# 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

**NLP** Natural Language Processing

**IR** Information Retrieval

**ML** Machine Learning

**DL** Deep Learning

**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 system, software developers typically consult several different kinds of documents, or artifacts, that assist them in their work [27, 42]. For example, when incorporating a new software library needed for a new feature, a developer might consult official application programming interface (API) documents for the library [38, 44] or question-and-answer developer forums [33, 40].

Many of the artifacts that developers consult contain unstructured text; for the purposes of this thesis, we refer to artifacts with unstructured text as natural language artifacts. To utilize ~~the~~ information in natural language artifacts, a developer must read the text to find the information that is relevant to the task being performed [4]. However, the sheer amount of information in these artifacts may prevent a developer from comprehensively assessing what is useful to their task [30]. Just within one kind of artifact, 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 certain task [12, 27]. 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 [30]. *GM: review references*

Finding information that assists a developer to complete a task can be a time-consuming and cognitively frustrating process [5, 38]. Therefore, we posit the need for approaches that assist developers in locating information in the different natural language artifacts sought as part of a software task.

## 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.

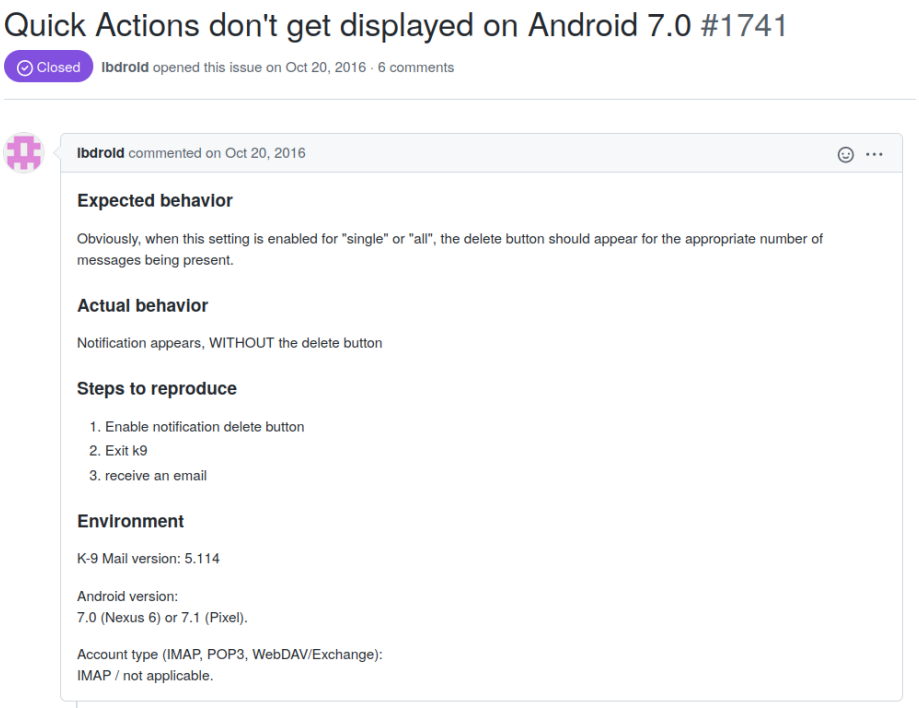


**Figure 1.1:** k-9 mail GitHub issue #1741 indicating that quick actions don't get displayed on Android 7.0

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 this task [22, 23, 39]. Often, this knowledge can be acquired from a developer's peers [41]. However, the fragmented and distributed nature of software development may prevent the developer from accessing their peers [22], instead

