Balancing Autonomy and Persona: Investigating Developer Preferences for Effective Human-Bot Interaction

by

Amir Ghorbani
M.Sc., University of Victoria, 2023

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Balancing Autonomy and Persona: Investigating Developer Preferences for Effective Human-Bot Interaction

by

Amir Ghorbani
M.Sc., University of Victoria, 2023

Supervisory Committee

Dr. Neil A. Ernst, Supervisor
(Department of Computer Science)

Supervisory Committee

Dr. Neil A. Ernst, Supervisor
(Department of Computer Science)

ABSTRACT

Software bots play a pivotal role in collective software development, promising enhanced productivity. While prior research has highlighted that excessive bot communication can lead to developer irritation, the broader array of human-bot collaboration attributes influencing developer preferences and their consequences remain less clear. This thesis delves into the key characteristics that shape developers' preferences for interactions between humans and bots, focusing on the context of GitHub pull requests.

Employing an exploratory sequential approach, we conducted interviews in Phase I, followed by a vignette-based survey in Phase II. The current thesis primarily reports on Phase II and its findings. A custom-designed vignette-based instrument was employed to survey open-source developers, recruiting participants from third-year software engineering students and the Prolific platform. Rigorous screening procedures ensured data collection from eligible participants only.

The study's results reveal a prevalent inclination among developers towards personable bots that demonstrate limited autonomy. Interestingly, the preferences appear to be influenced by developers' experience levels, with more seasoned developers exhibiting a preference for bots possessing greater autonomy. These empirical insights advocate for bot developers to enhance configurability options, allowing developers and projects to tailor bot behaviors according to individual preferences and project contexts.

Table of Contents

Supervisory Committee **ii**

Abstract **iii**

Table of Contents **iv**

List of Tables **vii**

List of Figures **ix**

Acknowledgements **xi**

Dedication **xii**

1 Introduction **1**

1.1 Bots in Pull Request discussions 2
1.2 Motivation 3
1.3 Problem Statement and Research Questions 4
1.4 Methodology 7
1.5 Contributions 7
1.6 Thesis Outline 9

2 Background & Related Work **10**

2.1 Definitions 10
2.2 Adoption and Applications 11
2.3 Benefits and Challenges 12
2.4 Perception of Bot Behavior (Phase I of the study) 16
2.5 Chapter Summary 19

3 Methodology and Experimental Setup **20**

3.1 Introduction . . . 20
3.2 Methodology . . . 20
3.2.1 Overview . . . 21
3.2.1.1 Independent Variables . . . 21
3.2.1.2 Dependent Variables . . . 22
3.2.1.3 Influential Factors . . . 23
3.2.2 Scenarios . . . 24
3.2.3 Vignettes . . . 24
3.3 Operationalization of the Constructs . . . 26
3.3.1 Autonomy . . . 27
3.3.2 Persona . . . 28
3.3.3 Design Breakdown . . . 29
3.4 Survey Deployment . . . 32
3.4.1 Stage I: User Verification and Consent Forms . . . 33
3.4.2 Stage II: Demographics and Screening Questionnaire . . . 35
3.4.3 Stage III: Survey Instructions and Scenarios . . . 38
3.5 Participant Recruitment . . . 38
3.5.1 Data Collection . . . 39
3.5.2 Demographics . . . 40
3.6 Chapter Summary . . . 42

4 Developer Perception and Preferences 43
4.1 Introduction . . . 43
4.2 Descriptive Analysis . . . 44
4.2.1 GitHub and PR Activity . . . 45
4.2.2 Bot Experience . . . 47
4.3 Statistical Modelling . . . 49
4.3.1 Linear Models . . . 51
4.3.2 Generalized Linear Models . . . 54
4.4 Open-ended Responses . . . 56
4.4.1 Model $\mathcal{M}_1$ Alignment . . . 59
4.4.1.1 Experienced GitHub Developers . . . 59
4.4.1.2 Inexperienced and Intermediate GitHub Developers . . . 60
4.4.1.3 Neutral Participants . . . 62
4.4.2 Model $\mathcal{M}_3$ Alignment . . . 62

4.4.2.1 Reactive Personable Bots 63
4.4.2.2 Proactive Factual Bots 63
4.4.2.3 Neutral Participants 64
4.5 Chapter Summary 66

5 Discussion, Future Work & Conclusion **67**
5.1 Discussion 67
5.1.1 Bot **Autonomy** and Developer's Perception 67
5.1.2 Bot **Personality** and Developer's Perception 69
5.2 Threats to Validity 69
5.3 Future Work 71
5.4 Conclusion 72

Bibliography **74**

List of Tables

Table 2.1 List of active bots on GitHub and their application[57] 13
Table 2.2 Themes, their Definitions, and evidence from the data compiled from Phase I of the study 17

Table 3.1 The merging of Persona and Autonomy constructs leads to a set of scenario combinations. It is noteworthy that each scenario is characterized by one variable being fixed at a particular value while the other variable varies between its two poles. 24
Table 3.2 demographic questions presented in the second stage of the survey instrument. 36
Table 3.3 Screening questions. 37
Table 3.4 Hiring iteration on Prolific and their respective recruitment rate. For all iterations, sex is balanced, and the only mandatory screening criteria applied on the Prolific control panel is Computer Programming. 40

