovery. Nalanda contains two subsystems: (1) a large-scale socio-technical graph system, named *Nalanda graph system*, and (2) a large-scale index system, named *Nalanda index system* that aims at satisfying the information needs of software developers.

To show the versatility of the Nalanda platform, we built two applications: (1) a software analytics application with a news feed named MyNalanda that has Daily Active Users (DAU) of 290 and Monthly Active Users (MAU) of 590, and (2) a recommendation system for related work items and pull requests that accomplished similar tasks (*artifact recommendation*) and a recommendation system for subject matter experts (*expert recommendation*), augmented by the Nalanda socio-technical graph. Initial studies of the two applications found that developers and engineering managers are favorable toward continued use of the news feed application for information discovery. The studies also found that developers agreed that a system like Nalanda artifact and expert recommendation application could reduce the time spent and the number of places needed to visit to find information.

ESEC/FSE '22, November 14–18, 2022, Singapore, Singapore

ACM ISBN 978-1-4503-9413-0/22/11…$15.00
https://doi.org/10.1145/3540250.3558949

CCS CONCEPTS

• **Software and its engineering → Programming teams**.

KEYWORDS

Collaborative software development, Socio-Technical Graphs, Recommender Systems for Software Engineering, Empirical study

ACM Reference Format:
Chandra Maddila, Suhas Shanbhogue, Apoorva Agrawal, Thomas Zimmermann, Chetan Bansal, Nicole Forsgren, Divyanshu Agrawal, Kim Herzig, and Arie van Deursen. 2022. Nalanda: A Socio-technical Graph Platform for Building Software Analytics Tools at Enterprise Scale. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22), November 14–18, 2022, Singapore, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3540250.3558949

1 INTRODUCTION

Building software is a highly collaborative process that requires awareness of the activities by many different stakeholders and interaction with many different artifacts such as files, pull requests, and work items. At the same time, large-scale software development creates lots of data about how people work with each other and with software artifacts. As a consequence, finding information can be hard, especially when software engineers work on large software projects with thousands of files and team members. A lot of times this knowledge about software development activity and expertise is hidden in the form of software development process data and the interaction map between stakeholders and artifacts. This data is hard to mine and represent in a form that allows practitioners to build applications on top of this data. This is primarily due to the scale at which this data is generated and the fact that this data is scattered across disparate data sources and systems. Therefore, it is hard to take full advantage of this data and extract hidden knowledge,

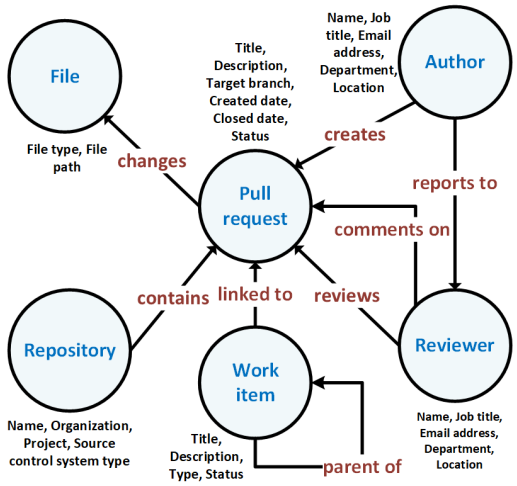

Figure 1: Nalanda's Graph Schema

without employing a plethora of tailored tools, customized for each source control system and the software development environment.

Socio-technical data that captures social and technical aspects of software development [39] is often captured in graph structures. For example, the Hipikat tool builds a project memory from past activities to support newcomers with software modification tasks [19]. In 2010, the Codebook framework was introduced with a focus on discovering and exploiting relationships in software organizations to support inter-team coordination [11]. Codebook provided a graph and a query language to support a wide range of applications: find the most relevant engineers, find out why a recent change was made, and general awareness of engineering activity [12]. Codebook was built for a single team with 420 developers only.

With the advent of cloud services, the scale at which software development happens and the volume of data generated during the software development process increased significantly [30, 34]. To address the challenges that come with scale, in this paper, we present a large-scale software analytics data platform named Nalanda[1] which is built on top of the software development activity data and the artifacts. Nalanda builds a socio-technical graph at *enterprise scale*, with thousands of repositories. Additionally, the Nalanda index system, helps with the search and recommendation of software development artifacts and the experts. Nalanda stores its graph in a native graph database and optimizes heavily to query complex relationships so that software analytics applications can operate directly on the graph via cloud services and a high degree of performance.

The Nalanda platform, which is a generic and *enterprise scale* software analytics data platform consists of two subsystems: the Nalanda graph system, which provides a *large scale* socio-technical graph of software data, and the Nalanda index system, which is an *enterprise scale* index system that can be used to support a wide range of software engineering tasks such as recommendation and search. Nalanda scales to enterprise-scale data from 6,500 repositories. The socio-technical graph has 37,410,706 nodes and 128,745,590 edges (The schema of the Nalanda graph is shown in Figure 1.) The index system contains 8,079,748 documents.

To show the *versatility* of the Nalanda platform, we describe two tools that have been built on top of the Nalanda platform and *deployed* at Microsoft: A software analytics news feed application built on top of the Nalanda graph system named MyNalanda, and a novel recommendation system (Nalanda artifact and expert recommendation application) leveraging the socio-technical graph for ranking the recommendations.

The goal of this paper is to describe the design, implementation, and deployment of the Nalanda graph system, the index system, and two successful applications (MyNalanda and the Nalanda artifact and expert recommendation application) built and deployed at Microsoft. We also share details about the extensive analyses and user studies that we conducted to evaluate the perceived usefulness of MyNalanda and Nalanda artifact and expert recommendation application from our deployments at Microsoft. Additionally, we share insights from building the Nalanda platform, MyNalanda, and the Nalanda artifact and expert recommendation application.

To that end, we explain the construction of the Nalanda graph system in Section 2 and the index system in Section 3. We explain the applications built leveraging these two systems i.e., MyNalanda in Section 4 and the Nalanda artifact and expert recommendation application in Section 5.

2 BUILDING THE NALANDA GRAPH

Key challenges in the construction of the Nalanda Graph are scale and consistency. In this section, we lay out what content we store in the Nalanda graph, from which sources we collect the data, and how we ensure that the graph is kept up to date and consistent as hundreds of thousands of events from thousands of repositories arrive on a daily basis.

2.1 Nalanda's Graph Schema

Nodes in the Nalanda graph represent the actors or entities involved in the software development life cycle, while the edges represent the relationships that exist between them.

Each node in the Nalanda graph has a type associated with it and attributes specific to that node type, as listed in Figure 1. The central node is the Pull Request, which has incoming edges from Author, Reviewer, Work item, and Repository nodes, and has a outgoing edges to File nodes changed by the pull request. A developer takes the role of an author when they make source code changes and submit pull requests and they assume the role of a reviewer when they perform code reviews. These are represented as user nodes in the Nalanda graph with different edge types but are listed here as two different nodes in Figure 1 for clarity. For File nodes, different types are distinguished, including source code, configuration, and project files. Files are edited by the authors via pull requests. Files are represented as nodes in the Nalanda graph with a second-order relationship established between the user and file nodes via a Pull request node.

