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

Chandra Maddila
chmaddil@microsoft.com
Microsoft Research

Apoorva Agrawal
t-aagraw@microsoft.com
Microsoft Research

Thomas Zimmermann
tzimmert@microsoft.com
Microsoft Research

Nicole Forsgren
niforsgr@microsoft.com
Microsoft Research

Kim Herzig
kimh@microsoft.com
Microsoft

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 files. With the speed of development increasing, information overload is a challenge for people developing and maintaining these systems. In this paper, we build a large scale socio-technical graph to address challenges of information overload and discovery, with an initial focus on artifacts central to the software development and delivery process. The Nalanda graph is an enterprise scale graph with data from 6,500 repositories, with 37,410,706 nodes and 128,745,590 edges. On top of this, we build software analytics applications including a newsfeed named MyNalanda, named after an ancient university and knowledge centre located in India.

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. Several platforms such as GH Torrent [21, 22], CodeMine [16], and World of Code [27] have been built to enable the development of large-scale data-driven tools. Many companies have adopted an open source-like culture within their organizations (inner source) [37] or store the code and assets for many projects in the same repository (monorepos) [31]. These strategies have been found to improve maintainability of source code and productivity of software engineers [19] as well as extensive code sharing and reuse and collaboration across teams [31]. In the absence of a monorepo for an organization, extra effort is needed to create connections between people who work in different repositories to prevent silos of poor communication.

The Nalanda platform provides a large-scale socio-technical graph of software data. Nalanda supports AI-infused applications at scale with high performance. (Section 2)
Table 1: Comparison between Codebook and Nalanda

|                  | Codebook | Nalanda graph |
|------------------|----------|---------------|
| # Nodes          | 200,000  | 37,410,706    |
| # Edges          | 350,000  | 128,745,590   |
| # Files          | 9700     | 14,537,998    |
| # Work items     | 3400     | 3,067,754     |
| # People         | 420      | 131,578       |
| # Repositories   | 1        | 6500          |
| Time period of data | 6 months | 33 months    |

- The Nalanda platform is versatile and can be used to support a wide range of software engineering tasks such as awareness, recommendation, and search. (Section 3)
- Insights around information and privacy needs on socio-technical development information in a corporate setting. (Section 4)

2 NALANDA DESIGN AND DEPLOYMENT

Nalanda graph is built using software development activity data. In this section we provide details about the schema of the Nalanda graph, the graph data pipeline, and the scale.

2.1 Nalanda Graph Schema

The schema of the Nalanda graph is shown in Figure 1. The nodes represent the actors or entities involved in the software development life cycle, while the edges represent the relationships that exist between them.

2.1.1 Nodes. Each node in the Nalanda graph has a type associated to it and attributes specific to that node type. For the full list of nodes and attributes, we refer to Figure 1. When developers make changes to source code, they create pull requests and submit them for code review. A pull request is represented as a pull request node in the Nalanda graph. 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. Both of them are represented as user nodes in the Nalanda graph with different edge types (we listed them as two different nodes in Figure 1 for illustration purposes). A file can be of different types: a source code file, a configuration file, a project file, etc. 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 user and file nodes via a pull request node.

2.1.2 Edges. 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 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 ‘contains’ edge is created between a repository node and a pull request node when a pull request is created in a source code repository. A ‘linked to’ edge is created between a pull request node and a work item node when developers link a pull request to work item in Azure DevOps. A ‘comments on’ edge is created between a reviewer node and a pull request node if they add code review comments. A ‘parent of’ edge is created between two work item nodes if they are linked by the developers with a parent-child relationship in Azure DevOps. A ‘reports to’ edge is created between two user nodes if one of them is the reporting manager of the other.

2.2 Augmented socio-technical graph

Some of the node types like pull request, work item, and file have text tokens in them (such as a title, description, path, or name). These text tokens reflect functional and technical concepts of the system and its application domain. We capture the relationships between such concepts (represented by word tokens) and the entities and actors, by expanding the socio-technical graph to have text tokens represented as nodes in the Nalanda graph (as shown in Figure 2). We use a simple tokenizer, based on word boundaries, to generate word tokens from the text and remove the stop words [3]. Then, we link the text nodes that appear in a pull request text, work item text, and file text to the respective entities. That establishes a second order relation between text tokens and the user nodes (for authors and reviewers). We also establish edges between text nodes based on their co-occurrence in the pull request text by using Pointwise Mutual Information (PMI) [30].

