2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)

Exploring Tools and Strategies Used During Regular Expression Composition Tasks

Gina R. Bai*, Brian Clee*, Nischal Shrestha*, Carl Chapman†, Cimone Wright†, Kathryn T. Stolee*
* Department of Computer Science, North Carolina State University, Raleigh, NC, USA
† Department of Computer Science, Iowa State University, Ames, IA, USA
{rbai2, nshrest, ktstolee}@ncsu.edu, briancleeisme@gmail.com, carlallenchapman@gmail.com, clwright@iastate.edu

***Abstract*—Regular expressions are frequently found in programming projects. Studies have found that developers can accurately determine whether a string matches a regular expression. However, we still do not know the challenges associated with composing regular expressions.**

We conduct an exploratory case study to reveal the tools and strategies developers use during regular expression composition. In this study, 29 students are tasked with composing regular expressions that pass unit tests illustrating the intended behavior. The tasks are in Java and the Eclipse IDE was set up with JUnit tests. Participants had one hour to work and could use any Eclipse tools, web search, or web-based tools they desired. Screen-capture software recorded all interactions with browsers and the IDE. We analyzed the videos quantitatively by transcribing logs and extracting personas. Our results show that participants were 30% successful (28 of 94 attempts) at achieving a 100% pass rate on the unit tests. When participants used tools frequently, as in the case of the *novice tester* and the *knowledgeable tester* personas, or when they guess at a solution prior to searching, they are more likely to pass all the unit tests. We also found that compile errors often arise when participants searched for a result and copy/pasted the regular expression from another language into their Java files. These results point to future research into making regular expression composition easier for programmers, such as integrating visualization into the IDE to reduce context switching or providing language migration support when reusing regular expressions written in another language to reduce compile errors.

***Index Terms*—Exploratory study, regular expressions, problem solving strategies, personas**

I. Introduction

Regular expressions are powerful programming tools and pervasively applied in text editors [1], search engines [2], and network security [3]. However, regular expressions are hard to read, write, maintain, and understand [4]–[6], and developers often leave their regular expressions under-tested [4], [7].

Various tools aim to help programmers with solving regular expression-related tasks. For example, some web-based regular expression testers (web tools) visualize regular expression strings for users [8], [9] and some websites explain and highlight the matching behavior in a debugging environment [10], [11]. Researchers have created educational games to encourage programmers to gain better understanding and experience in interpreting regular expressions in an enjoyable environment [12]–[15]. All of these efforts reflect a need for better support for regular expression composition.

However, regular expressions are largely under-studied in the hands of users. Researchers have surveyed developers about regular expression usage [4] and measured how well people can determine if a regular expression matches a string [5]. Yet, we still do not know *what tools and strategies* they use during regular expression composition tasks.

We ran an exploratory study during which 29 participants composed regular expressions to pass JUnit tests. This study provided 20 regular expression tasks with accompanying tests and participants were instructed to complete as many tasks as possible in the time allotted, using a prescribed (randomly generated) task order. After analyzing survey responses from and screen-capture videos generated by the participants in an hour-long lab activity, we are able to reveal their overall performance on the regular expression tasks and categorize their behaviors during composition. Our findings include:

- Visualization of regular expressions (i.e. from web tools) helps developers pass more tests in the tasks (Section V-B).
- Participants who consulted official documentation and tutorials for regular expressions are more likely to pass more tests in the regular expression tasks than those who consulted Q&A websites (e.g., Stack Overflow) (Section V-C).
- Participants who first tried to compose a regular expression solution instead of first using web search to find a solution are more likely to pass all the tests (Section V-D).
- When participants adapted solutions from other languages, 36.3% copied and pasted contents from websites (e.g., Q&A sites and web tools) to Eclipse and subsequently modified the regular expression syntax to correct compile errors (Section V-E).
- The most frequent personas were *intermediates*, representing participants who showed insufficient prior knowledge on regular expressions and little growth or success over the course of the experiment (Section VI).

As this is the first observational study of regular expression composition, we identify several avenues for future work. These include integrating better support into the IDE to reduce context switching between applications and providing support for migrating regular expressions between languages to reduce compile errors.

The rest of the paper is organized as follows. Section II presents the research questions, followed by the study design to address the research questions in Section III. The analysis

978-1-7281-1519-1/19/$31.00 ©2019 IEEE
DOI 10.1109/ICPC.2019.00039

is presented in Section IV, followed by results in Section V and Section VI. A discussion identifies implications and areas of future work in Section VII, followed by related work in Section IX and a conclusion in Section X.

II. Research Questions

The goal of our research is to understand the relationship between a participant successfully composing a regular expression and the strategies used or resources consulted. This exploratory study uses screen-capture software to observe how participants compose regular expressions to pass a set of test cases. This allows us to perform quantitative analysis over logs extracted from the videos and study the participants' behaviors.

We explore the following research questions:

- **RQ1:** *What tools and strategies do developers employ while solving regular expression tasks in the Eclipse IDE?* We analyzed the following dimensions that emerged from the data: tools (Eclipse built-in debugger, web tools); different sources of information (tutorials, API doc, Q&A sites); and strategies for composition, such as copy & paste or direct creation.
- **RQ2:** *Which personas emerge as representative of the task performance exhibited by the developers?* To categorize the personas, we used quantitative information. Quantitative information included: the average first time pass rate (the number of passed tests/the number of total tests in one task) per task, average improved pass rate per task, and the average number of test runs per task.

III. Study Design

This study was run in a lab environment. Participants were given a series of tasks to perform in the Eclipse IDE. This section details the tasks, procedures, participants, and data collected.

A. Tasks

There were 20 tasks available for participants to complete.[1] These tasks were selected for diversity of questions and to cover the most common language features of regular expressions [4]. Table I shows the complete list of tasks (*Task*), the number of JUnit tests in each task (*#JUnit*) and their sample regular expression solutions (*Sample Regular Expression Solutions*), including escaped slashes. The examples in Figure 1 and Figure 3 map to tasks `ValidEmail` and `NoVowelsWord` respectively.

A task includes source code with a blank regular expression, test cases to demonstrate expected behavior, and a textual description for the expected behavior. Figure 1 contains code for the `ValidEmail` task in our study that uses the `Pattern.matches()` function to validate an email address input string using a regular expression composed by a participant. Participants were expected to fill in the blank

[1]Artifacts: https://github.com/softwarekitty/regexCompositionStudy

```
1 /** A line of text will contain at most one newline
2  * and only then at the end of the string (this
3  * input will not have multiple lines). This
4  * function should take one line of text and verify
5  * that the entire string is composed of one valid
6  * email. Extra characters like whitespace before
7  * or after, or anything that would invalidate the
8  * email are not allowed (except newline at the
9  * end).
10 */
11 public class ValidEmail {
12   public boolean isValidEmail(String line) {
13   // TODO compose a regex to complete the challenge
14     String regex = "";
15     return Pattern.matches(regex, line);
16   }
17 }
```

Fig. 1. Code and Description for Task ValidEmail and One Sample Associated JUnit Test

```
1  public class ValidEmailTest {
2    private static ValidEmail validEmail = null;
3    @BeforeClass
4    public static void setup() {
5      validEmail = new ValidEmail();
6    }
7    @Test
8    public void testIsValidEmail_1() {
9      //a typical email
10     String anyLine =
           "###/+-?^_`{|}~$$$***@weird.do";
11     boolean correctAnswer = true;
12     boolean compositionAnswer =
           validEmail.isValidEmail(anyLine);
13     assertTrue(compositionAnswer == correctAnswer);
14   }
15   // ... More tests, eight test cases in total
```

Fig. 2. Code and Description for Task ValidEmail and One Sample Associated JUnit Test

string on line 14 such that the tests for the class (e.g, one sample test shown in Figure 2), will pass.

