Glossary

Q&A question-and-answer

DS$_{synthetic}$ Synthetic tasks dataset, comprising six synthetic tasks with annotations from 20 participants of text deemed as relevant in 20 associated artifacts with natural language text

DS$_{Android}$ Android tasks dataset, comprising 12,401 unique sentences annotated by three developers and originating from artifacts associated to 50 software tasks drawn from GitHub issues and Stack Overflow posts about Android development

DS$_{Python}$ Python tasks dataset, containing 28 natural language artifacts where 24 participants indicated text containing information that assisted them in writing a solution for three programming tasks involving well-known Python modules

API application programming interface

NLP Natural Language Processing

IR Information Retrieval

ML Machine Learning

DL Deep Learning

API Application Programming Interface

CNN Convolutional Neural Network

VSM Vector Space Model

BERT Bidirectional Encoder Representations from Transformers

TARTI Automatic Task-Relevant Text Identifier, our proof-of-concept semantic-based tool, which uses BERT to automatically identify and show text relevant to an input task in a given web page

Chapter 1

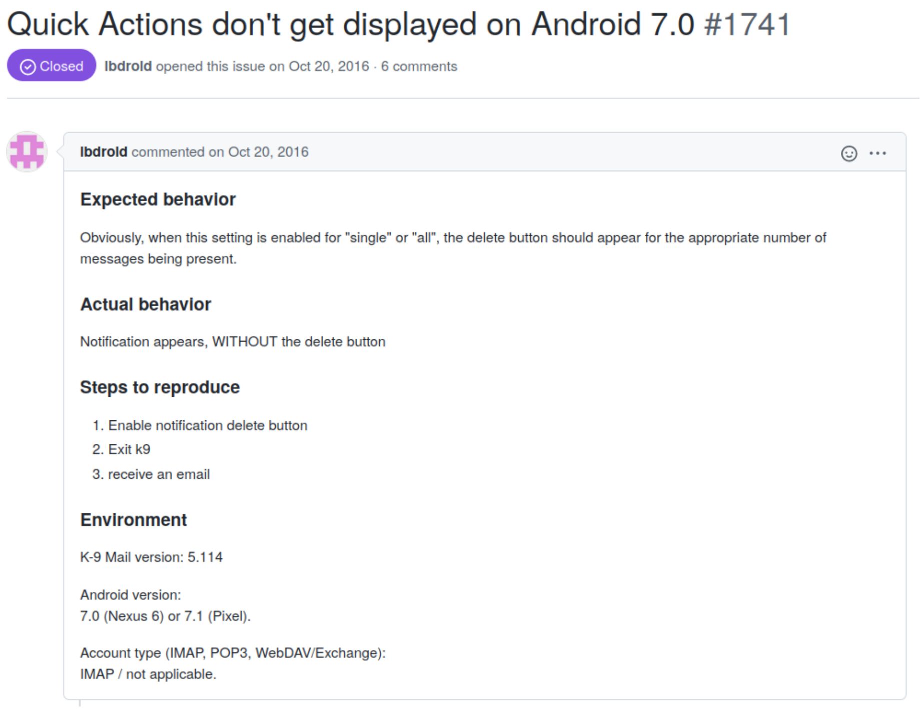
Introduction

When performing software tasks in large and complex software systems, software developers typically consult several different kinds of artifacts that assist them in their work [55, 81]. For example, when incorporating a new software library needed for a new feature, a developer might consult official API documents and guidelines for the library [74, 84] or question-and-answer developer forums for functionality, security and performance-related topics [65, 79].

Many of the artifacts consulted contain unstructured text [8]—we refer to artifacts with unstructured text as natural language artifacts—and a developer must read the text to find the information that is *relevant* to the task being performed. The sheer amount of information *within* these natural language artifacts may prevent a developer from comprehensively assessing what is useful to their task [61]. Just within one kind of document, API documentation, studies have shown that it can take 15 minutes or more of a developer's highly constrained time to identify information needed to perform a particular software task [22, 55] and a developer that fails to locate all, or most, of the information needed will have an incomplete or incorrect basis from which to perform a software task [61].

1.1 Scenario

To illustrate challenges in locating information useful for a task, let us consider an Android mail client application[1]. Figure 1.1 shows a task—in the form of a GitHub issue[2]—that indicates that the app notifications are not working as expected in Android 7.0.


Quick Actions don't get displayed on Android 7.0 #1741

Closed lbdroid opened this issue on Oct 20, 2016 · 6 comments

lbdroid commented on Oct 20, 2016

Expected behavior

Obviously, when this setting is enabled for "single" or "all", the delete button should appear for the appropriate number of messages being present.

Actual behavior

Notification appears, WITHOUT the delete button

Steps to reproduce

1. Enable notification delete button
2. Exit k9
3. receive an email

Environment

K-9 Mail version: 5.114

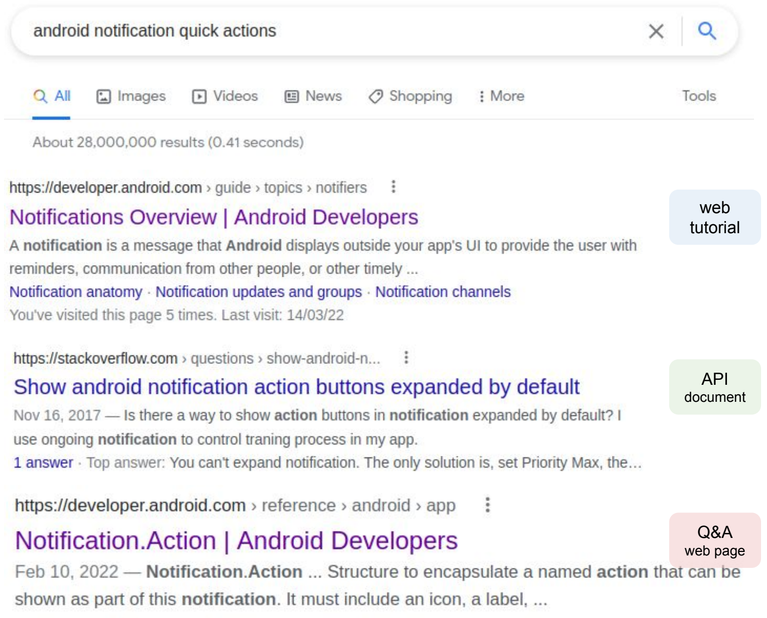
Android version:
7.0 (Nexus 6) or 7.1 (Pixel).

Account type (IMAP, POP3, WebDAV/Exchange):
IMAP / not applicable.

Figure 1.1: k-9 mail GitHub issue #1741 indicating that quick actions don't get displayed on Android 7.0

A developer assigned to this issue might not be familiar with how Android notifications work and thus, they will need additional knowledge to understand and resolve the bug [42, 44, 78]. More than often, this knowledge can be acquired from a developer's peers [80]. However, the fragmented and distributed nature of software development may prevent the developer from accessing their peers [42],

[1]https://github.com/k9mail/k-9
[2]https://github.com/k9mail/k-9/issues/1741


android notification quick actions

All · Images · Videos · News · Shopping · More · Tools

About 28,000,000 results (0.41 seconds)

https://developer.android.com › guide › topics › notifiers

Notifications Overview | Android Developers

A **notification** is a message that **Android** displays outside your app's UI to provide the user with reminders, communication from other people, or other timely ...

Notification anatomy · Notification updates and groups · Notification channels

You've visited this page 5 times. Last visit: 14/03/22

https://stackoverflow.com › questions › show-android-n...

Show android notification action buttons expanded by default

Nov 16, 2017 — Is there a way to show **action** buttons in **notification** expanded by default? I use ongoing **notification** to control traning process in my app.

