Purpose. This paper presents the findings of a review of the literature published in the twenty-first century in order to identify and analyze the current state of tools that track developer interactions with integrated development environments, as well as to recommend future research directions based on the actual state. Methodology. By systematically searching in five digital libraries we conducted a systematic review of the literature on data collection tools from integrated development environments published in the twenty-first century. Fifty-five papers were selected as primary studies. Findings. 55 articles were analyzed and the findings show that using an integrated development environment to collect usage data provides more insight into developer activities than it was previously possible. Usage data allows us to analyze how developers spend their time. With usage data, you can learn more about how developers create mental models, investigate code, conduct mini-experiments through trial and error, and what can help everyone improve performance. The research community continues to be highly active in developing tools to track developer activity. The findings indicate that more research is needed in this area to better understand and measure programmer behavior. Originality. For the first time, systematization and analysis of tools for tracking programmer's behavior in an integrated development environment have been carried out. Practical value. Our study contributes to a better understanding of the current state of research on programmer behavior in integrated development environments. An analysis of the study can help define a research agenda as a starting point for the creation of a novel practical tool.

Keywords: software development process; debugging; integrated development environment; literature review

Introduction

Traditionally, improving software development productivity has been an important challenge in software engineering. Given the vast differences in developer productivity, there is significant potential to improve support for the programming process by better understanding how developers approach software development and the individual challenges they face.

A software developer's productivity can be measured by observing and collecting certain types of events related to the use of the integrated development environment.

To assist developers in their daily work, you need to understand developers' activities, especially how they develop source code. Ideally, this can be done by observing developers in their actual work environment.

The majority of developers nowadays work in an integrated development environment (IDE). The usage of an IDE simplifies the development process. IDEs are popular among software engineers because they assist them with day-to-day tasks such as development and maintenance. Instead of using the version system repository as a data source, an alternative is to monitor the programmer's activities invisibly from the IDE he is using.
By getting information directly from the IDE, you can record much more detailed events. Researchers can learn about developer behavior in such IDEs, which can help them support (or refute) conclusions on developer behavior in various software engineering tasks and the use of related tools.

Data on how developers use their IDEs provide extra information into how they produce software. The IDE’s usage of data collection technologies allows for a more complete understanding of how developers work than was previously possible. The most obvious application of usage data is to examine how developers spend their time in the IDE by identifying usage log events and tracking the time between them. We can acquire a better understanding of the developer’s time allocation and uncover ways to save time by analyzing usage data. When we monitor a programmer’s interactions with an IDE, we can look for patterns in the flow of interactions that indicate that help is needed and provide it immediately.

**Purpose**

In this paper, we examine published research from the twenty-first century to investigate tools that track developer interactions with integrated development environments, based on the findings of a thorough literature review. The purpose of this research is to determine whatever tools have been developed in the research community and how they might be used. We identified research gaps and proposed future research directions based on our findings.

**Methodology**

The recommendations suggested by Kitchenham [25], which are among the most widely accepted in software engineering, were adopted to conduct a systematic literature review.

We used a well-defined procedure in our review, which included the following steps:
1. Identifying the research questions.
2. Conducting a database search.
3. Study selection.
4. Filtering studies by assessing their relevance.
5. Extraction of data.
6. A summary of the findings.
7. Writing a review report.

**Research questions.** We established the following research questions to determine the study’s scope:
1. How many tools for tracking developer activities with integrated development environments have been developed in the twenty-first century?
2. How have researchers used these tools to collect and analyze developer behavior?

The process we use to select relevant articles is shown in Figure 1.

![Fig. 1. Papers selection methodology](image-url)
Initial search. We searched five well-known digital libraries, such as Google Scholar, ACM, IEEE Xplore, ScienceDirect, and Mendeley. We chose these databases because they are very large, which makes this review exhaustive.

Unique. We merged the results from each database into a single set and removed duplicates.

Application of selection criteria. This stage allows us to determine whether the articles we receive are relevant to our topics. The abstract and main text of each article is evaluated to ensure that they fit the inclusion criteria.

The inclusion criteria are:
1. Twenty-first-century research that is relevant to our research questions and has been published in journals or conferences.
2. The article is written in English, Russian, or Ukrainian, and the full text of the article is available.

Snowballing. To decide whether to include a paper in the study, we looked at the list of all references and read the whole text of the article. To prevent overlooking potentially relevant studies, we employed the «snowballing» method, analyzing references for each selected study. The snowballing process was carried out in both directions (backward and forward).

For each chosen paper, data was collected and analyzed.

Findings

Now we can answer our research questions.

RQ 1. *How many tools for tracking developer activities with IDEs have been developed in the twenty-first century?*

Overall, we found 55 relevant papers, presented in Table 1.