The other test cases check three other valid email addresses: `"name@domain.com"`, `"1.2.3.4@crazy.domain.axes"` and `"!@B.gone"`. Four false test cases are covered as well: a twitter handle, a website url, a single word, and a sentence.

Another sample task `NoVowelsWord` is introduced in Figure 3. In this task, participants were expected to fill in the blank string on line 12 to pass the tests for the class. The `Matcher.find()` function searches for occurrences of the regular expressions in the text provided in the JUnit tests (e.g, one sample test shown in Figure 4).

Of the tasks, five used the `Pattern.matches()` function to validate the entire content of a string against a regular expression (*Validation* in *Type* column of Table I), and 15 used `Matcher.find()` to examine the strings against a given pattern and extract pertinent information from a string subject to a regular expression (*Extraction* in *Type* column of Table I).

We point out here that there are many possible correct solutions for any of the tasks. For example, both regular expressions `.+@.+` and `\\S+@\\S+.\\w` are candidates

TABLE I
REGULAR EXPRESSION PROBLEMS AND SAMPLE SOLUTIONS

Type	Task	#JUnit	Sample Regular Expression Solution
Extraction	AlternatingParity	7	`(?<!\\d)(([02468][13579])+[02468]?|([13579][02468])+[13579]?)(?!\\d)`
Extraction	GoogleKeywords	5	`.*www.google.com.*q=(.*)`
Extraction	OnMinuteEvents	5	`\\d{4}-\\d{2}-\\d{2}T\\d{2}:\\d{2}:00Z\\s+(.*)`
Extraction	PossessedPossessions	4	`[a-zA-Z]{2,}'s([a-zA-Z]{2,})`
Extraction	ReceiptScanner	4	`\\A\\s*(.+?)\\s*[\n]|\\\$?(\\d+(\\.\\d{2})?).*\\Z`
Extraction	RepeatedWords	5	`(\\w+),?\\1\\b\\`
Extraction	TSVParser	4	`([^\t\n]+)\t[^\t\n]+\t([^\t\n]+)`
Extraction	VerbPortion	5	`\\b([a-zA-Z]{2,})ing\\b`
Extraction	JavaIntDeclaration	5	`\\s*int\\s+\\w+\\s*=\\s*(\\d+);`
Extraction	LastDuplicateByte	6	`^.*([0-9ABCDEF]{2}).*?\\1.*$`
Extraction	ShortestDNA	5	`(AT(((?!AT).)*?)GC)`
Extraction	TrimWhitespace	6	`^\\s*(.*?)\\s*$`
Extraction	HasSoyIngredient	14	`\\b(Edamame|Kinnoko|Kyodofu|Miso|Natto|Okara|Shoyu|Soy(a)?| soybean(s)?|Tamari|Tempeh|Teriyaki|vegetable protein|Tofu|Yakidofu| Yuba|TSF|TSP|TVP)\\b`
Extraction	NoGremlins	6	`\\A[^\\f\\r\b]*\\z`
Extraction	NoVowelsWord	5	`\\b[bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ]{2,}\\b`
Validation	OutlineFormat	10	`^\\s*([a-z]|[A-Z]|i+|\\d+)\\.\\s+.*$`
Validation	ReverseSentences	5	`\\.(\\S+ )*\\S*[A-Z](\\s+\\.(\\S+ )*\\S*[A-Z])*`
Validation	SpacedWords	11	`^([a-zA-Z0-9]+\\s+)+([a-zA-Z0-9]+)?$\|^(\\s+[a-zA-Z0-9]+)+(\\s+)?$`
Validation	ValidEmail	8	`[A-Z0-9a-z.!#$%&'*+-/=?^_`{|}~]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,4}`
Validation	ValidPhoneNumber	13	`^(\\([0-9]{3}\\)|[0-9]{3})[ -]?[0-9]{3}[- ]?[0-9]{4}$`

```java
/** Returns true if any alphanumeric word in the
 * text contains no vowels. So the strange sentence:
 * "I have ctmpts training to go to!" should return
 * true. Input does not need to be a sentence, and
 * words are separated by whitespace as usual,
 * ignoring punctuation Words are composed of 2 or
 * more lowercase or uppercase letters.
 */
public class NoVowelsWord {
  public boolean hasNoVowelsWord(String content) {
    // TODO compose a regex to complete the
        challenge
    String regex = "";
    Pattern pattern = Pattern.compile(regex,
        Pattern.CASE_INSENSITIVE);
    Matcher matcher = pattern.matcher(content);
    return matcher.find();
  }
}
```

Fig. 3. Code and Description for Task NoVowelsWord and One Sample Associated JUnit Test

```java
public class NoVowelsWordTest {
  private static NoVowelsWord noVowelsWord = null;
  @BeforeClass
  public static void setup() {
    noVowelsWord = new NoVowelsWord();
  }
  @Test
  public void testHasNoVowelsWord_1() {
    //the word 'CTMPTS' has no vowels
    String anyLine = "April and Ron are taking
        CTMPTS training";
    boolean correctAnswer = true;
    boolean compositionAnswer =
        noVowelsWord.hasNoVowelsWord(anyLine);
    assertTrue(compositionAnswer==correctAnswer);
  }
  // ... More tests, five test cases in total
```

Fig. 4. Code and Description for Task NoVowelsWord and One Sample Associated JUnit Test

for correct solution to the `ValidEmail` task. The only requirement for successful completion is that all the test cases pass. Additionally, these sample solutions are smaller than the regular expressions found in open source projects, where size is measured by the number of nodes and edges in their DFA representations from RE2 (a tool that backtracks regular expressions) [16]. The average size of the sample solutions is 14 nodes and 37.5 edges, whereas the average size of 15,096 regular expressions from GitHub is 28 nodes and 75 edges [7]. In terms of complexity, measured as the ratio of edges to nodes, the two data sets are equivalent. Using smaller regular expressions is a design choice since participants have a short time to understand the desired behavior of the regular expression tasks and work on them.

B. Procedure

The study was conducted in a lab setting over two sessions, one hour each. Participants attended one lab session only. All desktop computers in the lab had screen capturing software and browsers installed prior to the study. Eclipse was preloaded with the study tasks. Participants were asked to use the lab computers only; no personal computers were allowed.

At the beginning of the study, participant were asked to complete a paper-based questionnaire which focused on educational classification and programming expertise as shown in Figure 5.

For the study tasks, each participant received a list of tasks from Table I defining the order in which they were to attempt the tasks. The order was randomly generated to avoid potential learning effects. Participants were given one hour to complete as many tasks as they could, and were told they were not

1) **What year are you in school (closest match)?**
   a) Freshmen b) Sophomore c) Junior d) Senior
   e) Graduate Student
2) **How would you rate your Java programming experience?**
   a) Novice b) Intermediate c) expert
3) **What is your experience with regular expressions?**
   a) I have never heard of them
   b) I have heard of them, but have no experience
   c) I have used them once or twice
   d) I use them regularly
   e) I am an expert
4) **What is your experience with JUnit tests?**
   a) I have never heard of them
   b) I have heard of them, but have no experience
   c) I have used them once or twice
   d) I use them regularly
   e) I am an expert
5) **How frequently do you use web searches while programming?**
   a) I have never heard of them
   b) I have heard of them, but have no experience
   c) I have used them once or twice
   d) I use them regularly
   e) I am an expert
6) **How many years have you been programming? __**
7) **How many years have you been programming in Java? __**

Fig. 5. Survey Questions

expected to finish all 20 tasks in the allotted time. They were allowed to use web resources and any default Eclipse utilities to assist in completing regular expression tasks. During the study, screens were recorded to facilitate analysis.