Table 4.1 Distribution of 56 respondents based on GitHub and PR activity. 46
Table 4.2 Candidate models for Autonomy preferences 50
Table 4.3 Comparison of standard error (`std error`), R-squared (`r-squared`), and `p-value` of the three models. The performance of each model for each pole of a given construct (*e.g.*, Reactive for Autonomy) is detailed in the respective column. 51
Table 4.4 Coefficients and Intercept values for $\mathcal{M}_1$ on Autonomy construct (scenario 1 and 2) 52
Table 4.5 AIC scores for GLMs with binomial response distribution and `logit` link function. For each scenario (column) the lowest AIC among GLM models with various different distribution strategies is marked with a green background colour. 55

Table 4.6 The computed values of the intercept and coefficient for $\mathcal{M}_3$ as it models Scenario 3 and 4. The *'Autonomy Choice'* columns present the preference of developers for a specific Autonomy value, which is either Reactive or Proactive. 56
Table 4.7 Open coding results for the open-ended question on rationale. Frequency of observed codes in the entire dataset is presented after each polarity value. 58
Table 4.8 Examples of human-centric design rationales provided by developers who preferred Reactive and Personable bots 64
Table 4.9 Example rationales provided by developers who preferred Proactive and Factual bots 65

List of Figures

Figure 3.1 Causal model illustrating the influence of various factors on the developer's preference (codified with `VignetteChoice_Autonomy` and `VignetteChoice_Persona`) 23
Figure 3.2 The survey instrument comprises a user interface (UI) sketch that includes three distinct sections. The topmost section features a textual prompt that explains the intended focus of attention. The middle section consists of two vignettes that present conversations in the style of GitHub pull request (PR) discussions. Finally, the bottom section comprises a multi-line text input field where participants can justify their preference or lack thereof 26
Figure 3.3 Scenario 1. 29
Figure 3.4 Scenario 2. 30
Figure 3.5 Scenario 3. 31
Figure 3.6 Scenario 4. 32
Figure 3.7 Stages of survey instrument in the order which participants go through them. 33
Figure 3.8 On the left side (a), we have a JavaScript function used to generate a random sequence of scenario IDs. We first group the scenarios that cannot appear after one another and then randomly insert a different scenario in between. On the right side (b), we listed all 8 possible sequences generated by this function. Scenarios with fixed Persona construct are colored blue and red for Autonomy. Note that how no two scenarios with the same color are adjacent to each other 34
Figure 3.9 Overall flow of participants in the system. 39
Figure 3.10 Gender, Age, and Professional Experience distribution in our dataset. 41

Figure 4.1 Summary of respondent choices across both vignettes, per construct. Left plot aggregates responses for scenarios 1 & 2 while the plot on the right side aggregates responses for scenarios 3 & 4. 44
Figure 4.2 Responses broken down into individual scenarios. 45
Figure 4.3 Preference of developers conditioned on GitHub activity level . 47
Figure 4.4 Preference of developers conditioned on PR activity level 47
Figure 4.5 Preference of developers conditioned on Bot Experience 48
Figure 4.6 $\mathcal{M}_1$ visualization on Scenario 1 53
Figure 4.7 $\mathcal{M}_1$ visualization on Scenario 2 53

ACKNOWLEDGEMENTS

In the culmination of this academic journey, I stand humbled and deeply appreciative, as I extend my gratitude to the many individuals and entities that have shaped the course of this endeavor.

First and foremost, I am indebted to the University of Victoria and its Computer Science Department for bestowing upon me the opportunity to engage in this pursuit of knowledge. The academic resources, guidance, and supportive environment provided by this esteemed institution have been integral to the realization of this work.

I extend my heartfelt appreciation to the Octera Research Team, whose unwavering assistance and guidance have been a guiding light throughout this undertaking. Their expertise and collaboration were invaluable, serving as a cornerstone upon which this research was built.

To my esteemed collaborators, Nathan Cassee, Derek Robinson, Adam Alami, Alexander Serebrenik, and Andrzej Wasowski, your collective insights, knowledge, and experiences have enriched my academic growth. Your contributions have not only shaped the content of this thesis, but have also molded me into a more adept scholar.

A special tribute is reserved for Neil Ernst, my dedicated supervisor, whose support transcended the academic realm. Your patience and understanding during personal challenges were instrumental in helping me navigate the complexities of academia and life. Your mentorship has been an invaluable gift, and I am immensely grateful for your unwavering guidance.

With heartfelt thanks,

Amir Ghorbani.

DEDICATION

To my cherished friends, spanning continents and cultures, your steadfast companionship has been a guiding light. Through the challenges of academia, your unwavering support has instilled in me the strength to persevere. This endeavor is as much a reflection of our shared aspirations as it is a testament to the enduring power of friendship.

To my esteemed parents and my younger brother, whose belief in the pursuit of knowledge has been a beacon of enlightenment, I owe immeasurable thanks. Your dedication to fostering curiosity and nurturing intellectual growth has shaped the scholar I have become. This dedication stands as an acknowledgment of your sacrifices and an expression of the profound respect and love I hold for you all.