2.3 Nalanda Platform Architecture

The Nalanda platform architecture is shown in Figure 3. 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 [16]. 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 [5].

The Nalanda platform builds upon Azure [2]. The key Azure services used are Azure batch, Azure CosmosDB, Azure SQL Server, Azure Blob Storage, and Lens explorer [6]. In this section, we explain how these services work together to realize the Nalanda platform.

As shown in Figure 3, the Nalanda graph platform is built using two independently operated pipelines: a data aggregation pipeline and a graph construction pipeline. Details about these two pipelines are explained in Section 2.4 and Section 2.5 respectively.

### 2.4 The Data Aggregation Pipeline

The data aggregation pipeline is responsible for fetching data from different data sources (most notably CloudMine) and make it available for the graph construction pipeline to process. This is indicated as step 1 in Figure 3.

CloudMine offers Azure DevOps data in two types of streams: the original ones, and incremental streams. The original streams contain all repository data since repository creation. The size of the original streams typically is in the order of hundreds of Terabytes. Incremental streams contain event data about the add, update or delete operations happened in Azure DevOps in the last three days. The size of the incremental streams typically in the order of tens of Gigabytes. We use both streams extensively to keep the data in the Nalanda graph consistent.

We use Lens Orchestrator [6] 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 [2]) 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.

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.

### 2.5 Graph construction pipeline

Once the data is available as CSV files in Azure Blob Storage, we process it using an Azure batch job to construct the Nalanda graph. We use Azure CosmosDB as our graph data store. We run the graph construction operation as an Azure batch service.

The separation of Bootstrap and Incremental pipelines offers substantial performance improvement in terms of run time and resource utilization. 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. More details about the performance of the pipeline are shown in Table 2. Note that for an increase of the number of repositories by a factor 20, the run time for the bootstrap mode was increased by a factor 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. We also leverage the performance optimizations offered by the Azure CosmosDB to maximize the throughput of our write operations.

Separating the bootstrapping activity as a different pipeline mode also offers us the flexibility to bootstrap any new or existing repository data in an independent and asynchronous manner.

### 2.5.2 Pipeline monitoring

For a large scale production pipeline such as Nalanda’s, it is important to set up a monitoring system that helps in getting notified as soon as possible when an issue is manifested. As shown in Figure 3, the Nalanda log management service writes a continuous log stream to an operational log store (hosted on Azure Blob Storage) for each pipeline run. As shown in Figure 4, the Nalanda monitoring service, which runs once a day, gathers logs from three streams listed below and sends an email digest of the issues:

- **Lens operational logs** These are generated by Lens. They give information about issues or exceptions occurred while connecting to the CloudMine data source, executing the scope scripts, and transferring the CSV files to Azure Blob Storage.

- **Nalanda operational logs** These are generated by the graph construction pipeline developed in C#. They contain exception or error information, stack traces, and other custom log messages.

- **Pipeline registry database** This database is primarily used by the Nalanda pipeline service for bookkeeping purposes, such as saving new file information, performing checkpoints, etc. Additionally, this database stores metadata about each pipeline run. The monitoring service uses this information to notify about any potential degradation in the job runs, anomalous behaviors (e.g., a sudden increase in the number of files to process), etc.

Once the data is gathered from all of these log streams, the Nalanda monitoring service applies a simple anomaly detection algorithm to detect potential issues. It checks for the existence of exceptions, changes in the job run time, etc., while filtering out transient issues surfaced by Azure. Finally, the service sends notification emails to the on-call engineers with detailed error messages and log files.

### 2.5.3 Data consistency and self healing

The Nalanda graph platform 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 like infrastructure issues, non availability of the Azure DevOps APIs, issues with the CloudMine crawlers, etc.

We devised a self-healing algorithm 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 the external data sources such as CloudMine. The steps of the self-healing algorithm are the following:

1. When the incremental pipeline is run, check for the time stamp of the oldest record in each CSV file.
2. Check the pipeline registry database for the last successful pipeline run.
3. Check if there is a difference of more than three days between those two timestamps, because incremental streams hold the data from the last three days.
4. If the above check yields true, trigger the bootstrap mode for the repositories for which data gap is detected.
5. Call the Lens APIs to disable the incremental job schedule. This will prevent the pipeline from fetching new data from the CloudMine store.
6. Trigger a bootstrap Scope script to fetch the data from the CloudMine original streams.
7. Update the pipeline registry to indicate that bootstrapping is happening. This locking mechanism ensures the incremental jobs are not run.
8. Once all the bootstrap files are processed, enable the incremental jobs by changing the status in the pipeline database.