C. Participants

In total, 34 participants were recruited through e-mails sent to students who had passed the introductory object-oriented programming course at Iowa State University. Every participant was required to fill out a survey before they started to solve the tasks. The participants included 25 undergraduate students and four graduate students. There were five additional students who had issues with screen-capture, so they were omitted from all analyses. We compensate every participant $20 in cash upon completion of the study.

Among the 29 participants from whom we collected data, there was an average of 4.16 years of programming experience, and an average of 3.26 years of Java experience. The survey results found that a majority of participants (76%) considered themselves as having intermediate Java programming knowledge though 20 participants had little or no experience with regular expressions. Participants either had no experience with JUnit tests at all (65.5%) or used them regularly (34.5%). All participants made use of web searches while programming.

D. Data Description

After removing the five videos that had screen capture issues, we ended up with 29 videos for analysis, 45-60 minutes each. This represents over 24 total hours of video, 94 total attempts at solving the tasks by the 29 participants, 1,097 total web searches, 3,401 websites visited in the browser, and 230 copy/paste interactions between the browser and the IDE.

IV. Analysis

To enable a quantitative analysis of the participants' problem solving processes, two authors designed a log to capture important events during the sessions. Next, these two authors transcribed the videos into the log format separately and then merged their logs. No inter-rater reliability was considered in this process.

A. Logged Events

Trigger events are used to identify when a log entry should be made. A trigger event is an on-screen event that prompts the transcriber to log the action. Once an event trigger is seen, the event is logged. The logs contain columns to describe the event. When a trigger event occurred, a row was added to the log and its associated column(s) were logged. A detailed description of the columns follows:

Time: Current time in the video when an event occurred.

Task: The name of the task in Eclipse.

Search: String from an online search query.

Website Visited: Current website visited.

Regex String: Regular expression in Eclipse or web tool.

Copy Paste: Type of copy-pasted item: test string or regular expression.

Debugger: True if using the Eclipse Debugger.

Eclipse: True if the event occurred while using Eclipse.

Web Tool: True if the event occurred while using a web tool.

JUnit Tests: True if the participant visited the JUnit tests.

Test Passed: The specific test(s) that passed.

Pass Rate: The pass rate of the executed JUnit tests, calculated as: $\frac{\#\ Passed\ JUnit\ Tests}{\#\ Total\ JUnit\ Tests}$

We consider the following as trigger events for the log entries:

- Application switch (logged: *Time*, *Eclipse*, *Web Tool*)
- Switch to browser (logged: *Time*, *Website Visited*)
- Search in browser (logged: *Time*, *Search*)
- Access website in browser (logged: *Time*, *Website Visited*)
- Access debugger or development environment in Eclipse (logged: *Time*, *Debugger*)
- Copy regular expressions or strings (logged: *Time*, *Copy Paste*, *Website Visited* or *Eclipse*)
- Paste regular expressions or strings (logged: *Time*, *Copy Paste*, *Website Visited* or *Eclipse*)
- Compose/Edit regular expressions (logged: *Time*, *Regex String*)
- Run JUnit tests (logged: *Time*, *Test Passed*, *Pass Rate*)
- Switch task in Eclipse (logged: *Time*, *Task*)
- Switch between JUnit tests and code in Eclipse IDE (logged: *Time*, *JUnit Tests*)

For example, consider a participant who 1) opens task *ValidEmail* and starts to work on it at 00:10:04, 2) switches to Chrome browser and 3) searches "valid email in regex Java"

TABLE II
PARTICIPANT'S BEHAVIORAL METRICS FOR A SUBSET OF PARTICIPANTS (8/29)

ParID	AFPR	AIPR	ATR	Persona Vector
krl	0% (0)	20% (L)	13.5 (H)	<0, 0/L, H>
b4r	0% (0)	50% (H)	68 (H)	<0, H, H>
dm8	15% (L)	18% (L)	5.3 (L)	<L, 0/L, L>
vrh	17% (L)	36% (H)	9.5 (L)	<L, H, L>
q3d	45% (H)	23% (L)	10.5 (H)	<H, 0/L, H>
clx	36% (H)	45% (H)	17 (H)	<H, H, H>
l6o	39% (H)	10% (L)	4.4 (L)	<H, 0/L, L>
nxb	39% (H)	33% (H)	3.7 (L)	<H, H, L>
Average:	29%	31%	10.1	—

at 00:10:34, 4) clicks on and visits a Stack Overflow result at 00:10:39, 5) copies an existing sample regular expression, `.+@.+`, at 00:12:15, 6) switches to Eclipse at 00:12:16, and 7) pastes into Eclipse at 00:12:17, then 8) runs the tests. In this scenario, the eight trigger events are: 1) switch task in Eclipse, 2) switch to browser, 3) search in browser, 4) access website in browser, 5) copy regular expressions or strings, 6) application switch, 7) paste regular expressions or strings, 8) run JUnit tests. The total columns being logged are: *Time*, *Eclipse*, *Search*, *Website Visited*, *Copy Paste*, *Regex String*, *Test Passed*, *Pass Rate*.

In the end, there were 11,644 total rows logged among all 29 participants.

B. Personas

Our process for determining personas was driven by quantitative observations, and was motivated by a work of Pruitt and Grudin, which suggests that personas provide an effective way to communicate qualitative and quantitative data [17]. We therefore use personas to synthesize and communicate quantitative observations about study participant behavior.

1) Metrics: To assist persona identification, we consider the following quantitative metrics:

- *Average first time pass rate per task (AFPR)*: the pass rate a participant produced in first JUnit test run in one task, on average. This reflects a participant's prior/initial knowledge of regular expressions. A high pass rate indicates a *knowledgeable* participant; a low pass rate indicates an *intermediate* participant; and a 0% pass rate indicates a *novice* participant.
- *Average improved pass rate per task (AIPR)*: the percentage of pass rate improved in one task, on average. This reflects a participant's learning progress during regular expression task composition.
- *Average number of test runs per task (ATR)*: the times a participant tested one task, on average. This reflects a participant's testing behavior. A high number of test run times indicates a *tester*.

2) Persona Vector Identification: We adopt the persona identification process used in the work of Dubey, et al. [18]. The first step of persona identification process is to calculate all the metrics discussed in Section IV-B1. Next, we classify the value of each metric as low (L) or high (H) against the average value among all participants, we also label 0 (0) if the average value is zero. Then we build persona vector to assist creation of personas. Table II shows the metrics values (*AFPR*, *AIPR*, *ATR*) and the average value of each metric among all participants, as well as the eight identified persona vectors (*Persona Vector*), each with one sample participant provided (*ParID*).

