Supporting Software Developers with a Holistic Recommender System

Luca Ponzanelli*, Simone Scalabrino†, Gabriele Bavota*, Andrea Mocci*,
Rocco Oliveto†, Massimiliano Di Penta‡ and Michele Lanza*
*Università della Svizzera italiana (USI), Switzerland — †University of Molise, Italy — ‡University of Sannio, Italy

***Abstract*—The promise of recommender systems is to provide intelligent support to developers during their programming tasks. Such support ranges from suggesting program entities to taking into account pertinent Q&A pages. However, current recommender systems limit the context analysis to change history and developers' activities in the IDE, without considering what a developer has already consulted or perused, *e.g.,* by performing searches from the Web browser. Given the faceted nature of many programming tasks, and the incompleteness of the information provided by a single artifact, several heterogeneous resources are required to obtain the broader picture needed by a developer to accomplish a task.**

We present Libra, a *holistic* recommender system. It supports the process of searching and navigating the information needed by constructing a *holistic* meta-information model of the resources perused by a developer, analyzing their semantic relationships, and augmenting the web browser with a dedicated interactive navigation chart. The quantitative and qualitative evaluation of Libra provides evidence that a *holistic* analysis of a developer's information context can indeed offer comprehensive and contextualized support to information navigation and retrieval during software development.

***Keywords*-Mining unstructured data, Recommender systems**

I. Introduction

Information seeking is a fundamental component of software development, aimed at constructing and enriching the developer's knowledge to solve the task at hand. The sources from which developers scavenge essential information elements are diverse, including teammates, project documentation, and online resources. Online resources have become the prominent reference to acquire knowledge [1], thus making the web browser one of the key instruments in the modern developer toolkit. Prominent examples of such resources are Q&A websites like Stack Overflow, forums, blogs, API documentation, and video tutorials [2].

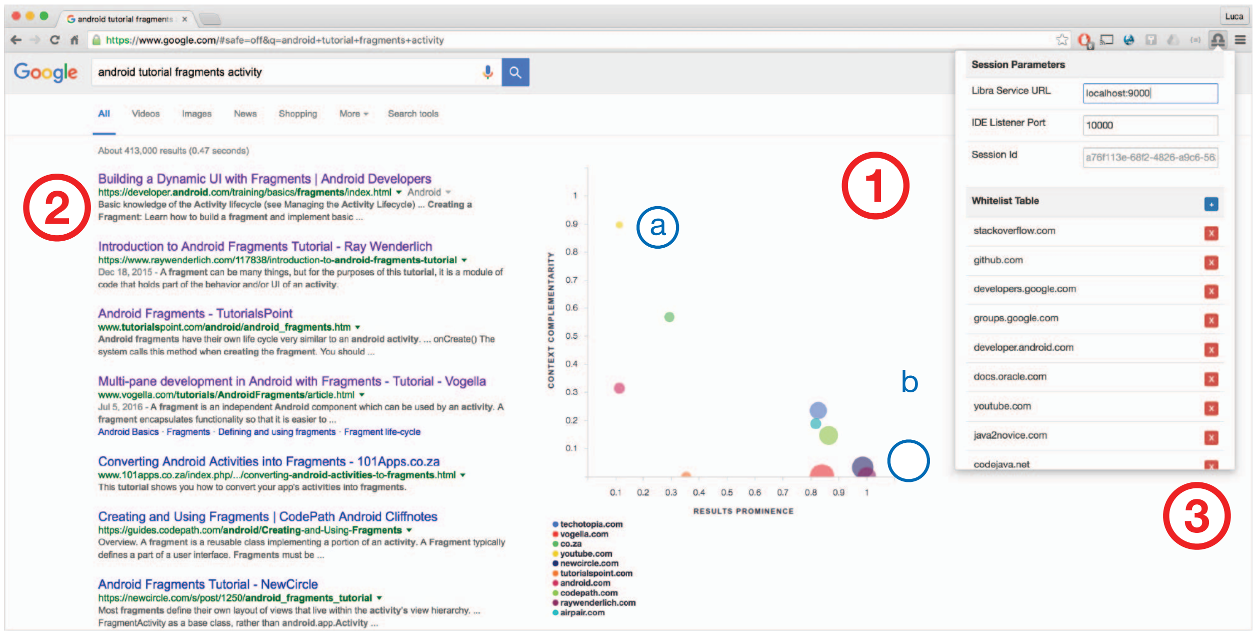
A guide to information seeking is provided by recommender systems for software engineering (RSSE) [3]. RSSEs suggest relevant artifacts to the developer, and may harness different information sources, *e.g.,* by mining API documentation [4], [5] and Q&A websites [6], synthesizing code examples from existing code bases [7]–[10], and extracting specific fragments of video tutorials [11], [12].

Many RSSE approaches work without considering the current knowledge context arising from the information seeking process happening in the task. Instead, they rely on historical information mined from repositories [13], or by considering the element being modified [6], [14]–[16].

Developers frequently search the web for the information fragments needed to complete a task [17]. Following an iterative approach [18], they inspect resources until they reach a satisfiable level of knowledge to solve a given task. This process can be described as a foraging loop to seek, understand, and relate information [19]. It can also be seen as a treasure hunt, where the map is progressively unveiled as hints are found along the way. Getting new hints to proceed towards the treasure requires one to search in the current zone of the map, facing riddles and tricks to get new pieces of the map, and eventually hunt down the treasure. Current RSSEs shortcut this process by pointing out a "candidate treasure" (*e.g.,* a Stack Overflow discussion for a given task) using only some small pieces of the map (*e.g.,* by only knowing what the developer is doing in the IDE). However, all pieces of the map are essential to proceed. The unveiled pieces of the map are the developers' *knowledge context*, continuously refined as they peruse new resources or modify existing code. In our vision a recommender system should provide continuous counseling to developers, guiding their information seeking process, taking into account what they are working on and what they already perused. The recommender should suggest to developers, in a timely fashion, pertinent artifacts given the current context: the developers' knowledge (*i.e.,* the already unveiled pieces of the map).

We propose LIBRA, a *holistic* recommender system that provides developers with real-time support for information navigation in the web browser. LIBRA monitors the developers' activity both in the web browser and in the IDE to track web search results, perused pages, and code written and modified by the developer. LIBRA models the knowledge context of the developer by considering all these resources, and constructs a *holistic* meta-information model of their contents. LIBRA's analysis does not consider the contents of resources as mere text, but takes into account their heterogeneous composition, including code fragments and exchange formats like XML and JSON. By holistically analyzing the contents of the knowledge context, LIBRA assists developers in selecting pertinent results from a web search by considering the *prominence* of a given resource, and the *complementarity* of a result with the gathered knowledge context. We evaluated LIBRA with two different studies to assess its usefulness during development activities, and its applicability in industrial contexts. Both studies showed that a holistic analysis of the developers' information context can offer comprehensive and contextualized support to information navigation and retrieval during development.

1558-1225/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSE.2017.17



android tutorial fragments
https://www.google.com/#safe=off&q=android+tutorial+fragments+activity
Google
android tutorial fragments activity
All
Videos
Images
News
Shopping
More
Search tools
Luca
1
2
3
a
b
CONTEXT COMPLEMENTARITY
RESULTS PROMINENCE
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

About 413,000 results (0.47 seconds)

Building a Dynamic UI with Fragments | Android Developers
https://developer.android.com/training/basics/fragments/index.html ▾ Android ▾
Basic knowledge of the Activity lifecycle (see Managing the Activity Lifecycle) ... Creating a Fragment: Learn how to build a fragment and implement basic ...

Introduction to Android Fragments Tutorial - Ray Wenderlich
https://www.raywenderlich.com/117838/introduction-to-android-fragments-tutorial ▾
Dec 18, 2015 - A fragment can be many things, but for the purposes of this tutorial, it is a module of code that holds part of the behavior and/or UI of an activity.

