A Smart Context-aware Program Assistant based on Dynamic Programming Event Modeling

Xuejiao Zhao[1,2], Hongwei Li[3], Yutian Tang[4], Dongjing Gao[5], Lingfeng Bao[6], Ching-Hung Lee[7]
[1]Joint NTU-UBC Research Centre of Excellence in Active Living for the Elderly (LILY), NTU, Singapore
[2]School of Computer Science and Engineering, NTU, Singapore
[3]School of Computer Information Engineering, Jiangxi Normal University, Nanchang, China
[4]Department of Computing, The Hong Kong Polytechnic University, Hong Kong
[5]School of Computer Science, Fudan University, Shanghai, China
[6]College of Computer Science, Zhejiang University, Hangzhou, China
[7]Division of Smart Product Design, School of Mechanical and Aerospace Engineering, NTU, Singapore
Email:{xjzhao, leechinghung}@ntu.edu.sg lihongwei@jxnu.edu.cn csytang@comp.polyu.edu.hk
gaodj14@fudan.edu.cn lingfengbao@zju.edu.cn

***Abstract*—In software development, there is a great demand for online information and resources. The traditional way for the developers to access online resources is through formulating keywords and searching in the web browser. The search results are limited by the keywords and the web browser also ignores the developers' working and search context. Tools that integrate information retrieval into the IDE are available, but they fail to perceive the developers' dynamic working context and use in the process of online search. In this paper, we present a context-aware program assistant called *amAssist*. *amAssist* monitors the developers' development events and models their working context dynamically, then integrates them with the entire online search process (e.g. keywords formulation, customized searching, search results annotation, etc.). We integrate *amAssist* into the Eclipse IDE. Our preliminary user study showed that by using our program assistant, developers can formulate keywords more accurately and acquire online information and resources more rapidly. Demo video: https://youtu.be/X4Tkjhc6wfU**

I. Introduction

Interleaving coding, web search, and learning have become a common practice in software development. There is a great demand for online information and resources in practice. The traditional way for the developers to access online information and resources is through formulating keywords and searching in the web browser. The quality of search results depends on the keywords used because the web browser can not perceive the developers' working and search context. If the developers can not formulate keywords to reflect their information needs accurately, they will spend a lot of time on information retrieval [1].

Tools that integrate information retrieval into the IDE are available [2], [3], [4]. However, they only extract a snapshot of the developers' working context and integrate it with online search shallowly. They model the developers' working context statistically (e.g., a selected program entity, an exception the developers currently encountered, etc.). The statistical working context is only use to augment the search queries rather than the entire online search process.

In this paper, we present a smart context-aware program assistant called *amAssist*. *amAssist* program assistant models the developers' development behaviors dynamically as their working context, and integrates them with the entire online search process (e.g. keywords formulation, customized searching, search results annotation, etc.). We unobtrusively monitor nine kinds of interactive development events (e.g. select, save, etc.) between the developers and the IDE in time series. These determine the developers' working focus over time. We then use an algorithm called *DOI* [5], [6] to compute the interest value of the involved API entities of these programming events (e.g. "select" the API entity "EditorPart.doSaveAs()"). With the *amAssist* program assistant, developers can use their working context to formulate their search query and customize their online search. The *amAssit* program assistant also annotates the list of search results and the content of web pages, which can assist the developers to locate relevant search results and related information on the web pages more rapidly. Our study shows that *amAssit* can help the developers formulate more specific queries with working context information. As such, the developers using *amAssist* can find and integrate relevant online programming resources more quickly with less search queries.

Our preliminary study suggests that the *amAssist* program assistant can increase developers' awareness of their working context over time. As such, it can help developers formulate more specific queries with working context information. It can also help developers to choose their interested and valuable web pages and locate useful information faster by context-aware search results annotation.

II. The *amAssist* program assistant

The architecture of the *amAssist* program assistant consists of two main parts as shown in Fig. 1: the backend component monitors and analyzes the developers' dynamic working context; the frontend search user interface (UI) presents the contextual information to the developers and allows them to use the context to search, refine, and browse online resources while they read code or program in the IDE.