TABLE III
RANKING OF REGULAR EXPRESSION TASKS BASED ON AVERAGE PASS RATE

Type	Task	Avg %	#Attempt	#100%
Validation	OutlineFormat	100%	1	1
Extraction	NoVolwelsWord	87%	3	2
Validation	ValidEmail	86%	8	5
Extraction	TrimWhitespace	84%	2	1
Validation	ValidPhoneNumber	81%	6	3
Validation	ReverseSentences	70%	4	2
Validation	SpacedWords	67%	7	2
Extraction	PossessedPossessions	63%	2	1
Extraction	NoGremlins	62%	4	0
Extraction	ShortestDNA	60%	3	1
Extraction	LastDuplicateByte	54%	4	0
Extraction	JavaIntDeclaration	53%	8	2
Extraction	VerbPortion	52%	5	1
Extraction	RepeatedWords	47%	6	2
Extraction	GoogleKeywords	42%	12	5
Extraction	TSVParser	42%	3	0
Extraction	AlternatingParity	32%	4	0
Extraction	HasSoyIngredient	32%	2	0
Extraction	ReceiptScanner	29%	7	0
Extraction	OnMinuteEvents	13%	3	0
Total		56%	94	28

V. RESULTS - RQ1

We describe the results in terms of *attempts*, where an *attempt* is a pairing of participant and task during which the participant ran the test cases. In total, the 29 participants viewed 121 tasks, but only ran the tests for 94, yielding 94 attempts for analysis.

A. Overall Correctness

Overall, participants ran tests for 94 attempts. Of these, 28 succeed with 100% of the unit tests passing, and the remaining 66 were abandoned with lower pass rates, either because the participants ran out of time (21) or switched tasks (45). The overall average pass rate across all 94 attempts was 56%. Of the 29 participants, 14 were successful in at least one attempt.

Table III shows the average pass rate (*Avg %*) on the JUnit tests on the attempts (*#Attempt*) on each task (*Task*) with its type listed in column *Type*. Column *#100%* shows the number of attempts that passed all JUnit tests on this task. Pass rate for an attempt is calculated by the maximum number of passed JUnit tests during the entire attempt over the number of total JUnit tests for the task. For example, if a participant passed 11/13 JUnit tests in `ValidPhoneNumber` at some point during the attempt, then the pass rate is 85% for this attempt. Both Table III and Table IV are sorted in descending order by average pass rate.

From Table III, we find that eight participants attempted the `ValidEmail` task and five of them successfully passed all JUnit tests, making this the third most passed task with an average pass rate of 86%.

Table IV categorizes the tasks (*Type*) and shows the number of tasks in each category (*#Tasks*), the average pass rate of attempts on these tasks (*Avg %*), the total number attempts on these tasks (*#Attempt*) as well as the number of attempts that achieved 100% pass rate (*#100%*). It also demonstrates that participants performed best in *validation* tasks, as all *validation* tasks were solved with pass rate of 78% on average, and 46% of the successful attempts were in *validation* type.

> Of 94 attempts, 28 succeed with 100% pass rate. *Validation* tasks represented 25% of the programming tasks (5/20), but represented 46% (13/28) of the attempts that achieved a 100% pass rate on the test cases.

B. Tools

During the study, participants were allowed to consult any resources at their disposal within the IDE and the browser, including web tools.

Of the 28 attempts that passed all JUnit tests, ten attempts involved web tools. Nine participants who actively interacted with web tools achieved a pass rate of 68% on average. They spent 13:05 minutes on average on web tools, which made up of 26.11% of their recorded sessions. The average for context switching was 11.89 times, as participants had to check the task specifications and test the regular expression strings in Eclipse IDE. Seven participants passed more test cases with the help of web tools, and six among them continued using web tools for remaining tasks while the other one switched to web tools near the end of the study and attempted no more tasks. All the web tools that participants used support visualization to a certain degree. Such visualizations can convert regular expression strings to graphical states for users [8], [9], and some can highlight the matching results for users in debugging environments [10], [11].

Six participants used the built-in debugger in Eclipse IDE on seven total attempts. These attempts achieved a 48% pass rate on average. Most participants who used it in one attempt did not return to it in subsequent attempts; only one participant accessed it twice.

> Participants who consulted web tools to visualize regular expression behavior passed more tests than those who did not consult web tools (67.6% vs. 54.6% in terms of pass rate).

C. Sources of Information

For information sources online, we classify the websites into two categories: Q&A sites (e.g., Stack Overflow, online forums) and D&T sites (official documentation and official/third-party tutorials), similar to prior work [19]. After consulting an online source, we looked at the percentage of improvement in the passing test cases. For example, participant #17 was working on the `ValidEmail` task and was passing 4/8 tests. This participant did a search for, "*regular expression java valid email*" and clicked on a result for *http://www.mkyong.com*. After looking up information in this website, the participant returned to the IDE and modified the regular expression directly. The modified regular expression achieved 6/8 passing tests, resulting in a 25% improvement in pass rate.

TABLE IV
TYPE OF TASKS AND AVERAGE PASS RATES

Type	#Tasks	Avg %	#Attempt	#100%
Validation	5	78%	26	13
Extraction	15	48%	68	15
Total	20	56%	94	28

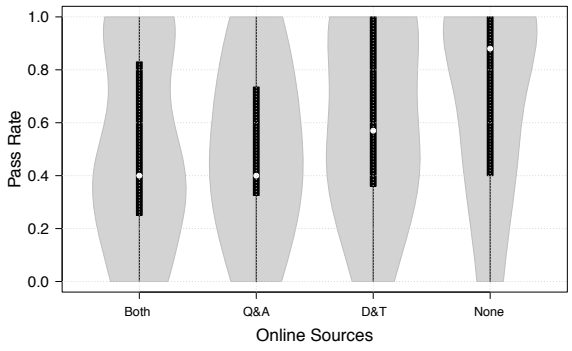
TABLE V
ONLINE SOURCES & AVERAGE IMPROVEMENT IN PASS RATE & AVERAGE PASS RATE

Online Sources	avgImp	avgPass	#Attempt
Q&A sites only	24%	50%	7
D&T sites only	35%	62%	13
Both Q&A and D&T sites	31%	51%	57
None	29%	70%	17
Total			94

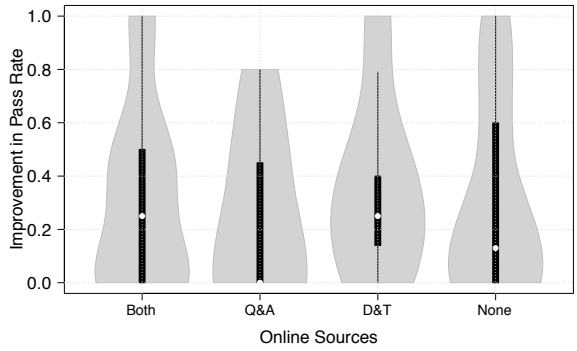
Table V lists the categories (*Online Sources*), their associated average improvement in pass rate (*avgImp*) and average pass rate after consulting each of several web sources (*avgPass*), and the number of attempts (*#Attempt*) for each category that was consulted. Of the 29 participants, 28 used search during their problem solving tasks. The most common resources consulted were Stack Overflow (visited 677 times), OracleDocs (visited 282 times) and Vogella (visited 92 times).

Participants who only consulted official documentation and tutorials for regular expressions improved their pass rate in JUnit tests by 35% on average, which provided the highest improvement in pass rate. Attempts being solved by consulting only Q&A sites led to the lowest improvement in pass rate; those being solved without consulting any online sources achieved highest pass rate on average. A potential explanation for this phenomenon is that participants who only relied on Q&A sites were not engaged in learning and thinking as actively as those who consulted documentation and tutorials, and those who did not search may have prior experience with regular expressions.

Figure 6 reveals the distribution of the pass rates (Figure 6(a)) and the improvement in pass rates (Figure 6(b)) for each category. In a violin plot, the white dot represents the median value and the width of the grey areas reflects the distribution of the data. For example, among attempts that only consult Q&A sites, Figure 6(b) indicates about half do not improve their pass rate, as the white dot is close to *0.0*. The widest part of the grey area is also around 0.0, indicating the highest probability of pass rate improvement for this category.