1 answer · Top answer: You can't expand notification. The only solution is, set Priority Max, the...

https://developer.android.com › reference › android › app

Notification.Action | Android Developers

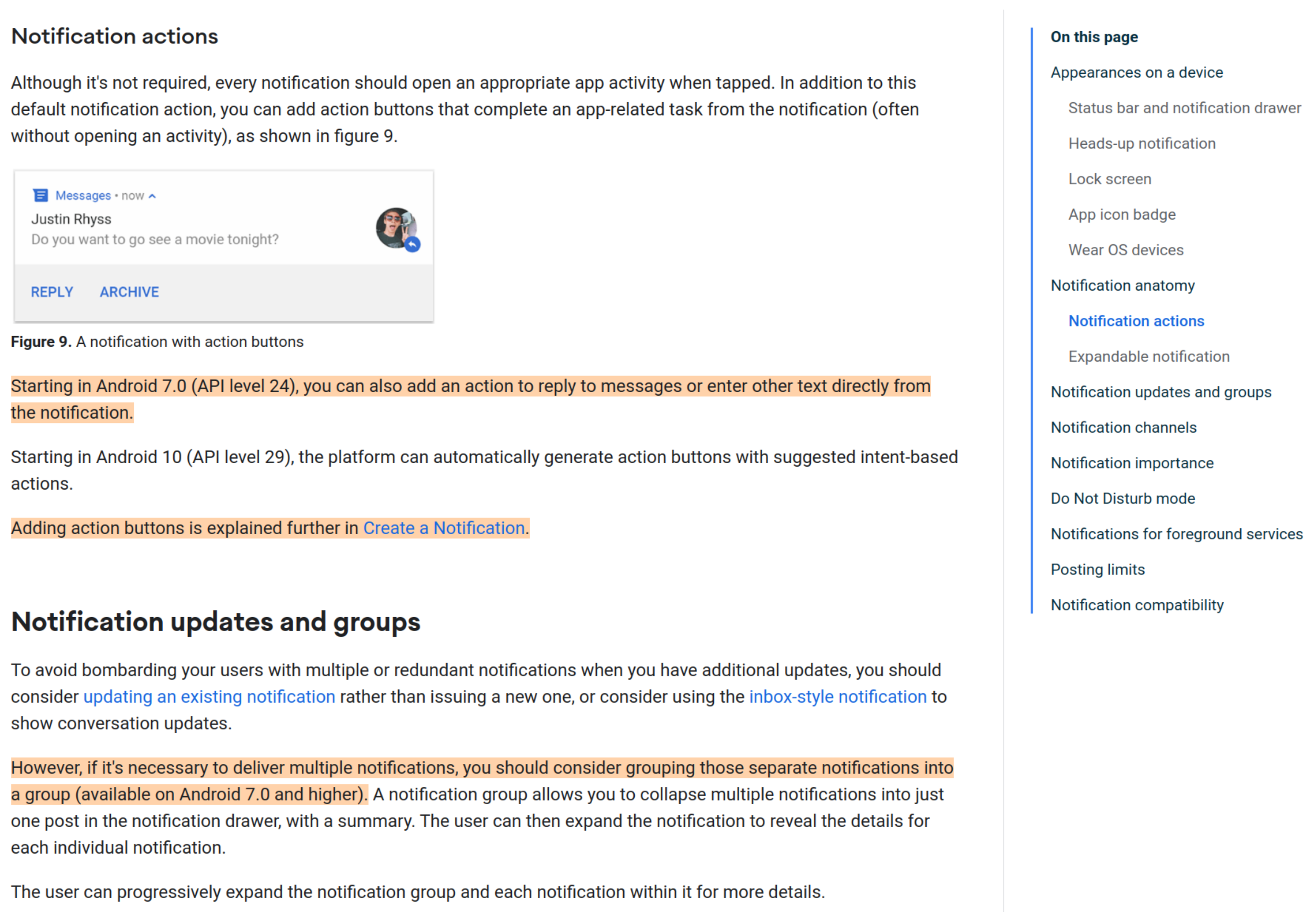
Feb 10, 2022 — **Notification.Action** ... Structure to encapsulate a named **action** that can be shown as part of this **notification**. It must include an icon, a label, ...

Figure 1.2: Search results showing artifacts of potential interest to the Android quick actions issue

instead they seek online web resources for information that may assist them in completing the task-at-hand [69, 88].

A common way to find software artifacts pertinent to the developer's task is through the usage of a web search engine [10, 44]. Figure 1.2 shows the artifacts resulting from a developer's search about android notifications for the task in Figure 1.1. We explain the information found in these artifacts and challenges associated with locating task-relevant text as follows.

A web tutorial is a document intended primarily to teach users how to use a technology [6], where a tutorial typically contains a series of structure topics that progressively explain concepts about a technology [35, 36]. Figure 1.3 provides an example of an official tutorial for the Android notifications. This tutorial contains approximately 200 sentences and, on the page's right-hand side, we find that this tutorial has nine sections, each with sub-sections of their own. Reading all sections



Notification actions

Although it's not required, every notification should open an appropriate app activity when tapped. In addition to this default notification action, you can add action buttons that complete an app-related task from the notification (often without opening an activity), as shown in figure 9.

Messages • now

Justin Rhyss

Do you want to go see a movie tonight?

REPLY ARCHIVE

Figure 9. A notification with action buttons

Starting in Android 7.0 (API level 24), you can also add an action to reply to messages or enter other text directly from the notification.

Starting in Android 10 (API level 29), the platform can automatically generate action buttons with suggested intent-based actions.

Adding action buttons is explained further in Create a Notification.

Notification updates and groups

To avoid bombarding your users with multiple or redundant notifications when you have additional updates, you should consider updating an existing notification rather than issuing a new one, or consider using the inbox-style notification to show conversation updates.

However, if it's necessary to deliver multiple notifications, you should consider grouping those separate notifications into a group (available on Android 7.0 and higher). A notification group allows you to collapse multiple notifications into just one post in the notification drawer, with a summary. The user can then expand the notification to reveal the details for each individual notification.

The user can progressively expand the notification group and each notification within it for more details.

On this page

Figure 1.3: Snapshot of the official Android notifications tutorial with highlights relevant to the GitHub issue #1741

of this document could potentially take 10 minutes or more[3] of a developer's time and, to save time, a developer would likely try to find the sections of the document most closely related to their task [44], but not without challenges. For example, using a web browser's search to find content that mentions the '*actions*' keyword, a developer would find eight different matches spread across four different sections and even within a matching section, such as the '*Notification actions*' (under focus in the figure), not all the text might be relevant to the developer, e.g., the section explains notifications for both Android version 7.0 and 12.0, but given that the developer's task is related to the former version, only the text highlighted (in orange) might be of relevance to the task in Figure 1.1.

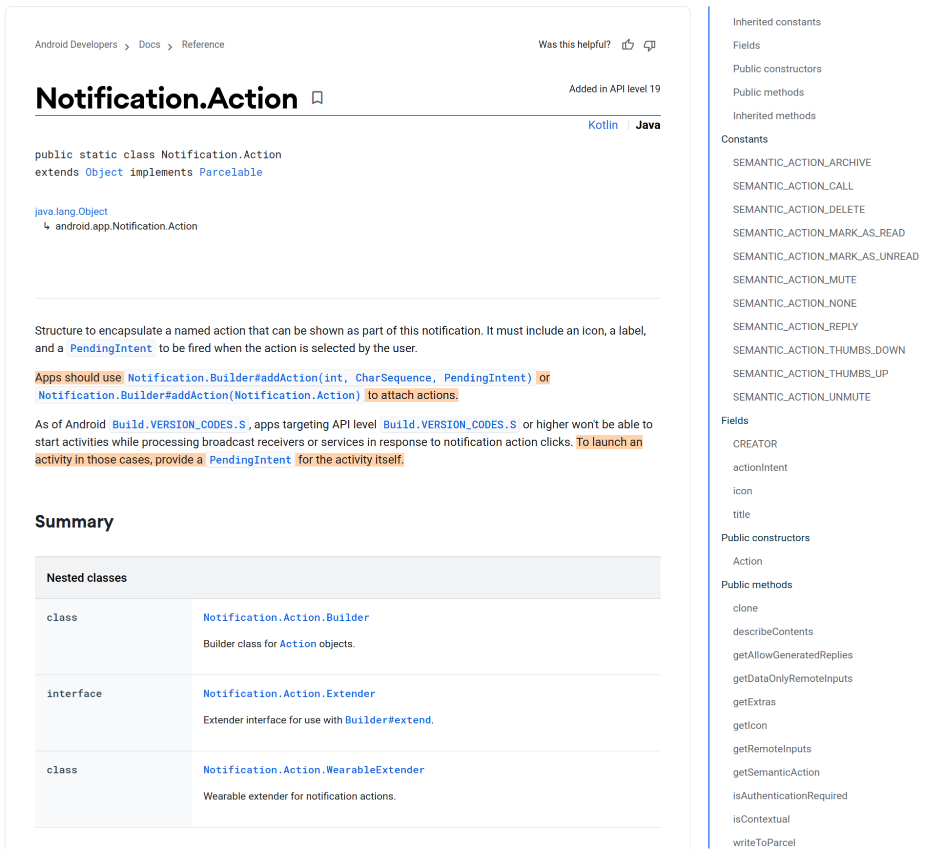
An Application Programming Interface (API) defines how two software libraries communicate [74]. For example, how the mail client software communicates with the native Android software. This communication happens via features and functionality exposed by the API [73] and instructions about its usage are often found in the API's reference documentation. Figure 1.4 shows part of the reference documentation for one of the classes of the Android API, namely '*Notification.Action*'. In the right-hand side of the figure, we find common information in this type of artifact, including a brief summary, constants and fields available, the class's constructor, and each of the functions or methods that this particular class provides, for a total of 35 distinct elements. To find the portions of the text relevant to their task, a developer unfamiliar with this document would face challenges similar to the ones described in the web tutorial artifact. Additionally, we note that complex APIs, such as Android, require combining several classes and method calls [74] to properly perform some instruction, what would mean inspecting at least three other classes with equally complex documentation to understand all the Android API elements used in the quick actions menu[4].