### 2.6 Scale

The Nalanda graph is built using the software development activity data from 6,500 repositories at Microsoft. We ingest data starting from 1st January, 2019, or from when the first pull request is created in a repository (whichever is newer).

---

**Table 2: Comparison of pipeline run time and number of records with the increase in number of repositories**

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

**Figure 4: Nalanda monitoring architecture**
work item, code review) of a developer, from multiple repositories, work items in their repository or across multiple repositories is a difficult and time consuming task. Azure DevOps provides a query editor [1] utility, but it is complex to use and it does not support multi-repository scenarios. Moreover, these tools operate in a work item centric fashion, i.e., the primary goal of the tool is to search for and find a work item one is interested in.

MyNalanda is a developer-centric (as opposed to workitem-centric) news feed based application built on top of the Nalanda graph. 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 team mates and other collaborators are working on without the hassle of going to different Azure DevOps repositories (if they know the repository names and locations) and fire multiple complex queries in the query editor.

3.1 MyNalanda Portal

MyNalanda is the first application we built on top of the Nalanda graph platform. 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. Azure DevOps provides a query editor [1] utility, but it is complex to use and it does not support multi-repository scenarios. Moreover, these tools operate in a work item centric fashion, i.e., the primary goal of the tool is to search for and find a work item one is interested in.

MyNalanda is a developer-centric (as opposed to workitem-centric) news feed based application built on top of the Nalanda graph. 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 team mates and other collaborators are working on without the hassle of going to different Azure DevOps repositories (if they know the repository names and locations) and fire multiple complex queries in the query editor.

3.1.1 MyNalanda homepage. A MyNalanda homepage from our production deployment is shown in Figure 5. MyNalanda has information organized in several sections, as explained below.

3 NALANDA APPLICATIONS

The Nalanda graph platform and the socio-technical graph can be used to build many applications to address the problems associated with information overload and satisfying the information needs of software developers. We discuss an application named MyNalanda, which is deployed in production at Microsoft, as well as two applications that can be built using the Nalanda graph.

3.2 Nalanda: A Socio-Technical Graph for Building Software Analytics Tools at Enterprise Scale ICSE 2022, May 21–29, 2022, Pittsburgh, PA, USA

Table 3: Node and edge types in the Nalanda graph

| Node type      | Count   | Edge type      | Count   |
|----------------|---------|----------------|---------|
| pull request   | 7,568,949 | creates        | 7,569,086 |
| work item      | 3,067,754 | reviews        | 39,447,635 |
| file           | 14,537,998 | contains       | 7,337,036 |
| user           | 131,578  | changes        | 65,706,621 |
| text           | 12,104,427 | parent         | 845,728  |
|                |          | linked to      | 7,094,597 |
|                |          | comments on    | 746,887  |
| Total nodes    | 37,410,706 | Total edges    | 128,745,590 |

Typically, the Nalanda graph 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 37,410,706 nodes and 128,745,590 edges. Detailed statistics of node and edge types are provided in Table 3.

News feed The centralized news feed is located in the middle of the page (indicated as ‘F’ in figure 5). 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. Users of MyNalanda can not only see their own activities, but also can see others’ activities, work items, and pull requests by visiting their homepage. Users have the ability to order the events in the news feed chronologically or by repository. Users can also apply filters on the repositories from which they would like to consume events from.

User details This section (‘A’ in figure 5) provides details about developers, such as name, email address, job title, and the programming languages they have expertise in. This helps in facilitating easy discovery of developers’ skills and their current projects based on their software development activity data.

Active items MyNalanda has four sections to show active items of a developers. This includes active repositories, active pull requests, active work items, and active code review requests. They are indicated as ‘B’, ‘C’, ‘D’, and ‘E’ respectively. Users of MyNalanda can prioritize the discovery of updates from repositories by following them. These items (and their updates) are shown at the top of the news feed. They can also unfollow an item if they would like to stop receiving updates about such repositories.

