The Impact of Affect, Scenario and Task Characteristics on Developer Decision-Making

A Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at George Mason University

by

Cassandra Bailey
Bachelor of Science
George Mason University, 2017

Director: Thomas LaToza, Professor
Department of Computer Science

Summer Semester 2020
George Mason University
Fairfax, VA

ACKNOWLEDGEMENTS

I would like to thank the numerous friends, coworkers, and colleagues who assisted me in my research. Dr. LaToza was of invaluable help – many thanks to his excellent mentorship and guidance. Special thanks to Abdulaziz for his analytical assistance, and to the rest of the Developer Experience Design Lab for their review. Finally, thanks go out to the amazing faculty of the CS and SWE departments at GMU for encouraging me, supporting me, and humbling me with their passion, experience and wisdom.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

THE IMPACT OF AFFECT, SCENARIO AND TASK CHARACTERISTICS ON DEVELOPER DECISION-MAKING

Cassandra Bailey, M.S.

George Mason University, 2020

Thesis Director: Dr. Thomas LaToza

Decision-making and strategy choice during software development are influenced by many factors, from the technologies we use to our own personal qualities. Better understanding these factors may help enable better understanding how and when development tools help developers. The goal of this thesis was to identify the relationship between contextual factors and the strategy selections that determine how developers approach problems. The factors that were examined were programming task, code scenario, and affect (mood). Task is the actual development task, such as debugging, implementing, or testing. Code scenario is a way to describe the type of code; our survey examines three scenarios: self-written code, framework code, and code containing callbacks. Affect describes psychological mood; it can be activated or deactivated (arousal), as well as positive or negative (valence). Our findings show that all three factors influenced the strategies participants chose. Participants with a highly positive

affect were over twice as likely to ask for help. Participants were twice as likely to use developer tools for a debugging task, as in an implementation task. Participants rarely wished to create diagrams when dealing with framework code, as compared to their own code. These findings deepen our understanding of developer strategy choice; these insights can be used to improve recommendation systems, IDE's, and other developer tools not only by improving the recommendations and content itself, but also by gaining more insight about when a developer might use those systems.

I INTRODUCTION

Decision-making by software developers has long been a focus of software engineering research [3][10][22]. Developers make many choices as they create and maintain software; each is an opportunity to improve developer experience and performance, and the quality of the software. For example, a developer might be asked to create a new widget for their company's application. Some developers may begin by looking at documentation or the existing surrounding code. Others may start by talking to the teammate that developed the related code. The task itself, creating a new feature, lends itself to certain strategies more than others. These differing strategies, and the factors that impact how they are selected, are the focus of this research.

First, we ask the question, *what types of factors impact developers and/or development work*? There are many factors that impact the productivity of developers: personal characteristics, such as mood or years of experience, as well as the characteristics of their work, like technologies used [7]. Developer qualities like affect and experience have been studied as a factor in software development [11][14][16] and in general human decision-making extensively [26][3][8][24][23]. The impact of these factors has primarily been examined through measuring its impact on job satisfaction, task completion, and performance [1][11][14][16]. While this documents the importance

of these factors, it does less to explain the mechanisms by which these factors impact development. That is, how does changing affect help developers to be more productive?

In this thesis, we examine how contextual factors impact development work by investigating how they impact strategy selection. Our study focuses on three contextual factors: *affect (mood)*, *task*, and *scenario*. A *contextual factor* is some element that creates or adds to the context of a programming problem and can be intrinsic (the programming patterns used in the code) or extrinsic (for example, characteristics or state of the developer). *Task* is the actual development task, such as debugging, implementing, or testing. Task characteristics have been examined as contextual factors previously [26]. *Code scenario* is a way to describe the type of code; our survey examines three scenarios: self-written code, framework code, and code containing callbacks. *Affect* describes psychological mood; it can be activated or deactivated (arousal), as well as positive or negative (valence).

The questions asked in this paper correspond to the three potential factors:

RQ1. Does the type of programming task (Debugging, testing/verification, implementation) impact the strategy choice developers use?

RQ2. Does code scenario (i.e., code you own, framework code, code with callbacks) impact the strategy choice developers use?

RQ3. Do changes in affect impact the strategy choice developers use?

Each of these research questions share a commonality – the impact of strategy choice. In this context, a *strategy* is a generalization of some development work. Similarly to the factors, we examined decision-making in such a way that it could be generalizable to

many software development situations, rather than a specific process(es). This led to the definition of strategies as a general action that can be taken when presented with any development problem in any scenario. While some strategies may coincide better with certain factors (for example, *Use dev tools* and the *debugging* task), the strategies were worded in such a way to mitigate these inherent relationships.

We conducted a study where participants were provided questions that represented different tasks and scenarios. For example, a question would include a code snippet representing the code scenario and then describe a simple task, like finding and fixing a bug. For each question, participants were asked to rank order the provided strategies based on the likelihood of their use to complete the task as it relates to the code snippet. This provides an examination of RQ1 and RQ2. In order to investigate RQ3, participants were randomly assigned one of five treatments: four different affect treatments, which are enumerated in SECTION III, or a control of no affect treatment, and were shown a corresponding affect-inducing video clip. Short video clips for inducing affect have been used previously in similar research [11][27][13].

We found that strategy selections varied, often substantially, depending on contextual factors, particularly in relation to the programming task. Participants were nearly four times as likely to search an online forum when presented with a code scenario using third-party libraries than a scenario containing only self-written code. Participants were nearly three times as likely to use print statements for debugging tasks than implementation tasks. Other strategies were avoided in specific situations - for example,

participants were very unlikely to read documentation or other artifacts for the debugging tasks.

Affect had an impact on some of the strategy choice as well. Participants with a slightly negative affect were very unlikely to select the *Experiment with edits* strategy. This suggests a complement to previous research that shows a slightly positive affect boosts creative thinking [26][24]. Additionally, the highly positive affect-induced participants were over twice as likely to ask for help from a colleague than those in the control group.

Improvements to recommendation systems and other supplementary systems could be made by using insights from these results. For example, one study found that altering the mood of participants improved their ability to predict the outcome of running algorithms [17]. Developers can use these insights to approach problems with different affects, as their decision-making will likely change. Recommendation systems can use knowledge about the code scenario, like the framework, to provide links to documentation and other resources. In addition, this research helps deepen our understanding of decision-making in developers. This, coupled with the ability to define and measure factors of a software development context, enables processes, IDE's, workplaces, and teams to better tailor themselves to developers.