978-1-5386-9443-5/18/$31.00 ©2018 IEEE
DOI 10.1109/ISSREW.2018.00-36

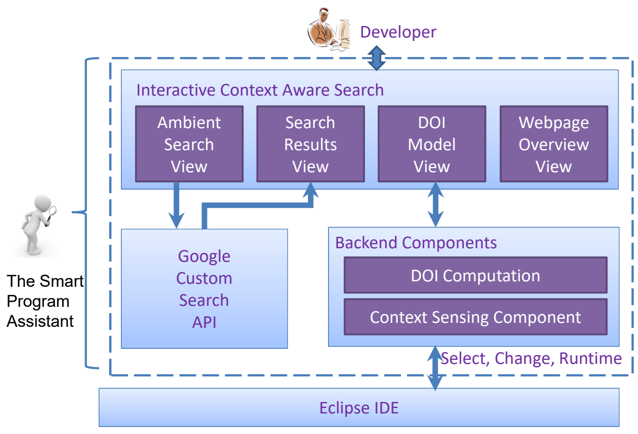


Fig. 1. The Architecture of the *amAssist* Program Assistant

A. *Context Sensing and DOI Computation*

The backend component consists of a context sensing component and a Degree-of-Interests (DOI) computation component.

1) Context Sensing Component: The *amAssist* program assistant has been integrated into the Eclipse IDE. It listens to the Eclipse workbench's events to monitor the developers' programming activities in the IDE. It monitors the programming events of the Eclipse workbench and uses these events to model the developers' working context. Note that the event listening occurs unobtrusively in the background without interrupting the developers' working flow.

We model each programming event as a tuple (t, a, P, L). t is the event index which is incremented by one when a new event is listened. a is the event category value and we summarize and abstract these events into five categories: $select$, $reveal$, $save$, $debug$, and $execute-exception$. These actions represent the fundamental activities that developers carry out in the IDE. P is a set of program entities involved in the event that the developer acts on, including compilation units, classes (including interfaces), methods, fields, and statements in the source code. P is resolved by the Java DOM/AST APIs and Eclipse JDT IJavaElement APIs.

L represents the libraries and framework API entities (e.g., classes, methods, and fields) extended by the source code entities. There are many online resources related to the usage or extension of libraries or framework APIs. However, the source code entities of the specific application are meaningless for online search because they are not standard name and only can be identified by the developers who named them. Therefor, the *amAssist* program assistant uses the Eclipse JDT DOM/AST APIs to search L (the libraries or framework APIs which extend from the source code entities P of an event). The *amAssist* program assistant also annotates the API entity that cause runtime exceptions and compilation errors.

2) DOI Computation Component: When developers read code or program, the *amAssist* program assistant uses the DOI algorithm to model their working context. The DOI algorithm assigns an interest value to every detected API entity and can detect unusual incidents (e.g., runtime exceptions) or important focus shifts in the working context. The initial interest value of an API entity of an event is assigned according to the essential interest level of current action and the API entity of the event. An action has one of four interest levels: $select < reveal < save$ or $debug < execute-exception$. Let l_a be the interest level of an action a, with $(1 \leq l_a \leq 4)$. An API entity has one of three interest levels: $normal < has-compile-error < cause-exception$. Let l_{api} be the interest level of an API entity p_{api}, with $(1 \leq l_{api} \leq 3)$.

Assume there is an event $e < t, a, P, L >$ and an API entity $p_{api} \in e.L$. The initial interest value of p_{api} in the event e (i.e., $v_{init}(p_{api}, e)$) is computed as $\gamma^{l_a} \times l_{api}$, where γ is an integer constant value ($\gamma > 1$). Based on the above definition, the highest initial interest value belongs to the API entities that throw an exception. The normal API entities with the action $select$ is assigned the lowest initial interest value.

