SCIENCE CHINA
Information Sciences

• LETTER •

September 2020, Vol. 63 190104:1–190104:3
https://doi.org/10.1007/s11432-019-2831-6

Special Focus on Big Data-Driven Intelligent Software Development for Onsite Programming

A quantitative benefit evaluation of code search platform for enterprises

Zhan SHI[1], Jie TANG[1], Hao YU[1], Yongxu XING[1], Zhiwei LIU[1*], Wei BAI[2] & Tao LI[2]

[1]*Baidu (China) Co., Ltd, Shanghai* 201210*, China;*
[2]*Baidu Online Network Technology (Beijing) Co., Ltd, Beijing* 100193*, China*

Received 19 September 2019/Revised 27 December 2019/Accepted 20 January 2020/Published online 28 July 2020

Citation Shi Z, Tang J, Yu H, et al. A quantitative benefit evaluation of code search platform for enterprises. Sci China Inf Sci, 2020, 63(9): 190104, https://doi.org/10.1007/s11432-019-2831-6

Dear editor,
With the size and complexity of modern software growing, developers need more efficient tools to accomplish all kinds of routine tasks, such as coding, compiling, and analyzing program structure [1]. Among these tasks, code search is the key software development activity [2]. To deal with the advent of large code repositories, various code search platforms are designed to maximize developers' productivity. With the help of these platforms, developers can easily figure out how to use an API or service by searching and reusing existing codes to improve efficiency tremendously.

The code search platform is an integrated development platform that provides various features to improve developer's efficiency, such as code search (search for code fragments) and code view (read codes of retrieved results). On how to evaluate the code search platform, there are many types of researches on evaluation metrics such as precision, recall, response time and other platform-related properties [3, 4]. For enterprises, the efficiency improvement conclusion is obvious but empirical. We do not know exactly how much efficiency is improved or how much time is saved using these platforms compared to traditional ways. A reasonable benefit evaluation in quantity can not only inform us of the value of code search platforms but also contribute to more reasonable resource allocation for supporting future features. Though the quantitative evaluation of efficiency improvement is important, there are few researches focus on this point.

In this study, based on user behavior data and code-related information collected from iSearch, we evaluate the benefit of code search platforms according to the time saved by using iSearch compared to the internal code host platform only. iSearch is a code search platform built in Baidu [5] that supports features like code fragments search, code navigation and code dependency analysis. iSearch contributes to productivity improvement in many aspects, here we choose two easily evaluated aspects, code search and code view. Traditionally, code search can be performed throw local IDEs if we know in advance which repositories may contain the desired code and then clone it. As now we can perform code search over the whole codebase of Baidu, the cloning process no longer needs and the time cost of cloning will be saved. Moreover, as code host platforms (like Github) usually do not support features like jumping to a definition or find references, if developers are reading code online they will spend lots of time understanding the dependency hierarchy, while iSearch can contribute to more efficient code reading. The saved time in the process of code search and code view can then be converted to the intuitive labor cost and we can figure out how much benefit the code search platform can bring.

From code search aspect. Developers search code for various purposes, which is classified into five

*Corresponding author (email: liuzhiweihome@gmail.com)

Table 1 Time saved of different categories of code search

Category	Percent (%)	With iSearch	Without iSearch	Saved time (s)	Average time (s)
API consumer needs help	22	R	E+B	30	6.6
Discover correct library for task	5	–	–	–	–
Example to build off of	3.5	S	CS	61.8	2.16
How to do something	3	–	–	–	–
Check implementation details	20	R	CS	61.8	12.36
Browsing	4	S	B+	15	0.6
Check commons style	1	S	CS	61.8	6.18
Name completion	1	S+	(CS)+	125	12.5
Reachability	8.5	G	(CS)+	125	10.62
Showing to someone else	4	B	B	0	0
Location in source control	3.5	S	CS	65	2.28
Why is something failing	10	–	–	–	–
Understand dependency	4.5	G	(CS)+	240	10.8
Side effects of proposed changes	1.5	G	(CS)+	240	3.6
Trace code history	5	B	C	61.8	3.09
Responsibility	3.5	B	C	61.8	2.16
Total saved time (s)	–	–	–	–	72.59

categories in [2]. The result of our survey on what code search platform needs to provide is similar to the study in [2], thus we adopt its categories for further analysis. The efficiency improvement for each purpose by iSearch varies. For each specific task, we conduct an A/B test with iSearch or using traditional ways such as a web search engine and get the saved time according to operations needed to accomplish the task. The detailed steps and time cost are listed in Table 1.

In Table 1, each letter abbreviation represents a single operation to accomplish one specific task. G (request for the view of call and dependency graph), R (jump to a definition and references), and S (perform code search) are common features that iSearch supports. B denotes browsing files that returned after queries or pages provided by iSearch directly such as file comparison, change history, etc. C represents the process of cloning target repositories and E is assumed to perform searches with a general web search engine. The sign + is used the same as regular expression, indicating one or more repetition. Then, each specific task can be represented by several separate successive operations.