These papers are all about a tool that keeps track of a developer's interactions with an integrated development environment.

The names of journals and conferences are listed in Table 1, along with the total number of papers from each source.

The year-by-year distribution of tools invented in the twenty-first century is depicted in Figure 2.

RQ 2. *How have researchers used these tools to collect and analyze developer behavior?*

We divided the selected tools into three groups that track: coding behavior, debugging behavior, and collaborative interaction.

Coding behavior. We have identified 44 tools to help you understand coding behavior [1–4, 7–12, 14–16, 21–24, 26–28, 30–40, 42–44, 46–48, 50–56].

These are IDE plugins that track and classify developer activity by listening for events related to developer behavior. Researchers used these plugins to determine how much time programmers spent writing code in the IDE. They were designed to track developer activity in the IDE by recording a range of code editing events, keystrokes, and keyboard shortcuts when building software. They allow you to replay a programming session using fine-grained typing logs and have a timeline.

One of the reasons for observing the development of algorithms and program texts is to control independent work.

These tools can also detect manual refactoring, which is reworking performed by the developer without the use of an IDE. BeneFactor [14] identifies developers' manual refactoring and provides them with reminders to employ automatic refactoring.

As a result, researchers are investigating new methods for collecting and analyzing data that might be used to characterize the coding workflow.

According to studies, code completion is one of the most commonly utilized features in IDEs [35].

These tools are used to evaluate the behavior of novice programmers in introductory programming classes [37]. Recorded transaction history contains not only information about edits, which shows how each source file was changed, but also the developer's interaction with the IDE (i.e., tool usage). The data also includes timestamps, which can be used to estimate how much time was spent on a specific task. The cornerstone for boosting hands-on learning and lowering time wasted in the software development process is monitoring development style approaches.

These tools make it possible to create systems that automatically monitor and evaluate the coding process, as well as provide adaptive feedback and programming skill assessments.
Fig. 2. Tools developed in twenty-first century per year
### Table 1

<table>
<thead>
<tr>
<th>Source</th>
<th>Venue</th>
<th>Total</th>
<th>Referred papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Comprehension</td>
<td>Conference</td>
<td>4</td>
<td>[1, 27, 40, 44]</td>
</tr>
<tr>
<td>Applied Informatics</td>
<td>Conference</td>
<td>1</td>
<td>[2]</td>
</tr>
<tr>
<td>Automated Software Engineering</td>
<td>Conference</td>
<td>3</td>
<td>[3, 19, 32]</td>
</tr>
<tr>
<td>Software Engineering</td>
<td>Conference</td>
<td>9</td>
<td>[5, 12, 14, 18, 24, 38, 43, 45, 52]</td>
</tr>
<tr>
<td>arXiv</td>
<td>Journal</td>
<td>1</td>
<td>[8]</td>
</tr>
<tr>
<td>IEEE Transactions on Software Engineering</td>
<td>Journal</td>
<td>1</td>
<td>[9]</td>
</tr>
<tr>
<td>Smalltalk Technologies</td>
<td>Conference</td>
<td>1</td>
<td>[10]</td>
</tr>
<tr>
<td>Dynamic languages</td>
<td>Conference</td>
<td>1</td>
<td>[11]</td>
</tr>
<tr>
<td>IEEE Software</td>
<td>Journal</td>
<td>3</td>
<td>[13, 22, 35]</td>
</tr>
<tr>
<td>Eye Tracking Research and Applications</td>
<td>Conference</td>
<td>1</td>
<td>[16]</td>
</tr>
<tr>
<td>Software Maintenance</td>
<td>Conference</td>
<td>1</td>
<td>[17]</td>
</tr>
<tr>
<td>Advanced Information Systems Engineering</td>
<td>Conference</td>
<td>1</td>
<td>[21]</td>
</tr>
<tr>
<td>Ethnographies of Code</td>
<td>Conference</td>
<td>1</td>
<td>[23]</td>
</tr>
<tr>
<td>Recommendation Systems for Software Engineering</td>
<td>Conference</td>
<td>1</td>
<td>[26]</td>
</tr>
<tr>
<td>Computer Science Education</td>
<td>Conference</td>
<td>1</td>
<td>[28]</td>
</tr>
<tr>
<td>Practical Aspects of Knowledge Management</td>
<td>Conference</td>
<td>1</td>
<td>[29]</td>
</tr>
<tr>
<td>Software Analysis, Evolution and Reengineering</td>
<td>Conference</td>
<td>2</td>
<td>[30, 42]</td>
</tr>
<tr>
<td>Innovation and Technology in Computer Science Education</td>
<td>Conference</td>
<td>1</td>
<td>[31]</td>
</tr>
<tr>
<td>Eclipse Technology Exchange</td>
<td>Conference</td>
<td>1</td>
<td>[33]</td>
</tr>
<tr>
<td>Visualizing Software for Understanding and Analysis</td>
<td>Conference</td>
<td>1</td>
<td>[34]</td>
</tr>
<tr>
<td>Object-Oriented Programming</td>
<td>Conference</td>
<td>1</td>
<td>[36]</td>
</tr>
<tr>
<td>Special Interest Group on Computer Science Education</td>
<td>Conference</td>
<td>2</td>
<td>[37, 54]</td>
</tr>
<tr>
<td>Software Quality, Reliability and Security</td>
<td>Conference</td>
<td>1</td>
<td>[41]</td>
</tr>
<tr>
<td>University of Lugano</td>
<td>Thesis</td>
<td>1</td>
<td>[46]</td>
</tr>
<tr>
<td>Software Process</td>
<td>Conference</td>
<td>1</td>
<td>[47]</td>
</tr>
</tbody>
</table>
### Journals and conferences of the selected papers