To the emblem of my homeland, Guilan, and the indomitable Mount Dorfak, you embody the essence of resilience against all odds. Just as you weather the storms with unwavering fortitude, so too does this thesis carry with it the spirit of determination and the desire to overcome. This dedication is a tribute to the land that has shaped my identity and to the mountain that stands as a symbol of the heights I can achieve.

Lastly, to the embodiment of Woman, Life, Freedom movement—may your triumphant spirit lead the way to a glorious revolution. In dedicating this work to the pursuit of a free and democratic Iran, I lend my voice to the chorus that yearns for liberty. May this thesis be a small yet meaningful step towards a future where happiness and democracy flourish.

With profound dedication and hope,

Amir Ghorbani.

Chapter 1

Introduction

Social coding platforms such as GitHub[1] have provided a valuable toolset for developers around the world to engage in open-source, collaborative, and transparent software development. GitHub is a web-based platform that provides version control and collaboration tools for developers. It allows individuals and teams to easily manage and track changes to their code projects. Using GitHub, developers can create and maintain repositories to store their code. They can track and manage different versions of their code using a version control system called Git[2], which allows them to keep a record of changes made over time. GitHub also offers features for issue tracking, code review, and project management, making it a comprehensive platform for software development.

In the context of GitHub, a pull request (PR) is a mechanism for proposing changes to a codebase and initiating a discussion among developers before merging those changes into the main project. It involves forking the original repository, creating a new branch, making desired modifications, committing changes, and finally submitting the pull request. Other developers can then review the proposed changes, provide feedback, and engage in a collaborative discussion. The project owners or maintainers can decide to merge the pull request, incorporating the changes into the main codebase, or close the pull request without merging if deemed inappropriate or unnecessary. Pull requests facilitate collaboration, code review, and help maintain code quality in software development environments[38, 32].

Indeed, human developers are not the only key participants in the software development process or the discourse surrounding pull requests. Through its Bots API,

[1] `https://www.github.com`
[2] `https://www.git-scm.com`

GitHub empowers developers to craft software entities, herein referred to as 'bots', that can be employed in various facets of cooperative open-source software development. These bots are software applications with variable degree of autonomy, functioning within the GitHub ecosystem to execute specific tasks [51]. They are capable of automating a wide array of activities, including, but not limited to, reacting to particular events such as pull requests, initiating and committing code modifications, generating issues, and providing feedback to contributors[24].

There are numerous real-world examples of bots functioning on GitHub. Consider `Dependabot`[3], which streamlines the process of updating project dependencies by initiating pull requests whenever a newer version of a dependency is accessible. `Probot`[4], a framework constructed using Node.js, streamlines the creation of GitHub Apps designed to manage automated operations, such as tackling stale issues, assigning tasks, or auto-responding to comments. Furthermore, the `Travis CI bot`[5], which is connected to the Travis CI continuous integration service, initiates a build process whenever a commit is enacted or a pull request is created, offering instantaneous feedback to developers concerning their code modifications.

In this research study, our focus is centred on the interactions between these bots and human developers (hereafter referred to as 'developers') within the context of pull request discussions on GitHub. **Specifically, we delve into the potential influence of various bot actions or behavioral characteristics on the preferences of developers.**

1.1 Bots in Pull Request discussions

In a standard pull request discussion, the process involves several key activities. It typically begins with code review, where team members examine the proposed changes to ensure they align with the project's standards and guidelines. This review often prompts a robust discussion about the changes' implications, possible improvements, or alternative solutions. The developer may commit additional changes based on this feedback, which are again subject to review. Automated tests, often run by bots, evaluate the stability and reliability of the changes. Once the participants approve the reviewed changes and all checks pass, the pull request gets approved. Finally, the

[3] `https://www.github.com/dependabot`
[4] `https://www.probot.github.io/`
[5] `https://www.travis-ci.com/`

approved changes are merged into the target branch, most often the main or master branch of the project.

Alternative activities during a pull request can enrich the process. Tools like issue linking, labels, draft pull requests, or external tools such as `GitKraken`[6] or `Sourcetree`[7], can facilitate the review process. For example, draft pull requests allow for progress sharing and discussion without initiating a formal review. Labels can efficiently categorize and manage pull requests. And issue linking can provide additional context for the changes. These alternative activities can vary depending on the team's preferences and project's needs.

Pull request discussions represent a complex medium of communication wherein text messages comprise a fraction of the overall dialogue[7, 13]. Developers engage in communication through numerous features including attaching labels to pull requests, allocating specific issues to other developers or flagging them in a comment for a response, appending emojis to comments, and of course, employing natural language as a mode of interaction. Bots of diverse complexity also partake in the dialogue, each acting in alignment with its unique objectives and settings. Certain bots exhibit unique traits that distinguish them from basic automation scripts, such as the ability to employ context-aware natural language for communication[52]. Some posses high computational power, leveraging machine learning algorithms for analytical tasks[25]. Beyond their demonstrated intellectual capabilities, some bots may also appear to exercise autonomy in their actions and may engage in communication via an established personality[46].