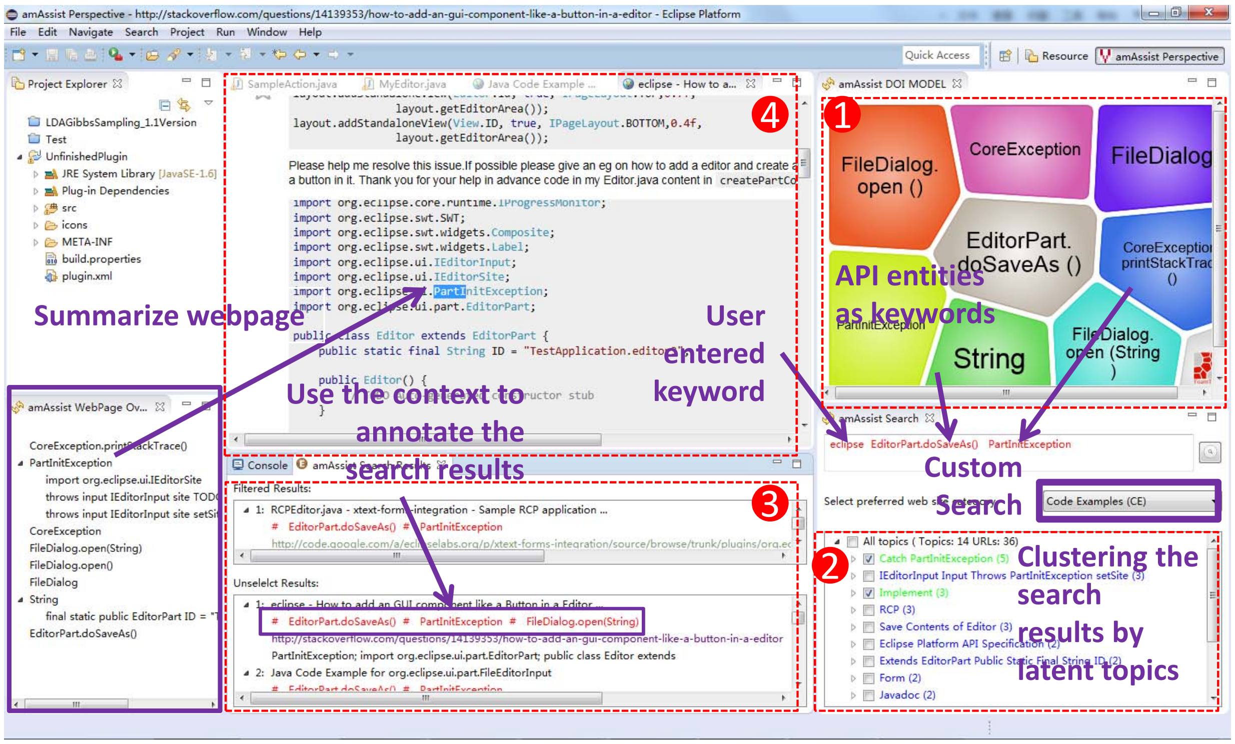
When a new event occurs, the *amAssist* program assistant updates the interest value of all the distinct API entities in the working context as follows. The *amAssist* program assistant assumes that the older an event $e < t, a, P, L >$ is, the less interest the developer has in the API entities involved in the event (i.e., $p_{api} \in e.L$). Thus, it decays the interest value of an API entity as new events arrive in the working context. The current interest value of p_{api} in the event $e < t, a, P, L >$ (i.e., $v(p_{api}, e)$) is computed as $v_{init}(p_{api}, e)/\gamma^{t_m - t}$ where t_m is the current event index.

The *amAssist* program assistant assumes that if developers act on an API entity p_{api} more often, they are more interested in p_{api}. Therefore, *amAssist* program assistant compute the interest value of p_{api} in the working context at the current event index t_m (i.e., $v(p_{api}, t_m)$) as $\sum_{1 \leq i \leq t_m} f(p_{api}, e(i))$. The function $e(i)$ returns the event at index i. The function $f(p_{api}, e(i))$ returns $v(p_{api}, e(i))$ if $p_{api} \in e(i).L$; otherwise it returns 0.

For example, a developer *"Reveal"* an API entity p_{api} – *"IProgressMonitor"*. The initial action interest value l_a is 2 and the initial API entity interest value l_{api} is 1. The initial interest value of *"IProgressMonitor"* is thus 4. After that, it decays the interest value of *"IProgressMonitor"* when new events arrive in the working context. So the next value of *"IProgressMonitor"* are 2, 1, 0.5, 0.25, 0.125 and so on.