Edges in the Nalanda graph represent the relationships between various actors and entities. Like nodes, edges can be of different types and can have properties associated with them. An edge is created between an author node and a pull request node when a

[1]Nalanda is named after an ancient university and knowledge center located in India. It is famous for its huge corpus of scriptures, books, and knowledge repositories.

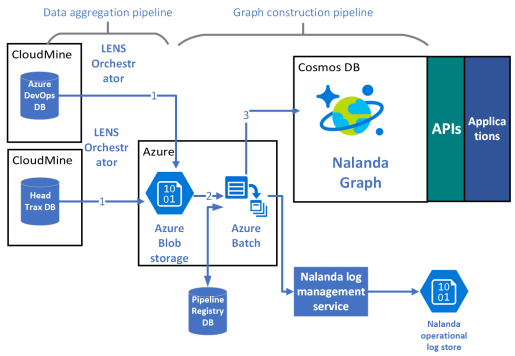

Figure 2: Nalanda's data collection and graph construction architecture

developer creates a pull request. Similarly, an edge is established between the reviewer and the pull request nodes when a developer is assigned a code review. A linked to edge is created when developers link a pull request to a work item, commonly done in Azure DevOps [3] to connect earlier, related, pull requests to new issues. Likewise, a parent of edge is created between two work items if they are linked by the developers with a parent-child relationship in Azure DevOps. Finally, a reports to edge is created between two user nodes if one of them is the reporting manager of the other.

2.2 Data Collection and Graph Construction

The Nalanda platform architecture is shown in Figure 2. The primary source of data for the Nalanda graph is Azure DevOps. Instead of directly crawling the Azure DevOps system for data, we leverage an intermediate data source called CloudMine [20]. The Nalanda platform takes the raw event data from CloudMine and processes it to create the nodes and edges of the Nalanda graph. The graph can be queried using the APIs we provide, or directly by means of the graph query language Gremlin [8].

The platform builds upon Azure [4]: key services used include Azure batch, Azure CosmosDB, Azure SQL Server, Azure Blob Storage, and Lens explorer [9]. As shown in Figure 2, the Nalanda platform is built using two independently operated pipelines: a data aggregation pipeline and a graph construction pipeline, as explained below.

2.2.1 Nalanda's Data Aggregation Pipeline. As indicated as step 1 in Figure 2, the data aggregation pipeline is responsible for fetching data from different data sources (most notably CloudMine) and making it available for the graph construction pipeline to process.

We use Lens Orchestrator [9] for orchestration and scheduling purposes. Lens has the ability to connect to multiple data sources and systems and move data around. In the aggregator pipeline, Lens first connects to the CloudMine data store (which is hosted on Cosmos [4]) and executes Scope scripts to gather data from various data streams, such as the pull request stream, the work item stream, the code review stream, etc. Lens saves the aggregated data in the form of CSV files in Cosmos. Then, Lens connects to Azure Blob Storage to temporarily store these CSV files for further processing. This intermediate store is required as CloudMine does not allow any other service (except Lens) to connect to and access the data files for security and compliance reasons.

Table 1: Comparison of pipeline run time and number of records with the increase in number of repositories

Mode	Run time		# Records to process	
	350 repos	6500 repos	350 repos	6500 repos
Bootstrap	9 hrs	28 hrs	1.05M	7.41M
Incremental	10 min	20 min	10K	57K

Additionally, we use the Lens job scheduling utilities to configure a job in Lens to run once every eight hours to pull the latest data from CloudMine and save it to the Azure Blob Storage.

2.2.2 Nalanda's Graph Construction Pipeline. Once the data is available in Azure Blob Storage, we process it using an Azure batch job to construct the Nalanda graph (Steps 2 and 3 in Figure 2). We use Azure CosmosDB as our graph data store.

When the batch job discovers new data files as generated by the aggregation pipeline, it updates the pipeline registry with new file information. This includes file names, size, timestamp, whether the file contains data from the bootstrap or the incremental stream, file processing status, processing duration, etc. The pipeline registry is a SQL database whose purpose is to serve as a transactional store for the graph construction pipeline. We create one row in the pipeline registry database for every file discovered. After the new data is downloaded, for each data file the corresponding node and edges are added to, deleted from, or updated in the Nalanda graph. Once all data for a file is read, its registry status is set to "completed".

The graph construction pipeline operates in two modes: bootstrap mode and incremental mode.

Bootstrap mode. This mode helps ingest all of a repository's data, from repository creation time until when it is run. Typically, we run this mode for a repository when it is being onboarded onto the Nalanda graph platform for the first time.

Incremental mode. This mode helps keep the data in the Nalanda graph updated without needing to read the massive original streams of CloudMine (whose size is in the orders of hundreds of Terabytes), using the incremental streams offered instead (with sizes in the order of tens of Gigabytes). We run the incremental pipeline once every eight hours.

These separate modes offer the flexibility to bootstrap any new or existing repository data in an independent and asynchronous manner. Furthermore, the separation of bootstrap and incremental pipelines offers substantial performance improvement in terms of run time and resource utilization, as illustrated in Table 1.

Refreshing the data by querying the original streams of CloudMine each time the pipeline is run, takes 28 hours for 6,500 repositories. As the incremental streams are substantially smaller, each incremental job finishes in 20 minutes, yielding an improvement of 98.8% in pipeline run time, for each pipeline run. Note that for an increase of the number of repositories by a factor of 20, the run time for the bootstrap mode was increased by a factor of 3 only. This is an effect of the careful design and implementation of the data pipeline by massively parallelizing the data processing code and enabling distributed processing on multiple Azure batch nodes.

Table 2: Nalanda node and edge types and their prevalence

Node type	Count
file	14,537,998
text	12,104,427
pull request	7,568,949
work item	3,067,754
user	131,578
repository	6,500
Total nodes	**37,410,706**

Edge type	Count
changes	65,706,621
reviews	39,447,635
creates	7,569,086
contains	7,337,036
linked to	7,094,597
parent of	843,728
comments on	746,887
Total edges	**128,745,590**

2.3 Data consistency and Self-healing

The Nalanda graph platform is a distributed system that works with multiple external data sources and large-scale data processing systems, which are prone to introduce data inconsistencies. Data gaps can manifest due to various factors related to infrastructure and availability of the CloudMine crawlers.

Detecting and remediating data gaps in such a massive distributed system is not a trivial task. We devised a novel self-healing system that detects data gaps and consistency issues proactively and performs self-healing. This helps the pipeline to guarantee data consistency irrespective of the failures manifested in external data sources such as CloudMine. The self-healing system uses the pipeline registry to monitor pipeline states and can switch from incremental to bootstrap mode if this is warranted.

When an incremental pipeline is run, the timestamp of the oldest record to be processed is compared with the timestamp of the last successful pipeline run. If the difference between these timestamps is bigger than three days (the period for which incremental streams hold their data), this means a data gap has occurred. To address this, the bootstrap mode is triggered for the repositories involved, and ongoing incremental pipelines are halted. Furthermore, the pipeline registry is updated to indicate that bootstrapping is taking place, thereby locking new incremental jobs.