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

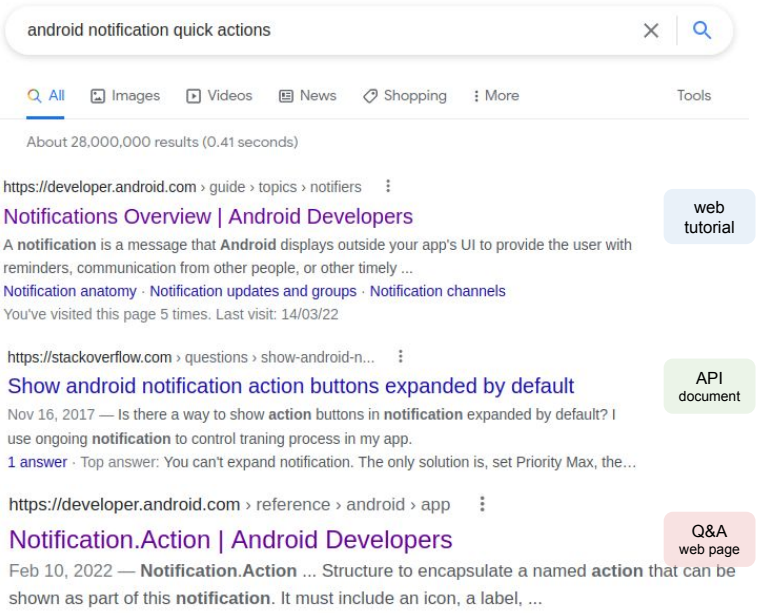


**Figure 1.2:** Search results showing artifacts of potential interest to the Android quick actions issue

they seek online web resources for information that may assist them in completing the task-at-hand [35, 45].

A common way to find software artifacts pertinent to the developer's task is through ~~the usage of~~ a web search engine [6, 23]. Figure 1.2 shows the artifacts resulting from a developer's search about android notifications for the task in Figure 1.1. The artifacts returned can each be considered to have a type. For instance, the artifacts returned represent web tutorials, API documents and question-and-answer artifacts, each of which can be considered a different type. Each type of artifact has different associated challenges for locating information useful to a task within them.

The first artifact in Figure 1.2 is a web tutorial, a document intended primarily to teach users how to use a technology [3] through a series of structure topics that progressively explain concepts about the technology [16, 17]. Figure 1.3 shows a portion of this artifact's content. The Android tutorial contains approximately 200 sentences and, on the page's right-hand side, we find that it has nine sections, each

### 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.

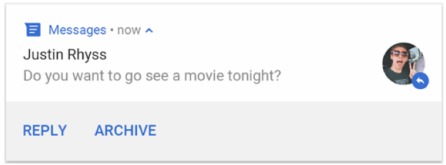


**Figure 9.** A notification with action buttons

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

with sub-sections of their own. Reading all sections of this document could potentially take 10 minutes or more[3] of a developer's time. To save time, a developer would likely try to find the sections of the document most closely related to their task [23]. 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. Not all of these matches will be relevant, requiring the developer to peruse each match and assess relevance. For instance, the '*Notification actions*' (under focus in the figure), 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 API contains software elements (e.g., classes and methods) that other developers are allowed and expected to use for a certain purpose [29], where instructions about an API usage are typically available in the API's reference documentation. such as how the mail client software communicates with the native Android software. This communication happens via features and functionality exposed by the API [37] and instructions about its usage are typically available 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 already described in the web tutorial. Additionally, we note that complex APIs, such as Android, require combining several classes and method calls [38] to properly perform some instruction, what would mean inspecting at least three other classes with equally complex documentation to find all the text explaining the API elements used in the quick actions menu[4].

The third artifact resulting from the developer's search about android notifica-

[3] Using a standard reading metric of 200 words per minute [18].