II THE CONTEXTUAL FACTORS

In this thesis, we examine the impact of three *contextual factors* on strategy selection. These include both factors intrinsic to the software (the programming style in which it is written), as well as extrinsic factors about the disposition and goal of the developer. They are *task*, *code scenario*, and *affect*.

A. Task

We examined three different types of programming tasks: implementation, verification, and debugging. Implementation involves writing new code, with new functionality. Successfully completing an implementation task results in new functionality. Verification is the task of testing or verifying the behavior of a certain component of the system, e.g. a class, method, or module. Successful completion ensures that the code artifact meets its requirements. For example, to verify a method, the requirements defining error handling, expected return values, expected inputs, and any edge cases should be met. Debugging is the task of localizing and fixing a defect. Successfully completing a debugging task results in the program no longer behaving in an unexpected way.

B. Code Scenario

Code scenario refers to the type of code that encompasses or is otherwise related focally to the task at hand. The situation faced by the programmer that requires some strategy to be chosen is often in the midst of existing code.

Three typical coding scenarios are examined here: Code that is entirely written and owned by the programmer (referred to in SECTION IV as “M”, for “My Code”), code that involves using a framework not written by the programmer (referred to in SECTION IV as “F”, for “Framework Code”), and code that involves concurrency and callbacks (referred to in SECTION IV as “C”, for “Callbacks/Concurrency”). The first two scenarios are complements of each other; the third scenario was selected because of the unique challenges and workflow of dealing with concurrency [1]. Though a typical software development task may encompass multiple code scenarios as they are defined here, the scenarios were purposefully developed to be simple and atomic, such that each scenario was accurately represented.

C. Affect

Affect defines the current emotional state or mood of a person. Affect is measured on two dimensions: valence and arousal. Valence measures the positive or negative nature of the emotion; for example, a “happy” affect corresponds to a positive valence, whereas a “sad” affect implies a negative valence. Arousal measures the extent to which the emotion is felt, or the intensity of the emotion. This two-dimensional representation defines four types of affect, which we examine in this thesis. Each dimension may be high or low. For arousal, these correspond to “Activated” and “Deactivated” in Figure 1

respectively. For valence, these correspond to “Pleasant” and “Unpleasant” respectively. This creates four types of affect: high valence/high arousal (HVHA), e.g., excited, low valence/low arousal (LVLA), e.g., sad or depressed, high valence/low arousal (HVLA), e.g., calm, and low valence/high arousal (LVHA), e.g., stressed or nervous. These abbreviations will be used throughout this paper to refer to the four affect treatments.

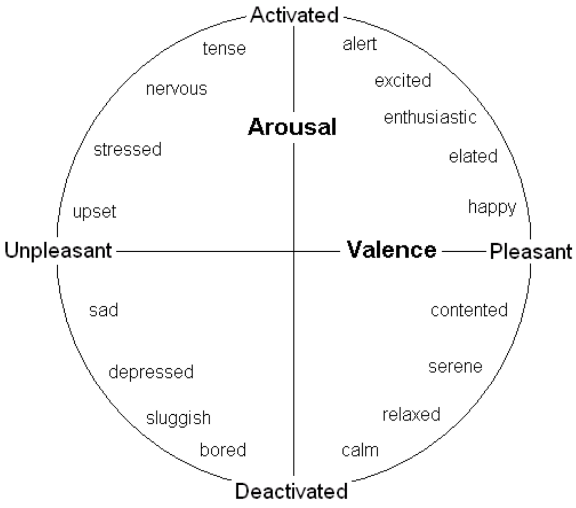

Figure 1 Two-dimensional representation of affect [27]

III RELATED WORK

Research has examined the impact of affect on decision-making and task performance, both generally and as it relates to software development. Other work has examined the different strategies developers may use to solve problems.

Affect has been extensively examined in psychology, with an abundance of work examining its impact on decision making. A positive affect has been shown to improve and boost creative thinking [7][24][14][23]. There is a lack of consensus about negative affect, though some literature suggests that it is more conducive to evaluative thinking, as people tend to focus on drawbacks and negative qualities during evaluation [24][4][8]. One study found that a negative affect improved evaluation of existing ideas and strategies; participants with a negative affect were able to more accurately assess ideas for their usefulness, whereas the positive-affect participants tended to overestimate usefulness [7].

In software development, affect can impact the behavior and decisions made by developers [17]. Research that examined the relationship between affect and debugging performance found that higher arousal (independent of valence) increased the number of debugging tasks that participants completed from 3.03 tasks to 4.59 tasks [16]. One study used physical activity to induce affect and found that higher valence and arousal leads to significantly improved debugging performance [16]. Other work has focused on

measuring affect in developers, with applications to improving developer experience. One study measured the affect of developers using mouse and keystrokes, and in a separate experiment showed that inducing positive affect had a positive impact on developer performance. This study, like the debugging performance study, also induced affect using video clips, and categorized treatments in two dimensions: valence and arousal. It found that arousal, not valence, was a significant factor in programming performance. A correlation was also found between self-reported mood levels and mouse/keyboard interactions [17].

Another body of work has examined the programming strategies that characterize how developers, teams, and industries work. Early work grouped programming strategies into differences into a programming persona which characterizes a work style: opportunistic, pragmatic, or systematic [9]. Opportunistic developers are exploratory in nature and focus on solving business problems. Systematic developers write code defensively and strive to build elegant solutions and gain deep understanding. Pragmatic developers are methodical, but not as much as systematic programmers, taking a "just-enough" investigation approach. One study found additional evidence that these distinctions exist (opportunistic and systematic developers were observed), independent of years of experience and domain knowledge. Opportunistic developers tended to try experimenting immediately, copying some sample code from the documentation and modifying it, building their code incrementally on top. Systematic developers spent significantly more time on the Concepts page of the documentation than opportunistic programmers (between 12.5% to 42.9%, versus 0% and 5.6%, respectively) [9].

Identifying programming personas has been argued to be beneficial, as it allows the developer to relate to an end-user (and other developers) in a personal sense, and to better anticipate the behaviors and use cases for them [12].

Other research has focused on conceptualizing programming strategies. One study defines explicit programming strategies as a process developers use [19]. Another focuses specifically on self-regulation and self-explanation strategies; monitoring and evaluating one's understanding, and explaining instructional material, respectively [6]. These conceptualizations capture the goal and task relationships to strategy choice and definition; these relationships are explored in our study, along with coding scenario and developer characteristics (affect), as contextual factors that influence the strategy choice, rather than inherent to the strategies.