B. *Interactive Context-Aware Search*

Fig. 2 shows the frontend search UI of the *amAssist* program assistant. The four red dotted boxes with numbers are the four views of *amAssist*: *DOI Model* view (No.1), *Ambient Search* view (No.2), *Search Results* view (No.3), and *Webpage Overview* view (No.4). The workflow of the *amAssist* program assistant also follows that sequence. The purple words, arrows, and solid boxes indicate the main functions of the *amAssist* program assistant. This UI not only integrates the entire online search process, but also uses the developers' working context at each step of the search.



```
                layout.getEditorArea());
layout.addStandaloneView(View.ID, true, IPageLayout.BOTTOM,0.4f,
                layout.getEditorArea());
```

Please help me resolve this issue.If possible please give an eg on how to add a editor and create a button in it. Thank you for your help in advance code in my Editor.java content in createPartCo

```
import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Label;
import org.eclipse.ui.IEditorInput;
import org.eclipse.ui.IEditorSite;
import org.eclipse.ui.PartInitException;
import org.eclipse.ui.part.EditorPart;

public class Editor extends EditorPart {
    public static final String ID = "TestApplication.editor";

    public Editor() {
        // TODO Auto-generated constructor stub
    }
```

CoreException.printStackTrace()
PartInitException
    import org.eclipse.ui.IEditorSite
    throws input IEditorInput site TOD
    throws input IEditorInput site setSi
CoreException
FileDialog.open(String)
FileDialog.open()
FileDialog
String
    final static public EditorPart ID = "
EditorPart.doSaveAs()

Filtered Results:
1: RCPEditor.java - xtext-forms-integration - Sample RCP application ...
EditorPart.doSaveAs() # PartInitException
http://code.google.com/a/eclipselabs.org/p/xtext-forms-integration/source/browse/trunk/plugins/org.ec

Unselelct Results:
1: eclipse - How to add an GUI component like a Button in a Editor ...
EditorPart.doSaveAs() # PartInitException # FileDialog.open(String)
http://stackoverflow.com/questions/14139353/how-to-add-an-gui-component-like-a-button-in-a-editor
PartInitException; import org.eclipse.ui.part.EditorPart; public class Editor extends
2: Java Code Example for org.eclipse.ui.part.FileEditorInput
EditorPart.doSaveAs() # PartInitException

eclipse EditorPart.doSaveAs() PartInitException

Select preferred web site category: Code Examples (CE)

All topics (Topics: 14 URLs: 36)
[✓] Catch PartInitException (5)
[ ] IEditorInput Input Throws PartInitException setSite (3)
[✓] Implement (3)
[ ] RCP (3)
[ ] Save Contents of Editor (3)
[ ] Eclipse Platform API Specification (2)
[ ] Extends EditorPart Public Static Final String ID (2)
[ ] Form (2)
[ ] Javadoc (2)

Fig. 2. The UI of the *amAssist* Program Assistant

1) DOI Model View: The *amAssit* program assistant ranks the API entities in the working context by their interest values. The flowchart of the DOI Model view is shown in Fig. 3. It uses an interactive foam tree *(Carrot Search API)* [1] to present a brief overview of the API entities that developers may be most interested in.

The *amAssist* program assistant can be configured to show the top N (by default $N = 10$) highest-interest-value API entities to the developer. The higher the interest value of an API entityis, the larger its bubble. The *amAssit* program assistant uses two strategies to update the foam tree. First, if the top 10 API entities N change, it will update the foam tree accordingly. Second, if the top 10 API entities N do not change but their interest values change, it will update the foam size at regular time interval T (by default $T = 10$ seconds).

2) Ambient Search View: The *amAssist* program assistant uses the *Google Custom Search API* [2] as the underlying search engine. The developers can choose one or more API entities in the foam tree to form the keywords in their search queries. They can customize the chosen API entities or enrich the chosen API entities with any words they want to use.