2.4 Scale

The Nalanda graph has been designed to accommodate thousands of repositories. At the time of writing, it holds the software development activity data from 6,500 repositories at Microsoft. We ingest data starting from January 1, 2019, or from more recent repositories when their first pull request is created.

To keep its graph up to date, the Nalanda platform processes 500,000 events per day. These events include new pull requests, updates or commits on those pull requests, pull request state changes, code review assignments, and code review comments. At the time of writing, the Nalanda graph contains 37 million nodes and 128 million edges as detailed in Table 2.

3 INDEXING NALANDA FOR INFORMATION RETRIEVAL

Many nodes in the Nalanda Graph contain text. To facilitate *search* over such text at the Nalanda scale, we need to create appropriate indexes. The actual indices needed may depend on the specific applications built on top of the Nalanda graph. In this section, we discuss the indices we create and how we ensure they remain up to date at scale.

For every search, we use the BM25 algorithm [35] for determining text similarity between a query and documents (pull requests, work items, ...) and ranking the results. BM25 is a bag-of-words model developed based on the probabilistic retrieval framework [1]. For a given query Q containing keywords q_1,...,q_n, the BM25 score for a document D is:

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i,D)\cdot(k_1+1)}{f(q_i,D)+k_1\cdot(1-b+b\cdot\frac{|D|}{\text{avgdl}})} \quad (1)$$

where $f(q_i, D)$ is q_i's term frequency in the document D, IDF(q_i) is q_i's inverse document frequency, $|D|$ is the length of the document D in words, and avgdl is the average document length in the text collection from which documents are drawn. k_1 and b are free parameters. We use standard recommended values ($b = 0.75$, $k_1 = 1.2$) for these constants [40].

3.1 The Nalanda Artifact Index

The Nalanda artifact index facilitates search through pull requests and work items. We index the metadata, titles, and descriptions of the artifacts (pull requests and work items). The metadata consists of elementary properties, namely project name, repository name, and organization name.

The Nalanda artifact index can be used to find relevant pull requests given a work item, feature, technical, or functional concept. Furthermore, with this index completed pull requests and work items can be used as a template and inspiration to solve similar problems. They provide code samples, expose code review comments, and help as informal documentation to learn the best practices. Additionally, they help understand the team or project-specific processes involved in getting such pull requests completed.

3.2 The Nalanda Expert Index

The Nalanda expert index is built to map subject matter experts (SMEs) to technical and functional skills. Experts can be of two types: functional and technical. Functional experts have expertise in specific functionality of a software product or service, such as the query optimizer in an RDBMS product or the ranker in a search engine product. Technical experts have expertise with a technology concept such as socket programming in Java.

The Nalanda expert index relies on a collaborative software development platform like Azure DevOps to mine and associate expertise with people. We match pull requests and work items (as also used for the artifact index) to their authors and contributors. The process of building the Nalanda expert index consists of the following steps:

(1) Find all pull requests and work items completed by a developer and extract key phrases from them. The key phrases include tokens in pull request minus English stopwords [5].
(2) Create a document using the data from step 1, which is the representation of a developer's skills.
(3) Repeat steps 1-2 for every developer and builds the corpus for the expert index

Similar to the Nalanda artifact index, we use the BM25 algorithm for querying the expert index corpus. The intuition is that if a developer makes frequent code changes related to a topic (functional or technical), they must be knowledgeable in that topic area. We represent the frequency of a topic in a developer's activity as term frequency in the BM25 index. Therefore, the more a topic appears in the document corpus constructed for that developer, the more weight that topic is given.

We leverage the Nalanda socio-technical graph for re-ranking the search results returned by the artifact and expert indices. A detailed analysis of the impact of employing the Nalanda graph in refining the search results is discussed in Section 5.3.

3.3 Scale

Building indices such as the artifact and expert index at scale is an expensive operation. We carefully crafted the system design to make the Nalanda index creation and refresh pipelines robust and tolerant to failures (details about implementation are explained in Section 5.2). The Nalanda search system has been built as a cloud-native service. This enables us to scale out the system horizontally with the increase in data and query volume. This also helps us in meeting high uptime Service Level Agreements (SLAs) requirements to move to production. Currently, the artifact index contains 8,018,320 documents and the expert index contains 61,428 documents.

We ingest data from 6,500 repositories. Our index data refresh pipeline, which runs once every week, completes in 65 minutes on average. The graph data refresh pipeline, which runs every 8 hours and finishes in 18 minutes. We optimized the API service to return the response in 1.7 seconds in accordance to the SLA requirements.

4 NALANDA APPLICATIONS (I): THE MYNALANDA PORTAL

The Nalanda graph and indexing platform can be used to build many applications to support software development teams and organizations in their daily work. The first application we discuss is MyNalanda, an online news feed in production at Microsoft, in which developers and managers alike can monitor ongoing software development activities.

4.1 MyNalanda Motivation

The motivation behind MyNalanda is that it is common practice for developers to work on multiple work items or pull requests at once. It is also common practice for developers at Microsoft to work on multiple source code repositories simultaneously. Microsoft does not have many large mono-repositories, but a lot of small or medium sized repositories. Keeping track of one's work items in their repository or across multiple repositories is a difficult and time consuming task. Moreover, these tools operate in a workitem-centric fashion, i.e., the primary goal of the tool is to search for and find a work item one is interested in.

By contrast, MyNalanda is a *developer-centric* news feed. Upon login, MyNalanda shows the activity (pull request, work item, code review) of a developer, from multiple repositories, in their homepage. Additionally, MyNalanda enables developers to discover what their teammates and other collaborators are working on without the hassle of going to different Azure DevOps repositories.

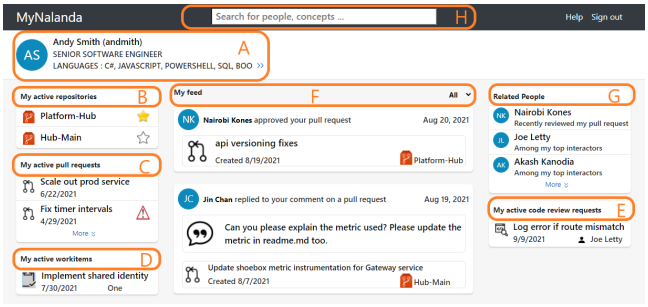

Figure 3: The MyNalanda homepage

4.2 The MyNalanda Homepage

For a user, the central hub in MyNalanda is their *homepage*. An example of such a MyNalanda homepage is shown in Figure 3. Its information is organized in the following sections:

News feed: The centralized news feed ('F' in Figure 3) is located in the middle of the page. The news feed shows events such as updates in pull requests, code review comments, and pull request status changes from all the repositories a developer works in. For managers, the news feed provides updates from their reports' activity.

User details: This section ('A' in Figure 3) provides details about developers, such as name, email address, job title, and their expertise (extracted from their software development activity data). This helps in facilitating easy discovery of developers' skills and their current projects.

Active items: There are separate sections for active repositories, pull requests, work items, and code review requests ('B'-'E' in Figure 3). Users of MyNalanda can prioritize the discovery of updates from these items by *following* or *unfollowing* them.