The research on decision-making and problem-solving in software development varies widely on the factors used to define the context. Some researchers focused entirely upon intrinsic factors like keywords in the code, stack traces, and logs [24]. Other research has defined some meta-characteristics of tasks that are subjective but still inherent to the task itself, rather than the developer. One study used contextual characteristics within an IDE to provide web resources to developers [7]. The system for recommending the proper resources was developed to analyze stack traces within the IDE, taking into account the exceptions, references, and degrees of importance of the former. Such a recommendation system could be further improved by including factors for other contextual factors, such as those researched in this paper. Another study developed a *Design- Pattern Recommender (DPR)* process to recommend software

design patterns based on a problem context [24]. This context was established based upon aforementioned intrinsic characteristics like keywords and object usage. Since recommendations are inherently context-specific, researching better ways to identify, measure, and interpret contextual factors will improve recommendation accuracy and quality.

IV EXPERIMENTAL DESIGN

We conducted an experiment consisting of two parts: Part I, the main questionnaire, and Part II, the follow-up survey. The online research experiment was performed in Part I. In Part II, a follow-up survey was sent to participants that voluntarily provided their email, to gain insight on what participants thought about the contextual factors and their impact on strategy choice.

A. Design

To answer each of the research questions, survey questions were developed to address RQ1 and RQ3, and the video clips were used to address RQ2. The survey questions developed each represent exactly one task and one code scenario; for example, one question asks the participant to do a debugging task, given the “My Code” code scenario.

Each question provided the same set of eight programming strategies for the participant to rank order, from most likely to least likely to use for the given scenario and task. Twenty-nine of the 45 total participants saw the strategies in the same order as presented in Table 2. The other 19 respondents saw the strategies in a randomized order for each question. This change is discussed in SECTION VI, and the data explored further in SECTION V.

B. Materials

The Part I questionnaire consisted of 9 questions. Each question represents a treatment of both code scenario (RQ2) and task (RQ1). Table 1 shows the cross-product of code scenario and task. Each entry in this table corresponds to precisely one question.

Table 1 Combinatorial representation of each question in Part I

	RQ2 – Code Scenario		
RQ1 - Task	M (my code)	F (framework code)	C (Callback)
D (debug)	DM	DF	DC
V (verification)	VM	VF	VC
I (implementation)	IM	IF	IC

A code scenario is presented to the participant in the form of a code snippet and some descriptive text. A series of three questions, each representing a task, is then presented to the participant to imagine themselves working on the task in the context of the given code scenario. This format is repeated twice more, for the remaining two scenarios. The order of the scenarios, as well as the tasks, were randomized using the Latin Square method [15].

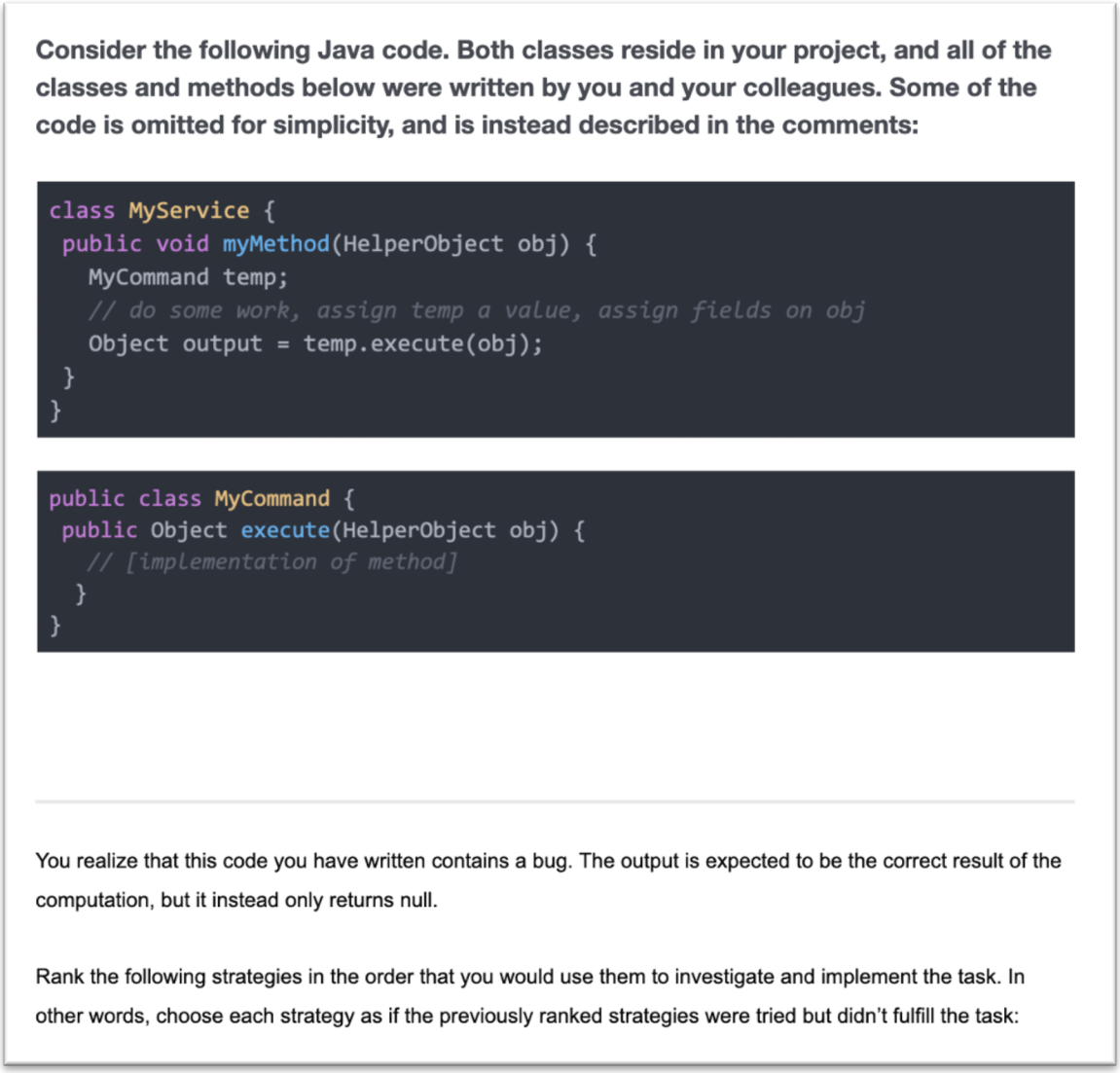

Figure 2 Sample Question: task: Debugging, code scenario: My Code

The strategies were presented as a list, which the participant ranked by rearranging each item of the list to form the order in which they would try each strategy for that question. Each item is simply the text shown in the *Strategy* column of Table 2. Throughout the paper, for brevity, the strategies will be referred to by their *Name* in Table 2.