(a) Online Sources vs. Pass Rate



(b) Online Sources vs. Improvement in Pass Rate

Fig. 6. Pass rates and pass rate improvements for attempts that access various online sources

Attempts being solved without consulting any sites achieved highest median pass rate . Attempts being solved by consulting only documentation and tutorial sites achieved 2^{nd} highest pass rate and gained the highest improvement in pass rate in terms of median. Although attempts being solved by consulting only Q&A sites had relatively the same median pass rate as attempts being solved by consulting both sites, they gained lower median improvement in pass rate than the latter.

> Participants who used official documentation and tutorials for regular expressions and did *not* consult Q&A sites, achieved the highest improvement in the correctness of their regular expressions (i.e., 35%, after consulting the online resource).

D. First Attempts

We recognized seven activities performed by participants: guess (compose from nothing), search, copy and paste ("C&P"), modify the regular expression directly (this process will not result in an empty string), test, debug, and use of web tools. Of the 94 attempts, 58 started with guessing in the IDE (*Guess*, 49 attempts) or in web tools (*Web Tools*, 9 attempts). Of those, 20 ultimately achieved success, resulting in a 34.5% success rate. The remaining 36 attempts started with *Search*. Of these, eight ultimately achieved success, resulting in a 22.2% success rate. This indicates that participants who guess and try to solve the task first, rather than searching for existing regular expression examples or solutions, are more likely to pass all JUnit tests.

> Participants who tried to compose a regular expression first instead of searching for a solution first are more likely to pass all the test (34.5% success vs. 22.2% success).

E. Copy and Paste

Participants sometimes copied regular expressions directly from search results, pasting them into the IDE. Copy and paste (C&P) was used in 33 of 94 attempts, and the average pass rate on these attempts was 45% (compared to an average pass rate of 62% for non-C&P attempts). Participants who used the C&P strategy improved their pass rates by 27% on average.

The most frequent online resources participants copied from were Stack Overflow, OracleDocs, Vogella, TutorialsPoint and Regular-Expressions. Participants copied possible solutions from miscellaneous Q&A websites as well.

Participants tested the copy-pasted regular expression strings directly as solutions in some cases (after a copy and paste, 36.3% of the time the regular expression was tested immediately after pasting), but more often participants modified the regular expression strings before testing (after a copy and paste, 57.7% of the time the regular expression was directly modified and then tested). Participants often modified the copied and pasted regular expressions to simply resolve compile errors (29 of 80 C&P from websites to Eclipse).

> Reuse of existing code improved pass rates on attempts by 27% on average. However, copied and pasted contents in 36.3% C&P interactions (from websites to Eclipse) were slightly modified in syntax to correct compile error.

VI. Results - RQ2

After following the persona vector identification process described in Section IV-B, we converged on a set of four unique personas: *Novice tester*, *Knowledgeable tester*, *Knowledgeable* and *Intermediate* (*Persona* and *Persona Vector* in Table VI). Table VI also contains the description of each persona (*Description*). As Table VI shows, the most frequent personas were the *intermediates*, but the distribution of persona types across the participants is relatively even.

Average descriptive characteristics, along with the interquartile ranges (IQR), for each persona are further listed in Table VII. Column *AvgRegexExp* and *AvgJUnitExp* introduces the average self-rated experience with regular expression and JUnit tests in pre-survey, where 0 maps to "I have never heard of them" and 4 maps to "I am an expert". *AvgJavaExp* introduces personas' self-reported years in programming with Java in pre-survey. This table also contains the average pass rate achieved (*AvgPassRate*), the average frequency of Google searches (*AvgSearches*), the average frequency of copy and paste (*AvgC&P*) and the average frequency of Stack Overflow

TABLE VI
SUMMARY OF PERSONA IDENTIFICATION AND DESCRIPTION

Persona	Persona Vector	Description
Novice tester (7/29)	< 0, 0/L, H > < 0, H, H >	No prior knowledge, frequently test, very likely (5/7) to significantly improve pass rate
Knowledgeable tester (5/29)	< H, 0/L, H > < H, H, H >	Sufficient prior knowledge, frequently test, about even chance (3/5) to significantly improve pass rate
Knowledgeable (8/29)	< H, 0/L, L > < H, H, L >	Sufficient prior knowledge, rarely test, even chance (4/8) to significantly improve pass rate
Intermediate (9/29)	< L, 0/L, L > < L, H, L >	Insufficient prior knowledge, rarely test, very likely (7/9) to slightly/no improve pass rate

TABLE VII
PERSONA STATISTICAL SUMMARY

Persona	RegexExp		JUnitExp		JavaExp		PassRate		Search		C&P		Stack		Docs	
	Avg	IQR	Avg	IQR	Avg	IQR	Avg	IQR	Avg	IQR	Avg	IQR	Avg	IQR	Avg	IQR
Novice tester	1.9	0.5	1.1	0	3.4	3.5	44.1%	23.5%	7.7	3.0	1.4	3.0	9.7	8.0	8.1	9.0
Knowledgeable tester	1.6	1.0	2.2	2.0	2.3	1.0	58.8%	32.0%	11.0	3.0	3.0	3.0	15.3	13.5	4.4	4.8
Knowledgeable	2.6	1.0	1.3	0	4.3	2.0	63.5%	44.0%	7.0	6.0	4.8	4.8	1.8	2.5	1.1	1.0
Intermediate	2.2	1.0	2.1	2.0	2.8	2.0	37.7%	12.0%	11.9	7.0	4.7	8.0	10.9	7.6	4.7	4.8

visits and documentation and tutorial site visits (*AvgStack*, *AvgDocs*). These factors reflect personas' overall correctness, search and copy/paste behaviors, and preference in sources of information. In Table VII, we can see that the *knowledgeables* were the most successful in terms of pass rate, followed by the *knowledgeable testers*, *novice testers*, and finally the *intermediates*. There is no significant relationship between personas and their experience in JUnit observed in this study.

Novice tester (7/29): *Novice testers* claimed that they have some experience with regular expressions and Java (3^{rd} in *AvgRegexExp*, 2^{nd} in *AvgJavaExp*), and they were observed spending much of their time reading documentation and tutorial sites (highest in *AvgDocs*). While they didn't achieve any success (AFPR of 0%) in the beginning of every attempt, they did eventually achieve some success (3^{rd} in *AvgPassRate*) and seemed to learned about fundamental topics.

Knowledgeable tester (5/29): Though the *knowledgeable testers* had the lowest average regular expression experience and lowest average Java experience, they appeared to quickly gain an understanding of regular expressions in Java and achieved success through heavy tinkering (2^{nd} in *AvgPassRate*). The *knowledgeable testers* tended to favor looking for sample solutions over learning from documentation and tutorial sites (highest in *AvgStack* and 3^{rd} in *AvgDocs*).

Knowledgeable (8/29): There were eight observed *knowledgeable* persona, and they achieved the highest average pass rate with 63.5%. They had the highest average regular expression experience and highest average Java experience, suggesting that familiarity with the language being used with regular expression development is as important as regular expression experience itself. *Knowledgeables* further had the lowest average Google searches, lowest average Stack Overflow site visits and lowest average documentation and tutorial site visits, suggesting preexisting knowledge of regular expressions in Java. *Knowledgeables* usually searched with specific keywords, such as "*regex lookahead*". This observation is supported by a prior work that found developers who have sufficient domain knowledge tended to look for very specific types of information [20]. The *knowledgeable* persona produced the highest average copy-and-paste, and an potential explanation is the usage of web tools during composition, which require copy-and-paste between web tools and Eclipse IDE.