The *amAssist* program assistant allows developers to customize the *Google Custom Search Engine* to search for their preferred websites by attaching one or more category labels to these websites. By default, the *amAssist* program assistant

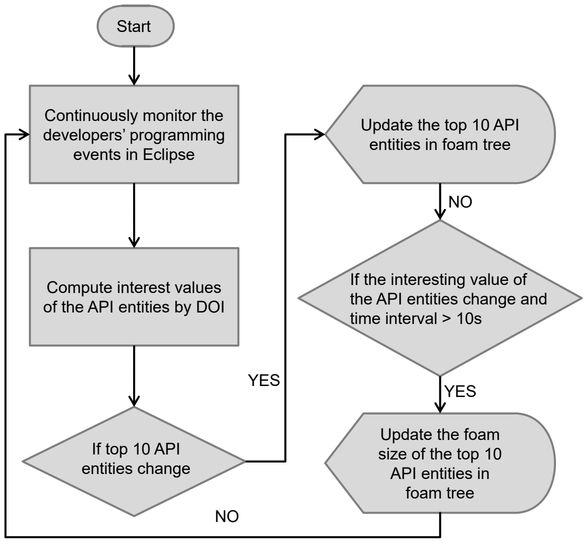

Start
Continuously monitor the developers' programming events in Eclipse
Compute interest values of the API entities by DOI
If top 10 API entities change
YES
NO
Update the top 10 API entities in foam tree
NO
If the interesting value of the API entities change and time interval > 10s
YES
Update the foam size of the top 10 API entities in foam tree

Fig. 3. The flowchart of DOI Model View

searches in four categories of popular programming-oriented websites, such as technical blogs (e.g., www.iteye.com), code examples (e.g., code.google.com, github), discussion forums (e.g., zhidao.baidu), and Q&A sites (e.g., stack overflow). Developers can "Select Preferred Website Category" to inform the *Google Custom Search Engine* about their preference for