Table 2 Names of strategies and corresponding full text

Name	Strategy
Experiment with edits	Experiment with various edits to see if those address the issue
Add print statements	Add print/logging statements to existing code
Read surrounding code	Read the immediately surrounding code
Check documentation	Check design documents, bug descriptions, email correspondence, or other artifacts
Search online forum	Search an online forum (e.g., Stack Overflow)
Use dev tools	Use developer tools (e.g., debugger, interface inspector, code/component graphs)
Ask for help	Ask for help from a colleague
Create diagrams	Create detailed diagrams manually

Part II of the experiment gathered the participant's opinions about their strategy choices. Figure 3 shows the questions given to the participants that opted in to Part II. We asked about strategy choice during the experiment as well as during development in general, to see if developers feel that factors like mood or task have an impact on their strategy choice.

Part II

1. What do you think is the biggest influence on which strategies you chose?
2. For each of the options you were given (refer to the list below), do you feel that there would be a certain programming task or scenario where you would choose each option first? If so, what would those scenarios be?
3. Did you feel like watching the video(s) affected your mood? Do you think it may have changed how you answered the questions?
4. When you're particularly frustrated or in an otherwise "bad" mood, do you think it affects how you code and the strategies you employ when coding? If so, how so?
5. What were your overall thoughts about the survey experience?

6. Were you surprised by the programming persona you were given? Why or why not?

Figure 3 Part II follow-up survey questions

C. Deployment

The study was administered remotely via a survey link, which was distributed through email and social media. To incentivize participation, the end of the survey redirects to a results page that ascribes a "programming style" to the participant: experimental, resourceful, or calculated, along with a corresponding description. Each of these styles were calculated by adding the "scores" (inverse of rank) each participant gave to each strategy for each question. Each style then corresponded to 2-3 strategies; for example, the Resourceful style included the following strategies: *Search online forum*, *Use dev tools*, and *Ask for help*.

The results page gave participants an overall style, as well as one for each type of task (Debugging, Implementation, Verification). It also showed a statistical breakdown of responses that fell into each category, overall and per task. This provided a way for the participant to compare their results with others. The page also provided multiple ways to share on social media or by copying the survey link directly. This was intended to incentivize participants to take the survey as well as to share it.

D. Participants

We recruited developers through postings on various social media platforms, as well as developer and engineering subgroups and mailing lists. A total of 45 participants completed the questionnaire. One-hundred and fifty respondents began the survey but exited before submitting answers. This includes those that chose not to participate after

reading the consent form. Web studies have greater access to potential participants, but the highly accessible and voluntary nature of a purely remote, asynchronous web study also increases the dropout rate [25][18]. Seventeen participants provided their email addresses after the questionnaire and Part II of the experiment was emailed to them. A total of six responses were recorded for Part II.

E. Procedure

Before answering the questions, participants were asked to watch either one or two video clips, depending on their condition. All participants were given the first video clip, as an affect neutralizer [29]. Participants in the control condition were advanced to the questionnaire after the first clip, while other participants were asked to watch one additional video to administer the affect treatment. After the completion of the Part I questionnaire, participants were asked to optionally provide their email. Those that provided their email were sent Part II of the study.

After completing Part I, participants were directed to a results page, where they could view their results, compare with the aggregated results of other participants, and share their results. Part II was distributed to those who provided their email and opted in to be contacted for a follow-up. Part II was also distributed via a survey link.

F. Data Analysis

To investigate the impact of each contextual factor on strategy selection, response data from Part I was collected for each participant. Each participant's response represents one affect treatment, as well as three task and three scenario treatments, for a total of 9 unique treatments per participant. Regression models were built for each of the strategies

to analyze the relative likelihood of that strategy being used, given the task, scenario, and affect. In addition, we examined the overall frequency of each of the strategies. This analysis was performed again, on the random-order and set-order subsets of the responses. Qualitative analysis was performed for the data collected in Part II.

V RESULTS

We first examine the overall frequency of strategy selections, and then examine the impact of each contextual factors or strategy selection. Figure 4 shows the frequency at which each strategy was selected in the top two or bottom two. The green bar represents the percentage of the time the strategy was ranked as either the number one or number two choice. The red bar similarly corresponds to the last and next-to-last ranks. *Read surrounding code* was the most popular, while *Create detailed diagrams* was most frequently ranked last. Note also that the green bars shrink, while the red bars grow, moving down Figure 4. This trend supports the idea that showing the participants the strategies in a particular order may have had an impact on their choice, since this was the set order for the strategies.