<table>
<thead>
<tr>
<th>Source</th>
<th>Venue</th>
<th>Total</th>
<th>Referred papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer Sciences and Information Technologies</td>
<td>Conference</td>
<td>2</td>
<td>[48, 49]</td>
</tr>
<tr>
<td>Instrumentation Technology Conference</td>
<td>Conference</td>
<td>1</td>
<td>[50]</td>
</tr>
<tr>
<td>Asia-Pacific Software Engineering Conference</td>
<td>Conference</td>
<td>1</td>
<td>[51]</td>
</tr>
<tr>
<td>Software Maintenance and Reengineering</td>
<td>Conference</td>
<td>1</td>
<td>[53]</td>
</tr>
<tr>
<td>Evaluation and Usability of Programming Languages and Tools</td>
<td>Conference</td>
<td>1</td>
<td>[55]</td>
</tr>
<tr>
<td>Software Engineering Research and Practice</td>
<td>Journal</td>
<td>1</td>
<td>[56]</td>
</tr>
</tbody>
</table>

---

**Debugging behavior.** We have identified 5 tools to help you understand debugging behavior [5, 9, 35, 41, 49].

Debugging is an unavoidable aspect of almost all software development projects, and it is usually more difficult and time-consuming than expected. These tools collect information on debugging in the IDE debugging infrastructure, how programmers debug, and what debugging tools and approaches are available. They help researchers collect and share data on interactive debugging attempts by developers [41]. Using their previous debugging sessions' knowledge, developers can go through call method sequences and find appropriate breakpoints.

Researchers can use such tools to uncover use patterns and smells that help them better understand how usable development environments are for debugging.

Breakpoints and step-by-step code examination are the most often used debugging functions, while sophisticated debugging options in IDEs are underutilized [5]. Developers often avoid complex debugger functions such as breakpoints, and prefer simpler debugging techniques such as printf debugging. Even when more efficient commands are available, users tend to utilize only a few debugging commands [9].

Programmers spend the majority of their time reading and interpreting source code, according to the research. They believe that running the application with a debugger numerous times is the most effective way to comprehend the code. This supports the hypothesis that debugging is used to both understand the source code and find bugs.

By examining how developers use IDEs, it is possible to uncover patterns of programmer behavior during debugging and identify the issues they face.

**Collaborative interaction.** We have identified 8 tools to help you understand collaborative interaction [6, 13, 17–20, 29, 45].

These tools make it easier for team members to keep track of each other's work and give them fast access to important information for group communication. They provide details about the files that other members of the team are working on and changing. Such information includes which code files are being modified, who is modifying them, and how they are being used.

The developer can use such a tool to see which team members are looking at which files, methods, and classes are now being changed.

These tools integrate collaboration features like text and VoIP chat to the programming environment. It displays which people are online as well as whether or not they are editing or debugging.

These tools are extensible, which means that they can be enhanced and integrated into other systems.

The key challenge these technologies confront is finding a balance between giving essential information about team members' actions while not overburdening developers with useless data.

**Recommendations.** The majority of research have created their own artifacts for designing and performing experiments that are not publicly available. As a result, they cannot be used to reproduce investigations or run new experiments. As a result, researchers should make their data available to other researchers who want to replicate their find-
ings, while accurately disclosing the criteria and elements they used to design and execute the investigations.

Usage data can be useful, but developers may have some concerns about the privacy of the data being collected and to whom the data is shared. These worries emerge mostly because the information gathered could disclose specific developers or portions of the source code that organizations are developing.