Related people This section (indicated as ‘G’ in figure 5) lists all of a developer’s collaborators. A developer’s collaborator is someone who reviews that developer’s pull requests or whose pull requests are being reviewed by the developer. We also consider the people who collaborate on a coding task or work item as collaborators. This section helps visualizing and developing awareness of who a developer collaborates with and how local software development communities are formed.

Search box The search box (indicated as ‘H’ in figure 5) can be used to find other developers and discover their activity. It also can help searching for technical and functional concepts by performing a simple look up over pull request and work item entities using the BM25 algorithm [7].

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 or MyNalanda. This makes it easy for developers to navigate back-and-forth from MyNalanda to 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 [29].

MyNalanda has to navigate through complex relationships to find the content presented in various sections. Nevertheless, 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”. The
graph representation of the data (the Nalanda graph) and its schema design contributes heavily to the query performance and fast load times of MyNalanda.

3.2 Concept and people search
Information needs of software developers go beyond simple code search [9]. As part of the MyNalanda user study (Section 4) we asked software developers at Microsoft to list the top-3 pain points when it comes to search and recommendation.

1. 81.08% of the developers would like to find the most relevant engineers to contact for a given feature, API (subject matter expert) with the help of a simple natural language query.
2. 74.32% of the developers would like to find people or teams their code or service is dependant on.
3. 66.22% of the developers would like to find related code, pull requests, bug reports, work items, etc.

The Nalanda graph can address these problems by leveraging the augmented socio-technical graph. The MyNalanda search bar currently has a basic implementation of concept search by using BM25 [7]. Once the BM25 index returns matches, we navigate the Nalanda graph to filter and rank results. We also use the graph to provide custom and context-sensitive results based on the user performing the search. Additionally, we use the concept embeddings learnt from the Nalanda graph to perform semantic search.

3.3 Neural reviewer recommendation
Code review is an integral part of any mature software development process, and identifying the best reviewer for a code change is a widely investigated problem within the software engineering research community. To date, most reviewer recommendation systems rely primarily on historical file change and review information; those who changed or reviewed a file in the past are the best positioned to review in the future. While these approaches are able to identify and suggest qualified reviewers, they may be blind to reviewers who have the needed expertise and have simply never interacted with the changed files before. The Nalanda graph can address this by leveraging the rich set of entities and their relationships present in the graph. In our ongoing efforts we employ a graph convolutional neural network and model the reviewer recommendation problem as a link prediction problem [38] in the Nalanda graph.

4 INFORMATION AND PRIVACY EXPECTATIONS
In this section, we describe a user study investigating perceptions of information and privacy when developer data is presented on Nalanda. We use mixed methods (interviews and survey) for evaluation. The study was reviewed and approved by the Microsoft Research Institutional Review Board, which is an IRB federally registered with the United States Department of Health & Human Services. A mixed-methods approach allowed us to understand user perceptions and explain emergent patterns that may not be solely available or apparent in telemetry data, particularly one with a platform that is not yet broadly rolled out to users. We situated our user study using MyNalanda because it provided participants with concrete examples of development information that aren’t typically available in a private development context without a monorepo, and that allowed us to surface questions about information sharing, information discovery, and expected privacy using real data, not just imagined scenarios. In this way, these insights have a higher likelihood of extending to other applications built on Nalanda.

4.1 Study Setup
4.1.1 Research Questions. We investigated the following:
We then showed a mock-up of MyNalanda and asked for reactions who were developers or engineering managers. While their work were recorded and automatically transcribed, and themes noted. (e.g., office or work-from-home). We included items asking about excerpts from all interviews and organized the themes by topic.

4.2 Results of the study
For each research question, 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.

4.2.1 What are the information needs of developers and engineering managers? (RQ1). 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.

A) 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) spoke about the usefulness of a feed that updates them about teammates’ current work. The ability to see detailed information about peers’ work is helpful when coordinating work, and is not readily available in many other platforms. The importance of information integration is echoed in prior studies investigating information characteristics of technical tools used to build systems [18].

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.

B) Survey: Based on themes from our interviews, we developed survey items to evaluate information integration and overload.