Related people: This section ('G' in Figure 3) visualizes who a developer collaborates with and how local software development communities are formed. A developer's collaborators include others who work together with the developer on a coding task or work item, or who are either being reviewed by that developer, or who are involved in reviewing a developers' pull request.

Search box: The search box ('H' in Figure 3) can be used to find developers and discover their activity. It also can help with searching for technical and functional concepts by leveraging the Nalanda artifact and expert indices (as explained in Section 3).

All elements in MyNalanda, such as pull requests, work items, people, and repositories have embedded URLs which take them to the corresponding item in Azure DevOps. This makes it easy for developers to navigate between MyNalanda and Azure DevOps.

Additionally, MyNalanda facilitates integration of other machine learning recommenders due to its extensible architecture. For example, overdue pull request are indicated with a subtle warning icon in the active pull requests section. This is powered by the Nudge machine learning models [32].

4.3 MyNalanda Usage

MyNalanda leverages the graph representation of the data (the Nalanda graph) and its schema design to navigate efficiently through complex relationships and find the content presented in various

sections. As a result, the MyNalanda homepage including the news feed and the other sections loads in less than a second. As quoted by one of the MyNalanda users *"It is simple, blazing fast to load, adapts to screen size"*.

Based on organic growth alone, MyNalanda reached 290 Daily Active Users (DAU), and 590 MonthlyActive Users (MAU), in the first six months of deployment of the Beta version at Microsoft.

4.4 MyNalanda Evaluation: Perceived Usefulness

To evaluate how developers and engineering managers perceive the usefulness of MyNalanda, we follow a mixed method research design involving interviews and surveys. With increasing sources of development-related information available, there remains an open question about how they want to access and integrate information about their own development activities and the development work done by their peers, and if current platforms are adequate. Through this evaluation we assess how MyNalanda matches the corresponding information needs.

4.4.1 *Evaluation Setup.*

Semi-structured Interviews. We conducted interviews to investigate information discovery and overload, and if users might use an interface like MyNalanda. Participants included five developers and two engineering managers; seven participants were men and zero were women. Semi-structured interviews were conducted remotely, and ranged from 30-45 minutes. Interview topics included interest in accessing information about their own and peers' development activity, information overload and how they typically get information (in both in-office and work-from-home contexts). We then showed a deployed version of MyNalanda and asked for reactions including if they would like it, what information they would find useful, and where they would want to see it. If the interviewee was an engineering manager, they were also asked what information they would be interested in seeing related to their team's work.

Immediately following the interviews, notes were taken by the interviewer to augment the transcription and interviews were coded for emergent themes. Following each subsequent interview, themes were revisited to see if any codes should be combined or separated. Once no new themes emerged (i.e., theoretical saturation), we concluded our interview phase. After seven interviews, themes remained consistent. Finally, we reviewed notable excerpts from all interviews and organized the themes by topic.

Surveys. Following interviews, we conducted surveys to validate and quantify the themes that emerged. Survey participants included full-time employees who were developers or engineering managers.

We designed our survey based on themes that emerged during our interviews, resulting in a 19-item instrument that took a median of 8.5 minutes to complete. Participants were shown their MyNalanda newsfeed (with their own development activity) and given the survey. Topics included demographics, usefulness of information included in their MyNalanda feed, information pain points and privacy concerns, and current and anticipated work location (e.g., office or work-from-home). We included items asking about usefulness of MyNalanda information, and preferences for possible features (based on jobs-to-be-done). We asked where respondents would like to see MyNalanda integrated (if at all), and their comfort level in sharing their development activity through something like MyNalanda. The full survey instrument is available online at research.microsoft.com [31].

The survey was sent to 2,000 people in total (1,400 developers and 600 engineering managers) with 144 responses (92 developers and 44 engineering managers), resulting in an 8% response rate after considering the 10% out of office responses. Our low response rate could be due to the fact that the survey is not a trivial one to fill in (cognitively and time it takes to complete the survey) and/or because we did not offer any incentives for participation. Of those respondents, 92 (67.65%) were developers (including software development engineers, senior software engineers, etc.) and 44 (32.35%) were engineering managers (including software engineering manager, software engineering lead, etc.). Our respondents included five women (6.17%), 73 men (90.12%), one who preferred to self-describe (1.32%), and two who preferred not to answer (2.63%). They reported an average of 10.15 years working at Microsoft, ranging from 0.6 to 29.9 years (standard deviation 7.13).

4.4.2 *Results of the Study.*

We first present findings from our semi structured interviews, with each noted as developer (D) or engineering manager (EM). We then present our survey results.

Interviews. Participants expressed two themes related to information needs: integration and overload. Information integration was echoed by many interviewees; we define this as having development-related information for self and others integrated into a single, easy-to-access interface. P1 (EM) and P2 (D) discussed easily *linking* design docs and associated artifacts like pull requests. P1 (EM) discussed the usefulness of graphic summaries for their teams' development activity across time periods; this reflects consolidation via visualization. P2 (D) called out that the ability to see detailed information about peers' work is helpful when coordinating work, and is not readily available.

P5 (D) spoke about viewing the information in different ways to deal with information overload and ensure they were not missing things: using a 'Most recent' view for chronologically-ordered information, an algorithmic 'Relevance' view to combine information across teams, and team-only view to further filter. This speaks to strategies they use to ensure keep up on development-related information both within and across teams they collaborate with. Similar sentiments were expressed by other interviewees.

Surveys. We asked participants to rate how useful each feature of MyNalanda was on a five-point Likert-type scale ranging from 1= Not at all useful to 5= Extremely useful. All items were optional; of those who took the survey, 85 answered questions about MyNalanda features (63 developers and 22 engineering managers). This allowed us to capture participant reactions to a real integrated information platform instead of a hypothetical one.

Table 3 lists the accumulated percentages of "Extremely useful" (Likert=5) and "Very useful" (Likert = 4) for each feature. Here, we see that *Active pull requests* (55.6% and 54.5% among developers and engineering managers, respectively) and *Active code review requests* (49.2% and 45.5% among developers and engineering managers, respectively) are the highest rated features, with *User details*

Table 3: MyNalanda survey Feedback

Feature	Cumulative (n=85)	Developers (n=63)	Managers (n=22)
Active pull requests (C)	55.3%	55.6%	54.5%
Active code review requests (E)	48.2%	49.2%	45.5%
Active repositories (B)	44.7%	46.0%	40.9%
Active work items (D)	34.1%	34.9%	31.8%
Feed (F)	28.2%	34.9%	29.1%
User Details (A)	24.7%	27%	38.2%
Related people (G)	20%	23.8%	9.1%

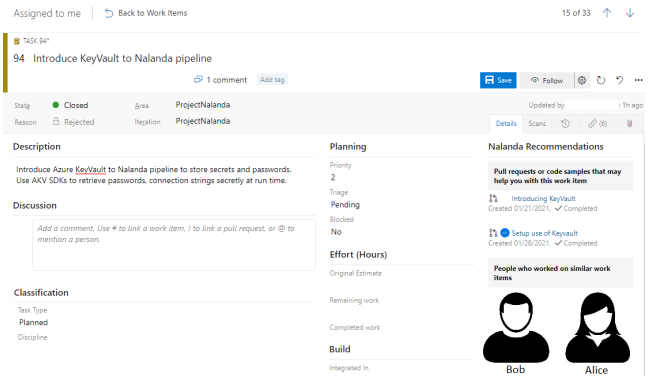

Figure 4: Nalanda recommendations in Azure DevOps

(27.0% and 38.2% among developers and engineering managers, respectively) and *Related People* (23.8% and 9.1% among developers and engineering managers, respectively) rated the lowest.