Intermediate (9/29): Despite the *intermediates* being familiar with regular expressions (2^{nd} in *AvgRegexExp*), they did not achieve great success (lowest in *AvgPassRate*). The *intermediates* tended to favor Google searching Q&A sites like Stack Overflow (highest in *AvgSearch* and 2^{nd} in *AvgStack*), and would frequently copy-and-paste regular expression strings (2^{nd} in *AvgC&P*).

VII. DISCUSSION

The combination of surveys and video logs has led to insights regarding the performance of participants, and what tools and problem solving strategies they used during regular expression composition.

A. Suggestions for regular expression writers

We suggest all developers come up with a solution before searching for solutions. We also suggest searching for regular expression related information, like the usage of special metacharacters, on official documentations, rather than for regular expressions to reuse; in our study, these behaviors led to higher success.

For *novice testers* and *knowledgeable testers*, we recommend using web tools. Compared to Eclipse IDE and JUnit tests, web tools that support dynamic testing can return matching results immediately. In addition, web tools usually provide both the regular expression tester and documentation on the

same window, which can reduce context switching for users. Since *novice testers* and *knowledgeable testers* test programs and check documentation frequently, they are very likely to benefit from using web tools.

We encourage the *knowledgeable testers* and the *intermediates* to think critically and adopt more documentation and tutorial sites as they search for online sources. Reusing existing code can improve pass rates on an attempt by 27% (Section V-E); however, our results show that people who learned from documentation and tutorial sites were able to improve the pass rates by 31%-35%. As demonstrated in Section V-D, participants who came up with their own solutions prior to looking for existing code were 1.7 times more likely to pass all the JUnit tests than those who searched first.

B. Implications for tool developers

Visualization, as provided by web tools, seemed to help developers with composition (Section V-B). To supplement web tools [8]–[10] that visualize regular expressions, there is a regular expression plugin for Eclipse IDE, QuickREx [21], that supports `java.util.regex`. This plugin suggests input metacharacters and highlights matching result for users, as most web tools do. However, no documentation search feature is available within the plugin. Beck, et al. concluded that regular expression experts agree that visual encoding of the regular expression is beneficial [22]. Other work on visualization also pointed out that visualization can support the task solving process by displaying useful information in a condensed way [23], enhance the programmer's understanding of the process of the execution [24], and serve as an external memory aid to help programmers tracking runtime execution [25]. Moreover, visualization can significantly help with conquering the design barrier where programmers have cognitive difficulties with visualizing solutions to programming problems [26]. Our results support these findings. Therefore, we suggest that features such as visualization and quick search for documentation should be integrated into the IDE to facilitate regular expression comprehension and reduce context switching for users. Tools for different languages that support regular expressions should be developed as well.

We also observed that some participants faced challenges in migrating regular expressions from web tools or sites they visited to Java due to differences in programming language representations of regular expression. As mentioned in Section V-E, 29 of 80 C&P from websites to Eclipse were edited to address the compile errors. For example, a single backslash in Python needs to be edited to a double backslashes in Java. This suggests that a language migration tool would be helpful.

VIII. Threats to Validity

Conclusion: Some tasks can produce a non-zero pass rate when tested against an empty regular expression. This situation might lead to an overestimation of participants' performance. In this study, two participants tested three tasks in total against an empty regular expression and gained 32.7% pass rate in average. This special case is rare, so the influence is small.

Editing and testing activities within web tools are not part of the IDE flow and thus not captured in our results.

Construct: Participants knew they were being observed which may have influenced their searching and composition behaviors.

Internal: Most participants attempted multiple tasks, yet we did not consider task order in the analysis of the results. It is possible that learning effects influenced pass rates of tasks attempted later in the study. However, as the tasks were assigned in random order, the impact should be distributed across all tasks.

External: Participants were students, and while students write regular expressions with some frequency, the results may not generalize to other populations. A replication with a more diverse set of developers is needed. In addition, years in programming (non-professionally) might not be the best indicator to measure the programming experience for students [27].

The 20 tasks in the case study have tasks with solutions that are less complex than regular expressions in the wild. Future studies will use tasks whose solutions are reflective of regular expressions found in source code repositories, and thus may not be representative of tasks for which programmers would use regular expressions. However, Host, et al. concluded that there is no significant difference between the correctness of students and professionals [28]. Runeson also confirmed that first year students, graduate students and industry professionals produced the same improvements between the different Personal Software Process (PSP) levels [29].

IX. Related Work

Regular Expression and Users: Several studies have investigated approaches to expedite regular expression processing on a large amount of text [30], [31]. However, few studies have focused on regular expression users, despite the fact that regular expressions are error-prone and hard to comprehend [4], [5]. Studies that include observations of users and regular expressions use survey [4], crowd-sourcing [5], or dynamic program analysis [7] as methodologies.

In this work, we observe programmer behavior directly using screen capture to better understand the tools and strategies developers use during regular expression composition.

Regular Expression Comprehension: Chapman and Stolee explored the contexts in which the professional developers use regular expressions, the most commonly used regular expression language features in Python, and the regular expression behavioral similarity among projects [4]. Through a survey of developers, the study concluded that developers complain about regular expressions being hard to read and write. A follow-up study on regular expression understandability [5] concluded that a regular expression's DFA is positively correlated with comprehension.

In this work, the median DFA size of the sample solutions are smaller than the regular expressions in the wild (i.e., 14 nodes vs. 28 nodes [7]). This indicates that the tasks in this

study are less complex, but were designed intentionally since the participants have a short time to work on the tasks.

IDE Interactions: Programmers work closely with Integrated Development Environments (IDEs), and Eclipse is a popular IDE for Java [32], [33]. A guide on how to collect and analyze general IDE usage data, which we used to guide the logged events in our study (Section IV-A), suggests recording commands invoked, files viewed, mouse clicks, and add-on tools used [34].

The choice to use screen capture software and transcribe logged events is not without precedence in the literature. To study programmers' actions, navigations and choices during software maintenance tasks, Ko, et al. transcribed 12 hours of screen-captured video across 10 developers' work, and logged each developer action [35].

Copy and Paste: Copy/paste is a popular technique to assist solving programming problems [36]–[38] as it can speed up the programming process [38]. Mann identified four copy/paste operations to address different scenarios in programming process, which are move, copy-identical, copy-and-change, and copy-once [39].

We investigate how participants use the copy and paste technique in regular expression composition, and compare this approach with direct editing performed by participants.

Visualization: Studies have shown that the lack of understanding of concepts/structures/syntax poses the most difficulties for programmers to solve a programming problem [24], [40]. To assist the programmer's understanding of the runtime program behavior, visualization tools have been suggested [24] and developed for educating purpose, such as Whyline [25], Storytelling Alice [41] and Gidget Game [42]. In the context of regular expressions, the website regexper.com [9] provides an automata-based visualization for regular expressions.

In this study, we explore what tools participants adopt during regular expression composition, and whether the web tools that support visualization of regular expressions better support composition.

Code Search and Problem Solving: Various studies have surveyed participants about why they use web search, their tools, and their selection criteria for code [43]–[45]. Professionals working in companies [46], [47] and students learning programming [44] are involved as participants in these studies. Some studies focused on directly observing participants' behaviors [48], while some other studies focused on collecting and analyzing search logs [19], [46], with participant solving the pre-selected tasks.

Researchers also studied the search activities within IDEs and developed tools to facilitate searches. An extension called Bing Developer Assistant has been developed for Microsoft Visual Studio, which can recommend previously written API sample code mined from public repositories and Q&A sites to developers, and hence boost their productivity by reducing context switches [49].