1.2 Motivation

Given the capabilities, level of autonomy, and displayed personalities of bots, they participate in pull request discussions and interface with developers as distinct entities. However, given the relative novelty of software bots, we currently lack a comprehensive understanding of the behavioural dynamics that influence developers' acceptance or rejection of collaboration with bots. Existing research primarily concentrates on understanding specific bots[33] characterizing types of bots[48, 52], and analyzing bot actions[36, 54], with less focus on the process of human-bot collaboration. While it is established that excessive notifications from bots can annoy developers[55], it

[6] `https://www.gitkraken.com`
[7] `https://www.sourcetreeapp.com`

remains unclear which additional aspects of bot communication hold significance for these developers.

Furthermore, the increasing complexity of bots, achieved by advancements in natural language processing and text generation, introduces an additional layer of complexity to human-bot interaction. As these bots acquire more anthropomorphic traits, they can present a facade of human-like personality[20]. This evolution in bot capabilities not only enriches the potential for more nuanced interaction but also necessitates a deeper exploration of its implications on the developer-bot dynamic.

1.3 Problem Statement and Research Questions

AUTONOMY: refers to the extent to which a bot can function independently, without requiring direct involvement or intervention from humans. Empirical evidence suggests that bots can significantly enhance the productivity of developers[36]. Developers can delegate monotonous and repetitious tasks to bots, as is the case with unit testing where a bot can automatically conduct unit tests on new changes committed to a pull request or perform commands issued by developers. Depending on their configuration by repository maintainers, bots can function with complete autonomy. This level of autonomy can streamline numerous processes, boosting developer productivity and reducing errors that arise from overlooked actions[54]. However, fully autonomous bots also have the potential to introduce challenges that could negatively impact developers' preferences through the generation of noise[56]. Unregulated autonomous bots can cause annoyance by issuing overly frequent notifications or initiating an excessive number of pull requests. Furthermore, the complete autonomy of bots complicates the attribution of responsibility when a bot inadvertently harms the project, particularly when crucial tasks such as bug detection and patching, or security screening, are delegated to the bot[43]. Consequently, we pose the following question:

RQ-1: ***How does the degree of Autonomy influence a developer's Preference for the bot's actions in a pull request discussion?***

Given the advantageous and challenging aspects of bot utilization, our first Research Question (**RQ-1**) seeks to delve into the interactions between developers and bots characterized by varying degrees of autonomy. Our aim is to ascertain if devel-

opers exhibit preference towards a specific bot solely based on its level of autonomy. In particular, we try to determine whether developers show a preference for Reactive bots, which require invocation from a developer to execute a task, or Proactive bots, which autonomously engage and participate in pull requests. Yet, this delineation does not clearly distinguish Proactive from Reactive bots. To illustrate, consider a bot that reacts to a sequence of events and actions initiated by a developer. Even though the bot's behavior is contingent upon a series of actions and events unfolding across different intervals, we argue that the notion of traceability plays a key role in determining a bot's degree of autonomy. We refine our definition of a Proactive bot based on the ease of tracing the originator developer who instigated the action sequence. If the initiating developer and the consequential actions leading to the bot's response can be readily discerned within the context of the ongoing discussion, we categorize the bot as Reactive. Conversely, if extensive scrutiny beyond the immediate discussion is requisite to ascertain the origin, we classify the bot as Proactive.

RQ-1.1: ***What elements factor into developers' preference for Proactive versus Reactive bots?***

Sub-Research Question 1.1 (**RQ-1.1**) expands upon **RQ-1** with the goal of discerning what factors sway a developer's inclination towards opting for a Reactive or Proactive bot. The relevance of this question rests on the existence of a significant correlation between the degree of bot autonomy and the developer's preference. Specifically, we converge on several factors including levels of GitHub activity, pull request activity, previous bot interaction experiences, and years of programming experience, and scrutinize whether they exert influence on a developer's choice.

PERSONA: An additional critical aspect of bot-developer interactions is the perceived personality of bots[34]. Persona is characterized as the collection of behaviors and traits exhibited by a bot, which are perceived as a cohesive personality entity. Bots can acquire certain anthropomorphic characteristics, such as the use of informal language or colloquialisms in their communication with developers, the inclusion of emojis in their responses as a means of expressing emotion, or the possession of a humanoid profile that enables them to portray a personality [20, 21]. However, there is currently limited research on the range of personalities that bots may exhibit in pull request discussions, and the dynamic between a bot's personality and developer

preferences remains unexplored. Given the global and diverse backgrounds of open-source developers on GitHub, investigating the exhibited personalities of bots and their impact on developer preferences constitutes the second objective of this study:

RQ-2: ***How does the degree of Persona influence a developer's Preference for the bot's actions in a pull request discussion?***

In Research Question 2 (**RQ-2**), mirroring the approach in **RQ-1**, our objective is to examine bot personalities and evaluate their impact on developers' preferences. Specifically, we focus on two personality types that bots can adopt and present to developers: Factual and Personable bots. We define Factual bots as those that behave as bot-like entities, communicating in a concise manner, while Personable bots appear more human-like and display a distinctive character. A noteworthy distinction between these two personality types is that while Factual bots exhibit uniformity in action and communication, each Personable bot may possess a unique character, imbuing it with more human-like attributes as opposed to the sterility of a mechanistic entity.