The third artifact resulting from the developer's search about android notifications is a post in a question-and-answer (Q&A) web page, Stack Overflow[5]. Figure 1.5 shows an example of the information found in this type of artifact. A question usually contains both text and code sniptes and it includes a set of tags

[3]Using a standard reading metric of 200 words per minute [37]

[4]https://github.com/k9mail/k-9/pull/1755/files

[5]https://stackoverflow.com/



Android Developers > Docs > Reference

Was this helpful?

Notification.Action

Added in API level 19

Kotlin | Java

```
public static class Notification.Action
extends Object implements Parcelable
```

java.lang.Object
↳ android.app.Notification.Action

Structure to encapsulate a named action that can be shown as part of this notification. It must include an icon, a label, and a `PendingIntent` to be fired when the action is selected by the user.

Apps should use `Notification.Builder#addAction(int, CharSequence, PendingIntent)` or `Notification.Builder#addAction(Notification.Action)` to attach actions.

As of Android `Build.VERSION_CODES.S`, apps targeting API level `Build.VERSION_CODES.S` or higher won't be able to start activities while processing broadcast receivers or services in response to notification action clicks. To launch an activity in those cases, provide a `PendingIntent` for the activity itself.

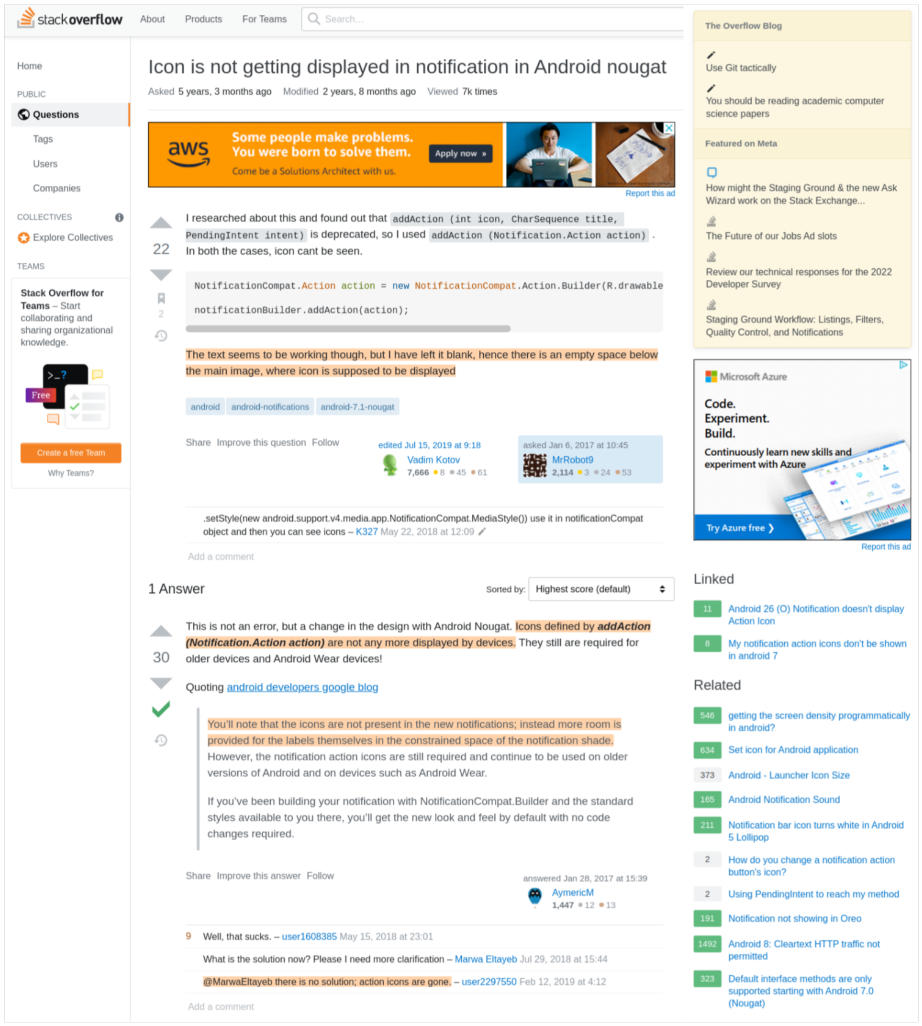
Summary

Nested classes	
`class`	`Notification.Action.Builder` Builder class for `Action` objects.
`interface`	`Notification.Action.Extender` Extender interface for use with `Builder#extend`.
`class`	`Notification.Action.WearableExtender` Wearable extender for notification actions.

Inherited constants
Fields
Public constructors
Public methods
Inherited methods
Constants
SEMANTIC_ACTION_ARCHIVE
SEMANTIC_ACTION_CALL
SEMANTIC_ACTION_DELETE
SEMANTIC_ACTION_MARK_AS_READ
SEMANTIC_ACTION_MARK_AS_UNREAD
SEMANTIC_ACTION_MUTE
SEMANTIC_ACTION_NONE
SEMANTIC_ACTION_REPLY
SEMANTIC_ACTION_THUMBS_DOWN
SEMANTIC_ACTION_THUMBS_UP
SEMANTIC_ACTION_UNMUTE
Fields
CREATOR
actionIntent
icon
title
Public constructors
Action
Public methods
clone
describeContents
getAllowGeneratedReplies
getDataOnlyRemoteInputs
getExtras
getIcon
getRemoteInputs
getSemanticAction
isAuthenticationRequired
isContextual
writeToParcel

Figure 1.4: Snapshot of the Android Notification.Action API reference documentation with highlights relevant to the GitHub issue #1741

about the problem's programming language or technology [83]. Each answer contains similar content and the counter that appears on left of a question and of each answer represents how many other users found them helpful (or not). The user who asked the question can also accept an answer (green check mark) if it correctly solved their problem. We also find a list of similar or related questions at the right portion of the page. This more structured format often assist a developer in navigating through the content of type of artifact [63], for example, a developer could read the problem and then the accepted answer to quickly find information that might be helpful to their task. Nonetheless, a series of factors make finding useful information challenging. For instance, only half of the Android questions on



Stack Overflow for Teams – Start collaborating and sharing organizational knowledge.

Icon is not getting displayed in notification in Android nougat

Asked 5 years, 3 months ago Modified 2 years, 8 months ago Viewed 7k times

aws
Some people make problems.
You were born to solve them.
Apply now »
Come be a Solutions Architect with us.
Report this ad

22

I researched about this and found out that `addAction (int icon, CharSequence title, PendingIntent intent)` is deprecated, so I used `addAction (Notification.Action action)` .
In both the cases, icon cant be seen.

```
NotificationCompat.Action action = new NotificationCompat.Action.Builder(R.drawable

notificationBuilder.addAction(action);
```

The text seems to be working though, but I have left it blank, hence there is an empty space below the main image, where icon is supposed to be displayed

android android-notifications android-7.1-nougat

Share Improve this question Follow edited Jul 15, 2019 at 9:18 Vadim Kotov 7,666 ● 8 ● 45 ● 61 asked Jan 6, 2017 at 10:45 MrRobot9 2,114 ● 3 ● 24 ● 53

.setStyle(new android.support.v4.media.app.NotificationCompat.MediaStyle()) use it in notificationCompat object and then you can see icons – K327 May 22, 2018 at 12:09

Add a comment

1 Answer Sorted by: Highest score (default)

30

This is not an error, but a change in the design with Android Nougat. Icons defined by ***addAction (Notification.Action action)*** are not any more displayed by devices. They still are required for older devices and Android Wear devices!