We asked participants to rate how useful each feature of MyNalanda was (including development activity, the news feed, and collaborators) 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 MyNa-
landa features (63 developers and 22 engineering managers). This
allowed us to capture participant reactions to a real integrated in-
formation platform instead of a hypothetical one. Table 4 lists the
accumulated percentages of "Extremely useful" (Likert=5) and "Very
useful" (Likert = 4) for each feature, reported for developers and
engineering managers. Not all engineering managers had activity
repos, but their direct reports did. Here, we see that Active pull
requests (55.6% and 54.5% among developers and engineering man-
gers, 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 (27.0% and 18.2% among
developers and engineering managers, respectively) and Related
People (23.8% and 9.1% among developers and engineering man-
gers, respectively) rated the lowest. The biggest difference between
developers and engineering managers is seen in the Feed (34.9% and
9.1% among developers and engineering managers, respectively) and
Related Data (28.2% and 27% among developers and engineering man-
gers, respectively) rated the lowest. The biggest difference between
developers and engineering managers is seen in the Feed (34.9% and
9.1% among developers and engineering managers, respectively). These
findings confirm what we heard in our interviews: developers and
engineering managers have different information needs. In particu-
lar, developers find a feed of detailed development-related in-
formation useful, given it’s what they do and need to keep track
of every day, while engineering managers are likely more interested
in team-level aggregates or trends over time. This echoes recent
industry research, showing that developers favor individual metrics
over team-level aggregates, which are often the focus of solutions
built for managers and leaders in organizations [25].

To evaluate information overload, we asked "How challenging
is it for you to currently discover and manage development ac-
tivity that is relevant for you?" Among our survey respondents,
53% indicate it is a minor concern and 47% indicate it is a major
concern. If given an opportunity to skip some information to help
deal with information overload, respondents indicated "comments
and replies... there’s so many of these it’s extremely noisy" and
"I’d like to see current state... on the level of work items and pull
requests." Nalanda platform provides for additional applications
that can address these challenges for users.

4.2.2 Do developers’ expectations about privacy related to their
development-related activity at work differ when the interface changes?
(RQ2) While most development-related information is available
and visible through existing source control systems and documents,
changing the format and presentation of information (e.g., via inte-
gration a news feed) may change expectations of privacy. Therefore,
we asked developers and managers how they would feel about their
work being available in a central feed like MyNalanda.

A) Interviews: Two themes related to privacy emerged: the desire
for personalized data, and an expectation that most development
work is public (even when shared in a new format or platform)
while desiring to keep some work interactions private.

One way to personalize news feeds is to provide filters for in-
dividual work and team work; this was mentioned by P2, P3, and
P5 (all D). While this feature was reported often by developers, it
could also serve to help coordinate work across teams. Relatedly,
personalization can include tools or systems a developer relies on,
assuring they have the most relevant information (P1, EM). Personal-
ization also includes algorithmic input into the news feed. This is
seen in this comment from P5 (D) "I am fine with algorithm running.
There is always a different view right like the ‘Most recent’ that can
have events in the chronological order and ‘Most relevant’ view can
have events filtered using an algorithm. I can’t trust the ‘Relevance’
view always, so there should be an option of ‘Recent’ view.”

In general, interviewees expected their development work to be
visible through MyNalanda, similar to how it is now visible through
source control and documents. However, they did add some privacy
expectations, notably that read activity would remain private (P1,
EM), and that any activity related to confidential projects would
remain hidden (P5, D). P2 (D) stated their mental model for privacy
this way: “Free to share, just make every item interesting.” This
proves the inherent, and perhaps growing, understanding
that developers are in effect trading their information for insights,
and as such expect the trade to be worth it.

B) Survey: Based on themes from our interviews, we developed
survey items to evaluate the themes identified during our interview:
algorithmic control over the ordering of their newsfeed and visibi-
licity of development-related information. We did not include items
about personalization because MyNalanda does not yet have these
features available. However, Nalanda platform provides for this to
be built into MyNalanda and other applications easily, addressing
development activity information needs addressed in interviews.
When asked if they would like algorithmic filtering, 24 (28%) responded Yes; 15 (18%) responded No, they preferred chronologi-
cal filtering; and 46 (54%) responded they would like the option to
switch between chronological and algorithmic filtering. This speaks
to the balance between providing users with easily-understood solu-
tions (e.g., chronological) and algorithmic solutions that may
provide “better” performance but cannot be easily understood or
explained by users. The trade-offs inherent in these differing ap-
proaches may speak to the comfort that users will have for other
Nalanda applications as well as other developer tools.

