International Journal of Teknology Informatics and Engineering
e-ISSN :2961-8215;.p-ISSN :2961-9068
Volume 1 No. 3 , Desember - 2022

IMPACT OF EMBEDDED CLS ON EYE TRACKING REPLICATION

Nuris Dwi Setiawan
Universitas Sains dan Teknologi Komputer

Bagus Sudirman
Universitas Sains dan Teknologi Komputer

Sigit Umar Anggono
Universitas Sains dan Teknologi Komputer

Abstract. Using different programming languages when software advancement is a familiar method in current software advancement. Nevertheless, using various languages that can hinder developer capacity is not widely known. This research simulated an unplanned controlled study examining the adoption of various languages in the situation of a directory programming task. Participants in this study were given programming tasks written in Java and one of three SQL-like embedded languages. Simple "SQL" over authority, "Java" program only, and a more Java-like hybrid embedded language. Furthermore, to transcribe the responses to the online quest and the participators' "task" solutions, the participators' eye movements were also recorded with an eye tracker.

"Eye Tracker" or in this study call as "Eye-Trc" is the methodology of the study of software development that has developed nowadays and grants more in-depth info about how developers accomplish programming tasks. This Eye-Trc method is used as a data collection method in this study. Eye-Trc data was get by thirty-one participators (university background and Industrial Background) for different programming tasks. To analyze the impact of inter-group inconstant and professional experience and in-group "task" variables on the dependent variable Time in completion, this study used a mixed model ANOVA. The outcome of this study indicates that an important impact on productivity was not found, this is different from the initial research because of the language used.

However, the same effect was found from the participators' expertise in programming activity indicating that more competent programmers were easy to full fill "polyglot programming tasks" more efficiently. In addition, it was raised that participators viewed the specimen code with the same proportion (time) for bringing "task" reckless of skills or language alternative provided. dominant-stage exploration management also remains mostly consistent over the experiences or language alternatives. Overall, it can be concluded that the programming stage of the linguist doesn't have an important impact. The top-stage strategies that participators used came to be identical reckless of the language alternative presented to them. As a suggestion for future research, the impact of various characteristics of polyglot programming languages should be studied in depth for the conclusions reached to remain correct across various polyglot programming contexts.

Keywords: Eye Tracking, Programming Language, Polyglot Programming, Computer Language Switching.

INTRODUCTION

"97% of open-source projects use more than two computer programming languages, and, on average, there are five programming languages. used in open-source projects" (Tomassetti&Torchiano, (2014); Mayer&Bauer, (2015)). Moreover, "developers claim to 'know' 10 different computer languages in survey responses" (Meyerovich & Rabkin, (2013)). However, recent research observes that studying a new computer language has important and risky instead for expert developers. "Polyglot programming has several stages of language-switching inspect, including Project-stage, file-stage, and embedded-stages" (Meyerovich & Rabkin, (2013)). Project-stage switch insertion by its name uses "project-stage polyglot" programming projects that are written in several languages but the different projects have one language. Embedded language switching in this context point to "one language", the "embedded language", and the language embedded in another, namely the "host language". The study of this topic improved and general in current years, "Most studies have focused on the effects of polyglot programming on code quality" (Morisio et al., (2012); Tomasetti&Torchiano, (2014)). "While some research focuses on different cultures while taking several programming languages, there are impacts and effects of polyglot programming on developer behavior that are largely uninvestigated". Stefic et al. (2013) observed "the effect of using embedded language translation online questionnaire-based study".

Study Name	Study Description	Methods Used
Study A	Online-Only Study	Online Questionnaire
Study B	Eye-Tracking Replication	Online Questionnaire, Eye Tracking

Table 1 "Study A" duplicated in "Study B"

The new version of this research was published in this study and was conducted twice. Participators fulfilled 6 tasks adopting "API" with multiple stages of "embedded language switching" or "Study A". "Study A" only noted and investigated participators' solvent to and "tasks" their communication data in the online learning surrounding. Next, a replication study (Study B) from Study A is presented with another mechanism of data set, namely "eye tracking" or shorted "Eye-Trc" stuff in this research. The file that is set as "Study A" = set in "Study B", besides in "Study B", the Eye-Trc data set with all website data. Eye-Trc is a system for software engineering studies that have expanded in current years. " Eye movements have been shown to pick up on much more discrete information during tasks than during interactions data or study just think out loud " (Fritz e6t al., (2015)). "Eye-Trc has been used to gain insight into how developers read" (Stelovsky, (1990); Rodeghero&McMillan, (2015)), "review" (Matsumoto, (2006); Maletic, (2012); Tamm, (2015); Fritz et al., (2015)) and "summarizes the sources code" (D'Mello et al., (2014)). Eye movement makes it possible to catch glimpses between material on screens and shifts in attention that participators deliberately notice themselves. Computer specialist activity is learning various aspects of programming. "Studies range from "programming-language" features such as syntax" (Stefik&Siebert, (2013)) and "type systems" (Hoppe&Hanenberg, (2013)) API design (Stylos&Myers, (2008)) and error effects (Becker, (2016)) "to studies trying to investigate the cognitive processes involved" (Brechmann et al., (2017); Tamm et al., (2015)).