Android Fragments - TutorialsPoint
www.tutorialspoint.com/android/android_fragments.htm ▾
Android fragments have their own life cycle very similar to an android activity. ... onCreate() The system calls this method when creating the fragment. You should ...

Multi-pane development in Android with Fragments - Tutorial - Vogella
www.vogella.com/tutorials/AndroidFragments/article.html ▾
Jul 5, 2016 - A fragment is an independent Android component which can be used by an activity. A fragment encapsulates functionality so that it is easier to ...
Android Basics · Fragments · Defining and using fragments · Fragment life-cycle

Converting Android Activities into Fragments - 101Apps.co.za
www.101apps.co.za/index.php/.../converting-android-activities-to-fragments.html ▾
This tutorial shows you how to convert your app's activities into fragments.

Creating and Using Fragments | CodePath Android Cliffnotes
https://guides.codepath.com/android/Creating-and-Using-Fragments ▾
Overview. A fragment is a reusable class implementing a portion of an activity. A Fragment typically defines a part of a user interface. Fragments must be ...

Android Fragments Tutorial - NewCircle
https://newcircle.com/s/post/1250/android_fragments_tutorial ▾
Most fragments define their own layout of views that live within the activity's view hierarchy. ... FragmentActivity as a base class, rather than android.app.Activity ...

● techotopia.com
● vogella.com
● co.za
● youtube.com
● newcircle.com
● tutorialspoint.com
● android.com
● codepath.com
● raywenderlich.com
● airpair.com

Session Parameters

Libra Service URL	localhost:9000
IDE Listener Port	10000
Session Id	a76f113e-68f2-4826-a9c6-56

Whitelist Table

stackoverflow.com
github.com
developers.google.com
groups.google.com
developer.android.com
docs.oracle.com
youtube.com
java2novice.com
codejava.net

Fig. 1. The LIBRA user interface.

Paper structure. In Section II we describe LIBRA, its architecture and its user interface. Section III details the *holistic* approach implemented in LIBRA's core. In Section IV we describe the controlled experiment to evaluate LIBRA's usefulness during development activities. Section V reports interviews with five industrial practitioners discussing LIBRA's applicability in industrial practice. Then, we survey the related work in Section VI, and conclude in Section VII.

II. LIBRA

LIBRA is a recommender system aimed at extending and integrating the two main modern software development tools—the IDE and the web browser—to support information seeking.

A. User Interface

Fig. 1 shows the LIBRA user interface as it appears in the web browser. It includes three components providing features to navigate the information space of the search engine[1].

Whenever a developer writes a query in a browser, a two-axes bubble chart (1) appears on the right side of the web page. Every bubble represents an entry in the results list (2) on the left side.

Hovering on a bubble highlights the corresponding entry in the results list, and fades out the others. If a developer hovers over a search result, LIBRA highlights the corresponding bubble, and fades out the others. The developer can access the URL of a search result by clicking on the corresponding entry in the results list or on the related bubble in the LIBRA chart.

The URL is then opened in a new tab, while the chart gets updated with the new context information, and the visited URL becomes part of the developer's context, including all the recently navigated resources and the code she recently wrote in the IDE. The chart provides additional support to navigate the information space by visualizing the following information:

Bubble Color: Resources are grouped by their domain and assigned a specific color. The bottom part of the chart contains an interactive legend reporting all domains found in the result set. Developers can click on a domain to highlight its results, fading out the others.

Context Complementarity: The y-axis represents the complementarity of the information provided by a search result with respect to the current context. The higher the position of a resource, the higher its complementarity with the developer's context, who can thus decide between broadening the context or sticking with resources similar to the ones already perused. For example, in Fig. 1 the resource browsed on youtube.com (a) has high complementarity (but low prominence).

Result Prominence: The x-axis allows to discriminate among the results returned by the search engine. The more a result is on the right side of the chart, the higher its prominence within the result set[2]. Developers can use this axis to avoid out of scope results, or results whose information is a subset of more prominent resources. For example, the resource browsed on raywenderlich.com (b) has high prominence (but low complementarity).

[1]LIBRA uses Google in its current implementation.

[2]Our exact definition of prominence is discussed in Section III.

Bubble Diameter: It represents the quantity of information provided by a resource with respect to the whole result set. Ranging between 10 and 25 pixels, the diameter is normalized on the maximum information content value. For example, the large red (semi)circle on the x-axis refers to resources on vogella.com, providing higher quantity of information than other resources shown in small circles, *e.g.,* the yellow youtube.com circle.

The developer can take advantage of the features described above to select resources that best suit the next step in the information seeking process. LIBRA also provides an option panel (3) where the developer can access basic information about the state of the application, and manage a white list of domains that can be freely tracked (*i.e.,* that can be part of the developer's context). A demo of LIBRA is publicly available[3].

B. Architecture

Fig. 2 shows the architecture of LIBRA and uses two types of arrows to denote two different phases performed by LIBRA: Solid arrows represent the tracking events, while the dashed arrows represent the events caused by the interaction of the developer with LIBRA.

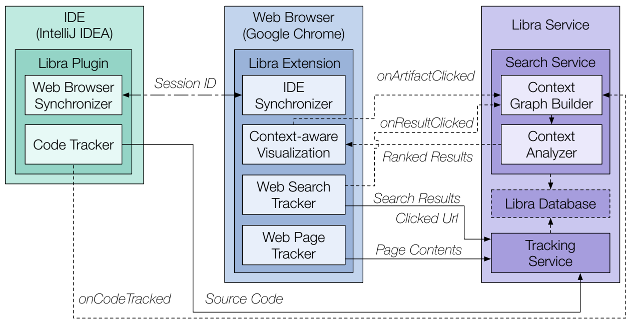


Fig. 2. The LIBRA architecture.

LIBRA is composed of three main components: (1) a plugin for the IntelliJ IDEA integrated development environment that takes care of tracking the modified and accessed source code, (2) a Google Chrome extension that tracks the web pages perused by the developer and augments the Google search result web page with the LIBRA user interface, and (3) a back-end service hosting LIBRA's analyzer as well as its data. The use of the Google search engine, IntelliJ IDEA, and Google Chrome are implementation choices adopted for our convenience.

Tracking Developer Context: Similarly to MYLYN [20], LIBRA aims at tracking the elements that are modified or created during a development task. LIBRA goes beyond the boundaries of the IDE, as it tracks developers' activities both in the IDE and in the web browser, thus targeting source code and web pages respectively. As depicted in Fig. 2, the *Libra Plugin* is responsible for tracking the code written or accessed by the developer. Whenever a developer opens a text file in the IDE (*e.g.,* Java code, XML files, documentation, or logs), the content is sent to the *Libra Service* to be parsed, modeled, and stored as context resource.

The browser's *Libra Extension* searches for queries performed on the search engine, as well as every other URL opened by the browser. The *Web Search Tracker* is responsible for checking whether a tab in Google Chrome corresponds to a search page, and, in such a case, to monitor new queries. When a search is performed, all URLs composing the result set of the search engine are sent to the *Tracking Service* to be processed and stored in a cache (used for the sake of performance). If the URL points to a document in HTML format, the service renders the page and extracts the text, while it uses Apache Tika to extract textual contents from binary files (*e.g.,* PDF, Word Documents)[4]. In case a URL points to a YouTube video, the service automatically extracts the English audio transcriptions as contents of the page using GOOGLE2SRT[5]. Such transcriptions are either automatically generated or written by the author of the video.

The *Web Page Tracker* keeps track of the URLs opened by the developer, and acts as a "remote crawler" of the service. Indeed, instead of asking the *Tracking Service* to crawl a certain URL, this component sends the whole rendered content to the *Libra Service* so that it can be parsed, modeled, and stored as context resource.

Developers interacting with LIBRA are assigned an ID generated by the *Libra Plugin* or the *Libra Extension*. Both components need to have synchronized IDs to identify the same user: every request sent to the service requires a check between the *IDE Synchronizer* and the *Web Browser Synchronizer* to set the same session ID on both sides (see double arrow in Fig. 2). This solution allows LIBRA to work with or without the IDE. The *Tracking Service* performs its operations if and only if the domain of the opened URL matches a white list of domains specified by the developer in the option panel. This limitation does not apply to the *Web Search Tracker*, which tracks any resource opened from a Google search while LIBRA is running. This ensures that the resources visualized in the bubble chart are the same reported in the search results' list.