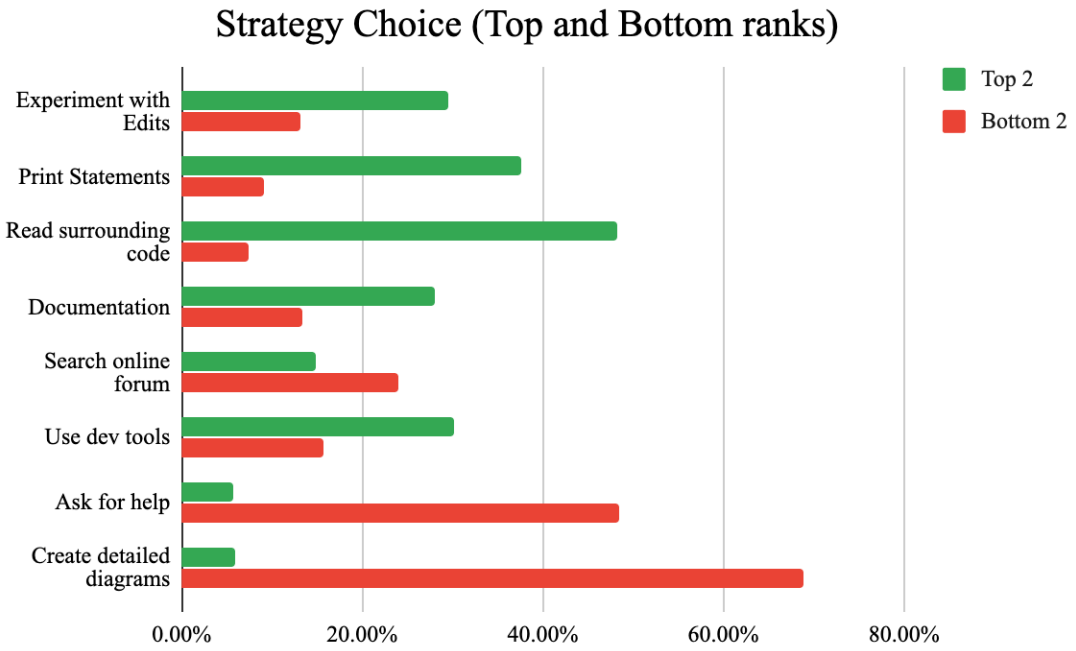

Figure 4 Top and bottom strategy choice frequency

To determine whether the randomization had an impact, the data was subdivided into responses for which the strategies were randomized, and those for which the strategies appeared in the order shown in Figure 4. Rank analysis was performed separately for each of the two subsets of data: respondents who received randomized order questions (36% of respondents) and those who received strategies in the set order. Figures 5 and 6 show these breakdowns for each strategy and rank. Each bar represents a rank, the bottom bar corresponding to the lowest rank, and the top bar to the highest rank. Comparing these charts to Figure 4 shows that the randomization had some impact on strategy selection, particularly for the strategies at the bottom of the non-randomized order. *Create detailed diagrams*, the strategy presented at the bottom of the ordered list,

went from being ranked last from 51% of the time to 36% of the time when the order was randomized.

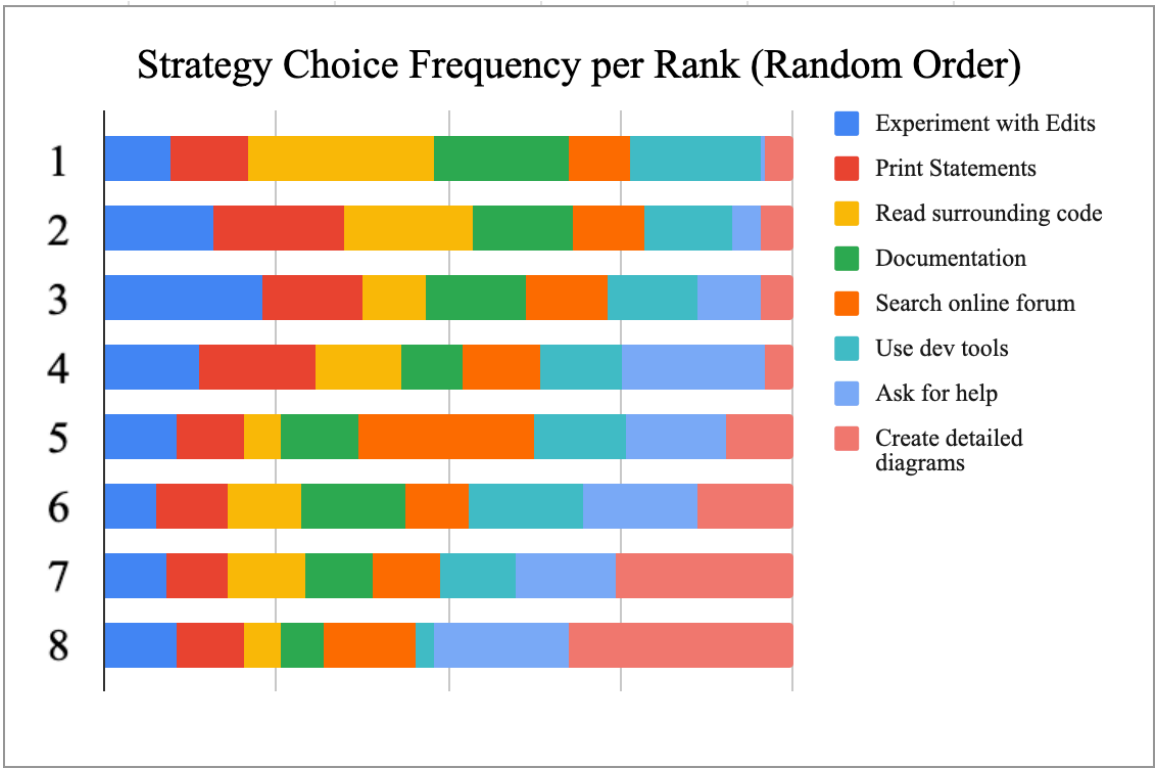

Figure 5 Per-rank strategy choice frequency for random-order subset

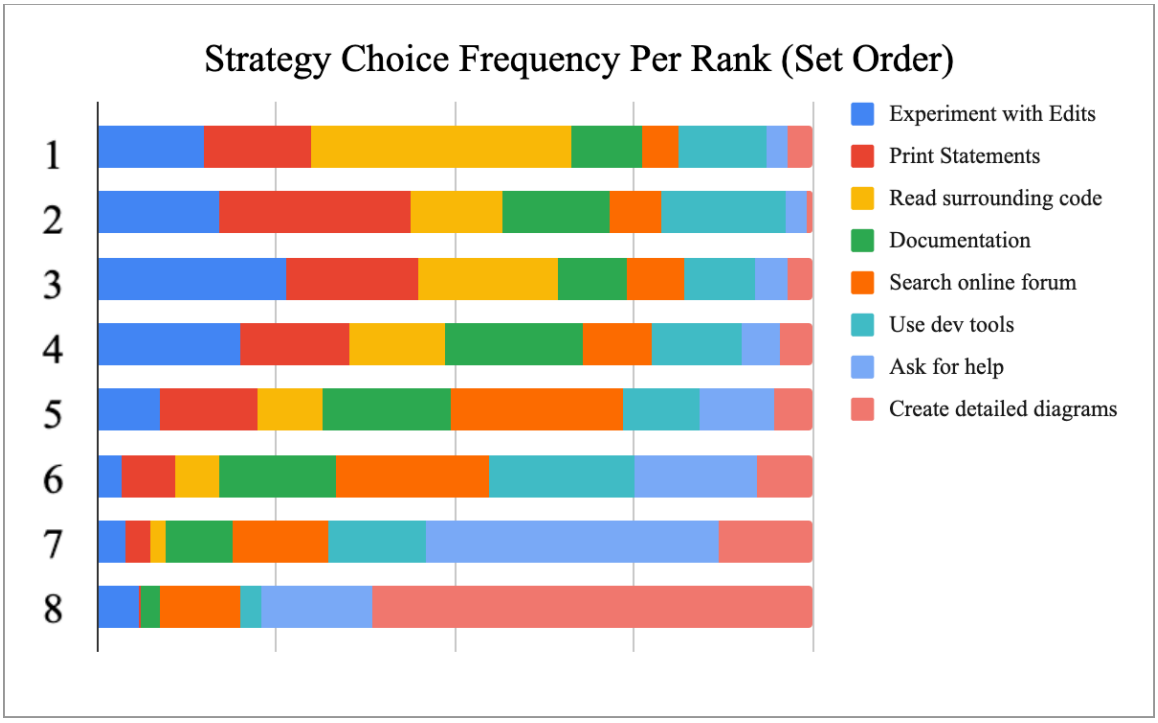

Figure 6 Per-rank strategy choice frequency for set-order subset

Next, we examine how scenario, task, and affect impacted strategy selection. A regression model was built for each strategy and the odds ratio (OR) examined to determine how much more likely that strategy would be used for different particular tasks, scenarios, and affects. Tables 3-10 depict these analyses.