RQ-2.1: ***What elements factor into developers' preference for Personable versus Factual bots?***

In Sub-research Question 2.1 (**RQ-2.1**), we further dissect the factors influencing developers' preferences between Personable and Factual bots. This question centers on identifying the specific elements that dictate a developer's inclination towards either a bot with a more human-like, personable persona or a bot with a factual, straight-to-the-point demeanor. To maintain experimental consistency, we employ the same parameters used in **RQ-1.1**. Aspects such as levels of GitHub activity, pull request activity, and previous experiences with bot interactions are considered to discern if they influence a shift in developers' preferences towards either the Personable or Factual bot personalities. This systematic approach allows us to maintain a coherent analytical framework across different aspects of our study.

1.4 Methodology

This thesis reports on the second part of a two-phase mixed-method[15] study. The initial phase was a comprehensive, exploratory interview study that aimed to understand which characteristics of bots shape human perceptions of bot behavior. Through these interviews, researchers gained insight into the types of bots respondents were familiar with. Researchers then delved into the behavioral aspects of bots that respondents favored. Interviewees revealed that even when bots were performing tasks of high utility, certain elements of the bot's behavior influenced developers' perceptions. These conversations culminated in the identification of a set of themes, emphasizing two key factors that impact perception: the extent of a bot's autonomous actions (Autonomy), and the personality portrayed by a bot (Persona)[23]. This understanding set the groundwork for our phase two study.

In the second phase of the study, we created a vignette-based survey instrument to evaluate the impacts of the two constructs (*i.e.*, Persona and Autonomy) on user preferences. A vignette is a graphic illustration that presents a hypothetical situation, to which research participants respond and reveal their perceptions. Given the bipolar nature (contrasting states) of each construct, four unique combinations emerge, termed as "Scenario". Within each scenario, one construct remains constant, while the other alternates between its states. In this survey, each scenario exposed respondents to two almost identical vignettes juxtaposed for comparison. For instance, consider a scenario where *Persona* remains consistent at the "Factual" level, yet one vignette showcases a "Proactive" bot, whereas its counterpart depicts a "Reactive" bot. These scenarios portrayed a bot participating in a pull request discussion within a software project on GitHub. Crucially, we altered the bot's behavior between the two scenarios to highlight either autonomy or persona, thereby representing different extremes of each construct. We then prompted developers to indicate their preference between the two illustrated bot behaviors.

1.5 Contributions

The primary contributions of this research are as followed:

1. We identify and define the constructs of bot Autonomy and Persona, coupled with a detailed operationalization of these constructs. This provides a solid

framework for understanding the varying behaviors and characters that bots can possess.

2. We assess the impact of bot Autonomy and Persona through a custom-developed vignette-based instrument. This tool enables a realistic and controlled study of bot behavior, enhancing the accuracy and applicability of our findings.

3. We present a series of statistically validated models, providing quantitative evidence regarding developers' preferences for specific bot behaviors. Our results suggest that developers tend to favor reactive, personable bots, marking a significant insight into the preferred design of bot interaction strategies.

4. We offer empirically-grounded, actionable recommendations to guide future development and use of bots in software development activities. Among these, we propose that the autonomy and persona of bots should be flexible and configurable according to the preferences and specific tasks of developers.

5. Lastly, our findings open new avenues for further research into the design of bot personas for specific projects and tailoring these personas for different developer demographics. This sets the stage for focused investigation into what constitutes an acceptable level of bot autonomy.

1.6 Thesis Outline

This thesis is presented in the following structure:

chapter 2 begins by clarifying the definition of software bots, specifically focusing on the ones defined by Erlenhov[19] and Storey[48]. It traces the evolution of bots and their multifaceted applications, then delves into the challenges and benefits associated with their use. The chapter concludes with an exploration of how developers perceive bot behavior, particularly in terms of autonomy and persona.

chapter 3 presents the study's research methodology and experimental design in detail. It explains why a quantitative experimental study was chosen, focusing on the advantages of this approach for supporting the developers' perception study. This chapter also illustrates the survey design process and the operationalization of Autonomy and Persona constructs. The process of survey deployment using SurveyMonkey, the participant recruitment strategies, and the participant screening processes are detailed. Lastly, it provides a thorough account of the data collection, cleaning, and anonymization process.

chapter 4 offers a detailed analysis of the quantitative data gathered, as well as an examination of the open-ended responses from the study. It delves into the descriptive analysis, followed by a comprehensive statistical analysis using Linear Models (LM) and Generalized Linear Models (GLM)[35]. The chapter also explains the adopted coding strategies for open-ended responses. The chapter concludes by presenting the findings of these analyses.

chapter 5 engages in a thorough discussion of the results presented in the previous chapter. It addresses potential threats to the validity of the study, highlighting the steps taken to mitigate these. The chapter also paves the way for future research directions based on the findings of the study. Lastly, it draws a conclusion, summarizing the key points and the overall significance of the research.

Chapter 2

Background & Related Work

2.1 Definitions

The concept of software bots encompasses a wide range of applications, each with specific goals that make it challenging to establish a single comprehensive definition. Consequently, various factors such as interactability, intelligence, personality, and operating context need to be taken into account when defining bots. Erlenhov *et al.* highlight the difficulty of achieving a unified definition due to divergent perspectives among industry practitioners and researchers[18, 19]. However, Seiffer *et al.* propose a definition that characterizes bots as software systems capable of autonomously emulating human behavior to perform tasks that were previously handled by humans[45]. In a similar definition, bots are considered autonomous agents existing within a software environment, making rational decisions independently, and assuming human roles and responsibilities[19, 5]).