Eye-Trc in Understanding Program

"Much research is done in the software engineering domain focusing on the sub-field of programmatic understanding" (Brooks, (1983)). In the area of program understanding, have utilized Eye-Trc of how the computer specialist understands the program and has provided insight. The other reason about Eye-Trc continues to evolve is

that it can give so much explanation. Fritz, (2015) “Eye-Trc data is more fine-grained than typical interaction data and can provide insight into how programmers read code”. Tamm et al., (2015) conducted a study to see differences in the way a person's eyes look at reading code versus the way one's eye reads words, besides, a difference between expert programmers and novice programmers reading code is also compared. Tamn et al., (2015) and Fritz et al., (2015) “brought in fourteen novices and nine professional software engineers, they experimented with tracking participators as they read Java code, and finally found that beginners view code in the same linear way as reading 80% of the time, not only that, they also found that novices read code in a similar way to how they read words while experts use different ways when reading the code” (Tamm et al., (2015)).

Research conducted by D'Mello et al., (2014) “summarizes the code “task” on a large Java open-source system and found that developers tend to see the call at most compared to the method signature”. Eye-Trc was adopted to calculate and research in several conditions of software engineering. Some studies have also shown that developers use extraneous websites as well as “Stack Over” and aid in understanding programs using Eye-Trc as a methodology. Eye-Trc data set while software engineering tasks have still in analyzed using visualization. This visualization helps explore congested eyes track and be verifiable by significant data. The reading activities of beginner and non-beginner computer specialists asked to comment on awareness questions for a C++ program. Eye movements are analyzed by dividing the program into logical code chunks. The outcome demonstrates that participators read the smaller method, the slices that participators jaded too much on improving the stage, and the difference between beginners and non-beginners only came when the method had greater readability.

Polyglot Programming

In the scientific literature, the advantages and disadvantages of polyglot programming at the stage of the human factor have been poorly explored (Visser et al., (2000)). “Using more appropriate language for a “task” leads to better productivity and maintenance is much easier by reducing the number of lines of project code” (Fjeldberg, (2008)). Suggested that incompatibility between programming languages is a barrier with differences varying over the language pairs. In this study, a different method was used, and the results are complementary. “Previous studies have shown that polyglot programming is very widespread and widely used in software development” (Tomassetti & Torchiano, (2014)). Mayer et al., (2015) “the frequency of polyglot programming in industrial projects and how developers view projects using several programming languages by conducting a survey with industry participators” (Le et al., (2017)). Mayer et al., (2015) “developers reported certain languages as better suited for certain tasks, and using multiple languages enabled requirements projects to translate into code more easily”. Nevertheless, developers also feel used of some programming languages in software projects is problematic to understand project and system changes. Mayer et al., (2015) “most of the relationship between the two languages occurs between such general-purpose languages Java and domain-specific languages such as SQL”.

A familiar problem is that developers report over-language links this is a bug created when a change is made, hesitate in selecting qualifiers adopted in both languages out of despair of cracking the project, and difficulty in understanding the program. Newly recorded anti-arrangement for polyglot programming projects, developer documentation, and bug reports containing common keywords relevant: “polyglot programming”, trusted sites used by developers: “Stack Overflow”, “GitHub issues”, “Bugzilla”, “IBM Developers”, and “developers android” to find “polyglot programming” principles began

to be analyzed. “While studies focus on the effects of polyglot programming on code quality” (Morisio et al., (2012); Tomasetti&Torchiano, (2014)) “There is little literature dealing with the effect measurements of polyglot programming on programmer productivity”. “One recent study explored the impact of polyglot programming on developers” (Stefik&Siebert, (2013)). Stefik&Siebert, (2013) “conducted a pilot study using an online questionnaire consisting of six database programming tasks and dividing participators into three different language groups with different stages of language acquisition”. “While some researchers have proposed the use of Eye-Trc as a methodology for studying polyglot programming tasks” (Konopka, (2015)), it remains unexplored in experiments studies.

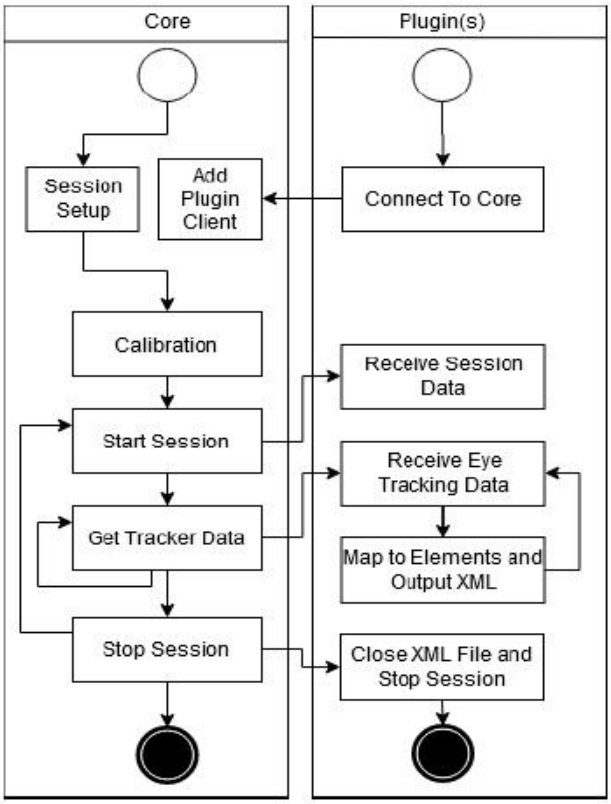

Figure 1: Core Architecture and iTrace Plugin

iTrace architecture

iTrace architecture is divided into core applications and various application plugins. The core application takes care of session management, interfacing with an eye tracker, and communicates with the plugin when its plugin communicates with the core application and maps views to lines other “IDE elements”. “When a plugin is connected to the core application, it is added to the plugin client list”. If the core utilities are started, a bundle will send to the different clients by session, and plugin data builds a suitable “XML file” and prepares to take stare files by the main utilization. When the plugin hooks up to the main utilization when the period is in running then this initial period bundle will send fastly.