RQ1: Does the type of programming task (Debugging, testing/verification, implementation) impact the strategy choice developers use?

Intuitively, different programming tasks call for different strategies. Our results offer evidence that this occurs in practice. Significant strategy choice variation was found for both non-baseline tasks. *Add print statements*, was nearly three times as likely to be used for the Debugging task than for the baseline Implementation task. *Check*

documentation was one fifth as likely to be used in the Debugging task as compared to the baseline (OR = 0.202, p = 0.000000000039337). Verification tasks were significantly more likely to involve use of the *Create detailed diagrams* strategy (OR = 1.7, p = 0.03205).

Table 3 Linear regression analysis for *Check documentation* strategy

Check documentation				
Variables	Odds ratio	SE ß	Wald	Sig. (p)
HVHA	1.45	0.29	0.37	0.2
HVLA	1.41	0.31	0.34	0.28
LVHA	0.88	0.33	-0.13	0.68
LVLA	2.1	0.36	0.74	0.04
Callback	0.81	0.22	-0.21	0.36
Framework	1.57	0.23	0.45	0.05
Debug	0.2	0.24	-1.6	< 0.01
Verification	0.4099	0.24	-0.89	< 0.01

Table 4 Linear regression analysis for *Use dev tools* strategy

Use dev tools				
Variables	Odds ratio	SE ß	Wald	Sig. (p)
HVHA	0.72	0.28	-0.33	0.24
HVLA	0.68	0.3	-0.38	2.08
LVHA	1.13	0.32	0.13	0.7
LVLA	1	0.35	< 0.01	0.99
Callback	0.73	0.23	-0.31	0.17
Framework	0.97	0.23	-0.02	0.9
Debug	2.01	0.23	0.7	< 0.01
Verification	1.64	0.22	0.49	0.03

RQ2: How does code scenario (i.e., code you own, framework code, code with callbacks) impact programming strategy selection?

Code scenario had a significant impact on strategy selection. For *framework code*, participants were 3.84 times as likely to *Search an online forum*, relative to *my code*, or code they own (p = 0.00000001818). Participants were also half as likely (OR = 0.51, p=0.0074) to use *Create detailed diagrams* when faced with the *framework* code scenario. Strategy selection for code with *callbacks* did not vary significantly from the baseline.

Table 5 Linear regression analysis for *Search online forum* strategy

Search online forum				
Variables	Odds ratio	SE ß	Wald	Sig. (p)
HVHA	0.77	0.28	-0.26	0.34
HVLA	1.66	0.3	0.51	0.1
LVHA	1.55	0.32	0.44	0.18
LVLA	0.99	0.39	-0.01	0.99
Callback	1.25	0.23	0.22	0.33
Framework	3.84	0.24	1.34	< 0.01
Debug	1.55	0.23	-0.01	-0.95
Verification	0.7	0.24	-0.82	< 0.01

Table 6 Linear regression analysis for *Create diagrams* strategy

Create diagrams				
Variables	Odds ratio	SE ß	Wald	Sig. (p)
HVHA	0.63	0.32	-0.46	0.15
HVLA	1.04	0.33	0.04	0.9
LVHA	1.18	0.35	0.17	0.63

LVLA	1.54	0.39	0.44	0.26
Callback	0.79	0.25	-0.23	0.35
Framework	0.51	0.25	-0.67	< 0.01
Debug	0.7	0.25	-0.35	0.17
Verification	1.7	0.25	0.53	0.03

In Part II of the experiment, one question asked participants to imagine certain tasks or scenarios where they would use that strategy first. Five of the six responses mention some specific task or situation where they would use a particular strategy. While some responses are related to a particular task, 4 of the 6 responses specifically describe code scenarios. For example, one participant said that they would try edits or print statements when working with an unknown language. Another mentioned that the code being authored by a particular developer would make them more inclined to reach out to that developer for help.

RQ3: How does affect impact programming strategy selection?

To examine the impact of affect on strategy choice, we built a regression model, using neutral affect as the control. Compared to neutral affect participants, participants with the two low valence treatments showed some significant changes in strategy selection. Additionally, one of the high valence treatments had an impact on strategy choice. Participants with the LVHA affect were 2.42 times more likely to *Add print statements* ($p = 0.0049$) as compared to the control condition.

Table 7 Linear regression analysis for *Add print statements* strategy

Add print statements				
Variables	Odds ratio	SE ß	Wald	Sig. (p)
HVHA	1.68	0.27	0.52	0.06
HVLA	1.29	0.31	0.25	0.42
LVHA	2.42	0.31	0.89	< 0.01
LVLA	1.25	0.35	0.23	0.52
Callback	1.42	0.23	0.35	0.13
Framework	0.68	0.23	-0.39	0.09
Debug	2.76	0.23	1.02	< 0.01
Verification	1.64	0.23	0.49	0.03

The LVHA group was also much less likely to *Read surrounding code* (OR = 0.171, p = 0.0000005454). The LVLA group also had a strong aversion to one specific strategy: *Experiment with edits* (OR = 0.09, p = 0.000000014186). This strategy was otherwise ranked higher than most of the other strategies in general (see Figure 2).

Table 8 Linear regression analysis for *Read surrounding code* strategy

Read surrounding code				
Variables	Odds ratio	SE ß	Wald	Sig. (p)
HVHA	0.63	0.31	-0.47	0.13
HVLA	0.41	0.33	-0.89	0.01
LVHA	0.17	0.35	-1.77	< 0.01
LVLA	2.27	0.44	0.82	0.06
Callback	1.1	0.23	0.09	0.68
Framework	0.57	0.24	-0.55	0.02
Debug	1.11	0.24	0.1	0.67

Verification	1.02	0.24	0.02	0.92

Table 9 Linear regression analysis for *Experiment with edits* strategy

Experiment with Edits				
Variables	Odds ratio	SE ß	Wald	Sig. (p)
HVHA	0.58	0.28	-0.55	0.05
HVLA	0.53	0.3	-0.64	0.34
LVHA	1.26	0.33	0.23	0.48
LVLA	0.09	0.42	-2.4	< 0.001
Callback	1.15	0.23	0.14	0.53
Framework	0.97	0.23	-0.03	0.9
Debug	1.41	0.23	0.34	0.13
Verification	1.22	0.23	0.19	0.39

The HVHA group, the highly activated positive affect treatment group, was 2.13 times more likely to *Ask for help from a colleague*. This strategy was generally unpopular, but the excited, enthusiastic affect correlated to the *Ask for help* strategy being ranked higher more frequently.