Interacting with Libra: The *Web Search Tracker* is responsible for instrumenting Google's result page such that LIBRA knows when a result is clicked by the developer. When this happens, the corresponding URL is opened in a new tab, and the URL is sent to the *Tracking Service* to become part of the context resources. The *Web Search Tracker* then notifies the *Search Service* to create and analyze the new context graph, and sends the results to the *Context-Aware Visualization* to display the updated information to the developer. Similarly, either the *Context Aware Visualization* in the browser, and the *Code Tracker* in the IDE, notify the *Search Service* whenever a result is opened from LIBRA's user interface or a new piece of code has been tracked from the IDE. In doing so the visualization is always updated to the last context available.

[3]http://libra.inf.usi.ch

[4]Supported file types are listed in the Tika website: http://tinyurl.com/hksl9hr.

[5]http://google2srt.sourceforge.net/en/

III. Holistic Approach

We detail the approach used by LIBRA to process and analyze the results returned from web searches. We discuss how we parse and model the information within artifacts, and how we designed *HoliRank*, an extension of PageRank [21] devised to analyze the complementarity and prominence of the results in a holistic fashion.

A. Content Parsing and Meta-Information Model

Whenever an artifact is sent to the *Tracking Service*, the contents are parsed using the StORMeD[6] island parser [22], capable of identifying complete and incomplete multi-language elements—*e.g.,* written Java, JSON, XML, Stack Traces—immersed in natural language paragraphs, and to model such contents as a Heterogeneous Abstract Syntax Tree (H-AST) that allows visiting and manipulating the results.

In our previous work [23], we developed the concept of meta-information system that allows to model specific aspects of the information. Following the same blueprint, we devised the following meta-information to model the contents of every resource processed by LIBRA:

- **Types**: It represents the set of Java types mentioned in a resource. We consider all the H-AST nodes matching a reference type (either fully qualified or simple name), and primitive types (*e.g.,* int, double).
- **Variable Declarations**: All H-AST nodes matching a variable and class field declaration.
- **Method Declaration**: All H-AST nodes matching a method declaration.
- **Method Invocations**: All H-AST nodes matching a method invocation and the name of the invoked method.
- **Identifiers**: All H-AST nodes matching an identifier that can be visited in any extracted constructs (*e.g.,* full method and class declarations).
- **XML Elements**: All H-AST nodes matching an XML element like a single tag (*i.e.,* `<tagname/>` or `<tagname>`) or a double tag (*e.g.,* `<tagname></tagname>`).
- **JSON Members**: All H-AST nodes matching a JSON member (*i.e.,* `"field": element`).
- **Natural Language**: We complement the meta-information with pure textual information, for example a term frequency map that can be reused to compute, for example, a textual similarity measure like *tf-idf* [24].

B. HoliRank: Holistic PageRank

The *PageRank* algorithm [21] identifies prominent pages within a graph of linked pages, *e.g.,* the World Wide Web. To compute the relevance of a page within a network of pages, *PageRank* models the behavior of a "random surfer", *i.e.,* a user who randomly surfs the web and (quoting Brin and Page) *"keeps clicking on links, never hitting back but eventually gets bored and starts on another random page"* [21].

[6]See http://stormed.inf.usi.ch

PageRank takes as input a directed graph G representing the hyperlinks between pages and returns a probability distribution P where each p_i represents the probability that the random surfer visits page i. The probability associated to a page represents its centrality in the network.

In our approach we need to calculate the centrality of a resource within a graph of resources. Since reconstructing explicit links is not possible due to the absence of an explicit reference like an URL or link in a page, the connection between two resources needs to be quantified differently. Specifically, we use a similarity graph. Approaches such as *LexRank* [25] (an unsupervised summarization algorithm based on *PageRank*) use a *tf-idf* [24] similarity graph as input, where an edge between two sentences exists if and only if the textual similarity is above a given threshold. *LexRank* runs *PageRank* on a similarity graph and selects the sentences with higher centrality to compose a summary.

The way *LexRank* computes the *PageRank* on a similarity graph can be adapted for LIBRA's purposes. Differently from *LexRank*, the vertices in the graph must be resources (web pages returned by the search engine) as in the original *PageRank*, and not sentences. Since the context includes heterogeneous resources, *i.e.,* source code from the IDE, web pages and other documents (*e.g.,* PDF files) returned by the search engine, relying on a pure textual similarity lowers the richness of the available meta-information system. We also need to consider the heterogeneity of the information: The resources tracked by LIBRA provide textual information, but also code information valuable to estimate connections between resources. Instead of working with pure textual similarity as in *LexRank*, we devised a *holistic* similarity function to take advantage of the additional information layer provided by the meta-information system: LIBRA models each resource with a variable number of meta-information types, depending on its contents.

Consider two resources R_x and R_y. Let $T_{x,y}$ be the set of shared types of meta-information between the resources, and let $M(R,t)$ be the meta-information of type t for the resource R. We define the similarity vector $V_{x,y}$ as:

$$V_{x,y} = \langle v_0, \ldots, v_{|T_{x,y}|} \rangle$$
$$\mathit{with}\ v_i = M(U_x, t_i) \sim M(U_y, t_i)\ \mathit{and}\ t_i \in T_{x,y}$$

where each element v_i of the vector $V_{x,y}$ represents the similarity value between two homogeneous meta-information, and ranges in the interval $[0,1]$. We calculate the general similarity between two resources R_1 and R_2 as the average of the vector V:

$$f_{sim}(R_x, R_y) = \overline{V}_{x,y}$$

which gives a value in the range $[0,1]$. We use this similarity function in the *PageRank* algorithm to compute the centrality of a resource within a similarity graph of the resources. We refer to this approach as "holistic PageRank" or *HoliRank*.

C. Analyzing Context Resources

Our approach is based on the metrics *context complementarity*, *result prominence*, and *information quantity*.

Context Complementarity measures the information intake provided by a resource in the current context of the developer. We use *HoliRank* to build the similarity graph CG of the recently used context resources (the code recently written/modified in the IDE and the recently navigated web pages). LIBRA considers as "recent" what the developer dealt with in the past four hours. This is a settable parameter. For each resource R in the search engine result set, we create an additional similarity graph CG_R by adding R to the set of vertices of CG, and for each vertex V_{CG} in CG we add an edge from R to V_{CG} whose weight is equal to $f_{sim}(R, V_{CG})$. For each graph CG_R, we run *HoliRank* to compute the centrality of the resource R, ranging in $[0, 1]$. The higher the centrality of R in the graph, the lower the context complementarity: a higher centrality implies a tight relationship with many resources of the context, indicating a low information intake of R since R is similar to what is already composing the context. We define context complementarity as:

$$\mathit{CtxComplementarity} = 1.0 - \mathit{HoliRank}(R, CG_R)$$

Result Prominence identifies prominent results among the search engine result set. Even though a set of results matched by a query can be more or less relevant, there is often an overlap of the information provided by different artifacts. For example, an artifact is a tutorial on a specific topic, while another artifact tackles a programming problem on the same topic. If a result overlaps with many other results, it probably provides diversified information in its contents. If we model a similarity graph of the result set, a high overlap of the information of a result R with other results in RS, would result in a more prominent (central) position of R in the graph. We build a similarity graph G_{RS} containing all results R in the results set RS. We use *HoliRank* to estimate the centrality of a resource R in the graph G_{RS}:

$$ResultProminence = HoliRank(R, G_{RS})$$

Information Quantity sums up the number of "elements" identified by our meta-information system. For example, for the *Natural Language* meta-information we consider the total amount of terms (after text preprocessing), to which we sum the number of declarators identified by *Method Declarators* meta-information, the ones identified by the *Variable Declarators* meta-information, *etc.* Counting information elements allows discriminating between two resources with the same size but with different contents. Consider for example two resources R_1 and R_2 having the same amount of characters, and—after preprocessing—the same terms. However, R_1 provides just text, while R_2 provides text and code. In this case, we consider the information quantity of R_2 is higher than the one of R_1.