Chrome Plugin Execution

There is some specific execution specialty of the chrome plugin that it differentiates from general plugin execution details, this specific execution took place due to the fact the Chrome plugin become constructed as a Chrome extension and did now no

longer have identical permissions for sources at the host machine. The first Chrome plugin's precise execution element that differs from preceding plugins is that it makes use of a WebSocket connection and iTrace-VisualStudio is used rather than the TCP socket connection utilized by iTrace-Eclipse. This is vital due to the fact Chrome extensions can not create sockets and as a substitute depends on WebSockets to hook up with the iTrace core. Additionally, Chrome extensions do now no longer permit local document access, so XML documents aren't constantly written whilst view information is acquired from the core. After the consultation ends, the consumer should manually click on a button to export the XML document. The document isn't stored withinside the identical listing because of the XML int document information as this is the simplest manner to shop the document withinside the unique shop document listing and save the document as a "Chrome Download".

Unlike other plugins, iTrace-Chrome will disconnect from Chrome after the session ends. When the session ends and the user exports his XML file, the connection between the Chrome plugin and the core app ends. If the user wants to record eye movements for additional tasks, the user will have to reconnect the plugin to the core application. To map gaze coordinates received from the core Chrome plugin application to site elements, a different mapping set is used for each element position. Use Javascript to get the DOM list item by eye coordinates and examine the combination of class, id, and tag name to determine the type of item being displayed. To know which lines will be displayed, each line of text in the usage environment section of the study is wrapped in a div so that each line can be individually selected. Finally, in the research environment used, the time used to synchronize the core application data with the plugin data was also written to a hidden div element when new data was processed, and every 5 seconds the website Logs this event time and records it along with the entry to end. Department.

Objection Coated

There have been some counterarguments regarding Chrome plugins. First, there is no way to enter the original file system. All file writing must be done in the browser's JavaScript engine, and the XML data must be written to the computer's file system using the file download API. Another challenge is that a generic Chrome plugin is not viable as it relies heavily on interfaces used in other studies. Additional logic should be added to the various sites suitable for the investigation to have the ability to characterize the view and map it to the definitionally important aspects. Perhaps there are popular Chrome plugins for Stack Overflow and website bug reporting, and GitHub, which follows the structure of the "Document Object Model", and dynamic content has logic to handle to map the appropriate required elements. there are a lot of them. Also, because the row and column information can change at any time due to window size changes, and because the DOM treats text nodes as one element, for large blocks of text there is no need to map row and column information to words. It is difficult to The row and column information should be obtained by adding an HTML wrapper around the text group with the row and column combination determined based on the font size of the text on the screen.

Lastly, manipulation and changing data be troublesome issues for all plugins. Data from the plugin can currently be adopted to investigate Eye-Trc data on non-static source code. Rows and columns of edited source code information can be adopted to get linguistic information about the source code being viewed. Editing is a complicated issue to deal with and account for in the Eye-Trc investigation. Presently, some solutions are being marked by the iTrace team but no solutions have yet been armed into any of the

plugins. To manipulate editing for this study, the study environment naturally saves text for solutions in each 5s to reduce this problem.

EXPERIMENT

Study A

Many details of the experiment setup remained consistent in "Study A" and "Study B". The same online location was adopted together by the same assignments. The essential diversity is the addition of Eye-Trc as a data collection system. Because of this further system, a mediator was appropriate to be new at the study to establish that authentical Eye-Trc data was gathered. "Study A" was aimed completely online without participator instruction, and "Study B" was aimed completely in person by a mediator current at the study.

Preliminary Design

This model is an iterative quantify the design in random participators that present and join into 1 of 3 empirical groups. In different groups is taken various language variants to complete the task. The same six tasks were presented to all participators insensitive to their group. The only variable is the language variant adopted to complete the task.

Table 2: Number of Participators and Distribution Between Groups

	Hybrid	String-Based	Object-Oriented
Students	5	5	6
Professional	4	6	5

Study Settings

Replication is fulfilled by adopting the original online platform that chooses for learning. This platform informs the participants of their rights and consent. Participators filled out a pre-questionnaire that was confidential and various participators were in 1 of the expertise groups. Questionnaires for additional information as well as the amount of programming expertise, the total programming work experience, and age were also provided. When the pre-questionnaire was completed participators were presented with a screen detailing the procedure for the remainder of the study. A moderator was served while the study to ensure that Eye-Trc data was correct and participators were being tracked properly. Before working on various tasks, the eye tracker is measured to establish high-quality data. Before the 1st assignment, participators had 5 minutes duration for reading sample code that serve as an example of their assignment and was written in their assigned language variant. This sample code is available for participators to read during the remainder of the study. The sample code is on the left side of the web page while the "solution area" of participators typing in their answers is on the right side under the time left to full fill the "task" as shown in Figures 2 and 3.