Based on this, we conclude that the active items (pull requests, code review requests, repositories, and work items) are the most valued features of MyNalanda, and that user details are primarily of interest to engineering managers.

5 NALANDA APPLICATIONS (II): ARTIFACT AND EXPERT RECOMMENDER

When a developer is working on a work item or feature, finding the relevant pull requests and code samples is one of the biggest pain points for developers [31]. The intensity of the problem multiplies in large organizations with teams working on multiple source code repositories. Similarly, finding functional and technical experts in large organizations is a difficult task. A lot of times, this involves the developers to mine git history, going through wiki pages, design documents, etc.

The Nalanda artifact and expert recommendation application implements a recommendation plug-in for Azure DevOps (AzDO) [3]. When a new work item is assigned to a developer, the plug-in triggers an API call to the Nalanda search API. The client passes the necessary input parameters, such as the search query (work item title and description), the work item owner, and the repository metadata. Upon receiving the results (artifact and expert recommendations), the client add recommendations to the work item page (in Azure DevOps). A sample work item recommendation page from live deployment (Beta version) is shown in Figure 4. Besides, through its easy-to-use APIs, the Nalanda artifact and expert recommendation application system powers other applications such as the *search box* in MyNalanda (as explained in Section 4.2).

When a work item or an issue is assigned to a developer, the Nalanda artifact and expert recommendation application uses a combination of title and description of the assigned work item as an input search query and provides recommendations about work items or pull requests that accomplished similar tasks. Additionally, the system also provides a list of subject matter experts whom a developer can reach out to seek help while working on that work item.

5.1 The Nalanda Ranking Algorithm

To construct the Nalanda Expert and Artifact recommenders, we devised a ranking algorithm consisting of three steps: 1) querying the artifact index (see Section 3.1) to get the relevant pull requests and work items, 2) querying the expert index (see Section 3.2) to get a list of relevant experts, and 3) re-ranking the results using the Nalanda graph. To that end, we take the following steps:

Step 1: We first construct a query as a combination of the title and description of the work item. Then, we tokenize the query using heuristics that we built for the software engineering domain, such as splitting strings into camel-cased or pascal-cased tokens. We also create n-gram based tokens since we found that bi-grams and tri-grams, such as `ImapTransfer` and `MailboxSyncEngine`, capture important information. Next, we filter out stop words [29].

We employ the BM25 algorithm, which takes care of prioritizing important tokens using Inverse Document Frequency (IDF) scores. It assigns more weight to the documents with a higher overlap with the search query tokens (term frequency), and calculates a relevance score as shown in Equation 1.

Step 2: We query the expert index, which contains one document per person. These documents contain the tokens mined from a developer's pull requests and work items history. We perform the same pre-processing explained in Step 1 on the query. The BM25 index returns a ranked list of experts.

Step 3: A heuristics-based filtering scheme is used to filter out results with relevance scores below a threshold. We determine the threshold empirically to set it at the 75th percentile of the relevance score distribution. We determined this value based on a series of experiments to optimize the accuracy of the results while reducing the size of the results set that is passed to the next step.

Step 4: We use the Nalanda socio-technical graph to assign proximity scores based on the edge distance between the person performing the search and the results returned by the BM25 index (obtained from Step-3). The proximity score is the length of the shortest path between two nodes. We use the proximity score to re-rank the results. For example, the proximity score is 1 if a pull request, work item, or people node is 1-edge away from a developer node, in their shortest path.

Step 5: We then pick the top-k results from the results set and return them to the user. Items that have a higher BM25 relevance score and are in close proximity to a developer are ranked higher. Furthermore, we use proximity score to break the tie between results where relevance scores are the same.

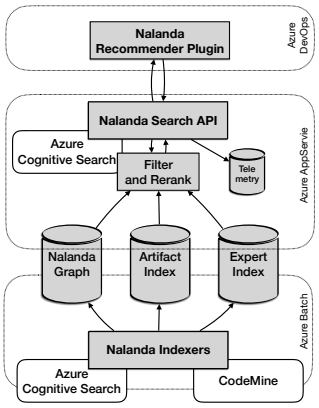

Figure 5: Nalanda artifact and expert recommendation application architecture

5.2 Implementation

We implemented the Nalanda search system on the Azure platform, with an emphasis on scalability to thousands of repositories. The underlying architecture is visualized in Figure 5.

5.2.1 Nalanda Indexers. The backbone of the recommendation system is the batch jobs creating and continuously updating the indices and the socio-technical graph, as discussed in Section 3 and shown at the bottom of Figure 5. We leverage CodeMine [21] to help us with aggregating source control system data from thousands of repositories.

We rely on Azure Cognitive Search (ACS) [2], which provides BM25 as a service, to store and access the indices. This helps us in alleviating the problems associated with service maintenance, uptime, and scale-out.

5.2.2 Search API. Given the indices and the socio-technical graph, the Nalanda Search API (shown in the middle of Figure 5) implements the search algorithm from Section 5.1. Each search query (typically initiated by the users in MyNalanda or through Azure DevOps) passed to the Nalanda search API is processed in real-time. The mean response time for the API call is 1.7 seconds.

The Nalanda API service asynchronously saves telemetry to an Azure SQL database without impacting the query performance. The telemetry includes the search query, user metadata, search results, click logs, and the API response time. We use this information to evaluate engagement and to improve the performance of the Nalanda search system and the API service.

5.3 Quantitative Evaluation

To understand the efficacy and usefulness of the Nalanda artifact and expert recommendation application, we conduct a large-scale offline evaluation and a user study.

5.3.1 Experiment Setup. We randomly sample 80,000 work items from the 6,500 repositories such that there are at least 10 work items selected from each repository. Subsequently, we use the title and description of each of these work items as the input to the Nalanda search API. We expect the right pull requests and people to be returned from the search API.

Since a recommendation system like ours does not exist in the company, we do not have a ground truth to conduct a large-scale evaluation. Therefore, we rely on the pull requests manually tagged by developers to the work items in Azure DevOps to build the evaluation dataset. To create the ground truth dataset for the expert recommendations, we leverage the private instance of StackOverflow deployed at Microsoft. Details about the index, ground truth, and test sets used for these experiments are shown in Table 4.

5.3.2 Results. We use two commonly used metrics for recommender systems: 1. Top K accuracy, which measures the number of times the correct item is found in the top K recommendations 2. Mean Reciprocal Rank (MRR), which calculates the reciprocal of the rank at which the first relevant document was retrieved [18].