In this work, participants use search as part of their problem solving strategies; we explore how often and whether search leads to success.

Personas in Software Engineering: Personas are an interaction design technique that establish fictional users of a system. The practice of personas was originally used from a marketing perspective [50]; however, Alan Cooper helped to shift their use towards software design and development [51]. Under Cooper's use of personas, designers focus on user goals and activity scenarios to guide software design [52], [53].

Another use of personas in software engineering is to support analysis of developers' behavior. Dubey, et al. explored testers' testing style and performance in crowdsourced testing and categorized them into six personas based on three quantitative metrics [18]. They also introduced hybrid personas in this study [18]. Stylos and Clarke adapted three personas from prior work [54], which are defined by a Visual Studio usability group based on participants' behavior [55]. They explored how different personas react to various constructors and suggest future work on comparing debugging strategies from the perspective of personas [55]. Ford, et al. provided a data-driven approach to identify clusters to create seven personas by interviewing and surveying software engineers on how they comprehend their tasks, collaborate with others and how they spend time [56].

In our work we adopt the use of observation-oriented persona development to better understand developers' behaviors during regular expression development.

X. Conclusion

Participants vary greatly in their programming experience and programming habits. In our study of 29 participants working on regular expression tasks, we revealed the strategies adopted, such as search, copy and paste, and use of web tools. We learned the participants performed best on *validation* tasks. We recommend guessing and composing solutions before searching, and learning from the documentations and tutorials instead of the existing sample solutions.

Further, we attributed personas to each of the research participants, grouping all 29 of them into four distinct archetypes. These four personas exemplify the behavioral patterns observed in the participants, and allow us to investigate which strategies used by each persona were effective or not. We found that the personas who tended to favor copy and pasting answers from Q&A sites were less successful than those who tried to learn from documentations and tutorials, or used heavy tinkering.

For future work, a study that requires participants to think-aloud is suggested to understand any learning barriers for regular expression composition. Another direction of future work is to explore the evolution of the regular expressions and technical mistakes made during regular expression composition.

Acknowledgements

We thank Peipei Wang for assistance with the regular expression complexity analysis. This work is supported in part by the NSF SHF #1714699 and #1749936, and the Harpole-Pentair endowment to Iowa State University.

REFERENCES

[1] T. Stubblebine, *Regular Expression Pocket Reference*, 2nd ed. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2007.

[2] H. Zhao, W. Meng, Z. Wu, V. Raghavan, and C. Yu, "Fully automatic wrapper generation for search engines," in *Proceedings of the 14th International Conference on World Wide Web*. ACM, 2005, pp. 66–75.

[3] D. Ficara, S. Giordano, G. Procissi, F. Vitucci, G. Antichi, and A. Di Pietro, "An improved dfa for fast regular expression matching," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 5, pp. 29–40, Sep. 2008.

[4] C. Chapman and K. T. Stolee, "Exploring regular expression usage and context in python," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, ser. ISSTA 2016. New York, NY, USA: ACM, 2016, pp. 282–293. [Online]. Available: http://doi.acm.org.prox.lib.ncsu.edu/10.1145/2931037.2931073

[5] C. Chapman, P. Wang, and K. T. Stolee, "Exploring regular expression comprehension," in *32nd IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2017, Piscataway, NJ, USA, 2017, pp. 405–416. [Online]. Available: http://dl.acm.org/citation.cfm?id=3155562.3155616

[6] Spishak, Eric and Dietl, Werner and Ernst, Michael D., "A Type System for Regular Expressions," in *Proceedings of the 14th Workshop on Formal Techniques for Java-like Programs*, ser. FTfJP '12. New York, NY, USA: ACM, 2012, pp. 20–26. [Online]. Available: http://doi.acm.org.prox.lib.ncsu.edu/10.1145/2318202.2318207

[7] P. Wang and K. T. Stolee, "How well are regular expressions tested in the wild?" in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2018. New York, NY, USA: ACM, 2018, pp. 668–678. [Online]. Available: http://doi.acm.org/10.1145/3236024.3236072

[8] "Debuggex: Online visual regex tester. javascript, python, and pcre." https://www.debuggex.com.

[9] "Regular expression visualizer using railroad diagrams." https://regexper.com.

[10] "Online regex tester, debugger with highlighting for php, pcre, python, golang and javascript." https://regex101.com.

[11] "Regexr: Learn, Build, & Test RegEx," https://regexr.com/.

[12] "Regex golf," https://alf.nu/RegexGolf.

[13] "Regex one," https://regexone.com.

[14] "Regex crossword," https://regexcrossword.com.

[15] A. Rosenfeld, A. Ade-Ibijola, and S. Ewert, "Regex parser ii: Teaching regular expression fundamentals via educational gaming." 09 2016.

[16] R. Cox, "Regular expression matching in the wild," *URL:http//swtch.com/~rsc/regexp/regexp3.html*, 2010.

[17] J. Pruitt and J. Grudin, "Personas: Practice and theory," in *Proceedings of the 2003 Conference on Designing for User Experiences*, ser. DUX '03. New York, NY, USA: ACM, 2003, pp. 1–15. [Online]. Available: http://doi.acm.org/10.1145/997078.997089

[18] A. Dubey, K. Singi, and V. Kaulgud, "Personas and redundancies in crowdsourced testing," in *12th International Conference on Global Software Engineering*, ser. ICGSE '17, Piscataway, NJ, USA, 2017, pp. 76–80. [Online]. Available: https://doi-org.prox.lib.ncsu.edu/10.1109/ICGSE.2017.7

[19] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer, "Two studies of opportunistic programming: Interleaving web foraging, learning, and writing code," in *SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '09, 2009, pp. 1589–1598. [Online]. Available: http://doi.acm.org/10.1145/1518701.1518944

[20] A. von Mayrhauser and A. M. Vans, "Program understanding behavior during debugging of large scale software," in *Papers Presented at the Seventh Workshop on Empirical Studies of Programmers*, ser. ESP '97. New York, NY, USA: ACM, 1997, pp. 157–179. [Online]. Available: http://doi.acm.org/10.1145/266399.266414

[21] "Quickrex plugin," https://github.com/netceteragroup/quickrex.

[22] F. Beck, S. Gulan, B. Biegel, S. Baltes, and D. Weiskopf, "Regviz: Visual debugging of regular expressions," in *Companion Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 504–507.

[23] J. H. Larkin and H. A. Simon, "Why a diagram is (sometimes) worth ten thousand words," *Cognitive Science*, vol. 11, no. 1, pp. 65–100, 1987. [Online]. Available: http://dx.doi.org/10.1111/j.1551-6708.1987.tb00863.x

[24] I. Milne and G. Rowe, "Difficulties in Learning and Teaching ProgrammingViews of Students and Tutors," *Education and Information Technologies*, vol. 7, pp. 55–66, 2002.

[25] A. J. Ko and B. A. Myers, "Designing the whyline: A debugging interface for asking questions about program behavior," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '04. New York, NY, USA: ACM, 2004, pp. 151–158. [Online]. Available: http://doi.acm.org/10.1145/985692.985712

[26] A. J. Ko, B. A. Myers, and H. H. Aung, "Six learning barriers in end-user programming systems," in *Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing*, ser. VLHCC '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 199–206. [Online]. Available: http://dx.doi.org/10.1109/VLHCC.2004.47

[27] J. Siegmund, C. Kästner, J. Liebig, S. Apel, and S. Hanenberg, "Measuring and modeling programming experience," *Empirical Softw. Engg.*, vol. 19, no. 5, pp. 1299–1334, Oct. 2014. [Online]. Available: http://dx.doi.org/10.1007/s10664-013-9286-4

[28] M. Höst, B. Regnell, and C. Wohlin, "Using students as subjects—a comparative study of students and professionals in lead-time impact assessment," *Empirical Software Engineering*, vol. 5, no. 3, pp. 201–214, Nov 2000. [Online]. Available: https://doi.org/10.1023/A:1026586415054

[29] P. Runeson, "Using students as experiment subjects an analysis on graduate and freshmen student data," in *Proceedings 7th International Conference on Empirical Assessment Evaluation in Software Engineering*, 2003, pp. 95–102.