IV. Study I: Controlled Experiment

The *goal* is to evaluate LIBRA in terms of its (i) ability in correctly assessing for each query search result its prominence and complementarity with respect to the context, and (ii) usefulness to developers during a development or maintenance task. The *context* consists of *participants*, *i.e.,* third-year Computer Science (CS) Bachelor students, and *objects*, *i.e.,* a University career management app and four maintenance tasks.

The study addresses the following research questions:

RQ$_1$: *How accurate is* LIBRA *in assessing the prominence and complementarity of query search results?* We investigate if the prominence and complementarity of information computed by LIBRA for a set of query search results Q_r is aligned with the developers' perception of prominence and complementarity.

RQ$_2$: *Does* LIBRA *help developers to complete their tasks correctly?* We investigate if the use of LIBRA helps developers when performing coding activities and to what extent—within an available time frame, and when working with or without LIBRA—they are able to correctly complete development and maintenance tasks.

A. Context Selection

We ran a controlled experiment with 16 3rd year CS Bachelor students at the end of a "Mobile Apps Development" course taught at the University of Molise (UniMol). Students learned about the design and implementation of Android apps, and were also required to develop an app.

We asked each participant to perform two programming tasks, one with and one without LIBRA. All tasks focused on the source code of `MyUnimol`, an app used by UniMol students to register for exams, visualize their marks, *etc.* All tasks were real implementation tasks performed by the `MyUnimol`'s developers in the past. We extracted these tasks from the app's issue tracker and versioning system (both repositories are private, and the study participants could not access them at any time). `MyUnimol` consists of 9k LOC.

We selected the four tasks from the issue tracker based on their type (two bug fixes and two enhancements) and difficulty (non-trivial, but doable in a limited amount of time). Then, we checked out from the versioning system the four snapshots of the app preceding the commit fixing each of the four issues. Participants worked on the specific version of the app related to the task they had to perform. The four tasks are:

T_1: When the user taps the "Logout" button in the upper right corner of the `MyUnimol` GUI, the app logs out the user without asking for confirmation. You are asked to add a confirmation dialog that pops up when the user taps logout. The dialog asks the user if she really wants to logout the app, providing as possible choices "Yes" and "No". If the user taps "Yes", the app logs her out, otherwise the confirmation dialog disappears. Make sure that the confirmation dialog is legible.

T_2: `MyUnimol` includes an address book with the contacts of all the University employees. A contact can have multiple phone numbers. When visualizing the details of a contact, all phone numbers associated to it are shown in a single string separated by a comma. This does not allow tapping on the phone number to start a call. You are asked to modify the view implementing the contact's details, showing each phone number associated to the contact in a separated field. It must be possible to tap a number to start a call.

T_3: There is a bug in the view allowing students to visualize personal details. The name and the student number shown on screen are not correct (*i.e.,* they are not the ones associated to the logged student who is visualizing her personal details). Also, tapping the "Back" button in this view makes the app crash. Fix the bug.

T_4: When a student logs in, `MyUnimol` loads in the home view a pie chart showing the exams already taken/to take. This also happens when the student comes back to the home view from another section of the app. The pie chart is shown through an animation that glitches, restarting multiple times (instead of loading the pie chart just once). Fix this animation.

B. Study Design and Procedure

Each participant was assigned two tasks to perform during the controlled experiment. We devised two possible pairs of tasks to assign to each participant: The first pair (T_1, T_2) includes two tasks related to the enhancement of existing features. The second pair (T_3, T_4) includes tasks dealing with bug-fixing activities. We wanted each participant to work on a pair of similar tasks (*e.g.,* two bug-fixing activities or two enhancements) having a comparable difficulty level in order to evaluate the effect of Libra during enhancement and bug-fixing tasks. For the purpose of RQ_2, this allows to observe the effect of LIBRA on the performance of the participants.

Participants were equally partitioned into the eight groups shown in Table I, reporting the study design.

TABLE I
STUDY I: DESIGN.

Group	Session 1	Session 2
A	T_1-LIBRA	T_2-NOLIBRA
B	T_1-NOLIBRA	T_2-LIBRA
C	T_2-LIBRA	T_1-NOLIBRA
D	T_2-NOLIBRA	T_1-LIBRA
E	T_3-LIBRA	T_4-NOLIBRA
F	T_3-NOLIBRA	T_4-LIBRA
G	T_4-LIBRA	T_3-NOLIBRA
H	T_4-NOLIBRA	T_3-LIBRA

The design is conceived in such a way that each participant worked both with and without LIBRA. To avoid learning effects, each participant had to perform different tasks across the two sessions. Different participants worked with and without LIBRA in different order and on two different tasks. When assigning participants to the eight groups, we made sure that their level of experience was (roughly) uniformly distributed across groups.

We collected the (claimed) experience of participants via a pre-questionnaire. We also collected information related to the typical sources of information participants consult during coding activities. We carried out a pre-laboratory briefing in which participants were trained on the use of LIBRA through a running example and the laboratory procedure was illustrated in detail. We made sure not to reveal the study research questions. The training was performed on tasks not related to the ones of the experiment to avoid a bias in the results.

Participants had to perform the study in two sessions of 75 minutes each, interleaved by a break of 30 minutes to avoid fatigue effects. During the break participants did not exchange information. Participants were allowed to use whatever they wanted to complete the tasks including any material available on the Internet. At the end of each session, each participant provided the code she implemented and answered a three-part post-questionnaire. The first part, to check for problems with the experimental design, was composed of questions in which participants had to express their level of agreement on a Likert scale going from 1 (absolutely no) to 5 (absolutely yes) to the following claims:

1) *The overall activity to be performed was clear.*
2) *The description of the task to implement was clear.*
3) *There was enough time to perform the task.*
4) *The task was easy to implement.*

The second part was aimed at collecting qualitative information about LIBRA's usefulness. The following questions were only answered by participants who completed a task performed with LIBRA:

1) *How useful were the prominence, complementarity, and information quantity indicators provided by* LIBRA*?* Possible answers used a 5-point Likert scale from 1 (not useful at all) to 5 (very useful) for each indicator. *Why? Please motivate your previous answer.*
2) *How often did you use* LIBRA *in your Web searches?* Possible answers on a five-point Likert scale: 1 (never), 2 (in ∼25% of the searches), 3 (in ∼50% of the searches), 4 (in ∼75% of the searches), 5 (always).
3) *How would you improve* LIBRA*?*

The third part of the questionnaire was aimed at collecting information useful to answer RQ_1 and was only answered by participants who just completed a task performed with LIBRA. We showed to the participant the visualization depicted by LIBRA for the last search she performed. For each of the web documents projected on the LIBRA chart, we asked: *Do you agree with the assessment of the prominence of the document performed by* LIBRA*?* The same question was also asked with respect to LIBRA's assessment of the document complementarity with the context. Both questions were answered with a Likert scale going from 1 (strongly disagree) to 5 (strongly agree).

C. Variable Selection and Data Analysis

A normality check using the Shapiro-Wilk test indicated a statistically significant deviation from normal distribution (p-value< 0.05); hence we use non-parametric statistics. For all tests we consider a significance level $\alpha = 5\%$.

We answer RQ_1 by showing boxplots of the participants' answers to the questions in the 3rd part of the post-questionnaire to evaluate LIBRA's assessment of the documents' prominence and complementarity.

We also statistically check, using the Wilcoxon signed-rank test [26], whether the average agreement is greater than 3 (*i.e.,* at least weak agreement), by testing the null hypotheses $H_{0pr} : \overline{pr} \leq 3$ and $H_{0cm} : \overline{cm} \leq 3$, where $\overline{pr}$ and $\overline{cm}$ are the average (perceived) document prominence and complementarity.

