ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2021, № 3 (93)

ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ ТА МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ

UDC 004.4(048.8)

O. O. ZHEVAHO[1*]

[1*]Dep. «Computer and Information Technologies», Dnipro National University of Railway Transport named after Academician V. Lazaryan, Lazaryana St., 2, Dnipro, Ukraine, 49010, tel. +38 (056) 373 15 35, e-mail marakonec@gmail.com, ORCID 0000-0003-0019-8320

An Overview of Tools for Collecting Data on Software Development and Debugging Processes from Integrated Development Environments

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.

doi: https://doi.org/10.15802/stp2021/242042

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2021, № 3 (93)

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.

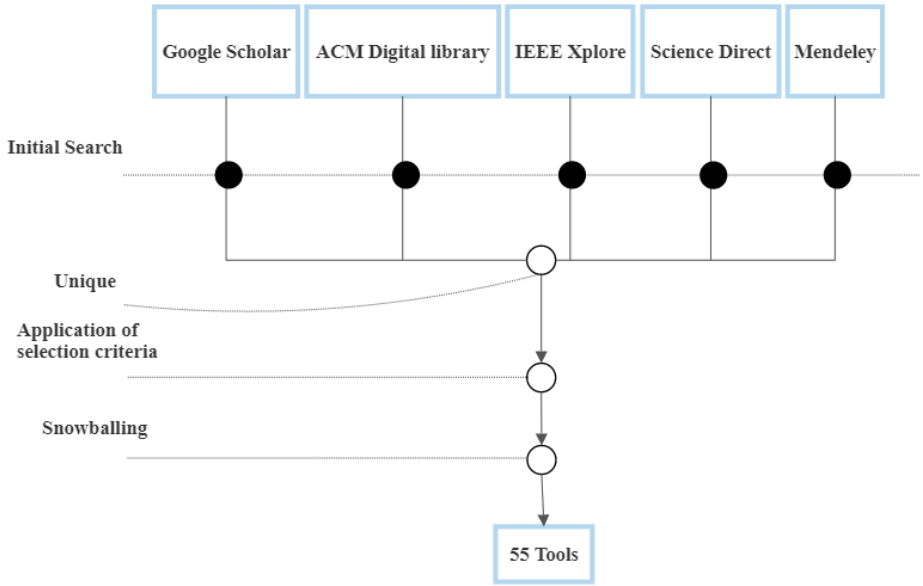

Fig. 1. Papers selection methodology

doi: https://doi.org/10.15802/stp2021/242042

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.

doi: https://doi.org/10.15802/stp2021/242042

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2021, № 3 (93)

Fig. 2. Tools developed in twenty-first century per year

doi: https://doi.org/10.15802/stp2021/242042

Table 1

Journals and conferences of the selected papers

Source	Venue	Total	Referred papers
Program Comprehension	Conference	4	[1, 27, 40, 44]
Applied Informatics	Conference	1	[2]
Automated Software Engineering	Conference	3	[3, 19, 32]
Foundations of Software Engineering	Conference	1	[4]
Software Engineering	Conference	9	[5, 12, 14, 18, 24, 38, 43, 45, 52]
Human Factors in Computing Systems	Conference	1	[6]
Current Trends in Theory and Practice of Informatics	Conference	1	[7]
arXiv	Journal	1	[8]
IEEE Transactions on Software Engineering	Journal	1	[9]
Smalltalk Technologies	Conference	1	[10]
Dynamic languages	Conference	1	[11]
IEEE Software	Journal	3	[13, 22, 35]
Systems, Programming, and Applications: Software for Humanity	Conference	2	[15, 20]
Eye Tracking Research and Applications	Conference	1	[16]
Software Maintenance	Conference	1	[17]
Advanced Information Systems Engineering	Conference	1	[21]
Ethnographies of Code	Conference	1	[23]
Recommendation Systems for Software Engineering	Conference	1	[26]
Computer Science Education	Conference	1	[28]
Practical Aspects of Knowledge Management	Conference	1	[29]
Software Analysis, Evolution and Reengineering	Conference	2	[30, 42]
Innovation and Technology in Computer Science Education	Conference	1	[31]
Eclipse Technology Exchange	Conference	1	[33]
Visualizing Software for Understanding and Analysis	Conference	1	[34]
Object-Oriented Programming	Conference	1	[36]
Special Interest Group on Computer Science Education	Conference	2	[37, 54]
International Journal of Technology Enhanced Learning	Journal	1	[39]
Software Quality, Reliability and Security	Conference	1	[41]
University of Lugano	Thesis	1	[46]
Software Process	Conference	1	[47]