Figure 2: Study Environment: Code Example

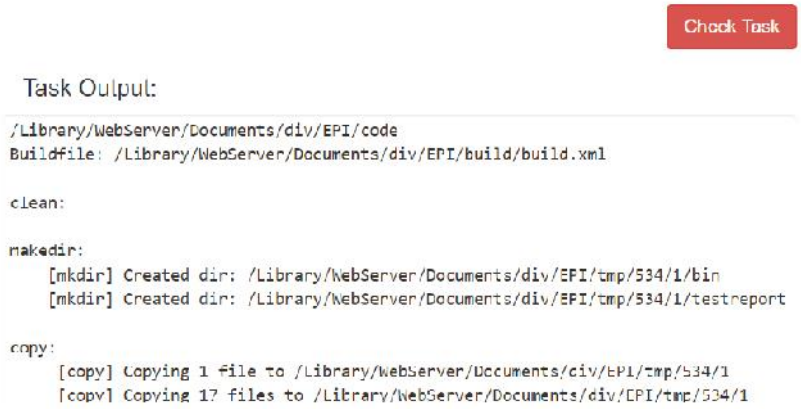

Figure 3: Study Environment: "task" Output

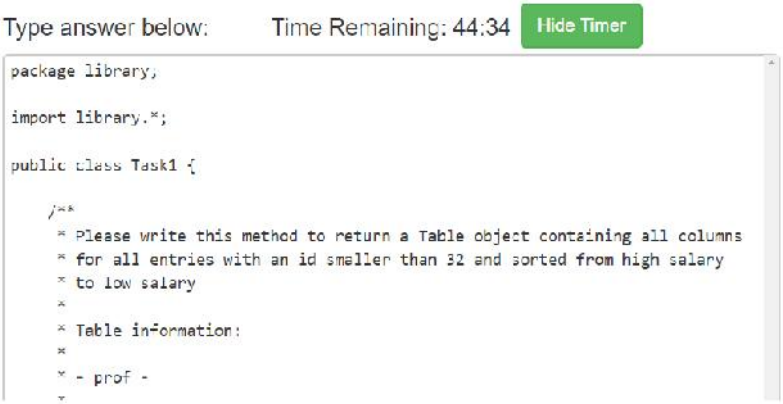

Figure 4 Study "situation, Solution Area, and Timer"

Task	Name	Description
Task 1	Simple Select	Select query with a single conditional and sort
Task 2	Moderate Select	Select query with a composite conditional
Task 3	Difficult Select	Select query with several composite conditionals
Task 4	Update	Update a single entry in an existing table
Task 5	Insert	Insert a single entry into an existing table
Task 6	Join Select	Select query requiring a join between two tables

Table 3 List of Tasks Used

Each “task” consists of one database task, if a participator is unable to full fill the “task” within forty-five minutes then the “task” will automatically come to end and the participator will move on to the next “task” after recalibration of the Eye-Trc. This “task” time limit was devised to avoid the analysis from catching up more than enough time and limited the maximum amount of time participators were promised to four and a half hours. During the “task” point, participators can check the accuracy of their answer with the remaining time before moving on to the next time using the p “check task” (button) below in the solution area. Their suggested solvent is then sent to the server where it is compiled along with the required additional classes and run against several unit tests. If a participator's solution passes the unit test, then the “task” output prints that the test was successful, and a pop-up is shown to the user prompting them to move on to the next task. If a participator's solution fails to compile or unit-tests, the “task” output will display a compilation error or unsuccessful test cases Participators can change and test their solvent as many times as they want until they complete the “task” or exceed the task's 45-minute time limit.

Mediation

Three different groups are built through different stages of language switching. This experimentally written API enables database queries. Designing different flavors of the API focuses on ideas for different querying approaches, requiring different amounts of language switching required to accomplish the task. Java code is employed at various levels of embedded SQL statements to interact with the database server with various API calls. The first group (Listing 1, Listing 4) uses a string approach that requires the API user to know the exact syntax of the SQL query they enter and does not provide support for type-checking the SQL query. The only error checking in this way is visible to the database, so a computer expert must detect query errors based on feedback from the database. Additionally, string-based queries must exactly match the SQL variant used by the database server. Nevertheless, any user with acceptable SQL action is provided with a flexible to allow them to take the entire SQL language.

Listing 1:” String-Based Design” Example

```
1 public Table query(Table table) throws Exception {
2     Query query = new Query();
3     query.Prepare("SELECT Field1, Field2 "
4         +"FROM table WHERE Field1 < 234 AND Field2 > 42 "
5         +"ORDER BY Field3 DESC");
6     Table result = table.Search(query);
7     return result;
8 }
```

Listing 2: “Object-Oriented Design” Examples

```
1 public Table query(Table table) throws Exception {
2   Query query = new Query();
3   query.AddField("Field1")
4     .AddField("Field2");
5   query.Filter(q.Where("Field1").LessThan(234).And("Field2").
      ↪ GreaterThan(42));
6   query.SortHighToLow("Field3");
7   Table result = table.Search(query);
8 }
```

The 2nd group adopted a methodical system which is becoming the API user to use several methods calls to construct queries. This variant also uses only one “programming language” and is confidential as a monoglot API, and it eradicates the demand to switch among the programming languages to write queries. This may impact productivity as it avoids switching costs. The third group adopted a combination method from the previous 2 language variants. The query-building process is detached into

various system calls, but in different stages of the query action adopting strings to create the equipment within that stage. This particular syntax falls between the usual language switches from “SQL” to “Java” by cause structure is nearest to the “host language”. The hue cycle of polyglots can check in this analysis, and also can be drawn as the hue cycle of language method decisions. In “string-based APIs”, SQL is precisely embedded into Java and there is no explicit network between languages in “Object-Oriented APIs”, a system in Java is built to complete SQL-like operations.