Quoting android developers google blog

> You'll note that the icons are not present in the new notifications; instead more room is provided for the labels themselves in the constrained space of the notification shade. However, the notification action icons are still required and continue to be used on older versions of Android and on devices such as Android Wear.
>
> If you've been building your notification with NotificationCompat.Builder and the standard styles available to you there, you'll get the new look and feel by default with no code changes required.

Share Improve this answer Follow answered Jan 28, 2017 at 15:39 AymericM 1,447 ● 12 ● 13

9 Well, that sucks. – user1608385 May 15, 2018 at 23:01

What is the solution now? Please I need more clarification – Marwa Eltayeb Jul 29, 2018 at 15:44

@MarwaEltayeb there is no solution; action icons are gone. – user2297550 Feb 12, 2019 at 4:12

Add a comment

The Overflow Blog

Use Git tactically

You should be reading academic computer science papers

Featured on Meta

How might the Staging Ground & the new Ask Wizard work on the Stack Exchange...

The Future of our Jobs Ad slots

Review our technical responses for the 2022 Developer Survey

Staging Ground Workflow: Listings, Filters, Quality Control, and Notifications

Linked

11 Android 26 (O) Notification doesn't display Action Icon

8 My notification action icons don't be shown in android 7

Related

546 getting the screen density programmatically in android?

634 Set icon for Android application

373 Android - Launcher Icon Size

165 Android Notification Sound

211 Notification bar icon turns white in Android 5 Lollipop

2 How do you change a notification action button's icon?

2 Using PendingIntent to reach my method

191 Notification not showing in Oreo

1492 Android 8: Cleartext HTTP traffic not permitted

323 Default interface methods are only supported starting with Android 7.0 (Nougat)

Figure 1.5: Snapshot of a Stack Overflow question about Android notifications with highlights relevant to the GitHub issue #1741

Stack Overflow have an accepted answer [65] and millions of questions have more than one answer [63]. Technologies also evolve and answers become obsolote [1]. Hence, despite the fact that structured data (i.e., votes and accepted answers) might assist a developer, finding task-relevant text in this type of artifact is also not trivial.

According to the discussion above, it is clear that if no tool support is provided, much of the process of locating text relevant to a task in a natural language artifact falls on the developer's shoulders [11, 29, 41] and given how quickly developers progress to use new kinds of technology to record pertinent information, the challenges we outlined are not exclusive to the three types of artifact in Figure 1.2, rather they are common to many kinds of natural language software artifacts [44, 81].

1.2 State of the art

Researchers have long recognized the value of assisting developers in locating information in the natural language artifacts sought as part of a software task. They have proposed many tools and approaches that combine Information Retrieval (IR), Natural Language Processing (NLP) and Machine Learning (ML) techniques, often using syntactic properties of the text, and an artifact's meta-data, to aid developers in locating the portion of the text that might be useful within these artifacts.

As examples of techniques available in the literature, Nadi and Treude consider that relevant information in Q&A web pages is often found in text with conditional clauses (i.e., sentences with the word '*if*') [63] while Robillard and Chhetri assume that relevant text in API documents mention a code element such as a class name or method signature [73], where they use such assumptions in techniques that identify this text automatically. These pre-define rules would fail to identify the relevant text in the artifacts of our running example (Figure 1.1) both because these syntactic rules do not take a developer's task into account and because these rules were derived from one type of artifact, implying that there are no guarantees that they extend to the different types of artifact that a developer might use.

Recognizing that relevance is often context-specific [8], i.e., information needs are associated with details of the task or characteristics of the developer performing it [73], many other software engineering researchers have proposed approaches that automatically identify information based on a specific context [79, 90, 92]. For example, Xu et al. identify relevant text in Stack Overflow posts based on how similar the text within an answer is with regards to a input query, which represents the developer's programming task. Although this assists in the automatic identification of context-specific information, the use of structural data by this and other techniques [45, 79] limits their adoption since it is evidently not possible to apply such techniques across different types of artifact that lack such meta-data [3, 8]. For example, Xu et al.'s approach only applies to one out of the three artifacts from the developer's search about android notifications (Figure 1.2) and for the artifact that it applies to, i.e., the Stack Overflow post in Figure 1.5, the approach would not identify relevant text in answers other than the accepted answer, what further illustrates some of the drawbacks on the usage of meta-data to guide the automatic identification of task-relevant text.

The discussion above evidences limitations of existing techniques in assisting developers in locating useful text across different kinds of artifacts. In summary:

1. it shows the need to investigate rules for the relevance of text considering how such text is found across different types of artifacts;
2. it shows the need for context-specific approaches; and
3. it asks how to relax limitations of existing artifact-specific approaches so that we can design more generalizable approaches for the automatic identification of text relevant to a developer's task.

1.3 Thesis

Our thesis statement is that:

> *A developer can effectively complete a software development task when automatically provided with text relevant to their task extracted from pertinent natural language artifacts by an artifact-type agnostic technique.*

Designing a generalizable technique means determining if the rules governing how natural language information is constructed across different types of artifact can guide us to information relevant to a task [39]. Hence, in the first part of this thesis, we present a formative study that ask what are common properties, if any, in the text deemed relevant to a software task and found across different types of artifacts? To answer this question, we report on a controlled experiment that examines the text that twenty developers deemed relevant in artifacts of different types associated with six software tasks.

By *characterizing* task-relevant text found in different kinds of artifacts, this study complements and adds to previous research that has examined text relevant to particular tasks and one kind of artifact [13, 40, 72, 73]. Notably, our analysis of natural language text in bug reports, API documents and Q&A websites inspected as part of this study show consistency in the meaning, or *semantics*, of the text deemed relevant to a task by a developer, suggesting that semantics might assist in the automatic identification of task-relevant text for the different types of artifacts a developer might use when performing a task [44, 54].

Approaches that interpret the meaning of the text have been successfully used for a variety of development activities, such as for finding who should fix a bug [91], searching for comprehensive code examples [79], or assessing the quality of information available in bug reports [14]. Nonetheless, we are not aware of the usage of semantic-based techniques in the design of a generalizable technique that assists developers in discovering task-relevant text across different types of artifacts. Hence, the second part of this thesis describes the investigation of a *design space* of six possible techniques that incorporate the semantics of words [20, 59] and sentences [25, 53] to automatically identify text likely relevant to a developer's task. Assessment of these techniques reveals that semantic-based techniques achieve recall comparable to a state-of-the-art technique aimed at one type of artifact [90], but that they do so across multiple artifact types.

Provided that we find consistency in the text that is relevant to a task within different kinds of artifacts and that semantic-based techniques can automatically identify such text; in the last part of this thesis, we consider whether these approaches can assist a software developer while they *work* on a task. We introduce TARTI[6], a web browser plug-in that automatically identifies and highlights text relevant to a developer's task in a web page and then, we present a controlled experiment that investigates benefits brought by using this tool, if any. We report how participants performed two Python programming tasks when provided with a curated set of artifacts associated with the tasks considered and assisted (or not) by TARTI. With this experiment, we compare the correctness of the solutions of each task performed by participants with and without tool assistance and we also detail

[6] Automatic Task-Relevant Text Identifier *AM: Tentative name :)*

the perceived usefulness of the text automatically identified and shown by the tool. Results indicate that participants found the text automatically identified useful in two out of the three tasks of the experiment and that tool support also led to more correct solutions in one of the tasks in the experiment. These results provide initial evidence on the role of semantic-based tools for supporting a developer's discovery of task-relevant information across different natural language artifacts.

1.4 Contributions

This thesis makes the following contributions to the field of software engineering:

- It details a formative study that characterizes task-relevant text across a variety of natural language artifacts, including API documentation, Q&A websites, and bug reports that are pertinent to six software tasks;
- It introduces six possible techniques that buil upon approaches that interpret the meaning, or semantics, of text to automatically identify task-relevant text across different kinds of software artifacts, where:
- It shows how the most promising semantic-based approaches that we have explored have accuracy comparable to a state-of-the-art approach tailored to one kind of artifact [90], i.e., Stack Overflow.
- It presents an empirical experiment that provides initial evidence on how TARTI assists a software developer while they work on a task.