The dependent variable to answer RQ_2 is *task completeness*. We asked a developer of `MyUnimol` (not involved in the study) to act as "evaluator" by reviewing the code implemented by the participants. The evaluator did not know the goal of the study nor which tasks were performed with/without LIBRA. We provided a checklist to assign a completeness score to each of the sub-tasks implemented by participants. The completeness percentage of each sub-task was proportional to its difficulty (as estimated by the authors) and complexity. For example, the checklist for task T_1 was: (i) confirmation dialog view implemented and linked to the logout button (+30%), (ii) behavior of the confirmation dialog implemented (+40%), (iii) proper UI theme set (+30%).

The main factor and independent variable is the presence/absence of LIBRA. Other potentially influencing factors are the (possible) different difficulty of the two tasks, the participants' (self-assessed) *Skills* in Java/Android development and years of *Experience* in Java/Android development.

To answer RQ_2, we show boxplots of task completeness distributions for the two treatments, and also compare the results using the Wilcoxon signed-rank test. Since we do not know *a priori* in which direction the difference should be observed, we use a two-tailed test. We also assess the magnitude of the observed difference using Cliff's delta (d) effect size [27], suitable for non-parametric data. Cliff's d ranges in the interval $[-1, 1]$ and is negligible for $|d| < 0.148$, small for $0.148 \leq |d| < 0.33$, medium for $0.33 \leq |d| < 0.474$, and large for $|d| \geq 0.474$.

We check the influence of the co-factors (ability, experience levels, and order in which tasks were performed) and their interaction with the main factor, using permutation test [28], a non-parametric alternative to ANOVA, which does not require normally distributed data. We set the number of iterations of the permutation test procedure to 500,000 to ensure that results did not vary over multiple executions.

To analyze the post-experiment questionnaire results we use descriptive statistics, and tested, using the Wilcoxon signed-rank test, the null hypothesis $H_{0ag} : \overline{ag} \leq 3$ ($\overline{ag}$ is the average agreement level), to assess if there has been a weak or strong agreement.

D. Study Results

The population involved in this study has 2.8 years of programming experience on average, with a maximum of 5 and a median of 3.0. They have a median of 2.5 years of Java programming experience (mean=2.2) and 3.5 months of Android development (maximum 1 year, minimum 1 month). Most of the participants learned how to develop an Android application while attending the "Mobile Apps Development" course at the University of Molise. Participants felt to have a good experience in Java programming with a median of three (medium experience), and a low experience in Android development (median=1.5, between very low and low). Concerning the sources of information exploited when programming, participants declared Q&A websites (median=5) as the most exploited, followed by forums (4), video tutorials (4), and official documentation (3.5).

How accurate is LIBRA in assessing the prominence and complementarity of query search results? Fig. 3 reports the level of agreement of the participants with the prominence and complementarity indicators provided by LIBRA for the ten documents retrieved by Google in the last search they did while performing the four tasks with LIBRA, thus adding up to 160 documents.

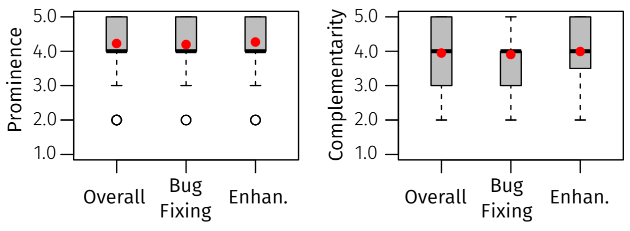


Fig. 3. Participants' agreement with LIBRA's indications of prominence and complementarity: 1=strongly disagree, 5=strongly agree.

Participants agreed with the prominence indicator (left side of Fig. 3) provided by LIBRA (median=4), and H_{0pr} can be rejected (p-value< 0.001). Only for five out of the 160 documents (3%), participants disagreed (Likert scale score=2) with LIBRA's prominence assessment. They agreed (4) for 62 (39%) or strongly agreed (5) for 70 (44%) documents. This is consistent both for bug fixing activities and enhancing existing features. Concerning the complementarity indicator (right side of Fig. 3), the agreement was fairly high (median 4), both for bug fixing and enhancement activities (in both cases the median=4). In this case, participants disagreed (2) with LIBRA's complementarity assessment on 8 documents (5%), while they agreed (4) for 73 (46%) or strongly agreed (5) for 43 (27%) documents. Thus, also the complementarity indicator provides precise insights to the LIBRA's users, and H_{0cm} can be rejected (p-value< 0.001) as well.

Does LIBRA help developers to complete their tasks correctly? Fig. 4 shows boxplots of completeness achieved by participants with (LIBRA) and without (NOLIBRA) LIBRA.

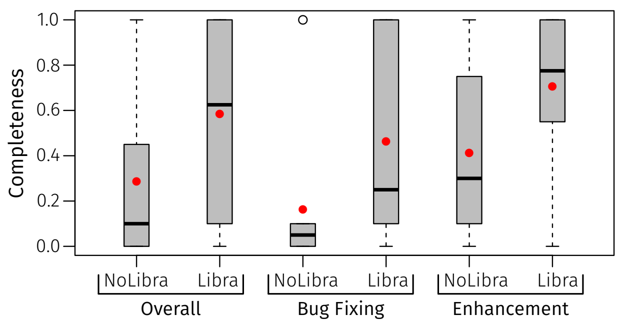


Fig. 4. Completeness achieved by participants with the two treatments.

Participants using LIBRA achieved a higher completeness. The LIBRA median is 62% (mean 58%) against the 10% median (mean 29%) of NOLIBRA. In other words, LIBRA allowed participants to achieve a median additional correctness of 52% (mean of 29%). The Wilcoxon paired test (Table II) indicates the presence of a statistically significant difference, with a p-value=0.035. Cliff's $d = 0.42$ indicates a *medium* effect size.

TABLE II
STUDY I: WILCOXON p-VALUE AND CLIFF'S d

Tasks	p-value	d
Overall	**0.035**	**0.42 (Medium)**
Bug Fix	0.170	0.50 (Large)
Enhancement	0.180	0.45 (Medium)

Fig. 4 and Table II report the completeness results for bug fixing and enhancement activities. LIBRA helped participants in both types of tasks, increasing the median completeness achieved for bug fixing activities by 25%, and for enhancement activities by 37%. The results of the Wilcoxon paired two-tailed test indicates that in both types of tasks the difference is not significant (p-value>0.05) and the effect size is medium and large for bug fixing and enhancement activities, respectively. The non-significant results is explainable by the low number of data points (8 participants for each type of task). Indeed, as explained above, the performance improvement provided by LIBRA is evident from the boxplots in Fig. 4 and statistically significant when considering the dataset as a whole.

The analysis of the post-questionnaires reveals that the usefulness of the information provided by LIBRA was also perceived by participants while performing the coding tasks. Table III reports the number of participants assessing the usefulness of the three indicators on each of the five levels in the considered Likert scale.

TABLE III
STUDY I: PERCEIVED USEFULNESS OF LIBRA'S INDICATORS

Indicator	Not useful at all	Not useful	Neutral	Useful	Very useful
Prominence	0	1	4	7	4
Complementarity	0	2	4	6	4
Info. Quantity	0	0	5	5	6

Out of the 16 participants, 11 (69%) found the prominence indicator useful or very useful, 4 remained neutral, while 1 found them not useful. Similar results were achieved for complementarity and information quantity. Participants used LIBRA in 59% of their web searches. 3 participants claimed to have used it only in 25% of their web searches. One of them was the most experienced in Android development, and achieved high completeness both with (100%) and without (90%) LIBRA, *i.e.,* he did not need to look online for help while performing the required tasks. The other two participants simply claimed to have used it only when they were not able to spot the useful web page to open in the search results.

We also statistically analyzed the effect of co-factors. Permutation tests indicated that none of the ability/experience factors collected in the pre-questionnaire had an effect on the task's completeness, nor it interacted with the study treatment, *i.e.,* availability of LIBRA (p-values were in all cases way greater than 0.05). Similarly, no significant effect of the task ordering was observed.