[30] Baeza-Yates, Ricardo A. and Gonnet, Gaston H., "Efficient text searching of regular expressions," in *Algorithms and Data Structures*, Dehne, F. and Sack, J. -R. and Santoro, N., Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1989, pp. 1–2.

[31] Baeza-Yates, Ricardo A. and Gonnet, Gaston H., "Fast text searching for regular expressions or automaton searching on tries," *J. ACM*, vol. 43, no. 6, pp. 915–936, Nov. 1996. [Online]. Available: http://doi.acm.org/10.1145/235809.235810

[32] G. Goth, "Beware the march of this ide: Eclipse is overshadowing other tool technologies," *IEEE Software*, vol. 22, no. 4, pp. 108–111, July 2005.

[33] G. C. Murphy, M. Kersten, and L. Findlater, "How are java software developers using the eclipse ide?" *IEEE Softw.*, vol. 23, no. 4, pp. 76–83, Jul. 2006. [Online]. Available: http://dx.doi.org/10.1109/MS.2006.105

[34] W. Snipes, E. Murphy-Hill, T. Fritz, M. Vakilian, K. Damevski, A. Nair, and D. Shepherd, "A practical guide to analyzing ide usage data," in *The Art and Science of Analyzing Software Data*, 2015.

[35] A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung, "An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks," *IEEE Trans. Softw. Eng.*, vol. 32, no. 12, pp. 971–987, Dec. 2006. [Online]. Available: http://dx.doi.org/10.1109/TSE.2006.116

[36] R. Koschke, R. Falke, and P. Frenzel, "Clone detection using abstract syntax suffix trees," in *Working Conference on Reverse Engineering*, 2006, pp. 253–262.

[37] B. Lague, D. Proulx, J. Mayrand, E. M. Merlo, and J. Hudepohl, "Assessing the benefits of incorporating function clone detection in a development process," in *International Conference on Software Maintenance*, Oct 1997, pp. 314–321.

[38] K. Narasimhan and C. Reichenbach, "Copy and paste redeemed," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Nov 2015, pp. 630–640.

[39] Z. A. Mann, "Three public enemies: cut, copy, and paste," pp. 31–35, July 2006.

[40] E. Lahtinen, K. Ala-Mutka, and H.-M. Järvinen, "A study of the difficulties of novice programmers," *SIGCSE Bull.*, vol. 37, no. 3, pp. 14–18, Jun. 2005. [Online]. Available: http://doi.acm.org.prox.lib.ncsu.edu/10.1145/1151954.1067453

[41] C. Kelleher, R. Pausch, and S. Kiesler, "Storytelling alice motivates middle school girls to learn computer programming," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '07. New York, NY, USA: ACM, 2007, pp. 1455–1464. [Online]. Available: http://doi.acm.org.prox.lib.ncsu.edu/10.1145/1240624.1240844

[42] M. J. Lee and A. J. Ko, "Investigating the role of purposeful goals on novices' engagement in a programming game," in *2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Sept 2012, pp. 163–166.

[43] S. E. Sim, C. L. A. Clarke, and R. C. Holt, "Archetypal source code searches: a survey of software developers and maintainers," in *Program Comprehension, 1998. IWPC '98. Proceedings., 6th International Workshop on*, Jun 1998, pp. 180–187.

[44] S. E. Sim, M. Umarji, S. Ratanotayanon, and C. V. Lopes, "How well do search engines support code retrieval on the web?" *ACM Trans. Softw. Eng. Methodol.*, vol. 21, no. 1, pp. 4:1–4:25, Dec. 2011. [Online]. Available: http://doi.acm.org/10.1145/2063239.2063243

[45] K. T. Stolee, S. Elbaum, and D. Dobos, "Solving the search for source code," *ACM Trans. Softw. Eng. Methodol.*, vol. 23, no. 3, pp. 26:1–26:45, Jun. 2014. [Online]. Available: http://doi.acm.org/10.1145/2581377

[46] Brandt, Joel and Dontcheva, Mira and Weskamp, Marcos and Klemmer, Scott R., "Example-centric Programming: Integrating Web Search into the Development Environment," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '10. New York, NY, USA: ACM, 2010, pp. 513–522. [Online]. Available: http://doi.acm.org/10.1145/1753326.1753402

[47] C. Sadowski, K. T. Stolee, and S. Elbaum, "How developers search for code: A case study," in *Foundations of Software Engineering*, ser. ESEC/FSE 2015, 2015, pp. 191–201. [Online]. Available: http://doi.acm.org/10.1145/2786805.2786855

[48] J. Singer, T. Lethbridge, N. Vinson, and N. Anquetil, "An examination of software engineering work practices," in *Conference of the Centre for Advanced Studies on Collaborative Research*, ser. CASCON '97, 1997. [Online]. Available: http://dl.acm.org/citation.cfm?id=782010.782031

[49] H. Zhang, A. Jain, G. Khandelwal, C. Kaushik, S. Ge, and W. Hu, "Bing developer assistant: Improving developer productivity by recommending sample code," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016. New York, NY, USA: ACM, 2016, pp. 956–961. [Online]. Available: http://doi.acm.org/10.1145/2950290.2983955

[50] N. Mikkelson and W. O. Lee, "Incorporating user archetypes into scenario-based design," in *Proc. UPA*, 2000.

[51] A. Cooper, *The Inmates Are Running the Asylum: Why High Tech Products Drive Us Crazy and How to Restore the Sanity (2Nd Edition)*. Pearson Higher Education, 2004.

[52] L. Schneidewind, S. Hörold, C. Mayas, H. Krömker, S. Falke, and T. Pucklitsch, "How personas support requirements engineering," in *International Workshop on Usability and Accessibility Focused Requirements Engineering*, ser. UsARE '12, Piscataway, NJ, USA, 2012, pp. 1–5. [Online]. Available: http://dl.acm.org.prox.lib.ncsu.edu/citation.cfm?id=2667081.2667082

[53] F. Anvari, D. Richards, M. Hitchens, and M. A. Babar, "Effectiveness of persona with personality traits on conceptual design," in *37th International Conference on Software Engineering - Volume 2*, ser. ICSE '15, Piscataway, NJ, USA, 2015, pp. 263–272. [Online]. Available: http://dl.acm.org.prox.lib.ncsu.edu/citation.cfm?id=2819009.2819048

[54] S. Clarke, "Measuring api usability," vol. 29, pp. S6–, 05 2004.

[55] J. Stylos and S. Clarke, "Usability implications of requiring parameters in objects' constructors," in *29th International Conference on Software Engineering*, ser. ICSE '07, Washington, DC, USA, 2007, pp. 529–539. [Online]. Available: http://dx.doi.org.prox.lib.ncsu.edu/10.1109/ICSE.2007.92

[56] D. Ford, T. Zimmermann, C. Bird, and N. Nagappan, "Characterizing software engineering work with personas based on knowledge worker actions," in *Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 394–403. [Online]. Available: https://doi-org.prox.lib.ncsu.edu/10.1109/ESEM.2017.54