Listing 3: Hybrid Design Example

```
1 public Table query(Table table) throws Exception {
2     Query query = new Query();
3     query.AddFields("Field1, Field2");
4     query.Filter("Field1 < 234 AND Field2 > 42");
5     query.SortHighToLow("Field3");
6     Table result = charts.Search(query);
7     return result;
8 }
```

Variable

Some reliant variables are adopted in the cloning study. The 1st vulnerable variable is the time to accurately the solution that was analyzed in the natural study. If the participator is unsuccess to full fill the task, the correct completion time is set to the total time spent trying to full fill the “task” which will be 45 minutes due to the task's time limit. The second dependent variable is the fixation's total time spent in the “area of significance” (A-OI) on the website where they took the study. There are 6 dominant-stage A-OIs current in the different tasks: “Check “task” Button, Timer, “task” Info, Solution Area, Code Samples, and “task” Output”.

Randomization

Once participators enter their college years defined experience groups based on their responses. The distribution of participators in each action group amid these 3 experimental analyses was monitored in the research setting. If this distribution was unequal among previous participators in the present participator's expertise group, the participators were defined as 1 of the marginalized groups. Once there is an identical allocation of experimental treatments within the experience group, then all groups are free to be assigned until each experimental group is filled again. This was done so that the group distribution remained even.

Blazing

When “Study A” was conducted in a dual-unseeing setting, this cloning was performed in a single-blind setting. The participators don’t know which group they were assigned to, but a moderator was needed to ensure that Eye-Trc data was collected properly. This moderator does not deliberately give the group assigned to the participator, but the moderator can see the “task” given to the participator and concludes the group assigned to the participator. However, to limit bias, moderators were instructed not to disclose any information regarding assigned tasks or groups to participators. They only know the information given to them in the learning situation. During the requirement, participators were notified that they would be participating in a study related to the effects of programming languages on computer specialists but the exact nature of the study and the group to which they were assigned were not provided.

RESULT

Pre-Clarification

Since participators have to write their solutions, code snippets are reported to the server every 5s. Moreover, to the code snippet, the event times of the "iTrace-Chrome plugin" are stored at the same time to synchronize the Eye-Trc data with this snippet. Different gaze sample is associated with a code snippet using an event record. The code snippet identical to the 1st glance in preoccupation is adopted to assist different preoccupations in a code snippet. To determine the fixation, the IVT-fixation filter is adopted by a speed brink of three seconds and a maximum distance of 75 ms. The smaller gap is fully by continuous interposition between the 2 endpoints of the gap.

	Task 1	Task 2	Task 3	Task 4	Task 5
Task 2	1				
Task 3	1	0.295			
Task 4	0.295	0.0733	< 0.001		
Task 5	0.0126	0.0152	< 0.001	1	
Task 6	0.0733	0.411	< 0.001	1	1

Table 4: Paired "t-test results" for the impact of tasks on completion duration

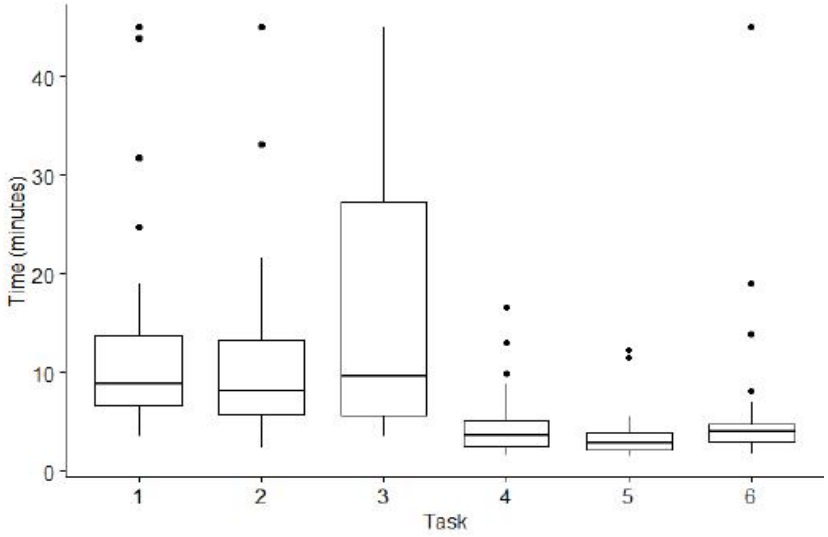

Figure 5 Completion Time For Each Task

Productivity Results

Several systems investigate the program distribution, in this appropriate situation, productivity is investigated by the duration of time, it takes to complete a task. Analyze the outcome for the 1st vulnerable variable, it is known that the moderate participator takes 550,81s to full fill the "task" by the common alteration of 614,93sand the time to fulfill the "task" ranges from 92,0s to 2702,0 s i.e. more than the time limit of 45 minutes. A combination system ANOVA was processed to analyze the effects between-group variables and expertise and within-group "task" variables on the vulnerable variable Time to Complete. Here it is found that the "task" has an important impact on completion time "($p < 0:001$), $F(5; 95) = 16:2555$". When trials are unbalanced, this difference in completion time by "task" could be due to a better understanding of the "task" type or "task" difficulty. To understand the various tasks, post-hoc analysis was used for paired t-tests with Bonferroni corrections. The results of these paired t-tests are shown in Table 4. This indicates that the three main items did not show significant differences between them. The last three tasks aren't much different either, but most of the three main tasks