This dissertation also contributes with three different datasets (*DS*) that can be used for replication purposes and future research in the field:

- $\text{DS}_{synthetic}$ provides a unique corpus of 20 natural language artifacts that include annotations from 20 participants of text deemed relevant to the six tasks in our study on characterizing task-relevant text;
- $\text{DS}_{android}$ is a dataset with 50 Android tasks and associated natural language artifacts with annotations from three developers of the text relevant to these tasks;

- DS$_{python}$ contains three Python tasks and it includes annotations from 24 participants that indicated what text assisted them in writing each task's solution.

1.5 Structure of the Thesis

GM: review *AM: I stopped here*

In Chapter 2, we present an overview of state of the art. The chapter details text analysis in software artifacts, existing approaches and tools that assist in the automatic identification of text, and how these tools fit under the umbrella of studies that seek to improve a developer's work.

Chapter **??** presents our empirical study to characterize task-relevant text. We provide details on the tasks and artifacts that we have selected for this study and then, we present our findings on the text considered relevant and the qualitative factors related to locating task-relevant text.

Chapter **??** describes the groundwork for producing the corpus (DS$_{android}$) that we use to evaluate the semantic-based techniques detailed in the chapter that follows. It describes the selection of tasks and artifacts pertinent to each task as well as how three human annotators identified relevant text in each of the artifacts gathered.

Chapter **??** details the semantic-based techniques we investigate for automatically identifying task-relvant text. The first two techniques that the chapter presents use word embeddings to identify likely relevant text via semantic similarity and via a deep neural network. A third sentence-level technique filters (or not) the output of the word-level techniques according to frame semantics analysis. We combine these techniques for a total of six possible techniques, assessing the text that they automatically identify for the tasks and artifact types available in the DS$_{android}$ corpus.

Chapter **??** details our empirical experiment investigating whether a tool embedding a semantic-based technique assists a software developer in locating information that helps them complete Python programming tasks. We begin by detailing experimental procedures and then, we report results from the experiment.

In Chapter **??**, we discuss challenges and decisions made throughout our work as well as implications from our findings.

Chapter **??** concludes this work by reflecting on the contributions in the thesis and by outlining potential future work. *AM: Will review where I will have future work*

Chapter 2

Related Work

Performing a task on a software system, like fixing a bug or adding a feature typically requires a developer to consult a number of different kinds of artifacts, such as API documentation, bug reports, community forums and web tutorials. Developers produce such natural language artifacts on a continuous basis [70] and there has been a steep growth in the number of natural language artifacts available [8, 84].

Natural language artifacts have become intrinsically tied to software development [49], what led software engineering researchers to both investigate properties of the text appearing in these artifacts [21, 52] and design approaches that can be embedded in tools that mine the textual data available to assist software developer performing some task [8, 18, 32]. In Sections 2.1 and 2.2 we respectively provide background information on the significant body of work that investigates properties of the text in software engineering artifacts and the techniques that automatically identify text of interest to a particular task.

Most of the techniques that extract information from natural language artifacts, including the work presented in this thesis, are examples of applications that assist help developers in performing several tasks during their work day [55]. Other applications have exploited textual data to provide guidance to developers reading software documents [75, 82] or to explain solutions for programming tasks found in community forums [79]. We discuss these and other applications in Section 2.3.

2.1 Natural Language Text in Software Artifacts

Software engineering researchers have investigated several questions on the nature of the text in software artifacts. These studies have focused on different kinds of artifacts and have deepen researchers' knowledge on the richness of information in natural language artifacts, serving as a first step towards the design of automatic tools for identifying and extracting text from these artifacts [5, 52].

A significant body of work has proposed taxonomies for the text available in development mailing lists [21, 33], API documents [52], bug reports [5], and others. In written development emails, Di Sorbo et al. have found that text can be classified according to its purpose (e.g., feature request, solution proposal, etc.) [21], suggesting that text under certain categories might be useful for developers performing specific tasks. They have sampled 100 emails and identified that text for feature requests often contain expressions in the form suggestions (e.g., '*we should add a new button*') while solution proposals are often expressed in the form of attempts (e.g., '*let's try a new method to compute cost*').

Maalej and Robillard have inspected the Java SDK 6 and .NET 4.0. API documentation [52] and proposed a set of categories (e.g., directives, purpose, rationale, functionality, etc.) that define how knowledge is structured in API documentation. This taxonomy led Robillard and Chhetri to find that directives—or pieces of information that the programmers cannot afford to ignore, such as instructions on how to efficiently perform some I/O operation—often follow certain linguistic patterns [73], what made they produce a catalog of patterns that capture directives found in the Java SDK 6 documentation.

Rastkar et al. observe that much of the text in a bug report resembles a conversation between different subjects [71] while Arya et al. argue that due to this conversational nature, a single bug report contains several threads each pertaining to some topic, e.g., reproducing a bug, discussing potential solutions, testing, or reporting progress [5]. Although the topicality of the content in bug reports might encourage the design of tools that automatically identify such text, researchers have also found high variance in the text of different issues within a software project [12], what poses significant challenges to approaches that gather information from multiple bug reports.

Although the studies described above shed light on many aspects of natural language text in software artifacts, they do not discuss common properties of the text that is relevant to some software task found in different kinds of artifacts. In Chapter **??** we study text in API documentation, bug reports, and community forums that is considered relevant to a software task.

2.2 Automatic Text Identification Approaches

Information useful to a software task can be buried in irrelevant text or attached to non-intuitive blocks of text, making it difficult to discover [73]. In this section, we detail tools and approaches from related work that seek to assist developers in identifying information within the natural language text of software artifacts.

2.2.1 Pattern Matching Approaches

Pattern matching approaches rely on regular expressions describing a sequence of tokens that represent a relevant text fragment [8]. Tokens can either represent words or linguistic elements extracted using Natural Language Processing (NLP).

As examples of pattern matching approaches, DeMIBuD [13] and Knowledge Recommender (Krec) [52, 73] are tools that detect relevant sentences in bug reports and API documentation, respectively. These tools use a set of patterns derived from annotated data to identify relevant text. Krec uses 361 unique patterns to detect relevant sentences mentioning a code element (e.g., a class name or method signature) in API documentation [73]. Likewise, DeMIBuD uses a set of 154 discourse patterns to detect sentences relevant to understanding a bugs expected behaviour and steps to reproduce it, which are essential to bug triaging tasks.

In Stack Overflow posts, Nadi and Treude [63] have both applied the original set of patterns from Krec [73] and proposed heuristics that use conditional clauses (i.e., sentences with the word '*if*') to identify text that help a developer decide whether they want to carefully inspect a Stack Overflow posts or skip it.

Although the heuristics and regular expressions used in the aforementioned studies are often light-weight and effective [8, 52], pattern matching approaches are specific to certain kinds of domain and types of artifact [27], what limits using them in a general manner.

2.2.2 Summarization Approaches

Extractive text summarization techniques are used in natural language artifacts in software engineering to produce a summary of the artifact's content. A summary represents key information that may help a developer complete their task [8]. There are summarization techniques based on both supervised and unsupervised learning [60] and one can summarize the entire content of an artifact or content specific to a input query, as in *query-based* summarization [28, 34].

A number of summarization approaches target bug reports and GitHub issues, largely focusing on identifying key information within these artifacts. Rastkar and colleagues [72] use a supervised learning approach to summarize the content of bug reports showing that conversational features used to summarize emails [62] can also be applied to bug reports while Lotufo et al. [51] proposed an unsupervised summarization approach that automates the identification of sentences that a developer would first read when a inspecting bug report.

While many summarization approaches largely rely on lexical aspects in text, researchers have also made use of structured data available in software artifacts [15, 67, 82]. For example, Ponzanelli et al. proposed a summarization approach that uses the number of votes an answer has on Stack Overflow as a filter to the text in the summary for a Stack Overflow post [67]. As another example, DeepSum [45] pre-processes a bug report dividing sentences containing software elements, the reporter of the bug, and any other sentences in the bug report to produce summaries containing more diverse information.

A smaller number of summarization approaches have focused on producing task specific summaries [79, 90]. These approaches pose the problem of finding task-relevant text as a query-based extractive summarization problem and tools such as AnswerBot [90] identify relevant text in Stack Overflow posts based on the content of the text, how similar that content is with regards to a input query (i.e., task) and the structured data available on each of the answers in a Stack Overflow post (i.e., number of votes or whether an answer is the accepted answer).