doi: https://doi.org/10.15802/stp2021/242042

Continuation of Table 1

Journals and conferences of the selected papers

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

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-

doi: https://doi.org/10.15802/stp2021/242042

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

1. Amann S., Proksch S., Nadi S. FeedBaG : An interaction tracker for Visual Studio. *2016 IEEE 24th International Conference on Program Comprehension (ICPC)* (Austin, 16-17 May 2016). Austin, 2016. P. 1–3. DOI: http://doi.org/10.1109/icpc.2016.7503741
2. Antal G., Végh Á. Z., Bilicki V. A methodology for measuring software development productivity using Eclipse IDE. *Proceedings of the 9th International Conference on Applied Informatics* (Eger, Jan. 29–Feb. 1 2014). Eger, 2014. Vol. 2. P. 255–262. DOI: http://doi.org/10.14794/icai.9.2014.2.255
3. Bao L., Ye D., Xing Z., Xia X., Wang X. ActivitySpace : A Remembrance Framework to Support Interapplication Information Needs. *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (Lincoln, 9-13 Nov. 2015). Lincoln, 2015. P. 864–869. DOI: http://doi.org/10.1109/ase.2015.90
4. Beller M., Gousios G., Panichella A., Zaidman A. When, how, and why developers (do not) test in their IDEs. *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (August 2015). 2015. P. 179–190. DOI: http://doi.org/10.1145/2786805.2786843
5. Beller M., Spruit N., Spinellis D., Zaidman A. On the dichotomy of debugging behavior among programmers. *Proceedings of the 40th International Conference on Software Engineering* (May 2018). 2018. P. 572–583. DOI: http://doi.org/10.1145/3180155.3180175

doi: https://doi.org/10.15802/stp2021/242042

6. Biehl J. T., Czerwinski M., Smith G. Robertson G. G. FASTDash : a visual dashboard for fostering awareness in software teams. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (April 2007). 2007. P. 1313–1322. DOI: http://doi.org/10.1145/1240624.1240823
7. Bieliková M., Polášek I., Barla M., Kuric E., Rástočný K., Tvarožek J., Lacko P. Platform Independent Software Development Monitoring : Design of an Architecture. *SOFSEM 2014 : Theory and Practice of Computer Science*. 2014. Vol. 8327. P. 126–137. DOI: http://doi.org/10.1007/978-3-319-04298-5_12
8. Caldeira J., Brito e Abreu F., Cardoso J., Ribeiro R., Werner C. Profiling Software Developers with Process Mining and N-Gram Language Models. *Arxiv*. 2021. URL: http://arxiv.org/abs/2101.06733v1
9. Damevski K., Shepherd D. C., Schneider J., Pollock L. Mining Sequences of Developer Interactions in Visual Studio for Usage Smells. *IEEE Transactions on Software Engineering* (1 Apr. 2017). 2017. Vol. 43. Iss. 4. P. 359–371. DOI: http://doi.org/10.1109/tse.2016.2592905
10. Dias M., Cassou D., Ducasse S. Representing code history with development environment events. *IWST-2013 – 5th International Workshop on Smalltalk Technologies.* 2013. P. 1–7.
11. Ebraert P., Vallejos J., Costanza P., Van Paesschen E., D'Hondt T. Change-oriented software engineering. *Proceedings of the 2007 international conference on Dynamic languages : in conjunction with the 15th International Smalltalk Joint Conference 2007* (August 2007). 2007. P. 3–24. DOI: http://doi.org/10.1145/1352678.1352680