Table 5 shows the results from the evaluation for different values of K. We can see that incorporating more attributes of the pull request, such as its title and description, improves both the MRR and accuracy considerably. Furthermore, re-ranking the results using the Nalanda graph also substantially improves the recommendations. Similar improvements can be noticed for the expert recommendation task too (Table 6).

Table 4: Evaluation Data Summary

	Artifact Recommendation	Expert Recommendation
Index data	Pull request title and description, socio-technical graph	Pull request title and description, socio-technical graph
Test set	Work Item title and description	Internal StackOverflow questions
Ground truth	Pull requests linked to the work item	People who answered the post and people who have answered at least five other questions with the same tag as the post
Data set	80K randomly sampled work items from the 6,500 repositories	10K randomly sampled questions and answers from StackOverflow

Table 5: Evaluation and Comparative study for Artifact Recommendation

	K = 3		K = 5		K = 10	
Indexed Properties	**Accuracy**	**MRR**	**Acc**	**MRR**	**Acc**	**MRR**
PR metadata	0.26	0.23	0.29	0.28	0.32	0.30
PR attributes						
+ PR title	0.38	0.36	0.43	0.41	0.51	0.48
+ PR description	0.49	0.47	0.53	0.51	0.60	0.59
+ socio-technical graph	**0.71**	**0.71**	**0.74**	**0.73**	**0.78**	**0.77**

Table 6: Evaluation and Comparative study for Expert Recommendation

	K = 3		K = 5		K = 10	
Indexed Properties	**Accuracy**	**MRR**	**Acc**	**MRR**	**Acc**	**MRR**
PR metadata	0.35	0.30	0.39	0.33	0.43	0.38
PR attributes						
+ PR title	0.51	0.46	0.54	0.49	0.59	0.53
+ PR description	0.60	0.54	0.64	0.59	0.69	0.61
+ socio-technical graph	**0.63**	**0.60**	**0.69**	**0.63**	**0.75**	**0.67**

Figure 6: Participants' evaluation of the Nalanda system

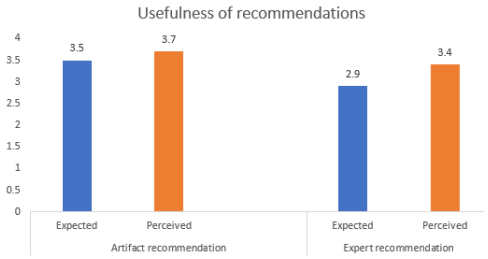

Figure 7: Expectations and perceptions of the Nalanda system

5.4 User Perception

We conducted a user study among developers regarding the usefulness of the recommendations. We selected ten participants (identified as P1–P10) from Microsoft to evaluate the recommendations on their recently completed work items.

5.4.1 Participants and Protocol. We conducted semi-structured interviews, which were conducted remotely, and ranged from 15-30 minutes. The average experience of the subjects is 7.7 years in the company and ranged from 10 months to 21 years.

We employ a one-group pretest-posttest pre-experimental design [16]. We used the Likert scale [37] for rating the responses. The respondents can provide their responses on a 1 to 5 scale, ranging from 'strongly disagree' to 'strongly agree'. We posed them the questions listed below.

When you are working on a work item, how useful would the recommendations be in completing the work item (on a scale of 1 to 5):

(1) You are going to refer to the work items and pull requests recommended as inspiration and informal documentation on accomplishing the work item.
(2) You would likely reach out to the recommended people for consultation on accomplishing the work item.

5.4.2 Results. **Do the users feel work item, pull request, and expert recommendations are useful?** The participants expressed that the Nalanda recommendation system can be a great value addition to the software development process. 60% of the participants rated 'agree' or 'strongly agree' when asked whether they find the artifact recommendations useful and 40% responded favorably to the expert recommendations.

Through question 2 we measure the difference between the expectations the participants had of a hypothetical recommendation system with the Nalanda artifact and expert recommendation application. This question measures the dependent variable for the user study (introduction of the Nalanda system). In Figure 6, the radar chart shows some differences between the participants' original expectations and their perception of the Nalanda recommendations.

These differences can be observed better in Figure 7, in which the averages of the rating are shown. The difference in expectation versus perception was more apparent with the expert recommendations compared to the artifact recommendations.

To offer an impression, we list some typical quotes (positive and negative) that we received from the developers.

> *"A tool like this will help greatly to understand the processes involved in pushing my changes through."*

> *"It is great to see the recommendations about people to talk to. My team is large, mostly remote, and I am new. So this is very helpful."*

> *"Finding people to talk to has never been a problem for me as I have been working in the same org for a while."*

6 DISCUSSION

6.1 Outlook

In the future, we anticipate both the Nalanda graph system and MyNalanda to be scaled out significantly inside Microsoft in terms of the number of repositories and users. Furthermore, we see opportunities for implementing the Nalanda graph platform as a service on top of the open-source software development data mined from systems like GitHub.

We also expect the Nalanda artifact and expert recommendation application to be employed at substantially more software development systems at Microsoft. Future research could entail including other types of useful recommendations such as internal and external documentation, tutorials, and recommendations from question-answer forums.

The rich socio-technical data in the Nalanda graph in combination with advances in deep learning and Graph Convolutional Networks (GCNs), hold promise for applications such as neural reviewer recommendations by leveraging the socio-technical structures. The Nalanda graph system lays the foundation to bring various techniques from the graph representation world [7], such as link prediction, social network analysis, etc. into the software engineering and analytics domain.

6.2 Threats and Limitations

6.2.1 Internal Validity. Conducting trustworthy experiments on data collected from thousands of repositories is challenging especially due to the problems of avoiding data leakage and obtaining credible ground truth. In our experiments, we addressed this (see Table 4). While our results are highly promising and an important first step, more experiments are needed to better understand the true nature of the graph's contribution to expert and artifact recommendation.

The risk of response bias is minimal in all our studies because all the participants of the user study are organizationally distant from the people involved in building this system. However, there remains a small chance that people in the user study may be positive about the system because they want to make the developers who are from the same company motivated and happy.

6.2.2 External Validity. The context of our evaluation and user studies is a software development company with a large number of developers. These developers work on a portfolio of products across many contexts and domains. By conducting a user study within one single company, we were able to control for factors like culture, tooling, frameworks, and programming languages. However, our results may not be generalizable across all developers in all contexts. Hence, our results are not verified in the context of other organizations or the open-source community. Therefore, our findings may be limited and warrant further research. Future work could investigate user interfaces, integrating our findings with design guidelines that span usability and technical and organizational complexity [27].

7 RELATED WORK

Graph Representations. Hipikat [19] is one of the earliest works to build a graph for software development entities like tasks, file versions, and documents, using a fixed schema. It was built for onboarding new hires quickly by providing easy access to relevant artifacts. Codebook [11] builds a prototype graph consisting of various software development entities. The data was mined from software repositories for a single team at Microsoft. Bhattacharya et al. [14] use graph representation of source code and bug tracking information to construct predictors for software engineering metrics like bug severity and maintenance efforts. Other applications of graph representations of software artifacts include visualizing relationships among project entities [38] and extracting changes in variability models [22]. Compared to all this work, the scale of the Nalanda graph is significantly larger with 37M nodes and 128M edges, and Nalanda has been designed to be actively used in a production setting.