took longer than the other last three tasks. This important difference was not found in the three comparisons. 'task' 1 was compared to 'task' 4 and 'task' 6 was compared to 'task' 1 and 'task' 2. These differences are illustrated in Figure 5. However, searching for the effect of variables between subject groups found the effect to be significant at completion (p = 0:2843); F(2; 19) = 1:3449 In the assigned group he was missing participants. Figure 6 shows that the hybrid group took longer on average, but this feature was analytically insignificant.

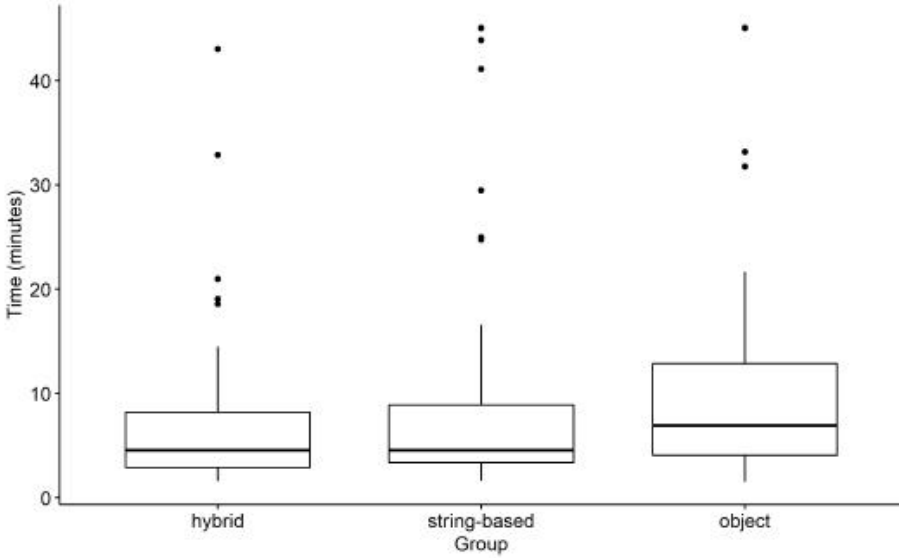

Figure 6 Completion Time For Each Group

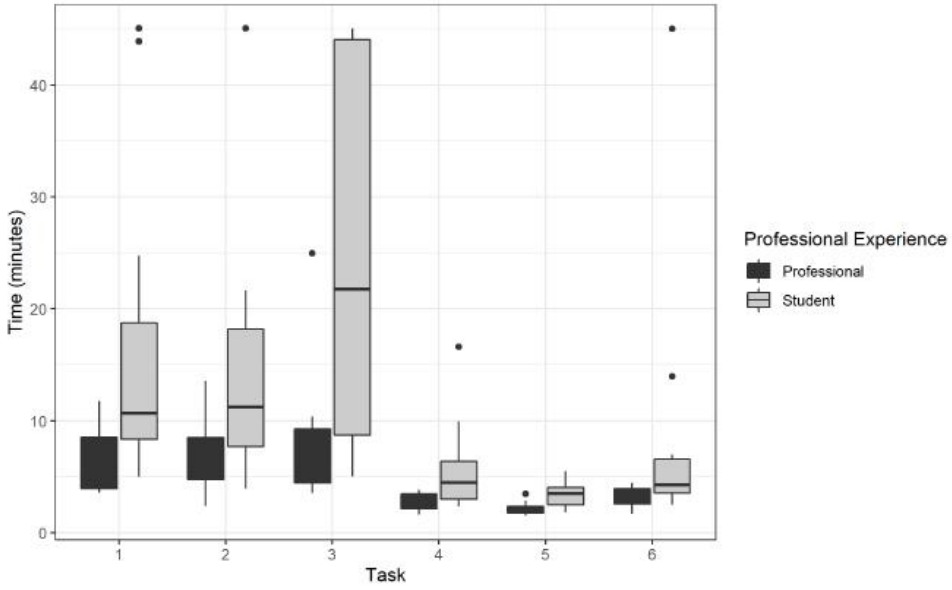

Figure 7 Completion Time for different task and Work Expertise Stages

Experience Results

Wandering the output of the ANOVA model further, the professional impact of timely expertise to completion can be seen. It was found that participators who described themselves as program professionals took an average of 4.93 minutes to complete it, and

others took an average of 12.1 minutes to complete different stacks. Moreover, the synergy of professional expertise and tasks does have an important impact “(p = 0:00435), F(5; 95) = 4:4964”. They show the moderate “task” faster, but no particular “task” helped experience to complete more quickly than the “task” itself or the participator's work experience described. Another metric that can be used to analyze participators is the degree to that they undeveloped their solutions with the analysis assignment button. The total duration participators used their solutions was normalized blending to the “task” time as tasks that took longer were more acceptable to check their solutions multiple times. A mixed ANOVA model was run to analyze the intergroup effects of group and professional experience variables and within-group assignment variables on the vulnerable variable Check Stage of Task.