To evaluate visibility of information, we asked, “How comfort-
able are you sharing information about your development activity
to others?” Among our respondents, 82.82% indicated they were
comfortable to share all of their development activities, 12.99% in-
dicated they were comfortable to share few of their development
activities, and the remaining 5.19% indicated they were not comfort-
able to share any of their development activities. When asked what
information they would not want shared, representative responses
included "[confidential] project work," "[specific] files [or] apps I have
open on my machine... no real-time eyeball tracking." These
findings shed light on the openness that developers have with their
activity data, even when shared in another format.

Table 4: MyNalanda survey Feedback

| Feature                        | Cumulative (n=85) | Developers (n=63) | Engineering 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%            | 49.9%                     |
| Active work items (D)        | 34.1%            | 34.9%            | 31.8%                     |
| Feed (F)                     | 28.2%            | 34.9%            | 9.1%                      |
| User Details (A)             | 24.7%            | 27%              | 18.2%                     |
| Related people (G)           | 20%              | 23.8%            | 9.1%                      |
4.3 Threats to validity

4.3.1 Internal validity. Our study was conducted during the COVID-19 pandemic. While we did not explicitly ask about remote work or the pandemic’s effects on our participant’s work, the possible effects on the study should not be ignored. As with any survey, there is always the possibility for bias. To reduce non-response bias, we kept the surveys as short as possible (median time to complete was 8.5 minutes). To minimize common-method bias, we included interviews in our research methodology and confirmed all findings with an external expert familiar with the project.

4.3.2 External validity. Our study’s context is a large software development company with tens of thousands of developers. These developers work on a portfolio of products across many domains and across many contexts, and by conducting a study within a single company, we were able to control for culture, tooling, and many other factors. However, our results may not be generalizeable across all developers in all contexts.

We investigated a platform that surfaced the work of developers and their colleagues; this allowed us to study important socio-technical factors like user perceptions of information needs and privacy. Early findings are insightful for teams in organizations that support large software development collaboration and coordination efforts. However, these results may not be true across all contexts, in particular those with different organizational cultures (such as those that don’t value transparency) or those where very different development stacks make data sharing difficult or impossible without a platform like Nalanda. 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 [24].

5 RELATED WORK

Graph Representations. Hipikat [15] is one of earliest works to build a graph for software development entities like tasks, file versions, and documents, using a fixed schema. It was built to onboard new hires quickly by providing easy access to relevant artifacts. Codebook [9] 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. [12] 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 [32] and extracting changes in variability models [17]. Compared to all this work, the scale of the Nalanda graph is significantly larger with 37M nodes and 128M edges, and Nalanda having been designed to be actively used in a production setting.

Social Networking. Singer et. al [36] describe a “social programming ecosystem” where developers are interested in knowing what other developers are working on. They want to find and collaborate with developers based on common interests. This work mentions social programming platforms like MasterBranch and CoderWall that have profile pages for developers. These profiles show the contribution of a developers to various projects, their skills, and gamify their achievements in the form of badges. These activities are compiled from a developer’s contribution to social code sharing platform like GitHub. They also found that developers would like to stay informed about the activity of coworkers who are not collocated by passively following their progress. One of the challenges in such platforms was that they aggregate activity only from a limited set of open source repositories.

Begel and Zimmermann proposed a newsfeed based solution for improving coordination based on the Codebook architecture [10]. In their 2018 work, Treude et al. [8] found that developers use news aggregation platforms like Reddit, Hacker News to be aware of the latest work in the software engineering domain. Such platforms also help them feel a part of the community by sharing their work, and getting feedback through comments and up votes. Our current work, through the MyNalanda application facilitate collaboration and information discovery in large enterprises like Microsoft and helps with accelerating the adoption of innersource practices.

Source Control Dashboards. Azure DevOps offers a dashboard for developers where they can see work items and pull requests assigned to them within a project. If a developer is working on multiple projects, they will have to navigate to the respective project pages to see the pull requests they are working on. Azure DevOps also provides a query editor [1] to discover the work items of other developers. However, it involves writing complex queries and the information presented does not persist, i.e., an expensive query needs to be executed every time someone would like to discover the work item activity of a developer.

GitHub offers a dashboard for developers on their homepage [4]. It displays the repositories, active pull requests and issues (work items). A key limitation of the GitHub dashboard is that one cannot see the activity of another developer. In contrast, MyNalanda helps finding the complete development activity of any developer. Another difference is the “Manager feed” in MyNalanda, which displays the development activity of all the developers reporting to a manager. This helps the discovery of the items worked by other team members. Additionally, MyNalanda is built to be an extensible system and provides value addition by integrating machine learning based recommenders such as Nudge [29].