Source Control Dashboards. GitHub offers a dashboard for developers on its homepage [6]. It displays the repositories, active pull requests, and issues (work items). A key limitation of the GitHub dashboard is there exists no notion of a "Team feed" in Github unlike MyNalanda. Team feed helps the discovery of the items worked on by other team members.

Information Needs. Ko et al. [28] studied information needs in colocated development teams. Fritz and Murphy [23] provide a list of questions developers ask for the most frequently sought-after information within a project. Information needs have also been studied in the context of change tasks [41], inter-team coordination [11] and software analytics [13, 15, 26] Through its applications, Nalanda can efficiently address most information needs related to people, code, and work items. For example, the answer to questions such as "Who is working on what" and "What are coworkers working on right now" is easily available in the MyNalanda application.

Artifact Recommendation systems for software developers. Recommendation systems for software engineering aim at assisting developers with activities such as code reusability, writing effective bug reports, etc. [36]. Tools like CodeBroker [42] help in finding the relevant code samples extracted from the standard Java documentation generated by Javadoc from Java source programs and deliver the suggestion to the Emacs editor. Anvik et al. [10] proposed a semi-automated method to assign bug reports to reporters based on their expertise using a machine learning algorithm. Mockus and Herbsleb[33] used quantity as a measure of expertise. Fu et al.[24] used the node2vec algorithm to convert file entities within projects into knowledge mappings. They proposed four features to capture the social relationships between developers. Devrec[43], a developer recommendation system, mines the development activities of developers in GitHub and StackOverflow to recommend collaborators for a given project. Hammad et al [25] use keywords from the textual content of commits. On the other hand, Canfora et al [17] use mailing lists and versioning systems to recommend experts for newcomers joining a software project. Compared to these approaches, the Nalanda artifact and expert recommendation application is designed to be highly scalable and provide responses in real-time.

8 CONCLUSION

In this paper, we seek to build a *large scale* software analytics data platform named Nalanda with two subsystems (the Nalanda graph systemand the Nalanda index system). The Nalanda graph system consists of a socio-technical graph encompassing the entities, people, and relationships involved in the software development life cycle. The Nalanda index system is an *enterprise scale* index system that can be used to support a wide range of software engineering tasks such as recommendation, and search.

We built the Nalanda graph system using software development activity data from 6,500 source code repositories. The graph consists of 37,410,706 nodes and 128,745,590 edges. To the best of our knowledge, it is the largest socio-technical graph built to date using private software development data. Similarly, the Nalanda index system contains 8,018,320 documents in its artifact index and 61,428 documents in its expert index with data ingested from 6,500 repositories at Microsoft.

The Nalanda platform and its applications (MyNalanda and Nalanda artifact and expert recommendation application) help in developing awareness of each other's work, and building connections between developers across repositories, while offering mechanisms to discover information while managing information overload. We also seek to address the problems of information discovery by finding related work items and experts for software developers.

Based on organic growth alone, MyNalanda has Daily Active Users (DAU) of 290 and Monthly Active Users (MAU) of 590. A preliminary user study shows that 74% of developers and engineering managers surveyed are favorable toward continued use of MyNalanda for information discovery. The Nalanda artifact and expert recommendation application, with the help of the socio-technical graph for customization, lifted the accuracy of artifact recommendations by 30.45 percentage points to 0.78. In a study with ten professional software developers, participants agreed that a system like Nalanda artifact and expert recommendation application could reduce the time spent and the number of places needed to visit to find information.

In the future, we anticipate both the Nalanda graph system and MyNalanda to be scaled out significantly inside Microsoft in terms of the number of repositories and users. We believe the systems and the techniques have applicability beyond Microsoft. Furthermore, we see opportunities for implementing the Nalanda graph platform as a service on top of the open-source data mined from platforms like GitHub.

REFERENCES

[1] 2009. The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends in Information Retrieval* 3 (01 2009), 333–389. https://doi.org/10.1561/1500000019

[2] Accessed 2021. Azure Cognitive Search. https://azure.microsoft.com/en-us/services/search/

[3] Accessed 2021. Azure DevOps. https://azure.microsoft.com/en-us/services/devops/.

[4] Accessed 2021. Azure services. https://azure.microsoft.com/en-us/services/

[5] Accessed 2021. English stop words. https://gist.github.com/sebleier/554280

[6] Accessed 2021. GitHub personal dashboard. https://docs.github.com/en/account-and-profile/setting-up-and-managing-your-github-user-account/managing-user-account-settings/about-your-personal-dashboard

[7] Accessed 2021. Graph Neural Network Techniques. https://neptune.ai/blog/graph-neural-network-and-some-of-gnn-applications

[8] Accessed 2021. Gremlin query language. https://docs.janusgraph.org/getting-started/gremlin/

[9] Accessed 2021. Lens Explorer. https://docs.microsoft.com/en-us/system-center/orchestrator/learn-about-orchestrator

[10] John Anvik, Lyndon Hiew, and Gail Murphy. 2006. Who should fix this bug? *28th International Conference on Software Engineering (ICSE)* 2006, 361–370. https://doi.org/10.1145/1134285.1134336

[11] Andrew Begel, Khoo Yit Phang, and Thomas Zimmermann. 2010. Codebook: Discovering and Exploiting Relationships in Software Repositories. In *Proceedings of the ACM/IEEE 32nd International Conference on Software Engineering*. Association for Computing Machinery, Inc. https://doi.org/10.1145/1806799.1806821

[12] Andrew Begel and Thomas Zimmermann. 2010. Keeping up with your friends: Function foo, library bar. dll, and work item 24. In *Proceedings of the 1st Workshop on Web 2.0 for Software Engineering*. 20–23. https://doi.org/10.1145/1809198.1809205

[13] Andrew Begel and Thomas Zimmermann. 2014. Analyze this! 145 questions for data scientists in software engineering. In *Proceedings of the 36th International Conference on Software Engineering*. 12–23. https://doi.org/10.1145/2568225.2568233

[14] Pamela Bhattacharya, Marios Iliofotou, Iulian Neamtiu, and Michalis Faloutsos. 2012. Graph-based analysis and prediction for software evolution. In *2012 34th International Conference on Software Engineering (ICSE)*. 419–429. https://doi.org/10.1109/ICSE.2012.6227173

[15] Raymond PL Buse and Thomas Zimmermann. 2012. Information needs for software development analytics. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 987–996. https://doi.org/10.1109/ICSE.2012.6227122

[16] Donald T. Campbell and Julian C. Stanley. 1963. Experimental and Quasi-Experimental Designs for Research.