An important difference based on the tasks that bring to participators F(5; 125) = 3:7094 (p = 0:00512) was found, this variation serve in Figure 8. “task” 1 is an elementary stage of “task” checking which is most likely. However, the professional experience was found to have no important impact on the examination “task” stage “F(1; 25) = 2:5274 (p = 0:1245)”. During participators' precise to the “hybrid API”, a variant arose to get somewhat decreased assignment check rates, this was a relatively small and insignificant difference. The interaction between the backgrounf and “task” “F(5; 125) = 0:3000 (0:8969)” backgrounf and group “F(2; 25) = 0:4863 (0:6206)” was not significant. Given this conclusion, it can be seen the experts analyze their assignments at approximately the same stage in a bringing “task” blind of the tasks they completed or the variation of the API they used.

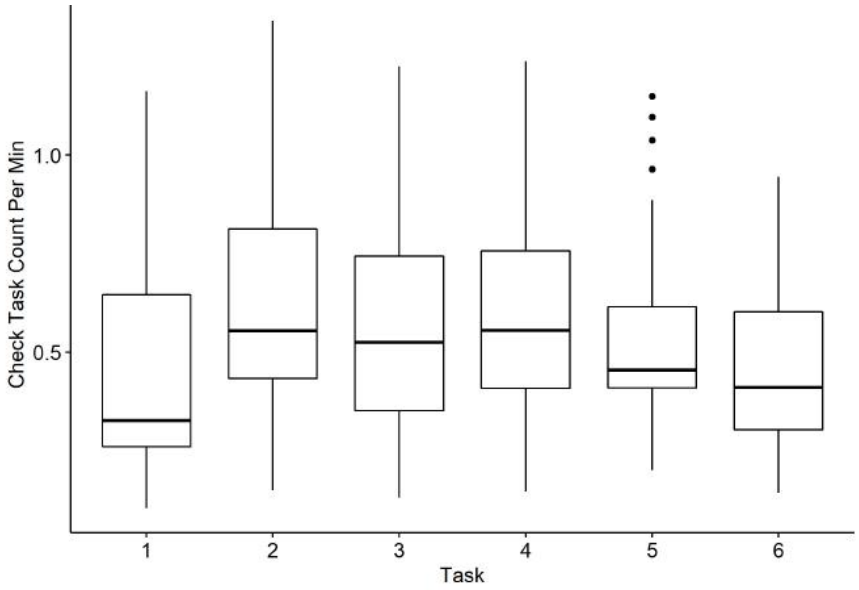

Figure 8 Checking “task” Stages by Task

Figure 9 Dominant degree A-OI: "Sample Code, Timer, Solution Area, Check Job Button, and Job Output Displayed"

Outcomes of View Behavior

Review Dominant degree "Areas of Significance" (AOI) on the Class website

There are 6-dominant-stage Areas of Significance, A-OIs, that are hunted while an Eye-Trc period. The allocation of the fixation's overall mean is shown in Table 5. This indicates that most "solution area" contains the code they are running which represents seventy-four point seventy-six percent of the "task" fixation duration. The sample code and "task" output are another A-OI that has a large number of case durations which contribute 18.30 percent and 5.32 percent of the complex time appropriately. The other A-OIs each account for less than one percent of the complex time most likely for the additional type of A-OIs. To further explore differences in the distribution of these top-stage A-OIs, 2 combine-methods ANOVAs were processed concerning the percentage of fixation duration and percentage of fixation amounts using within-subject variables from top-stage A-OI, and "task" along with between-subject variables from groups.

Website Area	Duration (Visits)
Check Task Button	0.55%(0.52%)
Sample Code	18.3%(18.66%)
Solution Area	74.76%(74.15%)
Task Info	0.97%(0.95%)
Task Output	5.32%(5.63%)
Timer	0.1%(0.1%)

Table 5 Allocation of Mean preoccupation period Over All Tasks in Dominant degree A-OI, allocation of Number of Fixations in Brackets

DISCUSSION AND IMPLICATIONS

Significance on Capacity

Different from "Study A", the assigned group participators had no important impact on computer specialist efficiency. The particular time or duration to carry out tasks grounded "API variant" the computer specialist used followed the same arrangement recognized in "Study A" indicating that the replication specimen volume is simple to analyze and that differences occur over "API groups", but fundamental various might exist. Several major differences were found between the tasks of the computer specialist in particular. This suggests that the ramification and complications of the "task" a

computer specialist is trying to entire will have a greater impact on the computer specialist's creativity than the stage of polyglot programming a computer specialist is using.

Significance on Experience

Differences were also found in the effect that professional computer specialists or casual trainees had on the time required to carry out a task. However, this disparate is not shocking considering that programmers with more experts are available to fulfill tasks faster than programmers with no more experience. That outcome comes around of outcome “Study A”. Whereas familiar experience appears to affect productivity in a polyglot programming task, this study could find no disparate in the capacity of expert, also not expert programmers based on the alternative of the “API”.