[1]http://project.carrot2.org
[2]https://developers.google.com/custom-search/

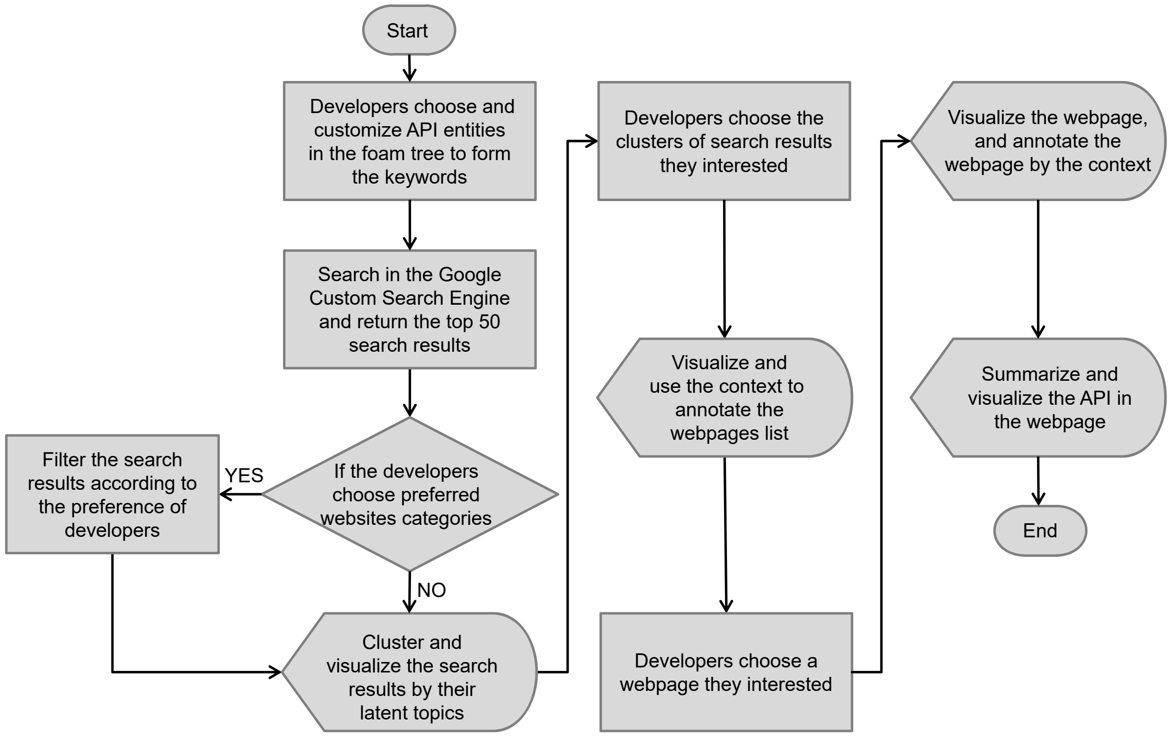

Start
Developers choose and customize API entities in the foam tree to form the keywords
Search in the Google Custom Search Engine and return the top 50 search results
If the developers choose preferred websites categories
YES
Filter the search results according to the preference of developers
NO
Cluster and visualize the search results by their latent topics
Developers choose the clusters of search results they interested
Visualize and use the context to annotate the webpages list
Developers choose a webpage they interested
Visualize the webpage, and annotate the webpage by the context
Summarize and visualize the API in the webpage
End

Fig. 4. The flowchart of Ambient Search, Search Results, and Webpage Overview View

a category of websites (e.g., code examples websites). The *amAssist* program assistant uses the BOOST mode of *Google Custom Search Engine* to promote the search results from the websites of the developers' preferred category, without excluding other websites.

3) Search Results View: The *amAssist* program assistant lists the top X (by default $X = 50$) search results returned by the search engine in the *Search Results* view. The *amAssist* program assistant also annotates the search results and corresponding web pages by the API entities of the working context, which can assist the developers in accessing search results they need more rapidly. For example, in Fig. 2, the red annotation of the first search result in "Unselect Results" view shows that this web page contains both the API entity used as keyword (i.e., $EditorPart.doSaveAs()$) and other API entities in the foam tree. These annotation augment the document surrogate [7].

Furthermore, the *amAssist* program assistant uses the Lingo algorithm [8] provided by *Carrot Search* to cluster the search results based on their latent topics. To this end, the *amAssist* program assistant retrieves the snippet of the top X search results.

The Lingo algorithm uses conceptually varied cluster labels first, then assigns web pages to the labels to generate the clusters [8]. Compared with other topic mining algorithms, Lingo can generate longer and more descriptive labels which can assist developers in understanding and selecting groups of necessary search results in less time. The *amAssist* program assistant displays the search results clusters in the lower part of the *Ambient Search* view. It shows the topics of the cluster and the number of search results in the cluster. Developers can filter the search results by selecting one or more cluster topics.

4) Webpage Overview View: To help developers grasp the key information from a particular web page, the *Webpage Overview* view of the *amAssist* program assistant summarizes the API entities in the foam tree and mentioned in the opened web page at the same time in a tree view. The tree can be expanded to show the places where the API entities appear on the web pages. The *amAssist* program assistant extracts an excerpt surrounding these places (e.g., here we use 5 words before and after the mentioned API entities.) This view can assist developers in navigating their interested content in the opened web page faster. The flowchart of the Ambient Search, Search Results, and Webpage Overview View is shown in Fig. 4.

III. Evaluation

We recruited 10 graduate students from the School of Computer Science, Fudan University to conduct a preliminary user study to evaluate our *amAssit* program assistant. The 10 participants were asked to search and use online program resources to debug and extend an existing Eclipse editor plugin. The participants had to fix one bug and add 2 features named *file save* and *content statistics*. Based on their programming experience, we matched the participants in pairs and allocated them to the experimental group or control group randomly. The experimental group used the *amAssist* program assistant while the control group used the Eclipse IDE and a web browser to perform the above task. Participants were not familiar with the Eclipse plugin development.

This preliminary study shows that compared to the control group, by using the *amAssist* program assistant, the experimental group was able to formulate search queries with more specific API keywords. For example, when the user opened a new editor, the existing implementation threw an *IllegalArgumentException*. Fixing this bug requires the knowledge of *EditorPart.openEditor()* and *IEditorInput* APIs. In order to get the resource to fix this bug, a developer of the control group used the keywords – *how to fix IllegalArgumentException*. In contrast, a developer of the experimental group used the keywords – *EditorPart.openEditor()* in our foam tree directly. This keywords contained more specific API entity derived by our program assistant, which facilitated the developer of the experimental group to find the related webpages faster.