Although we refrain from using structural data or assumptions on the content of an artifact because this might hinder the design of an automatic approach that identifies text relevant to some task in a more general manner, in Chapter **??** we compare the techniques that we explore in this thesis to the state-of-the-art.

2.2.3 Machine Learning Approaches

Machine Learning (ML) approaches take the text of a natural language software artifact and identify the sentences likely relevant to a particular software task using *supervised* or *unsupervised learning* methods [94].

Supervised learning approaches use a set of features and labeled data to train classifiers with the goal of identifying sentences relevant to certain software task. We have already presented supervised approaches that use text summarization (*i.e.,* [72]) and there are also approaches that identify relevant parts of software tutorials [35] or API documents [27, 52]. Despite their value, the cost and effort of hiring skilled workers to produce labeled data in software engineering artifacts has been a major limitation to the usage of supervised learning methods [4].

Unsupervised learning approaches do not require labelled data and determine relevant sentences according to properties inferred from the data. DeepSum [45] and Lotufo et al.'s [51] techniques are examples of unsupervised approaches in the scope of text summarization.

Other unsupervised approaches (*e.g.,* [36, 67, 68]) are mostly based around variations of the PageRank [64] or LexRank [23] algorithms. These algorithms represent all the text in an artifact as a graph. Then, they establish relationships (*i.e.,* weighted edges in the graph) between different sentences (*i.e.,* nodes in the graph) and select the nodes with highest weights as the most relevant ones. A crucial step in building the graph is in establishing relationships between nodes. Early approaches [36, 51] use Vector Space Model (VSM) [76] for this purpose while more modern ones [34, 79] use different word embeddings [9, 59], which we detail in Section 2.2.4.

2.2.4 Deep Learning Approaches

One substantial challenge of standard Machine Learning (ML) approaches is that researchers must engineer which features or properties of the text to use [24]. For example, Rastkar et al. uses conversational features in the text of a bug report to assist in determining which sentences to include in the bug report's summary [72] while Petrosyan and colleagues use linguistic and structural properties in the text of API documents to identify key text explaining API elements [66]. Given the specificity of such features, researchers have questioned the generalizability of standard ML approaches [27, 89].

In contrast to the human-engineered features, Deep Learning (DL) approaches allow the automatic extraction of features from training data through a series of mathematical transformations [19, 93]. Deep learning has led to groundbreaking advancements in many research areas (e.g., machine translation [50]) and, given its wide range of applications, we focus on its usage in natural language text appearing in software engineering artifacts [24, 46, 87].

Software engineering researchers have identified that the text in software task often differs from the text in the artifacts that are related to that tasks [34]. These so-called *lexical mismatches* [92] make it difficult to identify information of interest to a task and a number of studies have used DL neural embeddings [59] to bridge lexical gaps between the text of different software artifacts.

Neural, or word, embeddings produce vector representations in a continuous space, where words with similar meanings are typically close in the vector space model [31, 58]. Their usage has allowed researchers to improve the identification API elements pertinent to a programming task [92] or to more accurately assess the quality of the content in bug reports [14]. Word embeddings have become a common way to compare the semantic similarity of the text [57], being applied in query-based summarization techniques such as AnswerBot [90] or PageRank-based approaches such as HoliRank [68].

Many other DL studies in software engineering [24, 46, 87] use neural network architectures in a variety of software engineering tasks, including code comprehension [2, 56], community forum analysis [47, 86], or requirements traceability [16, 30]. DeepSum [45], which we described earlier, is an example of a summarization approach that uses an encoder-decoder architecture [17] to identify sentences of interest in bug reports. As another example, Xi et al. [88] use a Convolutional Neural Network (CNN) [43] to identify sentences pertaining to Di Sorbo et al.'s topics [21].

Few studies in software engineering have considered more modern neural networks [87] able to leverage hidden features not only between words but also between sentence pairs (e.g., BERT [20]). These models might assist in determining implicit relationships between the text in a task and the text in a relevant sentence within a software artifact. As such, Chapter **??** describes how we use BERT for automatically identifying text relevant to a software task.

2.3 Improving Developers' Productivity

The tools and approaches presented in Section 2.2 are examples of studies that help developers in locating information that assists them to complete a software task. These studies fit in the bigger context of software engineering research that facilitates or improves the quality of a developer's work [38, 55, 77].

Researchers have had a long interest in making knowledge bases that developers working on a task can benefit from. Hipikat [18] is a seminal tool that exemplifies this concept in action. It automatically tracks the artifacts used by a developer as part of a development tasks so that it can recommend these artifacts to any future developers who work on similar tasks [18]. Other tools like Strata summarize knowledge produced by developers while they navigate on the web so that other developers have a a head start when performing similar tasks [49].

As part of their work, developers engage in many sensemaking and decision making activities [78] and several studies have investigated how to provide means to better assist developers in performing such activities [7, 48, 49]. For example, tools such as Deep Intellisense display code changes, filed bugs, and forum discussions mined from an organization's database to better assist a developer understand the rationale behind the code they currently work on [32], and researchers have extended this idea to publicly available data, e.g., GitHub issues [85] or pull requests [26].

Although the studies mentioned above do not directly relate to this work of dissertation, we build upon many of the research procedures outlined in these and many other studies in the field. For example, in the evaluation of Strata Liu et al. describe procedures considering how a control and tool-assisted group complete information-seeking tasks [49], what assisted in the design of our experiment for evaluating an automated approach to task-relevant text identification (Chapter **??**).

Bibliography

[1] M. Allamanis and C. Sutton. Why, when, and what: Analyzing stack overflow questions by topic, type, and code. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 53–56. IEEE Press, 2013. ISBN 9781467329361. → page 7

[2] M. Allamanis, E. T. Barr, C. Bird, and C. Sutton. Suggesting accurate method and class names. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 38–49, 2015. → page 19

[3] V. Arnaoudova, S. Haiduc, A. Marcus, and G. Antoniol. The use of text retrieval and natural language processing in software engineering. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE)*, pages 949–950. IEEE Computer Society, 2015. → page 9

[4] A. Arpteg, B. Brinne, L. Crnkovic-Friis, and J. Bosch. Software engineering challenges of deep learning. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 50–59, 2018. doi:10.1109/SEAA.2018.00018. → page 18

[5] D. Arya, W. Wang, J. L. Guo, and J. Cheng. Analysis and detection of information types of open source software issue discussions. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 454–464, 2019. doi:10.1109/ICSE.2019.00058. → page 15

[6] D. M. Arya, J. L. Guo, and M. P. Robillard. Information correspondence between types of documentation for apis. *Empirical Software Engineering*, 25(5):4069–4096, 2020. → page 3

[7] M. Barnett, C. Bird, J. Brunet, and S. K. Lahiri. Helping developers help themselves: Automatic decomposition of code review changesets. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 134–144. IEEE, 2015. → page 20

[8] G. Bavota. Mining unstructured data in software repositories: Current and future trends. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 5, pages 1–12, 2016. doi:10.1109/SANER.2016.47. → pages 1, 8, 9, 14, 16, 17

[9] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, Dec. 2017. doi:10.1162/tacl_a_00051. URL https://www.aclweb.org/anthology/Q17-1010. → page 18

[10] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer. Two studies of opportunistic programming. *Proc. of the 27th Int'l Conf. on Human factors in computing systems - CHI 09*, page 1589, 2009. → page 3

[11] K. Byström and K. Järvelin. Task complexity affects information seeking and use. *Information Processing and Management*, 31(2):191–213, 1995. ISSN 03064573. → page 8

[12] O. Chaparro, J. M. Florez, and A. Marcus. On the vocabulary agreement in software issue descriptions. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 448–452, 2016. doi:10.1109/ICSME.2016.44. → page 15

[13] O. Chaparro, J. Lu, F. Zampetti, L. Moreno, M. Di Penta, A. Marcus, G. Bavota, and V. Ng. Detecting missing information in bug descriptions. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, pages 396–407, 2017. ISBN 9781450351058. doi:10.1145/3106237.3106285. → pages 10, 16

[14] O. Chaparro, C. Bernal-Cárdenas, J. Lu, K. Moran, A. Marcus, M. Di Penta, D. Poshyvanyk, and V. Ng. Assessing the quality of the steps to reproduce in bug reports. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 86–96, 2019. ISBN 9781450355728. doi:10.1145/3338906.3338947. → pages 10, 19