We also collected from the participants recommendations on how to improve our tool. These improvements mostly concern LIBRA's user interface and are currently being implemented.

Summing up. The study results indicated that both prominence and complementarity indicators reflect developers' perception of such measures, and are considered as useful/very useful indicators. LIBRA helped study participants to achieve a significantly better task completeness than the control group, though differences are not statistically significant when considering task types (*i.e.,* bug fixing and enhancement) separately due to the limited number of data points.

E. Threats to Validity

Threats to *construct validity* mainly concern imprecisions in the measurements made. A major challenge is to measure dependent variables related to **RQ**$_1$ (agreement with LIBRA) and above all **RQ**$_2$ (task completeness). For the former, we relied on developers' perceived agreement with LIBRA's assessment of documents' prominence and complementarity. For the latter we used a checklist-based approach. We are aware that results of such an approach might be influenced by the evaluator's subjectiveness, as well as by the weights we gave to each task (to account for its complexity).

Threats to *internal validity* concern confounding factors that could influence the results. First, as explained Section IV-B, we have used permutation test to analyze the effect of such factors, and also have been supported by the post-study questionnaires results. All participants strongly agreed about the clarity of the activity (mean 4.9, median 5, H$_{ag}$ rejected with p-value$<$ 0.001) and tasks (mean 4.8, median 5, H$_{ag}$ rejected with p-value$<$ 0.001). They weakly agreed on time (mean 4.1, median 4.5, H$_{ag}$ rejected with p-value$<$ 0.001), and had mixed opinions about the tasks' difficulty (mean 3.2, median 3, H$_{ag}$ not rejected with p-value=0.27). This should not be considered as a possible threat as it is normal to find people experiencing different difficulty and productivity levels. This indicates the absence of a possible ceiling effect.

TABLE IV
STUDY II: PARTICIPANT'S ANSWERS TO THE QUESTIONS EXPLICITLY ASKED

Developer	Giuseppe Socci	Luciano Cutone	Carlo Branca	Giovanni Grano	Matteo Merola
Position	Project Manager @ Genialapps	Project Manager @ IdeaSoftware	Developer @ Capgemini	Senior Developer @ Cedacri	Full Stack Developer @ Cleopa
Mobile Development Experience	5+ years	5+ years	1+ year	1+ year	1+ year
Do you find LIBRA useful?	Maybe	Absolutely yes	Yes	Absolutely yes	Maybe
Importance of prominence	Very high	Very high	Very high	High	Very high
Importance of complementarity	High	Very high	High	Medium	Medium
Importance of information quantity	Low	Very high	Medium	Low	Medium
Are you willing to use LIBRA for your activities?	Absolutely yes	Absolutely yes	Yes	Absolutely yes	Yes

Threats to *conclusion validity* concern the relationship between treatment and outcome. The main issue here is the possible presence of Type II errors—due to the limited number of study's participants—every time we could not reject a null hypothesis. In our study this happened when analyzing completeness results for different types of activity separately.

Threats to *external validity* concern the generalization of our findings. The controlled experiment has clear limitations (needed to achieve a high level of control) in terms of objects' characteristics and domain, and in terms of participants. To mitigate this threat due to the limited experience of the participants, we have conducted a second, qualitative study with experienced practitioners, described next.

V. STUDY II: INDUSTRIAL APPLICABILITY

A successful technological transfer is the main target objective for each prototype tool. Thus, the *goal* of this second study is to investigate LIBRA's industrial applicability by answering the following research question:

RQ$_3$: *Would practitioners consider exploiting* LIBRA *in their daily coding activities?*

The study *context* consists of the 5 *participants* listed in Table IV. We conducted semi-structured interviews to get qualitative feedback on both the tool and the underlying approach. Before each interview, one of the authors performed a demo of LIBRA to show its features to the participants. Then, we let the participant interact with the tool, performing web searches on the topics related to task T$_3$ of Study I. Each interview lasted ca. 1.5 hours and was based on a think-aloud strategy. After each interview we asked the questions listed in Table IV. The interviews were conducted by two of the authors.

A. Results

Table IV reports the participants' answers to the questions we asked to drive our interview. Giuseppe and Matteo expressed concerns about LIBRA's usability. In Giuseppe's opinion the graph-based interface provides too many details: "*You could think of a single metric that provides an indication of both complementarity and prominence, which could be used to indicate the overall usefulness of each page returned by Google. In this way it would be immediate for the user to identify which one is the better page for* LIBRA. *Then—and only if necessary—the user can analyze the chart to better understand why* LIBRA *is indicating a specific page.*"

We discarded this option since during a specific phase of a coding activity, a developer might be interested in reading documents that have a high prominence but a low complementarity with the context (*i.e.,* she may want to dig deeper into topics overlapped with her context), while in some other phases she might be interested in highly complementary documents (*i.e.,* she may want to broaden her knowledge). Thus, we do not see prominence and complementarity as direct indicators of "document quality". Similar usability concerns were expressed by Matteo.

Despite some reservations about the LIBRA's usability, both Giuseppe and Matteo expressed their desire to use LIBRA in their daily coding activities. Giuseppe liked the idea behind the tool, and would like to use it more: "*I should use* LIBRA *for much more time to better assess its usefulness. However, from this first experience I can say that* LIBRA *seems to be an interesting tool. I particularly like the idea to add information to the Google ranking. This can be particularly useful when you do not know exactly what you need, i.e., your query is rather generic*".

The other three participants provided enthusiastic comments about LIBRA, and would definitively like to use it while coding. One representative comment is the one by Luciano: "LIBRA *is a very interesting tool. Google provides accurate results in general. However, searching for pages related to software development is more challenging. When I use Google for my daily coding activities, I often need to open almost all documents in the first results' page to identify the most appropriate web page to read. In the hour I spent using* LIBRA, *I noticed that it allowed me to find the most appropriate web page quicker. Instead of analyzing ∼10 pages for each query, I have analyzed 2 or 3 pages, generally the one with the highest prominence, the one with the highest complementarity and (if needed) another one in between*". Giovanni particularly appreciated LIBRA's integration with the development workflow: "*I tried many different tools, but most of them are either hard to use or to integrate in the developer's workflow. The main strength of* LIBRA *is that it is integrated into the classical developer's workflow, which is programming and searching for information on Google, without adding any complexity:* LIBRA *does not create any barrier between the developer and her usual working environment;* LIBRA *just quietly guides the developer to the most useful results*". Carlo also appreciated LIBRA, and positively judged its usefulness and usability.

Participants agreed on the usefulness of prominence and complementarity, but less so for information quantity. Giuseppe explained: "*I do not care about information quantity since in my experience technical web pages are not so long*".

The participants provided several suggestions on how to improve LIBRA. Giuseppe suggested: "*provide an indication on the cohesiveness of the returned page with the query, to see how focused the page is with respect to the query. If you need information on a specific technology, you need a page that is extremely focused on the query, but if you need to learn a new technology, you prefer a less focused page*". Luciano, commenting LIBRA's indicators, explained: "*All the three indicators are crucial. It could be worthwhile to show the social importance of each page, i.e., how many times the page has been shared on social networks and/or how useful was the page for the developers, similarly to the mechanisms in Stack Overflow*". Giovanni expressed concerns about the way LIBRA tracks the web pages; "*I often open web pages of which I read a very limited part. If* LIBRA *tracks those pages and considers them as part of the context, it could provide misleading information about the documents' complementarity.*" To overcome this issue, Giovanni proposed to track the visiting time and weigh the importance of the pages in the context, ignoring pages visited for brief periods.

Summing up. This study provided positive feedback on the usefulness and practical applicability of LIBRA and its integration in a developer's workflow. Participants provided feedback on how to possibly improve LIBRA, *e.g.,* by providing a simplified user interface. Clearly, tools can always be improved, given sufficient time and human resources. However, we would like to emphasize that, stepping beyond mere implementation and UI concerns, the main contribution of the paper lies in the underlying holistic approach.