12. Foster S. R., Griswold W. G., Lerner S. WitchDoctor : IDE support for real-time auto-completion of refactorrings. *2012 34th International Conference on Software Engineering (ICSE)* (Zurich, 2-9 June 2012). Zurich, 2012. P. 222–232. DOI: http://doi.org/10.1109/icse.2012.6227191
13. Frost R. Jazz and the Eclipse Way of Collaboration. *IEEE Software* (Nov.-Dec. 2007). 2007. Vol. 24. Iss. 6. P. 114–117. DOI: http://doi.org/10.1109/ms.2007.170
14. Ge X., DuBose Q. L., Murphy-Hill E. Reconciling manual and automatic refactoring. *2012 34th International Conference on Software Engineering (ICSE)* (Zurich, 2-9 June 2012). Zurich, 2012. P. 211–221. DOI: http://doi.org/10.1109/icse.2012.6227192
15. Gu Z., Schleck D., Barr E. T., Su Z. Capturing and Exploiting IDE Interactions. *Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software* (October 2014). 2014. P. 83–94. DOI: http://doi.org/10.1145/2661136.2661144
16. Guarnera D. T., Bryant C. A., Mishra A., Maletic J. I., Sharif B. iTrace : Eye tracking infrastructure for development environments. *Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications* (June 2018). 2018. № 105. P. 1–3. DOI: http://doi.org/10.1145/3204493.3208343
17. Guzzi A., Pinzger M., van Deursen A. Combining micro-blogging and IDE interactions to support developers in their quests. *2010 IEEE International Conference on Software Maintenance* (Timisoara, 12-18 Sept. 2010). Timisoara, 2010. P. 1–5. DOI: http://doi.org/10.1109/icsm.2010.5609683
18. Hattori L., Lanza M. Syde : a tool for collaborative software development. *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering* (May 2010). 2010. Vol. 2. P. 235–238. DOI: http://doi.org/10.1145/1810295.1810339
19. Hegde R., Dewan P. Connecting Programming Environments to Support Ad-Hoc Collaboration. *2008 23rd IEEE/ACM International Conference on Automated Software Engineering* (L'Aquila, 15-19 Sept. 2008). L'Aquila, 2008. P. 178–187. DOI: http://doi.org/10.1109/ase.2008.28
20. Hundhausen C. D., Carter A. S. Supporting Social Interactions and Awareness in Educational Programming Environments. *Proceedings of the 5th Workshop on Evaluation and Usability of Programming Languages and Tools* (October 2014). 2014. P. 55–56. DOI: http://doi.org/10.1145/2688204.2688215
21. Ioannou C., Burattin A., Weber B. Mining Developers' Workflows from IDE Usage. *Advanced Information Systems Engineering Workshops*. 2018. Vol. 316. P. 167–179. DOI: http://doi.org/10.1007/978-3-319-92898-2_14
22. Jaspan C., Jorde M., Egelman C., Green C., Holtz B., Smith E., Hodges M., Knight A., Kammer L., Dicker J., Sadowski C., Lin J., Cheng L., Canning M., Murphy-Hill E. Enabling the Study of Software Development Behavior With Cross-Tool Logs. *IEEE Software* (Nov.-Dec. 2020). 2020. Vol. 37. Iss. 6. P. 44–51. DOI: http://doi.org/10.1109/ms.2020.3014573
23. Jekutsch S. *ElectroCodeoGram : An Environment for Studying Programming*. URL: http://www.mi.fu-berlin.de/wiki/pub/SE/ElectroCodeoGram/lancaster.pdf
24. Johnson P. M., Hongbing K., Agustin J., Chan C., Moore C., Miglani J., Shenyan Zh., Doane W. E. J. Beyond the Personal Software Process : Metrics collection and analysis for the differently disciplined. *25th Interna-*