[15] C. Chen, S. Gao, and Z. Xing. Mining Analogical Libraries in Q&A Discussions – Incorporating Relational and Categorical Knowledge into Word Embedding. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 338–348, 2016. doi:10.1109/SANER.2016.21. → page 17

[16] C. Chen, Z. Xing, Y. Liu, and K. O. L. Xiong. Mining likely analogical apis across third-party libraries via large-scale unsupervised api semantics embedding. *IEEE Transactions on Software Engineering*, 47(3):432–447, 2019. → page 19

[17] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014. → page 19

[18] D. Cubranic, G. Murphy, J. Singer, and K. Booth. Hipikat: a project memory for software development. *IEEE Transactions on Software Engineering*, 31 (6):446–465, 2005. doi:10.1109/TSE.2005.71. → pages 14, 20

[19] L. Deng and Y. Liu. *Deep Learning in Natural Language Processing*. Springer Publishing Company, Incorporated, 1st edition, 2018. ISBN 9789811052088. → page 19

[20] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. → pages 10, 20

[21] A. Di Sorbo, S. Panichella, C. A. Visaggio, M. Di Penta, G. Canfora, and H. C. Gall. Development emails content analyzer: Intention mining in developer discussions (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 12–23, 2015. doi:10.1109/ASE.2015.12. → pages 14, 15, 19

[22] S. Endrikat, S. Hanenberg, R. Robbes, and A. Stefik. How do API documentation and static typing affect API usability? In *Proc. of the 36th Int'l Conf. on SE*, pages 632–642, 2014. ISBN 978-1-4503-2756-5. → page 1

[23] G. Erkan and D. R. Radev. Lexrank: Graph-based lexical centrality as salience in text summarization. *J. Artif. Int. Res.*, 22(1):457–479, Dec. 2004. ISSN 1076-9757. URL http://dl.acm.org/citation.cfm?id=1622487.1622501. → page 18

[24] F. Ferreira, L. L. Silva, and M. T. Valente. Software engineering meets deep learning: a mapping study. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, pages 1542–1549, 2021. → pages 18, 19

[25] C. J. Fillmore. Frame semantics and the nature of language. *Annals of the New York Academy of Sciences*, 280(1):20–32, 1976. → page 10

[26] V. d. C. L. Freire et al. *Characterization of design discussions in modern code review.* PhD thesis, 2021. → page 20

[27] D. Fucci, A. Mollaalizadehbahnemiri, and W. Maalej. On using machine learning to identify knowledge in api reference documentation. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 109–119, 2019. → pages 16, 18, 19

[28] J. Goldstein et.al. Summarizing Text Document: Sentence Selection and Evaluation Metrics. *In Proceeding of SIGIR'99*, pages 121–128, 1999. → page 17

[29] M. K. Gonçalves, C. R. de Souza, and V. M. González. Collaboration, information seeking and communication: An observational study of software developers' work practices. *J. Univers. Comput. Sci.*, 17(14): 1913–1930, 2011. → page 8

[30] J. Guo, J. Cheng, and J. Cleland-Huang. Semantically enhanced software traceability using deep learning techniques. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pages 3–14. IEEE, 2017. → page 19

[31] Z. S. Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954. → page 19

[32] R. Holmes and A. Begel. Deep intellisense: A tool for rehydrating evaporated information. In *Proc. of the 2008 Int'l Working Conf. on Mining Software Repositories*, pages 23–26, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-024-1. → pages 14, 20

[33] Q. Huang, X. Xia, D. Lo, and G. C. Murphy. Automating intention mining. *IEEE Transactions on Software Engineering*, 2018. → page 15

[34] Q. Huang, X. Xia, Z. Xing, D. Lo, and X. Wang. Api method recommendation without worrying about the task-api knowledge gap. In *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 293–304, 2018. doi:10.1145/3238147.3238191. → pages 17, 18, 19

[35] H. Jiang, J. Zhang, X. Li, Z. Ren, and D. Lo. A more accurate model for finding tutorial segments explaining apis. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*,

volume 1, pages 157–167, 2016. doi:10.1109/SANER.2016.59. → pages 3, 18

[36] H. Jiang, J. Zhang, Z. Ren, and T. Zhang. An unsupervised approach for discovering relevant tutorial fragments for APIs. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pages 38–48, 2017. doi:10.1109/ICSE.2017.12. → pages 3, 18

[37] M. A. Just and P. A. Carpenter. A theory of reading: From eye fixations ot comprehension. *Psychological Review*, 87(4):329–354, 1980. doi:10.1037/0033-295X.87.4.329. → page 5

[38] M. Kersten and G. C. Murphy. Using task context to improve programmer productivity. In *Proc. of the 14th ACM SIGSOFT Int'l Symp. on Foundations of SE*, pages 1–11, 2006. ISBN 1-59593-468-5. → page 20

[39] W. Kintsch and T. A. van Dijk. Toward a model of text comprehension and production. *Psychological Review*, 85(5):363–394, 1978. ISSN 0033295X. → page 9

[40] A. J. Ko and B. A. M. and. A linguistic analysis of how people describe software problems. In *Visual Languages and Human-Centric Computing (VL/HCC'06)*, pages 127–134, Sep. 2006. → page 10

[41] A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *IEEE Trans. Softw. Eng.*, 32(12):971–987, Dec. 2006. ISSN 0098-5589. doi:10.1109/TSE.2006.116. → page 8

[42] A. J. Ko, R. DeLine, and G. Venolia. Information needs in collocated software development teams. In *29th International Conference on Software Engineering (ICSE'07)*, pages 344–353. IEEE, 2007. → page 2

[43] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012. → page 19

[44] H. Li, Z. Xing, X. Peng, and W. Zhao. What help do developers seek, when and how? In *2013 20th Working Conf. on Reverse Engineering (WCRE)*, pages 142–151, Oct 2013. → pages 2, 3, 5, 8, 10

[45] X. Li, H. Jiang, D. Liu, Z. Ren, and G. Li. Unsupervised deep bug report summarization. In *Proceedings of the 26th Conference on Program*

Comprehension (ICPC), pages 144–155, 2018. ISBN 978-1-4503-5714-2. doi:10.1145/3196321.3196326. → pages 8, 17, 18, 19

[46] X. Li, H. Jiang, Z. Ren, G. Li, and J. Zhang. Deep learning in software engineering. *arXiv preprint arXiv:1805.04825*, 2018. → page 19

[47] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, M. Lanza, and R. Oliveto. Sentiment analysis for software engineering: How far can we go? In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 94–104, May 2018. doi:10.1145/3180155.3180195. → page 19

[48] M. X. Liu, N. Hahn, A. Zhou, S. Burley, E. Deng, J. Hsieh, B. A. Myers, and A. Kittur. UNAKITE: Support developers for capturing and persisting design rationales when solving problems using web resources. 2018. → page 20

[49] M. X. Liu, A. Kittur, and B. A. Myers. To reuse or not to reuse? a framework and system for evaluating summarized knowledge. *Proceedings of the ACM on Human-Computer Interaction*, 5(CSCW1):1–35, 2021. → pages 14, 20

[50] A. Lopez. Statistical machine translation. *ACM Computing Surveys (CSUR)*, 40(3):1–49, 2008. → page 19

[51] R. Lotufo, Z. Malik, and K. Czarnecki. Modelling the 'hurried' bug report reading process to summarize bug reports. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 430–439, 2012. doi:10.1109/ICSM.2012.6405303. → pages 17, 18

[52] W. Maalej and M. P. Robillard. Patterns of knowledge in api reference documentation. In *IEEE Transactions on Software Engineering*, volume 39, pages 1264–1282, 2013. doi:10.1109/TSE.2013.12. → pages 14, 15, 16, 18

[53] A. Marques, G. Viviani, and G. C. Murphy. Assessing semantic frames to support program comprehension activities. In *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*, pages 13–24, 2021. doi:10.1109/ICPC52881.2021.00011. → page 10

[54] A. Meyer, E. T. Barr, C. Bird, and T. Zimmermann. Today was a good day: The daily life of software developers. *IEEE Transactions on Software Engineering*, 2019. → page 10

[55] A. N. Meyer, L. E. Barton, G. C. Murphy, T. Zimmermann, and T. Fritz. The work life of developers: Activities, switches and perceived productivity.