A more generic definition describes a bot as an interface that delivers services to the user, encompassing all the necessary components to provide such services. These services can be either internal or external to the bot, and can involve interactions with humans or even other bots. The use of software bots contributes value to services in numerous ways, including facilitating access and enabling automation[29]. Another study suggests that bots are interactive and intelligent services primarily aimed at enhancing developers' productivity[57]. While these definitions are broad, the key themes of interactability and intelligence consistently emerge as distinguishing characteristics of bots in contrast to conventional development tools[19]. It is important to note that tools that solely perform actions upon human invocation are not typi-

cally considered bots. Similarly, autonomous or intelligent agents functioning in the background without interacting with humans are not typically classified as bots[18].

Within this study, our attention is directed towards the definitions proposed by Storey and Zagalsky[48]. They define a software bot as a conduit or interface connecting users with services, typically through a conversational user interface (UI). Additionally, we incorporate the categorization of bots into three types, as identified by Erlenhov *et al.*, through interviews conducted with developers. The developers' definitions of bots varied based on their perception of the bot's primary utility, whether it was focused on facilitating **Chat** interactions, performing **Smart** actions, or demonstrating **Autonomous** behavior[19].It is imperative to highlight that this research does not delve into the intelligence quotient of a bot, and such a metric remains excluded from our definition.

2.2 Adoption and Applications

Bots play a significant role in open-source software (OSS) projects, serving various purposes and applications. In the context of repository maintenance, bot adoption by maintainers typically revolves around several categories, each serving specific purposes. These categories include code bots, test bots, DevOps bots, support bots, and documenting bots [48] and are defined as:

- **Code bots** are employed to enhance the efficiency and effectiveness of coding activities. They can assist in automating repetitive coding tasks, performing code formatting, suggesting code improvements, and even automating code reviews.

- **Test bots** are utilized to streamline the accessibility of complex static analysis tools. They help alleviate the burden of manual testing and can automatically run tests, perform static code analysis, and identify potential bugs or vulnerabilities in the codebase.

- **DevOps** bots focus on expediting the code deployment process. They automate various stages of software delivery, such as building, testing, and deploying code changes. DevOps bots help ensure smooth and efficient deployment pipelines, allowing for faster and more reliable code releases.

- **Support bots** play a crucial role in bridging the communication gap between users and developers. They can provide assistance, answer common questions, and offer guidance to users.

- **Documenting bots** contribute to the generation of documentation based on code commits and issue comments. They automatically extract relevant information from these sources and generate documentation, helping to keep project documentation up-to-date and reducing the manual effort required for maintaining comprehensive documentation.

These various categories of bots are adopted by repository maintainers to streamline development processes, enhance code quality, improve communication with users, automate tedious tasks, and ensure efficient documentation practices. An example of such bots is Dependabot, which specializes in managing dependency updates. However, there are also bots capable of undertaking more complex responsibilities, including code review, static analysis, bug detection, and even patching, leveraging artificial intelligence techniques[52].

An illustrative example of an intelligent bot is `SpotBugs`[1], which conducts static analysis on Java source code to identify bugs and detect poor coding practices. Bots can also be utilized for information categorization[48]. For instance, bots like `google-cla-bot` and `oca-clabot` are employed to identify contributors who have not signed the necessary Contributor License Agreements (CLA) for a particular project.

It is evident that bots come in various forms and serve diverse functions. Table 2.1 (adopted from Wyrich *et al.*) showcases the top 11 active bots on GitHub, ranked based on the number of pull requests they generate[57]. This metric is used as an indicator of bot activity and impact within the GitHub ecosystem. The table presents a summary of the bots' use cases, highlighting their specific roles and functionalities in different projects.

2.3 Benefits and Challenges

BENEFITS: Bots in software development offer several benefits that enhance productivity, collaboration, and code quality. By automating code review metrics and providing additional information, bots enhance feedback to developers, allowing them

[1] `https://www.spotbugs.github.io/`

#	**Bot Name**	**Use-Cases**
1	dependabot	Dependency update
2	pull	Keeping forks updated with upstream via automated PRs
3	renovate	Dependency update
4	pyup-bot	Dependency update for Python-based project
5	greenkeeper	Dependency update for JavaScript and NPM-based projects
6	snyk-bot	Dependency update
7	imgbot	Image optimization and compression
8	github-learning-bot	GitHub learning lab
9	everypoliticianbot	Collects data for EveryPolitician dataset
10	depfu	Dependency update for JavaScript and Ruby projects
11	scala-steward	Dependency update for Scala projects

Table 2.1: List of active bots on GitHub and their application[57]

to make informed decisions and improve their contributions[54]. Furthermore, bots reduce the effort of maintainers on trivial tasks, freeing up their time to focus on more important responsibilities. This reduction in manual labor leads to improved code quality and a higher standard of contributions[52, 51]. Bots also contribute to increased efficiency by automating routine tasks and streamlining the code review process, resulting in faster turnaround times and improved productivity[53, 54, 9].