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

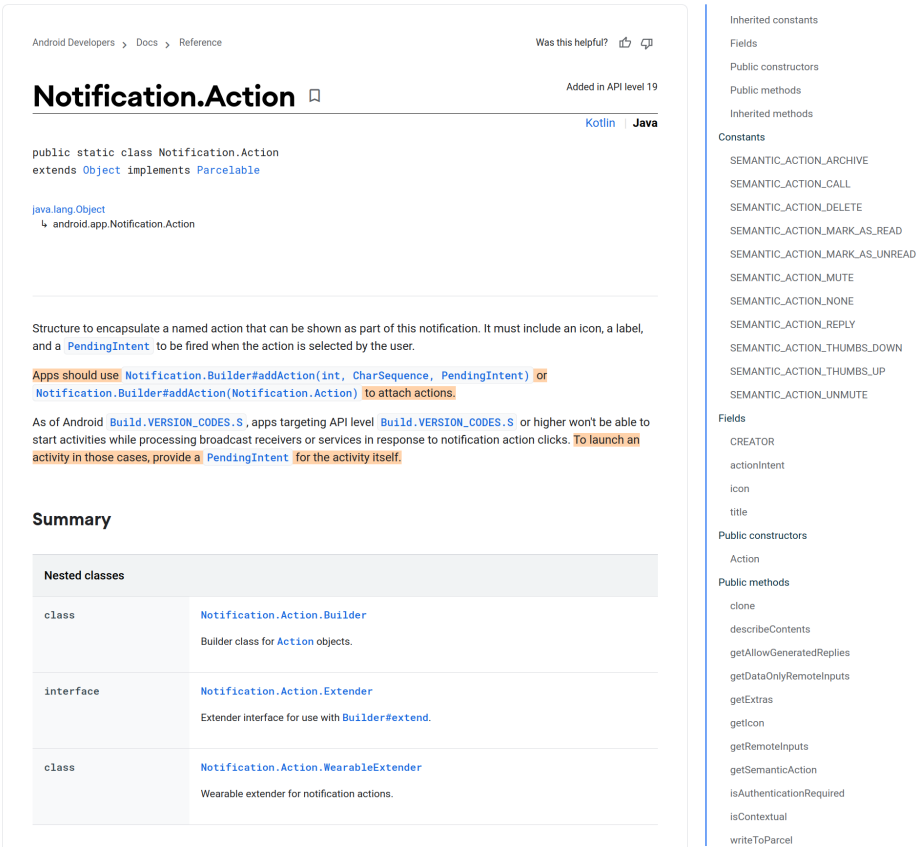


**Figure 1.4:** Snapshot of the Android Notification.Action API reference documentation with highlights relevant to the GitHub issue #1741

tions is a post in a question-and-answer (Q&A) web platform, 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 snippets and ~~it includes a~~ set of tags about the problem's programming language or technology [43]. 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 assists a developer in