The saved time in the fifth column means that to handle the task, less time is spent with features provided by iSearch compared to traditional ways. The saved time is based on both practical data analysis (such as the clone time cost) and empirical experience A/B test performed by five developers, i.e., the time cost to search for some queries with or without code search platform. The five developers are the first five authors of this study and their experience ranges from the most expert engineer (10 years) to the least experienced (several months–recent college graduate). R, G, and S operations correspond less than one second, B and E last for seconds. In this study, we take the average time cost of cloning key repositories (including main repositories of each product line and frequently updated repositories) as the average clone time of all repositories. We select 486 key repositories, count the number of users who have conducted code search more than once in each repository in workday and clone time. The average clone time, i.e., the time saved per search $T^{(i)}_{\text{clone}}$ is 61.8 s. The specific clone time cost of each repository is not given here because of short space.

Adding the saved time of each task with its percentage, the average saved time per code search is 72.59 s. It means that iSearch can save 72.59 s in average for each code search, compared to code host platform only. Then, the efficiency improvement, i.e., the saved time by code search is

$$E_{\text{improved}} = C_{\text{code\_search}} \times T_{\text{per\_code\_search}}, \qquad (1)$$

where $C_{\text{code\_search}}$ is the number of valid code searches, i.e., searches that satisfy users' requests, $T_{\text{per\_code\_search}}$ is the average saved time by iSearch and equals 72.59 s according to Table 1. We can further convert the efficiency improvement into labor cost by dividing E_{improved} by average working hours (8 h) and get how many humans can save with iSearch or other similar code search platforms.

iSearch has been applied on the base of Git code host platform in Baidu, which includes more than ten billion LOCs (lines of codes) with millions of LOCs added or deleted every day. Till now, more than half of developers (5000+) in Baidu have used common features supported by iSearch, such as code search, finding the definition and references. The click through rate (CTR) of code search now is about 40%. Although some of the clicks perhaps do not correspond to users' searches, there are also many queries that do not need to click, such as searches for macro definition, attributions, and query autocompletion. Thus, it is conservative to assume that the number of effective code searches is approximately half of click times. In the past year, all 1.75 million code search queries are performed. Then $C_{\text{code\_search}}$ in (1) can be taken as the number of half of click times, i.e., $1750000\times 20\% = 350000$.

Thus, the saved time or the efficiency improvement E_{improved} will be 7057 h or 882 humans per year.

From code view aspect. Compared to code host platforms such as Github, iSearch provides more features to avoid the cloning process, help developers find information more easily and read code more efficiently. Suppose there are developers who are willing to read code in online platforms, without code search platforms such as iSearch, they have to do that in code host platforms, which is inefficient if the aforementioned features are not supported. The wasted time or in other studies, saved time by iSearch can be evaluated by

$$T_{\text{code\_view}} = C_{\text{open\_file}} \times \Delta T_{\text{per\_open}}, \qquad (2)$$

where $C_{\text{open\_file}}$ is the number of times users browse files in the code host platform, $\Delta T_{\text{per\_open}}$ is the average saved time that users apply iSearch rather than common code host platform in the code view process. It is a weighted mean of a bunch of operations mentioned above. Similarly, the saved time by code view can also be converted to labor cost. Here, according to [6], about 58% of developers' time is spent on program comprehension, then the labor cost saved by iSearch will be $T_{\text{code\_view}}$ divided by 4.64 working hours.

To evaluate the efficiency improvement by typical features supported by iSearch in the code view process, we summarize the time cost of common development activities, such as file browsing with file tree, finding definition, references and example codes as well as comparing history and blame, performed by our developers with and without iSearch. The time saved is 5, 40, 60 and 31.8 s, respectively. There may still be other activities in daily development, here we only list the most common ones. If one browses a file, he will at least perform one of the operations. The precise proportion of each operation is hard to estimate. Each time a developer browses a file, he may conduct 'find a definition' several times while others may not. For simplification, we just add up them, i.e., the time saved of each time browsing files with the help of iSearch is about 136.8 s.

Till now, developers in Baidu browse files in the internal Git code host platform more than 19000 times every day. According to (2), the total saved time by iSearch by code view feature is 722 working hours or 156 labor cost per day.

Conclusion and future work. The code search platforms are designed to maximize developers' productivity. In this study, we present a method to quantitatively evaluate the benefits brought by code search platforms for enterprises. The efficiency improvement is estimated using the saved time that is calculated based on data acquired by iSearch. The analysis result shows that, in the current scale of users in Baidu, iSearch can save 7057 working hours from code search feature per year and 722 h from code view per workday, converting to labor cost is 882 per year and 156 per workday. The statistics can also be taken as reference to evaluate other code search platforms.

If code search platforms optimize existing tools and support more effective features, which will be studied in future, more users will adopt them and the cost saved for enterprises will also grow. Besides, we will evaluate the relevance and quality of iSearch. Now we are surveying on how the results are relevant to the queries and comparing our results with other code search platforms.

References

1 Potvin R, Levenberg J. Why Google stores billions of lines of code in a single repository. Commun ACM, 2016, 59: 78–87
2 Sadowski C, Stolee K T, Elbaum S. How developers search for code: a case study. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, 2015. 191–201
3 Oppenheim C, Morris A, McKnight C, et al. The evaluation of WWW search engines. J Documentation, 2000, 56: 190–211
4 Ali R, Beg M M S. An overview of Web search evaluation methods. Comput Electr Eng, 2011, 37: 835–848
5 Liu Z W, Xing Y X, Yu H, et al. Retrieval and management technology for industrial-scale massive code (in Chinese). J Softw, 2019, 30: 1498–1509
6 Xia X, Bao L, Lo D, et al. Measuring program comprehension: a large-scale field study with professionals. IEEE Trans Softw Eng, 2018, 44: 951–976