Additionally, bots promote consistency and standardization by enforcing coding guidelines and best practices[53, 18]. They ensure uniformity across projects, reducing errors and facilitating collaboration among team members. By handling tedious tasks such as code testing and continuous integration, bots improve the efficiency of these processes, saving time for developers. They can analyze data and provide valuable insights, aiding decision-making and process improvement[52]. Bots not only increase productivity but also offer scalability, and autonomy[19]. With their technical competence and learning capabilities, they become valuable contributors to software development, continuously improving their performance over time[18].

Moreover, bots are designed with pleasant and engaging personalities, making interactions enjoyable for users[28] and software bots that exhibit higher levels of anthropomorphism have been observed to foster trust among human users by creating a sense of familiarity[45, 30, 39, 27, 12, 11]. They integrate with communication channels, seamlessly providing access to software services. With their widespread adoption and advancements in natural language processing[19, 28], bots have become the de facto standard for interacting with software services. They offer flexibility in classification and creation, accommodating various interaction models, intelligence levels,

and purposes. Transparent interactions and clear identification as bots contribute to building user trust and further increasing their adoption rates[28].

CHALLENGES: The use of bots in software development is accompanied by several challenges and disadvantages that need to be addressed for their effective implementation. One of the most prevalent challenges is the introduction of noise[56, 54, 19] to human communication and development workflow, which can disrupt team cohesion and increase distractions[48, 31]. Additionally, there is a risk of unintentional triggering or misuse of bot capabilities by developers, potentially leading to inaccurate actions or sub-optimal feedback[56]. Bots may also lack contextual understanding, resulting in non-comprehensive feedback and an inability to provide actionable suggestions, diminishing the usefulness of their feedback[53, 56].

Another challenge is the breakdown of expectations regarding bot behavior, making it difficult to enforce inflexible rules and ensure consistent interactions[56, 28]. This can be particularly intimidating or unfamiliar for newcomers to open-source software (OSS) communities, raising concerns about bots intimidating or impersonating human contributors and causing stress among real contributors[19, 56]. User acceptance and trust are crucial factors, as some team members may be resistant or skeptical about relying on bots, necessitating efforts to build trust and ensure their acceptance[45, 19, 28]. Bots also need to demonstrate adaptability to handle unique or dynamic scenarios, and they should be seamlessly integrated into existing workflows and systems, which can be technically complex and time-consuming[56, 52, 18].

Maintenance and updates pose additional challenges, as bots require regular attention to stay relevant and effective, imposing additional overhead on development teams[56]. Ethical considerations are also important, including privacy, security, and fairness, to avoid potential risks or biases associated with bots[56]. Furthermore, limited decision support mechanisms can lead to potential errors or inadequate feedback, impacting the quality of the code review process[52, 48]. Challenges in social interactions among developers may arise, highlighting limitations in bots' ability to assist in non-technical aspects and improve social dynamics within teams[52, 19].

Improvements are needed to enhance bot capabilities, such as smarter learning mechanisms and better incorporation of user feedback[18]. Trust is a significant factor, as some personalities portrayed by bots may struggle to gain developers' trust in tasks that require deep understanding or modifications to code[28, 19]. Usability is another challenge, especially in achieving a balance between conversational ease and efficient task execution[19]. The specialization of bots may limit their broader capabilities[18],

contrasting them with non-specialist counterparts like Siri[2] or Cortana[3].

Technical and social complexities must be considered to ensure that bots possess the necessary technical competence, understand project requirements, and exhibit appropriate social behaviors[18]. User adoption is influenced not only by task accomplishment abilities but also by the personality and likability of bots[28]. Language and user interaction play crucial roles in user experiences, and careful design is required to prevent potential misuse and maintain user trust[19, 28]. Clear guidelines and more research on the negative impacts and challenges posed by bots are necessary to navigate the design process effectively[48].

2.4 Perception of Bot Behavior (Phase I of the study)

This work represents the Phase II of the study[23] and builds upon the findings obtained in Phase I. In Phase I, an exploratory sequential design was employed to address the question of *"What bot characteristics influence human perceptions of bot behavior?"* The qualitative phase involved conducting interviews to gather data, which then informed the subsequent quantitative phase (Phase II). The research team interviewed twelve open-source software developers from four distinct communities[4], aiming to gain a comprehensive understanding of their perceptions regarding bot behavior within open-source communities.

During Phase I, the interviews yielded preliminary insights into human perceptions of bot characteristics and behavior. The interviews culminated in the emergence of seven central themes: *Attitude*, *Autonomy*, *Persona*, *Task*, *Feelings*, *Project norm*, and *Role*. These themes, along with their definitions and representative quotations from the interviewees, are delineated in Table 2.2 (original table extracted from the paper):

> "The respondents identified that bots play significant *roles* in their projects, being seen primarily as assistants. Bots have a big influence on community perceptions and *project norms*, leading people to *feel* different emotions, whether annoyed, discouraged, or happy, among others. When we look at the *attitudes* behind these feelings, the biggest were how independent the bot was, and how it came across. Both *autonomy* and *persona* affected the interviewees *feelings* and *attitudes* towards a given bot[23]."