VI. RELATED WORK

Semantic code search and code search engines deal with retrieving code samples and reusable open source code from the Web. Different works [29]–[31] tackled this problem and provided capability of searching, ranking and adapting open source code. The mining of open source repositories has also been used to identify API and framework usage and to find relevant applications to be reused [32]–[34]. Other studies analyzed the habits of the developers in searching for code snippets [1], [35]–[37] or specific recommenders [18], and how general purpose search engines (*e.g.,* Google) outperform code search engines when retrieving code samples from the Web [38]. Sadowski *et al.* [17] conducted a study in Google to understand how developers search for code on the Web, finding that developers search for code very frequently.

We developed a layer on top of web search engines and of the IDE, to take into account the history of the information browsed by the developer. LIBRA helps in navigating the results of a search engine by considering the code-oriented nature of the artifacts, providing also prominence and complementarity information of different kinds of development-related resources.

Tracking Developer Context. A seminal tracking tool is MYLYN [20]. It assigns, according to a developer's task, a degree of interest (DOI) to code elements. Other work focused on tracking code modifications [39], or fine-grained interactions with either code and IDE [40]–[42]. Goldman *et al.* [43] bridged the gap between IDE and web browser, tracking interactions on both sides, and allow to link web pages (*e.g.,* API documentation) to code elements. In LIBRA we go beyond the IDE by including web resources as well. Sillitti *et al.* [44] developed PROM, a tool to monitor developers' activities on a workstation, *e.g.,* browsed files/resources. LIBRA goes beyond a simple monitoring, as it provides a visual means to help the developer navigating the search results.

Recommender systems. Various recommenders have been proposed, aimed at recovering traceability links, suggesting relevant project artifacts or code examples. Well-known examples are HIPIKAT [16], DEEPINTELLISENSE [45], STRATHCONA [46]–[48], and EROSE [13]. Other work focused on suggesting relevant documents, discussions and code samples from the web to fill the gap between the IDE and the web browser. Examples are MICA [49], FISHTAIL [15], DORA [50], and SURFCLIPSE [51]. Among the various sources available on the Web, Q&A Websites have been the target of many recommender systems. Other tools used Stack Overflow as main source for recommendation systems to suggest, within the IDE, code samples and discussions to the developer [4], [6], [14], [51]–[55]. LIBRA instead helps navigating the information space available after a web search by considering the information (*i.e.,* web resources and code) already acquired by the developer by tracking both the IDE and the web browser.

VII. CONCLUSIONS

A crucial activity in modern software development is the acquisition of pieces of information. These pieces are located in diverse places and are of a heavily heterogeneous nature. Instead of manually combing through the results of general purpose search engines or heeding the monochromatic suggestions of the recommender systems proposed so far, we tackled the problem by developing a *holistic* approach, based on a meta-information system, capable of dealing with the heterogeneous nature of web resources. Our approach, implemented in a tool named LIBRA, aids developers to interact with the suggestions, and seamlessly blends into their workflow. The empirical evaluation of Libra provides evidence that a *holistic* analysis of a developer's information context can offer comprehensive and contextualized support to information navigation and retrieval during software development.

Verifiability. We provide a replication package [56] containing the complete raw data of our first study, and the R scripts used for the analysis.

ACKNOWLEDGMENTS

Ponzanelli and Lanza thank the Swiss National Science foundation for the financial support through SNF Project ESSENTIALS, No. 153129. We thank all study participants and the `MyUnimol` developers.

References

[1] M. Umarji, S. Sim, and C. Lopes, "Archetypal internet-scale source code searching," in *Proceedings of OSS 2008 (The 4^{th} International Conference on Open Source Systems)*, 2008, pp. 257–263.

[2] L. MacLeod, M.-A. Storey, and A. Bergen, "Code, camera, action: How software developers document and share program knowledge using YouTube," in *Proceedings of ICPC 2015 (23rd IEEE International Conference on Program Comprehension)*, 2015, pp. 104–114.

[3] M. Robillard, R. Walker, and T. Zimmermann, "Recommendation systems for software engineering," *IEEE Software*, vol. 27, no. 4, pp. 80–86, 2010.

[4] P. Rigby and M. Robillard, "Discovering essential code elements in informal documentation," in *Proceedings of ICSE 2013 (35^{th} International Conference on Software Engineering)*, 2013, pp. 832–841.

[5] M. P. Robillard and Y. B. Chhetri, "Recommending reference API documentation," *Empirical Software Engineering*, vol. 20, no. 6, pp. 1–29, 2014.

[6] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza, "Mining stackoverflow to turn the IDE into a self-confident programming prompter," in *Proceedings of MSR 2014 (11^{th} Working Conference on Mining Software Repositories)*. ACM, 2014, pp. 102–111.

[7] R. P. L. Buse and W. Weimer, "Synthesizing API usage examples," in *Proceedings of ICSE 2012 (34^{th} International Conference on Software Engineering)*. IEEE, 2012, pp. 782–792.

[8] M. Acharya, T. Xie, J. Pei, and J. Xu, "Mining API patterns as partial orders from source code: from usage scenarios to specifications," in *Proceedings of ESEC/FSE 2007 (6^{th} joint meeting of the European Software Engineering Conference and the International Symposium on Foundations of Software Engineering)*. ACM, 2007, pp. 25–34.

[9] I. Keivanloo, J. Rilling, and Y. Zou, "Spotting working code examples," in *Proceedings of ICSE 2014 (36^{th} International Conference on Software Engineering)*. ACM, 2014, pp. 664–675.

[10] L. Moreno, G. Bavota, M. Di Penta, R. Oliveto, and A. Marcus, "How can I use this method?" in *Proceedings of ICSE 2015 (37^{th} IEEE/ACM International Conference on Software Engineering)*, 2015, pp. 880–890.

[11] L. Ponzanelli, G. Bavota, A. Mocci, M. Di Penta, R. Oliveto, M. Hasan, B. Russo, S. Haiduc, and M. Lanza, "Too long; didn't watch! extracting relevant fragments from software development video tutorials," in *Proceedings of ICSE 2016 (38^{th} International Conference on Software Engineering)*. ACM Press, 2016, pp. 261–272.

[12] L. Ponzanelli, G. Bavota, A. Mocci, M. Di Penta, R. Oliveto, B. Russo, S. Haiduc, and M. Lanza, "Codetube: Extracting relevant fragments from software development video tutorials," in *Proceedings of ICSE 2016 (38^{th} International Conference on Software Engineering)*. ACM Press, 2016, pp. 645–648.

[13] T. Zimmermann, P. Weißgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes," in *Proceedings of ICSE 2004 (26^{th} International Conference on Software Engineering)*. IEEE, 2004, pp. 563–572.

[14] L. Ponzanelli, A. Bacchelli, and M. Lanza, "Leveraging crowd knowledge for software comprehension and development," in *Proceedings of CSMR 2013 (17^{th} European Conference on Software Maintenance and Reengineering)*, 2013, pp. 59–66.

[15] N. Sawadsky and G. Murphy, "Fishtail: from task context to source code examples," in *Proceedings of TOPI 2011 (1^{st} Workshop on Developing Tools as Plug-ins)*. ACM, 2011, pp. 48–51.

[16] D. Cubranic and G. Murphy, "Hipikat: recommending pertinent software development artifacts," in *Proceedings of ICSE 2003 (25^{th} International Conference on Software Engineering)*. IEEE Press, 2003, pp. 408–418.

[17] C. Sadowski, K. T. Stolee, and S. G. Elbaum, "How developers search for code: a case study," in *Proceedings of ESEC/FSE 2015 (10^{th} joint meeting of the European Software Engineering Conference and the International Symposium on Foundations of Software Engineering)*, 2015, pp. 191–201.

[18] R. Holmes, "Do developers search for source code examples using multiple facts?" in *Proceedings of SUITE 2009 (Workshop on Search-driven Development: Users, Infrastructure, Tools and Evaluation)*, 2009, pp. 13–16.