doi: https://doi.org/10.15802/stp2021/242042

tional Conference on Software Engineering, 2003. Proceedings (Portland, 3-10 May 2003). Portland, 2003. P. 641–646. DOI: http://doi.org/10.1109/icse.2003.1201249

25. Kitchenham B. A., Charters S. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. URL: https://userpages.uni-koblenz.de/~laemmel/esecourse/slides/slr.pdf
26. Kobayashi T., Kato N., Agusa K. Interaction histories mining for software change guide. *2012 Third International Workshop on Recommendation Systems for Software Engineering (RSSE)* (Zurich, 4 June 2012). Zurich, 2012. P. 73–77. DOI: http://doi.org/10.1109/rsse.2012.6233415
27. Lijie Z., Godfrey M. W., Hassan A. E. Detecting Interaction Coupling from Task Interaction Histories. *15th IEEE International Conference on Program Comprehension (ICPC '07)* (Banff, 26-29 June 2007). Banff, 2007. P. 135–144. DOI: http://doi.org/10.1109/icpc.2007.18
28. Lyulina E., Birillo A., Kovalenko V., Bryksin T. TaskTracker-tool : A Toolkit for Tracking of Code Snapshots and Activity Data During Solution of Programming Tasks. *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (March 2021). 2021. P. 495–501. DOI: http://doi.org/10.1145/3408877.3432534
29. Maalej W., Happel H.-J. A Lightweight Approach for Knowledge Sharing in Distributed Software Teams. *Practical Aspects of Knowledge Management*. Vol. 5345. P. 14–25. DOI: http://doi.org/10.1007/978-3-540-89447-6_4
30. Maruyama K., Hayashi S., Omori T. ChangeMacroRecorder : Recording fine-grained textual changes of source code. *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)* (Campobasso, 20-23 March 2018). Campobasso, 2018. P. 537–541. DOI: http://doi.org/10.1109/saner.2018.8330255
31. Matsuzawa Y., Okada K., Sakai S. Programming process visualizer : A proposal of the tool for students to observe their programming process. *Proceedings of the 18th ACM conference on Innovation and technology in computer science education* (July 2013). 2013. P. 46–51. DOI: http://doi.org/10.1145/2462476.2462493
32. McIntyre M. M., Walker R. J. Assisting potentially-repetitive small-scale changes via semi-automated heuristic search. *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering* (November 2007). 2007. P. 497–500. DOI: http://doi.org/10.1145/1321631.1321718
33. McKeogh J., Exton C. Eclipse plug-in to monitor the programmer behaviour. *Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange* (October 2004). 2004. P. 93–97. DOI: http://doi.org/10.1145/1066129.1066148
34. Minelli R., Lanza M. Visualizing the workflow of developers. *2013 First IEEE Working Conference on Software Visualization (VISSOFT)* (Eindhoven, 27-28 Sept. 2013). Eindhoven, 2013. P. 1–4. DOI: http://doi.org/10.1109/vissoft.2013.6650531
35. Murphy G. C., Kersten M., Findlater L. How are Java software developers using the Eclipse IDE? *IEEE Software* (July-Aug. 2006). 2006. Vol. 23. Iss. 4. P. 76–83. DOI: http://doi.org/10.1109/ms.2006.105
36. Negara S., Vakilian M., Chen N., Johnson R. E., Dig D. Is It Dangerous to Use Version Control Histories to Study Source Code Evolution? *ECOOP 2012 – Object-Oriented Programming*. Berlin, 2012. Vol. 7313. P. 79–103. DOI: http://doi.org/10.1007/978-3-642-31057-7_5
37. Norris C., Barry F., Fenwick Jr. J. B., Reid K., Rountree J. ClockIt : collecting quantitative data on how beginning software developers really work. *ACM SIGCSE Bulletin*. 2008. Vol. 40. Iss. 3. P. 37–41. DOI: http://doi.org/10.1145/1597849.1384284
38. Omori T., Maruyama K. A change-aware development environment by recording editing operations of source code. *Proceedings of the 2008 international working conference on Mining software repositories* (May 2008). 2008. P. 31–34. DOI: http://doi.org/10.1145/1370750.1370758
39. Pärtel M., Luukkainen M., Vihavainen A., Vikberg T. Test My Code. *International Journal of Technology Enhanced Learning (IJTEL)*. 2013. Vol. 5, № 3/4. P. 271. DOI: http://doi.org/10.1504/ijtel.2013.059495
40. Parnin C., Gorg C. Building Usage Contexts During Program Comprehension. *14th IEEE International Conference on Program Comprehension (ICPC'06)* (Athens, 14-16 June 2006). Athens, 2006. P. 13–22. DOI: http://doi.org/10.1109/icpc.2006.14
41. Petrillo F., Soh Z., Khomh F., Pimenta M., Freitas C., Gueheneuc Y.-G. Towards Understanding Interactive Debugging. *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)* (Vienna, 1-3 Aug. 2016). Vienna, 2016. P. 152–163. DOI: http://doi.org/10.1109/qrs.2016.27
42. Proksch S., Nadi S., Amann S., Mezini M. Enriching in-IDE process information with fine-grained source code history. *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering*

doi: https://doi.org/10.15802/stp2021/242042

(SANER) (Klagenfurt, 20-24 Feb. 2017). Klagenfurt, 2017. P. 250–260. DOI: http://doi.org/10.1109/saner.2017.7884626