Significance of Behavioral Gaze

No significant differences were found in participators' top-stage gaze navigation behavior by the group to experience. This suggests the participator requested the “task” method negligible of the stage of language-switching they had to perform. Top-stage navigation behavior was found to change based on tasks with tasks at the opening time of the study hiring a greater stage of the excellent stage of evolution of the study. This might show that was relevant to the “task” performed deficient top-stage navigation or perhaps by dint of various in the tasks themselves. It was also found that the tasks differed significantly based on the participator's task. Nevertheless, since the tasks are provided in the same order. Significant differences were also found in indication-stage Transformation conduct based on expertise. This suggests that experts and neophytes take various methods for reading code while a polyglot programming task. It is not known even if that varies as a result of the programming of the polyglots themselves or the same variety that remain while assignments suggest one programming language.

CONCLUSION AND SUGGESTION

The outcome of a limitation of this research measuring the effect of polyglot data processing on computer specialist performance in this research is presented. This is the 1st research taking the Eye-Trc method in a “polyglot programming project”. The participators placed be 3 classes: a “string-based polyglot group”, “an object-oriented monoglot group”, and a “hybrid group”. The outcome here indicated the participators as experts that are available to fulfilling the polyglot data processing “task” expertly than other fellow students negligible of the group they were placed in. In contrast to authentic research, this answer can find an important impact on the group to which participators were placed on the usefulness of programmers supplementing projects. This outcome also indicated the navigational action of programmers continues widely and consistently related to the group in the participator's experience. For the next research, the impact of various class of polyglot programming languages should be studied more deeply to see whether the summary collected in this research hold across diverse polyglot data processing situation. Further studies in behavioral gaze also use to be carried out to regulate the exact brunt of assignment complications and programmers' navigational behavior.

BIBLIOGRAPHY

A. Stefik and S. Siebert. An empirical investigation into “programming-language” syntax. Trans. Comput. Educ., 13(4):19:1{19:40, Nov. 2013.

A. Van Deursen, P. Klint, and J. Visser. Domain-specific languages: An annotated bibliography. ACM Sigplan Notices, 35(6):26-36, 2000.

A. Vetro, F. Tomasetti, M. Torchiano, and M. Morisio. Language interaction and quality issues: an exploratory study. In Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement, pages 319-322. ACM, 2012.

BA Becker. A new metric to quantify repeated compiler errors for novice programmers. In Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education, pages 296{301. ACM, 2016.

B. Sharif, M. Falcone, and JI Maletic. An eye-tracking study on the role of scan time in finding source code defects. In Proceedings of the Symposium on Eye Tracking Research and Applications, ETRA '12 pages 381{384, New York, NY, USA, 2012. ACM.

F. Tomasetti and M. Torchiano. An empirical assessment of polyglot-ism in Git Hub. In Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, page 17. ACM, 2014.

H. Uwano, M. Nakamura, A. Monden, and K. ichi Matsumoto. Analyzing the individual performance of source code reviews using reviewers' eye movement. In Proceedings of the 2006 Symposium on Eye Tracking Research &Amp; Applications, ETRA '06, pages 133{140, New York, NY, USA, 2006. ACM.

H. -C. Fjeldberg. Polyglot programming. a business perspective. Ph.D. thesis, Master thesis, Norwegian University of Science and Technology, 2008.

J. Siegmund, N. Peitek, C. Parnin, S. Apel, J. Hofmeister, C. K astner, A. Begel, A. Bethmann, and A. Brechmann. Measuring neural efficiency of program comprehension. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering pages 140{150. ACM, 2017.

J. Stylos and BA Myers. The implications of method placement on API learnability. In Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering pages 105{112. ACM, 2008.

K. Kevic, BM Walters, TR Shaer, B. Sharif, DC Shepherd, and T. Fritz. Tracing software developers' eyes and interactions for changing tasks. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering pages 202{213. ACM, 2015.

LA Meyerovich and AS Rabkin. Empirical analysis of “programming-language” adoption. SIGPLAN Not., 48(10):1{18, Oct. 2013.

ME Crosby and J. Stelovsky. How do we read algorithms? a case study. Computer, 23(1):25{35, 1990.

M. Hoppe and S. Hanenberg. Do developers benefit from generic types?: an empirical comparison of generic and raw types in java. In Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications, OOPSLA 2013, part of SPLASH 2013, Indianapolis, IN, USA, October 26-31, 2013, pages 457{474. ACM, 2013.

M. Konopka. Combining eye tracking with navigation paths for identification of cross-language code dependencies. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, page 1057{1059, New York, NY, USA, 2015. Association for Computing Machinery.

P. Mayer and A. Bauer. An empirical analysis of the utilization of multiple programming languages in open source projects. In Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering, page 4. ACM, 2015.

P. Mayer, M. Kirsch, and MA Le. On multi-language software development, cross-language links and accompanying tools: a survey of professional software developers. Journal of Software Engineering Research and Development, 5(1):1, 2017.

P. Rodeghero and C. McMillan. An empirical study on the patterns of eye movement during summarization tasks. In 2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), volume 00, pages 1{10, Oct. 2015.

P. Rodeghero, C. McMillan, PW McBurney, N. Bosch, and S. D'Mello. Improving automated source code summarization via an eye-tracking study of programmers. In Proceedings of the 36th international conference on Software engineering, pages 390{401, 2014.

RE Brooks. Towards a theory of the comprehension of computer programs. Intl J. of Man-Machine Studies, 18(6):543{554, 1983.

T. Busjahn, R. Bednarik, A. Begel, M. Crosby, JH Paterson, C. Schulte, B. Sharif, and S. Tamm. Eye movements in code reading: Relaxing the linear order. In 2015 IEEE 23rd International Conference on Program Comprehension, pages 255{265, May 2015.