Information Needs. Ko et al. [26] studied information needs in collocated development teams. Fritz and Murphy [20] provide a list of questions developers ask for most frequently sought after information within a project. Information needs have also been studied in the context of change tasks [35], inter-team coordination [9] and software analytics [11, 13, 23] 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.

The user study in this paper contributes new insights on information and privacy needs in the context of development activity. Themes that emerged were the importance of information integration and and expectation that most development work is public but also a desire to keep some work interactions private.
6 CONCLUSION

In this paper, we seek to build a large scale socio-technical graph encompassing the entities, people and relationships involved in the software development life cycle. We aim to bring the benefits of inner-sourcing and mono-repositories to enterprises like Microsoft by leveraging the Nalanda socio-technical graph. The Nalanda platform and its applications help in developing awareness of each others work, building connections between developers across repositories, while offering mechanisms to manage information overload.

We built the Nalanda graph 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. Thanks to our design-for-scale, our initial application, MyNalanda, is fast for users, with the entire private software development data. Thanks to our design-for-scale, our initial application, MyNalanda, is fast for users, with the entire home page, news feed, and other sections having an end-to-end load time of less than a second. MyNalanda aims to help software developers with activity discovery and information overload, especially when they work with multiple repositories and source control systems. Furthermore, to illustrate the versatility of the Nalanda platform, we discuss two other applications presently under construction: a neural reviewer recommender, and a concept search system for finding software developers. Additionally, we describe our initial application, MyNalanda, its design, implementation, and use.

We conducted a user study (with 144 respondents) to understand the information and privacy expectations of software developers when using applications built on top of the Nalanda graph. 74% of the study participants, comprising engineers and engineering managers, are favorable towards using applications that facilitate information discovery using the Nalanda graph, such as MyNalanda, and use them as their primary source to discover activity across various repositories and source control systems.

In the future, we anticipate both the Nalanda graph and MyNalanda to be scaled out significantly inside Microsoft in terms of number of repositories and users. Looking at the success, favorability, and the scale of the deployment of Nalanda graph and MyNalanda inside Microsoft, we believe the systems and the techniques has applicability beyond Microsoft. Furthermore, we see opportunities for implementing the Nalanda graph platform as a service using open source data from, e.g., GitHub. Lastly, the rich data in the Nalanda graph in combination with advances in deep learning and graph convolutional networks, hold promise for applications beyond feeds, search, and reviewer recommendation.

7 ACKNOWLEDGEMENTS

We thank B. Ashok, Christian Bird, and Denae Ford for their support of Nalanda and their valuable feedback. We also thank the interview and survey participants for their incredible insights.

REFERENCES

[1] Accessed 2021. Azure DevOps query editor. https://docs.microsoft.com/en-us/azure/devops/boards/queries?view=azure-devops&tabs=queries
[2] Accessed 2021. Azure services. https://azure.microsoft.com/en-us/services/
[3] Accessed 2021. English stop words. gist.github.com/sebleier/554280
[4] 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