43. Robbes R., Lanza, M. SpyWare : a change-aware development toolset. *Proceedings of the 30th international conference on Software engineering* (May 2008). 2008. P. 847–850. DOI: http://doi.org/10.1145/1368088.1368219
44. Rothlisberger D., Nierstrasz O., Ducasse S., Pollet D., Robbes R. Supporting task-oriented navigation in IDEs with configurable HeatMaps. *2009 IEEE 17th International Conference on Program Comprehension* (Vancouver, 17-19 May 2009). Vancouver, 2009. P. 253–257. DOI: http://doi.org/10.1109/icpc.2009.5090052
45. Salinger S., Oezbek C., Beecher K., Schenk J. Saros : an eclipse plug-in for distributed party programming. *Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering* (May 2010). 2010. P. 48–55. DOI: http://doi.org/10.1145/1833310.1833319
46. Shron Y. *EclipsEye Spying onEclipse*. URL: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=064DE0D8481B3C9FD39F875EA79AD59C?doi=10.1.1.709.9455&rep=rep1&type=pdf
47. Shin H., Choi H.-J., Baik J. Jasmine : A PSP Supporting Tool. *Software Process Dynamics and Agility.* 2007. Vol. 4470. P. 73–83. DOI: http://doi.org/10.1007/978-3-540-72426-1_7
48. Shynkarenko V., Zhevago, O. Visualization of program development process. *2019 IEEE 14th International Conference on Computer Sciences and Information Technologies (CSIT)* (Lviv, 17-20 Sept. 2019). Lviv, 2019. P. 142–145. DOI: http://doi.org/10.1109/stc-csit.2019.8929774
49. Shynkarenko V., Zhevaho O. Development of a toolkit for analyzing software debugging processes using the constructive approach. *Eastern-european Journal of Enterprise Technologies*. 2020. Vol. 5. Iss. 2 (107). P. 29–38. DOI: http://doi.org/10.15587/1729-4061.2020.215090
50. Sillitti J., Succi, Vernazza. Collecting, integrating and analyzing software metrics and personal software process data. *2003 Proceedings 29th Euromicro Conference* (Belek-Antalya, 1-6 Sept. 2003). Belek-Antalya, 2003. P. 336–342. DOI: http://doi.org/10.1109/eurmic.2003.1231611
51. Sison R. Personal software process (PSP) assistant. *12th Asia-Pacific Software Engineering Conference (APSEC'05)* (Taipei, 15-17 Dec. 2005). Taipei, 2005. P. 8. DOI: http://doi.org/10.1109/apsec.2005.87
52. Snipes W., Nair A. R., Murphy-Hill E. Experiences gamifying developer adoption of practices and tools. *Companion Proceedings of the 36th International Conference on Software Engineering* (May 2014). 2014. P. 105–114. DOI: http://doi.org/10.1145/2591062.2591171
53. Soetens Q. D., Demeyer S. ChEOPSJ : Change-Based Test Optimization. *2012 16th European Conference on Software Maintenance and Reengineering* (Szeged, 27-30 March 2012). Szeged, 2012. P. 535–538. DOI: http://doi.org/10.1109/csmr.2012.70
54. Steinert B., Cassou D., Hirschfeld R. CoExist : Overcoming Aversion to Change Preserving Immediate Access to Source Code and Run-time Information of Previous Development States. *ACM SIGPLAN Notices*. 2013. Vol. 48. Iss. 2. P. 107–118. DOI: http://doi.org/10.1145/2480360.2384591
55. Yoon Y., Myers B. A. Capturing and analyzing low-level events from the code editor. *Proceedings of the 3rd ACM SIGPLAN workshop on Evaluation and usability of programming languages and tools* (October 2011). 2011. P. 25–30. DOI: http://doi.org/10.1145/2089155.2089163
56. Yuan X., Vega P., Yu H., Li Y. *A Personal Software Process Tool for Eclipse Environment*. 2005. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.121.1638&rep=rep1&type=pdf

О. О. ЖЕВАГО[1*]

[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

doi: https://doi.org/10.15802/stp2021/242042

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2021, № 3 (93)

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'яти 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 й допускаючи помилки, а також з'ясувати, 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 програміста в інтегрованому середовищі розробки. **Практична значимість.** Отримані результати сприяють кращому розумінню поточного стану досліджень поведінки програмістів 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

1. Amann, S., Proksch, S., & Nadi, S. (2016). FeedBaG: An interaction tracker for Visual Studio. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)* (pp. 1-3). Austin, USA. DOI: http://doi.org/10.1109/icpc.2016.7503741 (in English)
2. Antal, G., Végh, Á. Z., & Bilicki, V. (2014). A methodology for measuring software development productivity using Eclipse IDE. In *Proceedings of the 9th International Conference on Applied Informatics* (Vol. 2, pp. 252-262). Eger, Hungary. DOI: http://doi.org/10.14794/icai.9.2014.2.255 (in English)
3. Bao, L., Ye, D., Xing, Z., Xia, X., & Wang, X. (2015). ActivitySpace: A Remembrance Framework to Support Interapplication Information Needs. In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 864-869). Lincoln, USA. DOI: http://doi.org/10.1109/ase.2015.90 (in English)
4. Beller, M., Gousios, G., Panichella, A., & Zaidman, A. (2015). When, how, and why developers (do not) test in their IDEs. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (pp. 179-190). DOI: http://doi.org/10.1145/2786805.2786843 (in English)
5. Beller, M., Spruit, N., Spinellis, D., & Zaidman, A. (2018). On the dichotomy of debugging behavior among programmers. In *Proceedings of the 40th International Conference on Software Engineering* (pp. 572-583). DOI: http://doi.org/10.1145/3180155.3180175 (in English)
6. Biehl, J. T., Czerwinski, M., Smith, G., & Robertson, G. G. (2007). FASTDash: a visual dashboard for fostering awareness in software teams. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 1313-1322). DOI: http://doi.org/10.1145/1240624.1240823 (in English)
7. Bieliková, M., Polášek, I., Barla, M., Kuric, E., Rástočný, K., Tvarožek, J., & Lacko, P. (2014). Platform Independent Software Development Monitoring: Design of an Architecture. In *SOFSEM 2014: Theory and Practice of Computer Science* (Vol. 8327, pp. 126-137). DOI: http://doi.org/10.1007/978-3-319-04298-5_12 (in English)
8. Caldeira, J., Brito e Abreu, F., Cardoso, J., Ribeiro, R., & Werner, C. (2021). Profiling Software Developers with Process Mining and N-Gram Language Models. *Arxiv*. Retrieved from http://arxiv.org/abs/2101.06733v1 (in English)
9. Damevski, K., Shepherd, D. C., Schneider, J., & Pollock, L. (2017). Mining Sequences of Developer Interactions in Visual Studio for Usage Smells. *IEEE Transactions on Software Engineering* (Vol. 43, Iss. 4, P. 359-371). DOI: http://doi.org/10.1109/tse.2016.2592905 (in English)