The ratio of the API entities chosen from the foam tree in most of the experimental group developers' queries varied from 0.37 to 0.5. Only one developer used 0.18 of the API entities in the foam tree, which was less than the other four developers. This developer is an experienced Eclipse plugin developer. This demonstrates that the high utilization of the API entities in foam tree and the working context in the foam tree can summarize the developers' information needs effectively, especially for beginners.

The average number of keywords for the experimental group and control group were 19.40 and 15.60 respectively, which shows that compared to the control group, the experimental group can formulate more keywords.

The average search time that the experimental group needed to complete all the tasks was 6.8 times, which was much lower than the 10.4 the control group needed. This result shows that compared to the control group, the experimental group can search fewer times to finish the same tasks.

In addition, according to our observations, the experimental group could locate relevant online programming resources and start integrating relevant online resources faster. Although by no means conclusive, this result suggests that the *amAssit* program assistant based on the developers' dynamic programming event modeling can help the developers to formulate better keywords and locate necessary online programming resources more rapidly while they work in the IDE.

IV. Related Work

Methods for monitoring the users' interaction with software tools and documents and infering their interests or working context have been developed, including Read and Write Wear [9], FishEye view [10], Mylar model [6], and concern code [11]. The *amAssist* program assistant uses DOI model was inspired by these corresponding works. To be comparable to the existing works, the *amAssist* program assistant tracks more comprehensive programming events as the developers interact with the IDE and program entities.

Tools such as Strathcona [12], CodeBroker [13], MFIE [14], and Suade [15] attempt to use context for code searches. There are some researches used crowdsourcing knowledge to assist with software development, e.g. Seahawk [2], BluePrint [3], and Dora [4], etc. GraPacc [16] used a snapshot of the currently edited code to search and rank relevant API usage patterns based on a database of API usage patterns mined from open source projects. HelpMeOut [17], [18] suggested program editing to fix compilation errors based on program editing by other programmers that fix bugs. These tools only use the current code context and do not search for online resources. These existing tools consider context as a set of program entities at the current snapshot of the code. In contrast, the *amAssist* program assistant considers context as a stream of programming events. Furthermore, existing tools use the program context only to augment the search query. In comparison, the *amAssist* program assistant incorporates the working context in the thorough search procedure tightly and deeply.

V. Conclusion

This paper presented our *amAssist* program assistant. This tool unobtrusively monitors and analyzes the developers' dynamic working context. It supports an interactive context-aware search of online programming resources while developers are coding or reading code in the IDE. Our preliminary study suggests that the *amAssist* program assistant can increase developers' awareness of their working context over time. As such, it can help developers search for more context-aware queries. It can also help developers to choose web pages which meet their requirement and locate useful information faster by context-aware search results annotation.

In the future, we will enhance the *amAssist* program assistant with advanced *Google Custom Search Engine* features, for example tweaking the ranking of the search results using the interest values of the API entity keywords or refining the search results using the API entities that are not used as keywords. We will also integrate the knowledge graph of the software engineering domain [19], [20], [21] with some advanced retrieval methods [22] based on knowledge graph to enhance the search engine. More useful programming resources (e.g. hyperlinks in Stack Overflow [23], and API [24], etc.) will be introduced into the tool to further improve programmer development efficiency. We will conduct a more comprehensive user study to evaluate the *amAssist* program assistant and the advantages and disadvantages of the deep integration of developers' dynamic working context while searching online.

VI. Acknowledgments

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its IDM Futures Funding Initiative.

References

[1] G. C. Murphy, D. Notkin, and K. Sullivan, "Software reflexion models: Bridging the gap between source and high-level models," *ACM SIGSOFT Software Engineering Notes*, vol. 20, no. 4, pp. 18–28, 1995.