Table 10 Linear regression analysis for *Ask for help* strategy

Ask for help				
Variables	Odds ratio	SE ß	Wald	Sig. (p)
HVHA	2.13	0.29	0.76	0.01
HVLA	1.56	0.31	0.45	0.16
LVHA	1.43	0.33	0.39	0.28
LVLA	1.39	0.36	0.33	0.35

Callback	0.92	0.23	0.07	0.75
Framework	0.77	0.23	-0.27	0.25
Debug	0.72	0.22	-0.32	0.15
Verification	0.95	0.23	-0.04	0.83

VI THREATS TO VALIDITY

This study is subject to a number of potential threats to internal and external validity. The survey design and administration could potentially have uncontrolled factors. Initially, the survey gave each participant the list of strategies to rank for each question in a deterministic order; in other words, the participant saw the strategies initially in the same order for every question. However, some participant feedback revealed that this may have an impact on how the participants were ordering the strategies, especially so for the bottom ranks. After receiving this participant feedback, the strategies were randomized. Sixteen responses included this randomization (36%). Analyzing the data separately does show some correlation between the non-random order and the rank ascribed to the strategy; however, most of the observations from the overall dataset remained apparent in the randomized subset.

There are also some inherent threats to validity of administering the survey remotely via an anonymous survey link. We attempted to mitigate these concerns in a few ways. First, in order to ensure participants were watching the entirety of the video clips, we placed a timer in the survey that prevented advancement to the next page for the duration of the clip. While this ensures that participants cannot immediately skip past the videos, there were some usability issues, particularly on mobile devices, that made it

difficult for some users to advance to the next pages. We also limited each IP address to one response, to mitigate the potential for retakes.

Another threat to the internal validity of the experiment arise from a lack of explicit validation for some of the experimental materials. Four of the clips in this experiment were used to induce the four affect treatments, HVHA, HVLA, LVHA, and LVLA. Clips used to induce these exact treatments were validated in previous research [16]. However, two of these clips were in poor video/audio quality, so similar clips were selected for this experiment. For HVLA, the former clip, “Landscapes”, was a slideshow of nature landscapes set to soothing classical music. A clip similar to this was found online and used for our experiment. For HVHA, the former clip was a series of pranks on strangers, and the replacement was also a prank video. I did not independently validate these two replacement clips - the latter of which requires particular scrutiny. This comedic video depicts a series of pranks in which the pranksters eat food loudly in a quiet library, filming reactions of strangers as they do. We chose to use this clip because of its similarity to a HVHA treatment used previously [16].

An additional potentially confounding variable is the level of Java experience of the participant. Participants were not screened for their Java knowledge, so it is unclear what impact, if any, varying skill levels have on strategy choice.

Another concern with the internal validity of the study is the possibility of participant fatigue. Due to the same tasks and scenarios being used in different combinations, and the same eight strategies being provided for every question, the repetition could have had a confounding impact. This concern is exacerbated by the

remote participation, where participants cannot be monitored, and they may be more likely to get distracted or fatigued.

VII DISCUSSION

This study further extends the body of work to understand developer tendencies, decisions, and styles, with a broad range of applications in recommendation systems, IDE improvement, and process framework development. Developers tend to change their strategies based on some of the contextual factors we explored, but there was also a level of uniformity to the responses (mean std. dev. = 1.4), implying that there are certain strategies that are preferable, independent of any factors we explored. One possibility is that there are indeed programming styles, e.g. opportunistic, pragmatic, and systemic. A strategy like *Experiment with edits* is more likely to be used by an opportunistic developer, and opportunistic developers are more common [12][9].

Previous research on affect and decision-making has often identified a relationship between positive affect and creative thinking [26][24][14][23]. Our results showed that those with the HVHA treatment – a positive, excited affect – were 2.13 times more likely to ask for help from a colleague. Participants with either of the two negative affect treatments tended to use different strategies as well. For the low-valence, high-arousal affect (LVHA), the *Add print statements* strategy was 2.42 times more likely to be used. Negative moods have been shown in the past to correlate to evaluative, rather than generative (creative) thinking. When adding print statements, a developer already has some idea of how to proceed, or at least what information they might need to

proceed. In other words, using print statements is a more evaluative strategy than a creative one. This might contribute to its increased usage in the LVHA group. The LVHA group also experienced a sharp decrease in usage of the *Read surrounding code* strategy, an otherwise popular strategy. This strategy falls on the opposite side of the evaluation-ideation spectrum. A developer may not know what they are looking for, and the first step of idea generation is to gather information - this developer may start by reading the surrounding code. However, developers that are less likely to use this strategy may be less inclined to use such an open-ended strategy. Those in the LVHA group, perhaps, were more inclined to immediately begin evaluating their initial idea(s) rather than trying to introduce more.

The other low valence affect also had a significant impact on one strategy choice in particular. The LVLA group was significantly less likely to *Experiment with edits* (OR = 0.09). This lower-energy negative mood thus had a different impact than its higher-energy counterpart, LVHA. This could be due to the proactive nature of the *Experiment with Edits* strategy, making it unappealing to participants with a deactivated mood.

Not only are developers likely to have different strategies depending on their mood, the efficacy of actually executing those strategies changes [6]. This research could be used to provide mood-specific suggestions and support for developers, especially if mood can be measured passively as some research has shown [17].

Of the three contextual factors examined, task had the strongest impact on strategy selection. The relationship between task and strategy choice is intuitive; *Use dev tools* and *Add print statements* both were more likely to be used for the Debugging task,

while *Check documentation* was much less likely. One of the less popular strategies overall, *Create detailed diagrams*, was more likely to be used for the Verification task. This could be due to the way strategies are often generalized: with respect to tasks rather than any other contextual factor [7][19][6]. The findings of this research support this intuition; anticipating the strategies a developer might use in a given context is significantly easier with knowledge of the task at hand, even without additional context. Part II of the experiment showed that when given the opportunity to enumerate examples of tasks or scenarios, participants tended to enumerate scenarios more than tasks.