doi: https://doi.org/10.15802/stp2021/242042

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)
11. Ebraert, P., Vallejos, J., Costanza, P., Van Paesschen, E., & D'Hondt, T. (2007). Change-oriented software engineering. In *Proceedings of the 2007 international conference on Dynamic languages: in conjunction with the 15th International Smalltalk Joint Conference 2007* (pp. 3-24). DOI: http://doi.org/10.1145/1352678.1352680 (in English)
12. Foster, S. R., Griswold, W. G., & Lerner, S. (2012). WitchDoctor: IDE support for real-time auto-completion of refactorings. In *2012 34th International Conference on Software Engineering (ICSE)* (pp. 22-232). Zurich, Switzerland. DOI: http://doi.org/10.1109/icse.2012.6227191 (in English)
13. Frost, R. (2007). Jazz and the Eclipse Way of Collaboration. In *IEEE Software* (Vol. 24, Iss. 6, pp. 114-117). DOI: http://doi.org/10.1109/ms.2007.170 (in English)
14. Ge, X., DuBose, Q. L., & Murphy-Hill, E. (2012). Reconciling manual and automatic refactoring. In *2012 34th International Conference on Software Engineering (ICSE)* (pp. 211-221). Zurich, Switzerland. DOI: http://doi.org/10.1109/icse.2012.6227192 (in English)
15. Gu, Z., Schleck, D., Barr, E. T., & Su, Z. (2012). Capturing and Exploiting IDE Interactions. In *Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software* (pp. 83-94). DOI: http://doi.org/10.1145/2661136.2661144 (in English)
16. Guarnera, D. T., Bryant, C. A., Mishra, A., Maletic, J. I., & Sharif, B. (2018). iTrace: Eye tracking infrastructure for development environments. In *Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications* (No 105, pp. 1-3). DOI: http://doi.org/10.1145/3204493.3208343 (in English)
17. Guzzi, A., Pinzger, M., & van Deursen, A. (2010). Combining micro-blogging and IDE interactions to support developers in their quests. In *2010 IEEE International Conference on Software Maintenance* (pp. 1-5). Timisoara, Romania. DOI: http://doi.org/10.1109/icsm.2010.5609683 (in English)
18. Hattori, L., & Lanza, M. (2010). Syde: a tool for collaborative software development. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering* (Vol. 2, pp. 235-238). DOI: http://doi.org/10.1145/1810295.1810339 (in English)
19. Hegde, R., & Dewan, P. (2008). Connecting Programming Environments to Support Ad-Hoc Collaboration. In *2008 23rd IEEE/ACM International Conference on Automated Software Engineering* (pp. 178-187). L'Aquila, Italy. DOI: http://doi.org/10.1109/ase.2008.28 (in English)
20. Hundhausen, C. D., & Carter, A. S. (2014). Supporting Social Interactions and Awareness in Educational Programming Environments. In *Proceedings of the 5th Workshop on Evaluation and Usability of Programming Languages and Tools* (pp. 55-56). DOI: http://doi.org/10.1145/2688204.2688215 (in English)
21. Ioannou, C., Burattin, A., & Weber, B. (2018). Mining Developers' Workflows from IDE Usage. A*dvanced Information Systems Engineering Workshops, 316*, 167-179. DOI: http://doi.org/10.1007/978-3-319-92898-2_14 (in English)
22. Jaspan, C., Jorde, M., Egelman, C., Green, C., Holtz, B., Smith, E., … & Murphy-Hill, E. (2020). Enabling the Study of Software Development Behavior With Cross-Tool Logs. *IEEE Software* (Vol. 37, Iss. 6, pp. 44-51). DOI: http://doi.org/10.1109/ms.2020.3014573 (in English)
23. Jekutsch, S. *ElectroCodeoGram: An Environment for Studying Programming*. Retrieved from http://www.mi.fu-berlin.de/wiki/pub/SE/ElectroCodeoGram/lancaster.pdf (in English)
24. Johnson, P. M., Hongbing Kou, Agustin, J., Chan, C., Moore, C., Miglani, J., … & Doane. (2003). Beyond the Personal Software Process: Metrics collection and analysis for the differently disciplined. In *25th International Conference on Software Engineering, 2003. Proceedings* (pp. 641-646). Portland, USA. DOI: http://doi.org/10.1109/icse.2003.1201249 (in English)
25. Kitchenham, B. A. & Charters, S. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. Retrieved from https://userpages.uni-koblenz.de/~laemmel/esecourse/slides/slr.pdf (in English)
26. Kobayashi, T., Kato, N., & Agusa, K. (2012). Interaction histories mining for software change guide. In *2012 Third International Workshop on Recommendation Systems for Software Engineering (RSSE)* (pp. 73-77). Zurich, Switzerland. DOI: http://doi.org/10.1109/rsse.2012.6233415 (in English)
27. Lijie Z., Godfrey, M. W., & Hassan, A. E. (2007). Detecting Interaction Coupling from Task Interaction Histories. In *15th IEEE International Conference on Program Comprehension (ICPC '07)* (pp. 135-144). Banff, Canada. DOI: http://doi.org/10.1109/icpc.2007.18 (in English)
28. Lyulina, E., Birillo, A., Kovalenko, V., & Bryksin, T. (2021). TaskTracker-tool: A Toolkit for Tracking of Code Snapshots and Activity Data During Solution of Programming Tasks. In *Proceedings of the 52nd ACM*