[19] P. Pirolli and S. Card, "The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis," in *Proceedings of International Conference on Intelligence Analysis*, 2005, pp. 2–4.

[20] M. Kersten and G. Murphy, "Using task context to improve programmer productivity," in *Proceedings of FSE 2014 (The 22nd International Symposium on the Foundations of Software Engineering)*. ACM Press, 2006, pp. 1–11.

[21] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," in *Proceedings of WWW 1998 (7^{th} International Conference on World Wide Web)*. Elsevier Science Publishers B. V., 1998, pp. 107–117.

[22] L. Ponzanelli, A. Mocci, and M. Lanza, "Stormed: Stack overflow ready made data," in *Proceedings of MSR 2015 (12^{th} Working Conference on Mining Software Repositories)*. ACM Press, 2015, pp. 474–477.

[23] ——, "Summarizing complex development artifacts by mining heterogenous data," in *Proceedings of MSR 2015 (12^{th} Working Conference on Mining Software Repositories)*. ACM Press, 2015, pp. 401–405.

[24] C. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[25] G. Erkan and D. R. Radev, "Lexrank: Graph-based lexical centrality as salience in text summarization," *Journal of Artificial Intelligence Research*, vol. 22, no. 1, pp. 457–479, 2004.

[26] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures (fourth edition)*. Chapman & All, 2007.

[27] R. J. Grissom and J. J. Kim, *Effect sizes for research: A broad practical approach*. Lawrence Associates, 2005.

[28] R. D. Baker, "Modern permutation test software," in *Randomization Tests*. Marcel Decker, 1995.

[29] S. Reiss, "Semantics-based code search," in *Proceedings of ICSE 2009 (31^{st} International Conference on Software Engineering)*. IEEE, 2009, pp. 243–253.

[30] S. Thummalapenta, "Exploiting code search engines to improve programmer productivity," in *Proceedings of OOPSLA 2007 (22^{nd} conference on Object-Oriented Programming Systems and Applications)*. ACM, 2007, pp. 921–922.

[31] S. Thummalapenta and T. Xie, "Parseweb: a programmer assistant for reusing open source code on the web," in *Proceedings of ASE 2007 (22^{nd} International Conference on Automated Software Engineering)*. ACM, 2007, pp. 204–213.

[32] C. McMillan, M. Grechanik, D. Poshyvanyk, C. Fu, and Q. Xie, "A source code search engine for finding highly relevant applications," *Transactions on Software Engineering*, vol. 38, no. 5, pp. 1069–1087, 2012.

[33] C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu, "Portfolio: finding relevant functions and their usage," in *Proceedings of ICSE 2011 (33^{rd} International Conference on Software Engineering)*. ACM, 2011, pp. 111–120.

[34] S. Thummalapenta and T. Xie, "Spotweb: Detecting framework hotspots and coldspots via mining open source code on the web," in *Proceedings of ASE 2008 (23^{rd} International Conference on Automated Software Engineering)*. IEEE, 2008, pp. 327–336.

[35] S. Bajracharya and C. Lopes, "Mining search topics from a code search engine usage log," in *Proceedings of MSR 2009 (6^{th} Working Conference on Mining Software Repositories)*, 2009, pp. 111–120.

[36] S. K. Bajracharya and C. Videira Lopes, "Analyzing and mining a code search engine usage log," *Empirical Software Engineering*, vol. 17, no. 4-5, pp. 424–466, 2012.

[37] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi, "Mining internet-scale software repositories," in *Proceedings of NIPS 2007 (21^{st} Annual Conference on Neural Information Processing Systems)*. MIT Press, 2007, pp. 929–936.

[38] S. Sim, M. Umarji, S. Ratanotayanon, and C. Lopes, "How well do search engines support code retrieval on the web?" *Transactions on Software Engineering and Methodology*, pp. 1–25, 2011.

[39] C. Parnin and C. Gorg, "Building usage contexts during program comprehension," in *Proceedings of ICPC 2006 (14^{th} IEEE International Conference on Program Comprehension)*, 2006, pp. 13–22.

[40] R. Minelli, A. Mocci, and M. Lanza, "I know what you did last summer – an investigation of how developers spend their time," in *Proceedings of ICPC 2015 (23^{rd} IEEE International Conference on Program Comprehension)*, 2015, pp. 25–35.

[41] R. Minelli, A. Mocci, R. Robbes, and M. Lanza, "Taming the ide with fine-grained interaction data," in *Proceedings of ICPC 2016 (24^{th} International Conference on Program Comprehension)*, 2016, pp. 1–10.

[42] S. Amann, S. Proksch, S. Nadi, and M. Mezini, "A study of visual studio usage in practice," in *Proceedings of SANER 2016 (23^{rd} IEEE International Conference on Software Analysis, Evolution, and Reengineering)*, 2016, pp. 124–134.

[43] M. Goldman and R. Miller, "Codetrail: Connecting source code and web resources," *Journal of Visual Languages & Computing*, pp. 223–235, 2009.

[44] A. Sillitti, A. Janes, G. Succi, and T. Vernazza, "Collecting, integrating and analyzing software metrics and personal software process data," in 29th *EUROMICRO Conference 2003, New Waves in System Architecture*, 2003, pp. 336–342.

[45] R. Holmes and A. Begel, "Deep intellisense: a tool for rehydrating evaporated information," in *Proceedings of MSR 2008 (*5th *Working Conference on Mining Software Repositories)*. ACM, 2008, pp. 23–26.

[46] R. Holmes, R. J. Walker, and G. C. Murphy, "Strathcona example recommendation tool," in *Proceedings of ESEC/FSE 2005 (joint meeting of the* 10th *European Software Engineering Conference and the* 13th *International Symposium on Foundations of Software Engineering)*, 2005, pp. 237–240.

[47] R. Holmes, R. Walker, and G. Murphy, "Approximate structural context matching: An approach to recommend relevant examples," *Transactions on Software Engineering*, vol. 32, no. 12, pp. 952–970, 2006.

[48] R. Holmes and G. Murphy, "Using structural context to recommend source code examples," in *Proceedings of ICSE 2005 (*27th *International Conference on Software Engineering)*. ACM, 2005, pp. 117–125.

[49] J. Stylos and B. A. Myers, "Mica: A web-search tool for finding api components and examples," in *Proceedings of VL/HCC 2006 (Symposium on Visual Languages and Human-Centric Computing)*, 2006, pp. 195–202.

[50] O. Kononenko, D. Dietrich, R. Sharma, and R. Holmes, "Automatically locating relevant programming help online," in *Proceedings of VL/HCC 2012 (Symposium on Visual Languages and Human-Centric Computing)*, 2012, pp. 127–134.

[51] M. Rahman, S. Yeasmin, and C. Roy, "Towards a context-aware ide-based meta search engine for recommendation about programming errors and exceptions," in *Proceedings of CSMR/WCRE 2014 (Software Maintenance, Reengineering and Reverse Engineering)*, 2014, pp. 194–203.

[52] J. Cordeiro, B. Antunes, and P. Gomes, "Context-based recommendation to support problem solving in software development," in *Proceedings of RSSE 2012 (*3rd *International Workshop on Recommendation Systems for Software Engineering)*. IEEE Press, 2012, pp. 85–89.

[53] W. Takuya and H. Masuhara, "A spontaneous code recommendation tool based on associative search," in *Proceedings of SUITE 2011 (ICSE Workshop on Search-driven Development: Users, Infrastructure, Tools and Evaluation)*. ACM, 2011, pp. 17–20.

[54] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza, "Prompter: A self-confident recommender system," in *Proceedings of ICSME 2014 (*30th *International Conference on Software Maintenance and Evolution)*. IEEE, 2014, pp. 557–580.

[55] ——, "Prompter: Turning the IDE into a self-confident programming assistant," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2190–2231, 2016.

[56] L. Ponzanelli, 2016. [Online]. Available: http://libra.inf.usi.ch/