IEEE Transactions on Software Engineering, 43(12):1178–1193, 2017. doi:10.1109/TSE.2017.2656886. → pages 1, 14, 20

[56] Q. Mi, J. Keung, Y. Xiao, S. Mensah, and Y. Gao. Improving code readability classification using convolutional neural networks. *Information and Software Technology*, 104:60–71, 2018. → page 19

[57] R. Mihalcea, C. Corley, C. Strapparava, et al. Corpus-based and knowledge-based measures of text semantic similarity. In *Aaai*, volume 6, pages 775–780, 2006. → page 19

[58] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. 2013. → page 19

[59] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems (NIPS)*, pages 3111–3119, 2013. → pages 10, 18, 19

[60] L. Moreno and A. Marcus. Automatic software summarization: The state of the art. *Proc. - 2017 IEEE/ACM 39th Int'l Conf. on SE Companion*, pages 511–512, 2017. → page 17

[61] G. C. Murphy, M. Kersten, M. P. Robillard, and D. Čubranić. The emergent structure of development tasks. In A. P. Black, editor, *ECOOP 2005 - Object-Oriented Programming*, pages 33–48, 2005. ISBN 978-3-540-31725-8. → page 1

[62] G. Murray and G. Carenini. Summarizing spoken and written conversations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 773–782, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics. URL http://dl.acm.org/citation.cfm?id=1613715.1613813. → page 17

[63] S. Nadi and C. Treude. Essential sentences for navigating stack overflow answers. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 229–239, 2020. doi:10.1109/SANER48275.2020.9054828. → pages 6, 7, 8, 16

[64] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. URL http://ilpubs.stanford.edu:8090/422/. Previous number = SIDL-WP-1999-0120. → page 18

[65] C. Parnin, C. Treude, L. Grammel, and M.-A. Storey. Crowd documentation: Exploring the coverage and the dynamics of api discussions on stack overflow. *Georgia Institute of Technology, Tech. Rep*, 11, 2012. → pages 1, 7

[66] G. Petrosyan, M. P. Robillard, and R. De Mori. Discovering information explaining API types using text classification. In *Proc. of the 37th Int'l Conf. on SE - Volume 1*, pages 869–879, Piscataway, NJ, USA, 2015. IEEE Press. ISBN 978-1-4799-1934-5. → page 19

[67] L. Ponzanelli, A. Mocci, and M. Lanza. Summarizing complex development artifacts by mining heterogeneous data. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 401–405, 2015. doi:10.1109/MSR.2015.49. → pages 17, 18

[68] L. Ponzanelli, S. Scalabrino, G. Bavota, A. Mocci, R. Oliveto, M. Di Penta, and M. Lanza. Supporting software developers with a holistic recommender system. In *Proc. of the 39th Int'l Conf. on SE*, pages 94–105, Piscataway, NJ, USA, 2017. IEEE Press. ISBN 978-1-5386-3868-2. → pages 18, 19

[69] N. Rao, C. Bansal, T. Zimmermann, A. H. Awadallah, and N. Nagappan. Analyzing web search behavior for software engineering tasks. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 768–777, 2020. doi:10.1109/BigData50022.2020.9378083. → page 3

[70] S. Rastkar. *Summarizing software artifacts*. PhD thesis, University of British Columbia, 2013. → page 14

[71] S. Rastkar and G. C. Murphy. Why did this code change? In *Proceedings of the 2013 International Conference on Software Engineering (ICSE)*, pages 1193–1196, 2013. ISBN 978-1-4673-3076-3. → page 15

[72] S. Rastkar, G. C. Murphy, and G. Murray. Summarizing software artifacts: A case study of bug reports. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE)*, pages 505–514, 2010. ISBN 9781605587196. doi:10.1145/1806799.1806872. → pages 10, 17, 18

[73] M. P. Robillard and Y. B. Chhetri. Recommending reference api documentation. *Empirical Software Engineering*, 20(6):1558–1586, Dec. 2015. ISSN 1382-3256. doi:10.1007/s10664-014-9323-y. → pages 5, 8, 10, 15, 16

[74] M. P. Robillard and R. Deline. A field study of API learning obstacles. *Empirical Software Engineering*, 16(6):703–732, 2011. → pages 1, 5

[75] M. P. Robillard, A. Marcus, C. Treude, G. Bavota, O. Chaparro, N. Ernst, M. A. Gerosa, M. Godfrey, M. Lanza, M. Linares-Vásquez, et al. On-demand developer documentation. In *2017 IEEE International conference on software maintenance and evolution (ICSME)*, pages 479–483. IEEE, 2017. → page 14

[76] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, Nov. 1975. ISSN 0001-0782. doi:10.1145/361219.361220. URL https://doi.org/10.1145/361219.361220. → page 18

[77] C. Satterfield, T. Fritz, and G. C. Murphy. Identifying and describing information seeking tasks. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pages 797–808, 2020. → page 20

[78] J. Sillito, G. C. Murphy, and K. De Volder. Questions programmers ask during software evolution tasks. In *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 23–34, 2006. → pages 2, 20

[79] R. F. Silva, C. K. Roy, M. M. Rahman, K. A. Schneider, K. Paixao, and M. de Almeida Maia. Recommending comprehensive solutions for programming tasks by mining crowd knowledge. In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, pages 358–368, 2019. doi:10.1109/ICPC.2019.00054. → pages 1, 8, 10, 14, 17, 18

[80] G. Singer, U. Norbisrath, E. Vainikko, H. Kikkas, and D. Lewandowski. Search-logger analyzing exploratory search tasks. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 751–756, 2011. → page 2

[81] J. Starke, C. Luce, and J. Sillito. Searching and skimming: An exploratory study. In *2009 IEEE International Conference on Software Maintenance*, pages 157–166, 2009. doi:10.1109/ICSM.2009.5306335. → pages 1, 8

[82] C. Treude and M. P. Robillard. Augmenting API documentation with insights from stack overflow. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 392–403, 2016. doi:10.1145/2884781.2884800. → pages 14, 17

[83] C. Treude, O. Barzilay, and M.-A. Storey. How do programmers ask and answer questions on the web? *Proc. of the 33rd Int'l Conf. on SE*, page 804, 2011. ISSN 0270-5257. → page 6

[84] M. Umarji, S. E. Sim, and C. Lopes. Archetypal internet-scale source code searching. In B. Russo, E. Damiani, S. Hissam, B. Lundell, and G. Succi, editors, *Open Source Development, Communities and Quality*, pages 257–263, 2008. ISBN 978-0-387-09684-1. → pages 1, 14

[85] G. Viviani, M. Famelis, X. Xia, C. Janik-Jones, and G. C. Murphy. Locating latent design information in developer discussions: A study on pull requests. 2019. doi:10.1109/TSE.2019.2924006. → page 20

[86] S. Wang, N. Phan, Y. Wang, and Y. Zhao. Extracting api tips from developer question and answer websites. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 321–332. IEEE, 2019. → page 19

[87] C. Watson, N. Cooper, D. N. Palacio, K. Moran, and D. Poshyvanyk. A systematic literature review on the use of deep learning in software engineering research. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 31(2):1–58, 2022. → page 19

[88] X. Xia, L. Bao, D. Lo, P. S. Kochhar, A. E. Hassan, and Z. Xing. What do developers search for on the web? *Empirical Softw. Engg.*, 22(6): 3149–3185, Dec. 2017. ISSN 1382-3256. → pages 3, 19

[89] Y. Xiao, J. Keung, Q. Mi, and K. E. Bennin. Bug localization with semantic and structural features using convolutional neural network and cascade forest. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, pages 101–111. ACM, 2018. → page 19

[90] B. Xu, Z. Xing, X. Xia, and D. Lo. AnswerBot: Automated generation of answer summary to developers' technical questions. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 706–716, 2017. doi:10.1109/ASE.2017.8115681. → pages 8, 10, 11, 17, 19

[91] X. Yang, D. Lo, X. Xia, L. Bao, and J. Sun. Combining word embedding with information retrieval to recommend similar bug reports. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pages 127–137, 2016. doi:10.1109/ISSRE.2016.33. → page 10

[92] X. Ye, H. Shen, X. Ma, R. Bunescu, and C. Liu. From word embeddings to document similarities for improved information retrieval in software engineering. In *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, pages 404–415, 2016. ISBN 9781450339001. doi:10.1145/2884781.2884862. → pages 8, 19

[93] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021. → page 19

[94] D. Zhang and J. J. Tsai. *Machine learning applications in software engineering*, volume 16. World Scientific, 2005. → page 18