Steps such as encryption of critical pieces of information can be implemented to reduce worries about the confidentiality of obtained data. Developer names, window headers, file names, and source code identifiers, for example, might be hashed to obfuscate the data and limit the danger of acquiring information identifying the developer or the projects and code they are working on.

Originality and practical value

The tools for tracking programmer activity in an integrated development environment were systematized and studied for the first time.

This systematic literature review complements existing research on tools that track developer interactions with the integrated development environment in three ways:

1. An examination and demonstration of all twenty-first-century tools.
2. A summary of tool development issues that have been resolved.
3. Making recommendations for future research.

We believe that conducting this systematic literature review at this time is crucial because it brings together all of the past research and can help researchers avoid misusing IDE use tracking technology in software engineering research.

As a result, it can serve as a starting point for future research into programmer behavior in an integrated development environment.

Conclusions

We conducted a systematic literature review to determine the current state of tools that track developer interactions with integrated development environments. We found 55 papers that were related to the creation of a tool for tracking developer engagement in IDEs. We also provided advise to the software development community and a list of ideas for academics interested in developing a tool to track developer activity.

Our findings contribute to a better understanding of where programming behavior research in integrated development environments stands right now. An examination of the research findings can assist in the development of a research agenda, which can subsequently be used to create a new practical tool.

The research community continues to be highly active in developing methods to track developer activity. Further research in this area is needed to better understand and evaluate programmer behavior, according to the findings.

LIST OF REFERENCE LINKS


42. Proksch S., Nadi S., Amann S., Mezini M. Enriching in-IDE process information with fine-grained source code. 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering


О. О. ЖЕВАГО

1 Каф. «Комп’ютерні інформаційні технології», Дніпровський національний університет залізничного транспорту імені академіка В. Лазаряна, вул. Лазаряна, 2, Дніпро, Україна, 49010, тел. +38 (056) 373 15 35, ел. пошта marakonec@gmail.com, ORCID 0000-0003-0019-8320
Огляд інструментів збору даних про процеси розробки та налагодження програмного забезпечення з інтегрованих середовищ розробки

Мета. У цій статті передбачено провести огляд літератури, опублікованої у двадцять першому столітті, із метою виявлення та аналізу поточного стану інструментів, які відслідковують взаємодію розробників з інтегрованими середовищами розробки, а також надати рекомендації для подальших досліджень на основі поточного стану. Методика. Шляхом систематичного пошуку в п’яті електронних бібліотеках ми провели детальний огляд літератури щодо інструментів збору даних з інтегрованих середовищ розробки. Було відібрано 55 інструментів. Результати. Аналіз отриманих інструментів показує, що використання інтегрованого середовища розробки для збору даних дозволяє краще зрозуміти дії розробників, ніж це було можливо раніше. Дані із середовищ розробки дозволяють нам аналізувати, як розробники проводять свій час, дізнатися більше про те, як вони створюють ментальні моделі, досліджують код, проводять міні експерименти, роблячи спроби і допускаючи помилки, а також з’ясовувати, що може допомогти кожному підвищити продуктивність. Дослідницьке товариство продовжує розробляти інструменти для відслідкування активності розробників. Отримані дані свідчать, що в цій галузі необхідні додаткові дослідження, щоб краще зрозуміти поведінку програмістів. Наукова новизна. Уперше проведено систематизацію та аналіз інструментів відслідкування поведінки розробників в інтегрованому середовищі розробки. Практична значимість. Отримані результати сприяють кращому розумінню поточного стану досліджень поведінки розробників в інтегрованих середовищах розробки та можуть допомогти визначити план як відправну точку для створення нового практичного інструменту.

Ключові слова: процес розробки програмного забезпечення; налагодження; інтегроване середовище розробки; огляд літератури

REFERENCES


10. Dias, M., Cassou, D., & Ducasse, S. (2013). Representing code history with development environment events. *IWST-2013 - 5th International Workshop on Smalltalk Technologies*, 1-7. (in English)


40. Parnin, C., & Gorg, C. Building Usage Contexts During Program Comprehension. In 14th IEEE International Conference on Program Comprehension (ICPC'06) (pp. 13-22). Athens, Greece. DOI: http://doi.org/10.1109/icpc.2006.14 (in English)


Creative Commons Attribution 4.0 International
doi: https://doi.org/10.15802/stp2021/242042 © O. O. Zhevaho, 2021


51. Sison, R. (2005). Personal software process (PSP) assistant. In 12th Asia-Pacific Software Engineering Conference (APSEC'05) (pp. 8). Taipei, Taiwan. DOI: http://doi.org/10.1109/apsec.2005.87 (in English)


Received: February 12, 2021
Accepted: June 11, 2021