[2] A. Bacchelli, L. Ponzanelli, and M. Lanza, "Harnessing stack overflow for the ide," in *Recommendation Systems for Software Engineering (RSSE), 2012 Third International Workshop on*, pp. 26–30, IEEE, 2012.

[3] J. Brandt, M. Dontcheva, M. Weskamp, and S. R. Klemmer, "Example-centric programming: integrating web search into the development environment," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 513–522, ACM, 2010.

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

[5] M. Kersten and G. C. Murphy, "Using task context to improve programmer productivity," in *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, pp. 1–11, ACM, 2006.

[6] M. Kersten and G. C. Murphy, "Mylar: a degree-of-interest model for ides," in *Proceedings of the 4th international conference on Aspect-oriented software development*, pp. 159–168, ACM, 2005.

[7] M. Hearst, *Search user interfaces*. Cambridge University Press, 2009.

[8] S. Osiński, J. Stefanowski, and D. Weiss, "Lingo: Search results clustering algorithm based on singular value decomposition," in *Intelligent information processing and web mining*, pp. 359–368, Springer, 2004.

[9] W. C. Hill, J. D. Hollan, D. Wroblewski, and T. McCandless, "Edit wear and read wear," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 3–9, ACM, 1992.

[10] A. Cockburn and M. Smith, "Hidden messages: evaluating the efficiency of code elision in program navigation," *Interacting with Computers*, vol. 15, no. 3, pp. 387–407, 2003.

[11] M. P. Robillard and G. C. Murphy, "Automatically inferring concern code from program investigation activities," in *Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on*, pp. 225–234, IEEE, 2003.

[12] R. Holmes, R. J. Walker, and G. C. Murphy, "Strathcona example recommendation tool," *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 5, pp. 237–240, 2005.

[13] Y. Ye and G. Fischer, "Reuse-conducive development environments," *Automated Software Engineering*, vol. 12, no. 2, pp. 199–235, 2005.

[14] J. Wang, X. Peng, Z. Xing, and W. Zhao, "Improving feature location practice with multi-faceted interactive exploration," in *Proceedings of the 2013 International Conference on Software Engineering*, pp. 762–771, IEEE Press, 2013.

[15] F. W. Warr and M. P. Robillard, "Suade: Topology-based searches for software investigation," in *Proceedings of the 29th international conference on Software Engineering*, pp. 780–783, IEEE Computer Society, 2007.

[16] A. T. Nguyen, H. A. Nguyen, T. T. Nguyen, and T. N. Nguyen, "Grapacc: a graph-based pattern-oriented, context-sensitive code completion tool," in *Proceedings of the 2012 International Conference on Software Engineering*, pp. 1407–1410, IEEE Press, 2012.

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

[18] B. Hartmann, D. MacDougall, J. Brandt, and S. R. Klemmer, "What would other programmers do: suggesting solutions to error messages," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1019–1028, ACM, 2010.

[19] X. Zhao, Z. Xing, M. A. Kabir, N. Sawada, J. Li, and S.-W. Lin, "Hdskg: Harvesting domain specific knowledge graph from content of webpages," in *Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on*, pp. 56–67, IEEE, 2017.

[20] X. Zhao, "Hdso: Harvest domain specific non-taxonomic relations of ontology from internet by deep neural networks (dnn)," *BSR winter school Big Software on the Run: Where Software meets Data*, p. 74.

[21] C. Chen, S. Gao, and Z. Xing, "Mining analogical libraries in q&a discussions–incorporating relational and categorical knowledge into word embedding," in *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, vol. 1, pp. 338–348, IEEE, 2016.

[22] C.-H. Lee, Y.-H. Wang, and A. J. Trappey, "Ontology-based reasoning for the intelligent handling of customer complaints," *Computers & Industrial Engineering*, vol. 84, pp. 144–155, 2015.

[23] J. Li, Z. Xing, D. Ye, and X. Zhao, "From discussion to wisdom: web resource recommendation for hyperlinks in stack overflow," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pp. 1127–1133, ACM, 2016.

[24] F. Thung, "Api recommendation system for software development," in *Automated Software Engineering (ASE), 2016 31st IEEE/ACM International Conference on*, pp. 896–899, IEEE, 2016.