[5]https://stackoverflow.com/

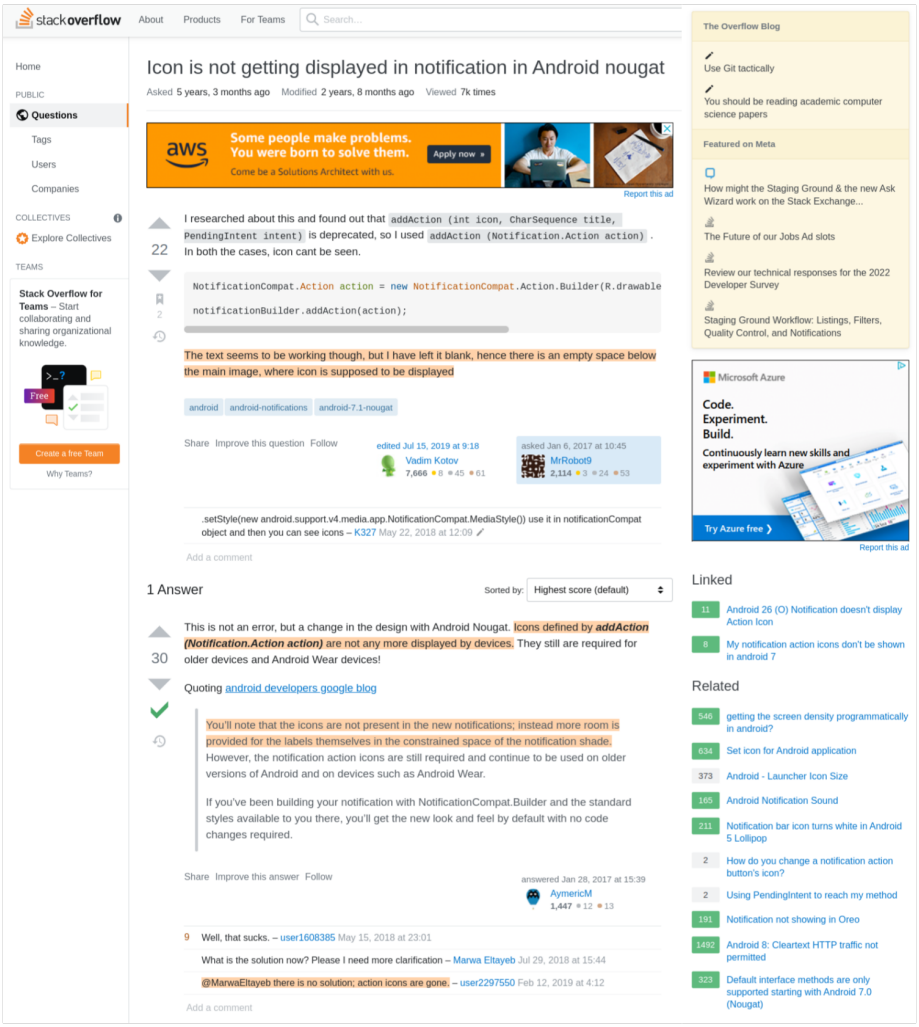


**Figure 1.5:** Snapshot of a Stack Overflow question about Android notifications with highlights relevant to the GitHub issue #1741

navigating through the content in this type of artifact [31], 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 have an accepted answer [33] and millions of questions have more than one answer [31]. Technologies also evolve and answers become obsolote [1]. Hence, despite the fact that structured data might assist a developer, finding task-relevant text in this type of artifact is also not trivial.

At this point, 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 [7, 15, 21]. Given how quickly developers progress to use new kinds of technology to record pertinent information, the aforementioned challenges are not exclusive to the three types of artifact that we have discussed, rather they are common to many kinds of natural language software artifacts [23, 42].

## 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 and, to understand the state of the art, this section discusses some of the artifacts studied and approaches that extract text from these artifacts to assist developers in performing certain software activities. Although we do not claim that the list of artifacts and techniques presented hereafter is complete, they comprise common artifact types sought by developers [23, 44] and many of the techniques used to parse the natural language text in them [2].

In API documentation, Robillard and Chhetri investigated how to automatically identify sentences key to using an API element [37]. Their approach uses regular expressions to match the text in this kind of artifact to a set of patterns derived from the Java SDK 6 documentation. More complex approaches, i.e., Machine Learning (ML) and Deep Learning (DL), were also used by Fucci et al. with the purpose of automatically identifying sentences with directives, concepts, examples, and other types of information included in API documents [14].

In development mailing lists, Panichella et al. proposed an automatic approach

leveraging Information Retrieval (IR) to extract text with useful information that assists in understanding code elements inspected by a developer [32] while Di Sorbo et al. use linguistic patterns—extracted via Natural Language Processing (NLP)—to automatically categorize the content of development emails, what might assist developers in finding text specific to some category, e.g., finding text about the solution for a bug or text discussing potential new features for a system [11].

In bug reports, researchers have mostly applied text summarization as a means of identifying text that a developer would first read when inspecting a bug report, which might assist developers in finding useful information for their task; and both ML and DL techniques have been investigated for this purpose, e.g., [25] or [24].

In community forums and Q&A pages, researchers have explored both lexical and syntactic approaches for the automatic identification of sentences that would help a developer quickly decide if the content in these artifacts is relevant to a developer's task [31]. Other approaches have also considered extracting information relevant to a developer's programming task combining query-based summarization and meta-data available on Stack Overflow [40, 46].

Although effective, these and other techniques target specific types of artifacts and, given how quickly developers progress to using new kinds of technology to record pertinent information, it may be difficult to apply such artifact-centric approaches to cover the range of artifacts sought by a developer as part of a software task and, for the techniques that do consider multiple types of artifacts (e.g., [34]), automatically identifying text likely relevant to a developer's task falls outside of their scope.