doi: https://doi.org/10.15802/stp2021/242042

Technical Symposium on Computer Science Education (pp. 495-501). DOI: http://doi.org/10.1145/3408877.3432534 (in English)

29. Maalej, W., & Happel, H.-J. (2008). A Lightweight Approach for Knowledge Sharing in Distributed Software Teams. In *Practical Aspects of Knowledge Management* (Vol. 5345, pp. 14-25). DOI: http://doi.org/10.1007/978-3-540-89447-6_4 (in English)
30. Maruyama, K., Hayashi, S., & Omori, T. (2018). ChangeMacroRecorder: Recording fine-grained textual changes of source code. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)* (pp. 537-541). Campobasso, Italy. DOI: http://doi.org/10.1109/saner.2018.8330255 (in English)
31. Matsuzawa, Y., Okada, K., & Sakai, S. (2013). Programming process visualizer: A proposal of the tool for students to observe their programming process. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education* (pp. 46-51). DOI: http://doi.org/10.1145/2462476.2462493 (in English)
32. McIntyre, M. M., & Walker, R. J. (2007). Assisting potentially-repetitive small-scale changes via semi-automated heuristic search. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering* (pp. 497-500). DOI: http://doi.org/10.1145/1321631.1321718 (in English)
33. McKeogh, J., & Exton, C. (2004). Eclipse plug-in to monitor the programmer behaviour. In *Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange* (pp. 93-97). DOI: http://doi.org/10.1145/1066129.1066148 (in English)
34. Minelli, R., & Lanza, M. (2013). Visualizing the workflow of developers. In *2013 First IEEE Working Conference on Software Visualization (VISSOFT)* (pp. 1-4). Eindhoven, Netherlands. DOI: http://doi.org/10.1109/vissoft.2013.6650531 (in English)
35. Murphy, G. C., Kersten, M., & Findlater, L. (2006). How are Java software developers using the Eclipse IDE? *IEEE Software* (Vol. 23, Iss. 4, pp. 76-83). DOI: http://doi.org/10.1109/ms.2006.105 (in English)
36. Negara, S., Vakilian, M., Chen, N., Johnson, R. E., & Dig, D. (2012). Is It Dangerous to Use Version Control Histories to Study Source Code Evolution? In *ECOOP 2012-Object-Oriented Programming* (Vol. 7313, pp. 79-103). Berlin, Heidelberg. DOI: http://doi.org/10.1007/978-3-642-31057-7_5 (in English)
37. Norris, C., Barry, F., Fenwick Jr., J. B., Reid, K., & Rountree, J. (2008). ClockIt: collecting quantitative data on how beginning software developers really work. *ACM SIGCSE Bulletin, 40*(3), 37-41. DOI: http://doi.org/10.1145/1597849.1384284 (in English)
38. Omori, T., & Maruyama, K. (2008). A change-aware development environment by recording editing operations of source code. In *Proceedings of the 2008 international working conference on Mining software repositories* (pp. 31-34). DOI: http://doi.org/10.1145/1370750.1370758 (in English)
39. Pärtel, M., Luukkainen, M., Vihavainen, A., & Vikberg, T. (2013). Test My Code. *International Journal of Technology Enhanced Learning (IJTEL), 5*(3/4), 271. DOI: http://doi.org/10.1504/ijtel.2013.059495 (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)
41. Petrillo, F., Soh, Z., Khomh, F., Pimenta, M., Freitas, C., & Gueheneuc, Y.-G. (2016). Towards Understanding Interactive Debugging. In *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)* (pp. 152-163). Vienna, Austria. DOI: http://doi.org/10.1109/qrs.2016.27 (in English)
42. Proksch, S., Nadi, S., Amann, S., & Mezini, M. (2017). Enriching in-IDE process information with fine-grained source code history. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)* (pp. 250-260). Klagenfurt, Austria. DOI: http://doi.org/10.1109/saner.2017.7884626 (in English)
43. Robbes, R., & Lanza, M. (2008). SpyWare: a change-aware development toolset. In *Proceedings of the 30th international conference on Software engineering* (pp. 847-850). DOI: http://doi.org/10.1145/1368088.1368219 (in English)
44. Rothlisberger, D., Nierstrasz, O., Ducasse, S., Pollet, D., & Robbes, R. (2009). Supporting task-oriented navigation in IDEs with configurable HeatMaps. In *2009 IEEE 17th International Conference on Program Comprehension* (pp. 253-257). Vancouver, Canada. DOI: http://doi.org/10.1109/icpc.2009.5090052 (in English)