This research could be further extended by determining ways to identify the type of task being performed, and subsequently provide contextual support and recommendations to the developer. Leveraging this insight can benefit the software engineering practice as a whole, by advising the design of IDE's and plugins for features like recommendation systems, autofill and autocomplete, and other assistance mechanisms. Deepening our understanding of how developers make decisions and why strategies are chosen also provides greater clarity on impactful factors for further research, like affect. Furthermore, the strategies developed in the course of this experiment could be used in future research about developer decision-making.

VIII CONCLUSIONS

Our research builds upon the interdisciplinary body of work that examines developer behaviors and decision-making. Decision-making is performed in a context, one that can be augmented with recommendations for that context; our research sought to explore definitions, measurements, and relationships to that context that would be leverageable to improve developer experience. Understanding the strategies developers use (as well as the measurable, actionable factors that influence them) is critical as the amount of developers, technologies, and software increase to meet the demand of the modern world. As we further this understanding, systems developers use and take part in can be made more efficient, effective, and tailored to each developer, thus improving software quality and the software development practice overall.

REFERENCES

[1] Ahmed, S. and Bagherzadeh, M. 2018. What do concurrency developers ask about? a large-scale study using stack overflow. *Empirical Software Engineering and Measurement (ESEM '18). Association for Computing Machinery*. pp. 1-10.

[2] Alaboudi, A. and LaToza, T. 2020. Using Hypotheses as a Debugging Aid. *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*.

[3] Ashoori, M., Bellamy, R. K. E., and Weisz, J. D. 2015. Creating the Mood: Design for a Cognitive Meeting Room. *Human Factors in Computing Systems (CHI EA '15), Extended Abstracts.* Association for Computing Machinery. pp. 2001-2006.

[4] Bartolic, E., 1999. Effects of experimentally-induced emotional states on frontal lobe cognitive task performance. *Neuropsychologia*, vol. 37 no. 6, pp.677-683.

[5] Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., et al. 2001. *Manifesto for agile software development*.

[6] Bielaczyc, K., Pirolli, P. and Brown, A., 1995. Training in Self-Explanation and Self-Regulation Strategies: Investigating the Effects of Knowledge Acquisition Activities on Problem Solving. *Cognition and Instruction*, vol. 13, no. 2, pp.221-252.

[7] Cheney, P. H. 1984. Effects of individual characteristics, organizational factors and task characteristics on computer programmer productivity and job satisfaction. *Information & Management*, 7(4), pp. 209-214.

[8] Clapham, M. M. 2001. The effects of affect manipulation and information exposure on divergent thinking. *Creativity Research Journal,* vol. 13, no. 3-4, pp. 335–350.

[9] Clarke, S. 2007. "What is an End User Software Engineer?". Dagstuhl Seminar Proceedings. *Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany*.

[10] Cordeiro, J., Antunes, B., and Gomes, P. 2012. Context-based recommendation to support problem solving in software development. *Recommendation Systems for Software Engineering (RSSE)*. pp. 85-89.

[11] Dieste, O., Aranda, A.M., Uyaguari, F., Turhan, B., Tosun, A., Fucci, D., Oivo, M., Juristo, N. 2017. Empirical evaluation of the effects of experience on code quality and programmer productivity: An exploratory study. *ESE*. pp. 1–86.

[12] Grudin, J. 2006. Why Personas Work: The Psychological Evidence. pp. 642-664.

[13] Hewig, Johannes & Hagemann, Dirk & Seifert, Jan & Gollwitzer, Mario & Naumann, Ewald & Bartussek, Dieter, 2005. A Revised Film Set for the Induction of Basic Emotions. *Cognition and Emotion*, vol. 19.

[14] Isen, A., Johnson, M., Mertz, E. and Robinson, G.. 1985. The influence of positive affect on the unusualness of word associations. *Journal of Personality and Social Psychology*, vol. 48, no. 6, pp. 1413-1426.

[15] Jacobson, M. T.; Matthews, P. 1996. "Generating uniformly distributed random latin squares". *Journal of Combinatorial Designs,* vol.4, pp. 405-437.

[16] Khan, I.A., Brinkman, W.P., and Hierons, R. M. 2011. "Do moods affect programmers' debug performance?," Cognition, Technology & Work, vol. 13, no. 4, pp. 245–258.

[17] Khan, I. A. 2016. *Towards A Mood Sensitive IDE To Enhance The Performance Of Programmers*. LAP Lambert Academic Publishing.

[18] Krantz J. H., Dalal, R. 2000. Validity of web-based psychological research. *Birnbaum MH (ed) Psychological experiments on the internet*. pp. 35–6

[19] LaToza, T., Arab, M., Loska, D., and Ko, A. J. 2020. Explicit Programming Strategies.

[20] Leutenmayr, S. and Bry, F. 2011. Liquid decision making: an exploratory study. *Information Integration and Web-based Applications and Services (iiWAS '11).* Association for Computing Machinery. pp. 391–394.

[21] Maranzato, R.P., Neubert, M., Herculano, P. 2011. Moving back to scrum and scaling to scrum of scrums in less than one year. *Object oriented programming systems, languages, and applications companion (OOPSLA '11).* pp. 125-130

[22] Meng, M., Steinhardt, S. and Schubert, A., 2019. How developers use API documentation. *Communication Design Quarterly*, vol. 7, no.2, pp. 40-49.

[23] Phillips, L., Bull, R., Adams, E. and Fraser, L.. 2002. Positive mood and executive function: Evidence from Stroop and fluency tasks. *Emotion*, vol. 2, no. 1, pp. 12-22.

[24] Politis, J. and Houtz, J. C. 2015. "Effects of Positive Mood on Generative and

Evaluative Thinking in Creative Problem Solving," *SAGE Open*, vol. 5, no. 2.

[25] Reips, U. D. 2000. The web experiment method: advantages, disadvantages, and solutions. *Birnbaum MH (ed) Psychological experiments on the internet*. pp. 89–117

[26] Snowden, P. and Dawson, L. 2011. Creative feelings: The effect of mood on creative ideation and evaluation. ACM: Creativity and Cognition. pp. 393-394.

[27] Trimmer, P.C., Paul, E. S., Mendl, M. T., McNamara, J. M., and Houston, A. I. 2013. On the Evolution and Optimality of Mood States. *Behavioral Sciences.* no. 3. pp. 501-521.

BIOGRAPHY

Cassandra Bailey graduated from Battlefield High School, Haymarket, Virginia, in 2011. She received her Bachelor of Science from George Mason University in 2017. She has been employed as a software engineer since graduation.