[5] Accessed 2021. Gremlin query language. https://docs.janusgraph.org/getting-started/gremlin/
[6] Accessed 2021. Lens Explorer. https://docs.microsoft.com/en-us/system-center/orchestrator/learn-about-orchestrator/view/azure-devops
[7] Giambattista Amati. 2009. BM25. Springer US, Boston, MA, 257–260. https://doi.org/10.1007/978-0-387-39946-9_921
[8] Mauricio Aniche, Christoph Trende, Igor Steinmacher, Igor Wiese, Gustavo Pinto, Margaret-Anne Storey, and Marco Aurélio Gerosa. 2018. How Modern News Aggregators Help Development Communities Shape and Share Knowledge. In 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE). 499–510. https://doi.org/10.1109/ICSE.2018.3180180
[9] Andrew Begel, Khaso Yi Phang, and Thomas Zimmermann. 2010. Codebook: Discovering and Explaining Relationships in Software Repositories. In Proceedings of the ACM/IEEE 32nd International Conference on Software Engineering. Association for Computing Machinery, Inc.
[10] Andrew Begel and Thomas Zimmermann. 2010. Keeping up with your friends: Function fo0, library bar, dll, and work item 24 In Proceedings of the 1st Workshop on Web 2.0 for Software Engineering, 20–23.
[11] 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.
[12] Pamela Bhattacharya, Marios Iliofotou, Julian Neamtiu, and Michalis Faloutsos. 2010. Graph-based analysis and prediction for software evolution. In 2012 36th International Conference on Software Engineering (ICSE). 419–429. https://doi.org/10.1109/ICSE.2012.622173
[13] 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.
[14] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Huenuine Ponde, Jared Kaplan, Harri Edwards, Yura Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03574 (2021).
[15] Davor Cubranic, Gail Murphy, Janice Singer, and K.S. Booth. 2005. Hikariat: A project memory for software development. Software Engineering, IEEE Transactions on 31 (07 2005), 446 – 465. https://doi.org/10.1109/TSE.2005.51
[16] Jacek Czerwonka, Nachi Nagappan, Wolfram Schulte, and Brendan Murphy. 2013. CODEMINE: Building a Software Development Data Analytics Platform at Microsoft. IEEE Software (July 2013).
[17] Nicolas Dintzner, Arie Deursen, and Martin Panging. 2018. FEVER: An approach to analyze function-oriented changes and artefact co-evolution in highly configurable systems. Empirical Software Engineering 23 (04 2018).
[18] Nicole Forsgren, Alexandra Durckova, Paul F Clay, and Xuequn Wang. 2016. The Integrated user satisfaction model: Assessing information quality and system quality as second-order constructs in system administration. Communications of the Association for Information Systems 38 (2016), 803–839.
[19] Nicole Forsgren, Dustin Smith, Jez Humble, and Jessie Frazelle. 2019. 2019 accelerate state of deops report. (2019).
[20] 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.
[21] Georgios Gousios. 2013. The GHTorrent dataset and tool suite. In 2013 10th Working Conference on Mining Software Repositories (MSR). IEEE, 233–236.
[22] Georgios Gousios and Diomedis Spinellis. 2012. GHTorrent: GitHub’s data from a firehose. In 2012 9th IEEE Working Conference on Mining Software Repositories (MSR). IEEE, 12–21.
[23] Henning 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 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 568–579.
[24] Pooya Jafarian, David Botta, Fahimeh Raja, Kristie Hawkey, and Konstantin Bermovov. 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.
[25] Errini Kalliamvakou, Nicole Forsgren, Liz Redford, and Shonte Stephenson. 2021. Octoverse Spotlight 2021: The Good Day Project—Personal analytics to make your work days better. Access: 2021-10-06.
[26] 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.
[27] Yuexing 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 36th International Conference on Mining Software Repositories (MSR). IEEE, 134–154.
[28] Chandra Maddila, Apoorva Agrawal, Tom Zimmermann, Nicole Forsgren, Kim Herzig, Arie van Deursen, Chandra Maddilla, 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.

[29] Chandra Shekhar Maddila, Sai Surya Upadrashta, Chotan 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

[30] Christopher D. Manning and Hinrich Schütze. 1999. Foundations of Statistical Natural Language Processing. The MIT Press, Cambridge, Massachusetts.

[31] 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.

[32] 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

[33] 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.

[34] Nischal Shrestha, Colton Botta, Titus Barik, and Chris Parnin. 2020. Here we go again: Why is it difficult for developers to learn another programming language?

In 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE). IEEE, 691–701.

[35] Jonathan Sillito, Gaïl 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.

[36] Leif Singer, Fernando Figueira Filho, Brendan Cleary, Christoph Trende, Margaret-Anne Storey, and Kurt Schneider. 2013. Mutual Assessment in the Social Programmer Ecosystem: An Empirical Investigation of Developer Profile Aggregators. In Proceedings of the 2013 Conference on Computer Supported Cooperative Work (San Antonio, Texas, USA) (CSCW ’13). Association for Computing Machinery, New York, NY, USA, 103–116. https://doi.org/10.1145/2441776.2441791

[37] Klaas-Jan Stol, Paris Avgiouri, Muhammad Ali Babar, Yan Lucas, and Brian Fitzgerald. 2014. Key factors for adopting inner source. ACM Transactions on Software Engineering and Methodology (TOSEM) 23, 2 (2014), 1–55.

[38] Muhaz Zhang and Yixin Chen. 2018. Link Prediction Based on Graph Neural Networks. In Advances in Neural Information Processing Systems, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. Curran Associates, Inc.