doi: https://doi.org/10.15802/stp2021/242042

45. Salinger, S., Oezbek, C., Beecher, K., & Schenk, J. (2010). Saros: an eclipse plug-in for distributed party programming. In *Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering* (pp. 48-55). DOI: http://doi.org/10.1145/1833310.1833319 (in English)
46. Sharon, Y. *EclipsEye Spying onEclipse*. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=064DE0D8481B3C9FD39F875EA79AD59C?doi=10.1.1.709.9455&rep=rep1&type=pdf (in English)
47. Shin, H., Choi, H.-J., & Baik, J. Jasmine: A PSP Supporting Tool. In *Software Process Dynamics and Agility* (Vol. 4470, pp. 73-83). DOI: http://doi.org/10.1007/978-3-540-72426-1_7 (in English)
48. Shynkarenko, V., & Zhevago, O. (2019). Visualization of program development process. In *2019 IEEE 14th International Conference on Computer Sciences and Information Technologies (CSIT)* (pp. 142-145). Lviv, Ukraine. DOI: http://doi.org/10.1109/stc-csit.2019.8929774 (in English)
49. Shynkarenko, V., & Zhevaho, O. (2020). Development of a toolkit for analyzing software debugging processes using the constructive approach. *Eastern-european Journal of Enterprise Technologies*, *5*(2(107)), 29-38. DOI: http://doi.org/10.15587/1729-4061.2020.215090 (in English)
50. Sillitti, J., Succi, & Vernazza. (2003). Collecting, integrating and analyzing software metrics and personal software process data. In *2003 Proceedings 29th Euromicro Conference* (pp. 336-342). Belek-Antalya, Turkey. DOI: http://doi.org/10.1109/eurmic.2003.1231611 (in English)
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)
52. Snipes, W., Nair, A. R., & Murphy-Hill, E. (2014). Experiences gamifying developer adoption of practices and tools. In *Companion Proceedings of the 36th International Conference on Software Engineering* (pp. 105-114). DOI: http://doi.org/10.1145/2591062.2591171 (in English)
53. Soetens, Q. D., & Demeyer, S. (2012). ChEOPSJ: Change-Based Test Optimization. In *2012 16th European Conference on Software Maintenance and Reengineering* (pp. 535-538). Szeged, Hungary. DOI: http://doi.org/10.1109/csmr.2012.70 (in English)
54. Steinert, B., Cassou, D., & Hirschfeld, R. (2012). CoExist: Overcoming Aversion to Change Preserving Immediate Access to Source Code and Run-time Information of Previous Development States. *ACM SIGPLAN Notices*, *48*(2), 107-118. DOI: http://doi.org/10.1145/2480360.2384591 (in English)
55. Yoon, Y., & Myers, B. A. (2011). Capturing and analyzing low-level events from the code editor. In *Proceedings of the 3rd ACM SIGPLAN workshop on Evaluation and usability of programming languages and tools* (pp. 25-30). DOI: http://doi.org/10.1145/2089155.2089163 (in English)
56. Yuan, X., Vega, P., Yu, H., & Li, Y. (2005). *A Personal Software Process Tool for Eclipse Environment*. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.121.1638&rep=rep1&type=pdf (in English)

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

doi: https://doi.org/10.15802/stp2021/242042