[17] Gerardo Canfora, Massimiliano Di Penta, Rocco Oliveto, and Sebastiano Panichella. 2012. Who is Going to Mentor Newcomers in Open Source Projects?. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering* (Cary, North Carolina) *(FSE '12)*. Association for Computing Machinery, New York, NY, USA, Article 44, 11 pages. https://doi.org/10.1145/2393596.2393647

[18] Nick Craswell. 2009. *Mean Reciprocal Rank*. Springer US, Boston, MA, 1703–1703. https://doi.org/10.1007/978-0-387-39940-9_488

[19] Davor Cubranic, Gail Murphy, Janice Singer, and K.S. Booth. 2005. Hipikat: A project memory for software development. *Software Engineering, IEEE Transactions on* 31 (07 2005), 446 – 465. https://doi.org/10.1109/TSE.2005.71

[20] Jacek Czerwonka, Nachi Nagappan, Wolfram Schulte, and Brendan Murphy. 2013. CODEMINE: Building a Software Development Data Analytics Platform at Microsoft. *IEEE Software* (July 2013). https://doi.org/10.1109/MS.2013.68

[21] Jacek Czerwonka, Nachiappan Nagappan, Wolfram Schulte, and Brendan Murphy. 2013. CODEMINE: Building a Software Development Data Analytics Platform at Microsoft. *IEEE Software* 30, 4 (2013), 64–71. https://doi.org/10.1109/MS.2013.68

[22] Nicolas Dintzner, Arie Deursen, and Martin Pinzger. 2018. FEVER: An approach to analyze feature-oriented changes and artefact co-evolution in highly configurable systems. *Empirical Software Engineering* 23 (04 2018). https://doi.org/10.1007/s10664-017-9557-6

[23] Thomas Fritz and Gail C. Murphy. 2010. Using Information Fragments to Answer the Questions Developers Ask. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1* (Cape Town, South Africa) *(ICSE '10)*. Association for Computing Machinery, New York, NY, USA, 175–184. https://doi.org/10.1145/1806799.1806828

[24] Chenbo Fu, Mingming Zhou, Qi Xuan, and Hong-Xiang Hu. 2017. Expert recommendation in oss projects based on knowledge embedding. In *2017 International Workshop on Complex Systems and Networks (IWCSN)*. 149–155. https://doi.org/10.1109/IWCSN.2017.8276520

[25] Maen Hammad, Haneen Hijazi, Mustafa Hammad, and Ahmed Otoom. 2020. Mining expertise of developers from software repositories. *International Journal of Computer Applications in Technology* 62 (01 2020), 227. https://doi.org/10.1504/ijcat.2020.106581

[26] Hennie Huijgens, Ayushi Rastogi, Ernst Mulders, Georgios Gousios, and Arie van Deursen. 2020. Questions for data scientists in software engineering: a replication. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 568–579. https://doi.org/10.1145/3368089.3409717

[27] Pooya Jaferian, David Botta, Fahimeh Raja, Kirstie Hawkey, and Konstantin Beznosov. 2008. Guidelines for designing IT security management tools. In *Proceedings of the 2nd ACM Symposium on Computer Human interaction For Management of information Technology*. 1–10. https://doi.org/10.1145/1477973.1477983

[28] Amy J Ko, Robert DeLine, and Gina Venolia. 2007. Information needs in collocated software development teams. In *29th International Conference on Software Engineering (ICSE'07)*. IEEE, 344–353. https://doi.org/10.1109/ICSE.2007.45

[29] Hans Peter Luhn. 1960. Key word-in-context index for technical literature (kwic index). *American Documentation* 11 (1960), 288–295.

[30] Yuxing Ma, Chris Bogart, Sadika Amreen, Russell Zaretzki, and Audris Mockus. 2019. World of code: an infrastructure for mining the universe of open source VCS data. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 143–154. https://doi.org/10.1109/MSR.2019.00031

[31] Chandra Maddila, Apoorva Agrawal, Tom Zimmermann, Nicole Forsgren, Kim Herzig, Arie van Deursen, Chandra Maddila, Nicole Forsgren, and Kim Herzig. 2021. *Appendix to Nalanda: A Large-Scale Socio-Technical Graph of Entities and Relationships in Software Development Environment*. Technical Report MSR-TR-2021-28. Microsoft. https://aka.ms/MSR-TR-2021-28

[32] Chandra Shekhar Maddila, Sai Surya Upadrasta, Chetan Bansal, Nachiappan Nagappan, Georgios Gousios, and Arie van Deursen. 2021. Nudge: Accelerating Overdue Pull Requests Towards Completion. *CoRR* abs/2011.12468 (2021). arXiv:2011.12468 https://arxiv.org/abs/2011.12468

[33] 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 2002)*. IEEE, 503–512. https://doi.org/10.1109/ICSE.2002.1007994

[34] Rachel Potvin and Josh Levenberg. 2016. Why Google stores billions of lines of code in a single repository. *Commun. ACM* 59, 7 (2016), 78–87. https://doi.org/10.1145/2854146

[35] Stephen E. Robertson, Steve Walker, Micheline Hancock-Beaulieu, Aarron Gull, and Marianna Lau. 1992. Okapi at TREC. In *Proceedings of The First Text REtrieval Conference, TREC 1992, Gaithersburg, Maryland, USA, November 4-6, 1992 (NIST Special Publication, Vol. 500-207)*, Donna K. Harman (Ed.). National Institute of Standards and Technology (NIST), 21–30. http://trec.nist.gov/pubs/trec1/papers/02.txt

[36] Martin Robillard, Robert Walker, and Thomas Zimmermann. 2010. Recommendation Systems for Software Engineering. *IEEE Software* 27, 4 (2010), 80–86. https://doi.org/10.1109/MS.2009.161

[37] John Robinson. 2014. *Likert Scale*. Springer Netherlands, Dordrecht, 3620–3621. https://doi.org/10.1007/978-94-007-0753-5_1654

[38] Anita Sarma, Larry Maccherone, Patrick Wagstrom, and James Herbsleb. 2009. Tesseract: Interactive visual exploration of socio-technical relationships in software development. In *2009 IEEE 31st International Conference on Software Engineering*. 23–33. https://doi.org/10.1109/ICSE.2009.5070505

[39] Anita Sarma, Larry Maccherone, Patrick Wagstrom, and James D. Herbsleb. 2009. Tesseract: Interactive visual exploration of socio-technical relationships in software development. *2009 IEEE 31st International Conference on Software Engineering* (2009), 23–33. https://doi.org/10.1109/ICSE.2009.5070505

[40] Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. 2008. *Introduction to information retrieval*. Vol. 39. Cambridge University Press Cambridge.

[41] Jonathan Sillito, Gail C Murphy, and Kris De Volder. 2008. Asking and answering questions during a programming change task. *IEEE Transactions on Software Engineering* 34, 4 (2008), 434–451. https://doi.org/10.1109/TSE.2008.26

[42] Yunwen Ye and Gerhard Fischer. 2002. Supporting Reuse by Delivering Task-Relevant and Personalized Information. In *Proceedings of the 24th International Conference on Software Engineering* (Orlando, Florida) *(ICSE '02)*. Association for Computing Machinery, New York, NY, USA, 513–523. https://doi.org/10.1145/581339.581402

[43] Xunhui Zhang, Tao Wang, Gang Yin, Cheng Yang, Yue Yu, and Huaimin Wang. 2017. DevRec: A Developer Recommendation System for Open Source Repositories. 3–11. https://doi.org/10.1007/978-3-319-56856-0_1