## 1.3 Thesis Statement

The scenario we presented indicates that many kinds of artifacts can be consulted to help with a task and that a developer faces challenges in finding the text within these artifacts is relevant to the task at hand. In surveying the state of the art, we demonstrated that existing techniques are tied to particular artifacts. This thesis aims to overcome these limitations. We posit 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 a generalizable technique.*

On potential method for designing a generalizable technique might arise from determining if there are similar rules governing how text relevant to a task is constructed across different types of artifacts [19]. Hence, we 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 present a formative study 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 [8, 20, 36, 37]. 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 and in the design of a more generalizable technique.

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 [47], searching for comprehensive code examples [40], or assessing the quality of information available in bug reports [9]. Nonetheless, we are not aware of their usage in techniques that assists developers in discovering task-relevant text over different types of natural language artifacts. To this end, this thesis describes the investigation of a design space of six possible techniques that incorporate the semantics of words [10, 28] and sentences [13, 26] 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 [46], 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, 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 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 assisted (or not) by TARTI, where experimental results indicate that participants found the text automatically identified by the tool useful in two out of the three tasks of the experiment. Furthermore, tool support also led to more correct solutions in one of the tasks in the experiment. Therefore, 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 build 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 [46], i.e., Stack Overflow.
- It presents an empirical experiment that provides initial evidence on how a semantic-based tool, TARTI, assists a software developer in completing a software task.

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

[6] Automatic Task-Relevant Text Identifier

- $\text{DS}_{synthetic}$ 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;

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

## 1.5 Structure of the Thesis

In Chapter **??**, we describe background information and previous approaches that seek to assist developers in finding useful information in natural language artifacts. The chapter details the empirical analysis of text 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, how we observe consistency on the semantics of the task-relevant text, and what are common challenges faced by developers trying to locate task-relevant text.

Chapter **??** describes the groundwork for producing the corpus ($\text{DS}_{android}$) that we use to evaluate the 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-relevant text. The first two techniques that the chapter presents use word embeddings to identify likely relevant text via semantic similarity and via a 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 $\text{DS}_{android}$ corpus, where we find that the neural network and the frame semantics techniques are the most promising ones for automatically identifying task-relevant text.

Chapter **??** details our empirical experiment investigating whether TARTI—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*

# 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 8

[2] 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 8

[3] 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

[4] 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. → page 1

[5] A. Begel and B. Simon. Novice software developers, all over again. In *Proc. of the Fourth Int'l Workshop on Computing Education Research*, pages 3–14, 2008. ISBN 978-1-60558-216-0. → page 1

[6] 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

[7] 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

[8] 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. → page 10

[9] 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. → page 10

[10] 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. → page 10

[11] 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. → page 9

[12] 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

[13] 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

[14] 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. → page 8

[15] 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

[16] 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. → page 3

[17] 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. → page 3

[18] 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

[19] 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 10

[20] 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

[21] 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

[22] 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

[23] 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

[24] X. Li, H. Jiang, Z. Ren, G. Li, and J. Zhang. Deep learning in software engineering. *arXiv preprint arXiv:1805.04825*, 2018. → page 9

[25] 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. → page 9

[26] 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

[27] 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. → page 1

[28] 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. → page 10

[29] M. Monperrus, M. Eichberg, E. Tekes, and M. Mezini. What should developers be aware of? an empirical study on the directives of api documentation. *Empirical Software Engineering*, 17(6):703–737, 2012. → page 5

[30] 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

[31] 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 8, 9

[32] S. Panichella, J. Aponte, M. Di Penta, A. Marcus, and G. Canfora. Mining source code descriptions from developer communications. In *2012 20th IEEE International Conference on Program Comprehension (ICPC)*, pages 63–72. IEEE, 2012. → page 9

[33] 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, 8

[34] 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. → page 9

[35] 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

[36] 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. → page 10

[37] 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

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

[39] 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. → page 2

[40] 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, 9, 10

[41] 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

[42] 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

[43] 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

[44] 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, 8

[45] 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. → page 3

[46] 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 9, 10, 11

[47] 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