[2] `https://www.apple.com/ca/siri/`
[3] `https://www.microsoft.com/en-us/cortana`
[4] FOSSASIA, ROS, Coala, and RTEMS

Theme (freq.)	Definition	Evidence from the data
Attitude (104)	How a human perceives bot actions	*"So, it's totally fine with me if the [bot] reviews and the bots review and say, you know, this is not fine"* (P10).
Autonomy (41)	Does a human have to invoke/moderate a bot	*"... if [a bot] just goes and tells you something's broken, it's not as good as if it can, like, fix it for you"* (P8).
		"a bot is only as good as the automation it gives you" (P8).
Persona (67)	Aspects of a bot's character as perceived by a human	*"It [a bot] should be a little cool. Some sarcasm, some funny joke so to keep up the energy and enthusiasm in development"* (P3).
Task (144)	Tasks which a bot can perform	*"the bot has been configured to automate the various tests. So, let's say you create a pull request, then what happens that depending on the dependencies, and the CI/CD workflows of the of your particular project, whatever pull request someone is making,...And they might be using some custom tests like Travis CI, so that is something that these custom bots do it can run some tests that are being built particularly for that particular organization*" (P1).
Feelings (28)	Emotional response of a human evoked by either a bot or a human	*"... in case if it is a bot [reviewing the PR] ... and he's [sick] simply rejecting my PR, then I will feel a little bad, and my perception towards that bot will actually change."* (P3).
Project Norm (32)	Implicit practices within an OSS project	*"RTEMS is actually a community that works over emails, as of now we don't use bots"* (P12)
Role (14)	The role(s) a bot or human play within a project	*"So, this bot can do [style checks]. It is time-consuming to ask the reviewer for his review ... We can say the bot becomes one of the reviewers. "* (P2).

Table 2.2: Themes, their Definitions, and evidence from the data compiled from Phase I of the study

These findings not only align with the existing literature on bot-human interaction[52, 53, 19] but also emphasize two prominent themes, namely Autonomy and Persona, which served as guiding factors for the design of the second phase. The data collected in Phase I demonstrated that the themes of Autonomy and Persona evoked strong emotional responses among the interviewees:

"In Phase 1 data, the themes of **Autonomy** and **Persona** seem to evoke the strongest reactions amongst our interviewees. For example, the lack of rational and factual explanations subsequent to bot actions can have ramifications on how the developer feels towards the bot's actions. P3 expressed disappointment if a bot would reject his PR with no justification for such action. They stated: *"in case if it is a bot doing the task, and a bot is not commenting on what error I made, and he's simply rejecting my PR, then I will feel a little bad, and my perception towards that bot will actually change"* (P3). P9 expressed stronger action: *"I think as a newcomer, it feels a lot worse, because if it's my first contribution and a bot rejected me, I'm not contributing more to this project. I'm done with this project* (P9).

Similarly, too much autonomy takes joy out of the process of contributing. For example, P2 and P6 explained that they would not feel the same "joy" if their PRs were accepted by a bot. They explained: *"so, the thing with PRs is that you get the joy of finally getting merged after a lot of suggestions from the reviewers. And that joy is really not comparable for a bot just merging it with any reviews."* (P2) and *"I might have probably been more happier [sic] if a human being accepted my PR"* (P6)[23]."

2.5 Chapter Summary

In this chapter, we have provided a comprehensive examination of the benefits and challenges associated with the integration of bots in software development processes (section 2.3). Notably, the benefits include enhanced productivity, collaboration, code quality, and developer experience, as facilitated by bots' capabilities to automate routine tasks, offer valuable insights, and streamline code review processes[54, 52, 51, 9]. However, despite these advantages, there are inherent challenges that warrant attention. These encompass noise introduction, lack of contextual understanding, expectation breakdown, and concerns over acceptance and trust among team members[56, 54, 19, 48, 31].

Phase I of the study (section 2.4), established during earlier research[23], sought to understand the human perception of bot characteristics in software development, especially within open-source communities. The study employed an exploratory sequential design to address the central question of what bot characteristics influence human perceptions of bot behavior. Interviews conducted with twelve developers from various open-source communities brought to light seven significant themes influencing these perceptions: Attitude, Autonomy, Persona, Task, Feelings, Project norm, and Role.

The implications of bot Autonomy and Persona were particularly significant, eliciting strong emotional responses from the interviewees. Developers expressed feelings of disappointment and frustration when a bot autonomously rejected their PR without justification, potentially leading to decreased contribution. Similarly, the joy of having a PR accepted was diminished when the action was performed by a bot rather than a human reviewer.

In conclusion, while bots have become integral to software development and offer numerous advantages, their use and implementation are not devoid of challenges and complexities. Understanding human perceptions of bot behavior, as highlighted in the preliminary phase of the study, offers valuable insights into the aspects of bot Autonomy and Persona. These insights provide a foundation upon which to develop strategies for effective bot-human collaboration and underline the necessity for further research to address the inherent challenges associated with bot integration in software development processes.

Chapter 3

Methodology and Experimental Setup

3.1 Introduction

This chapter will go over the research methodology that was used to answer **RQ-1** (*do developers prefer reactive or proactive bots?*) and **RQ-2** (*do developers prefer personable or factual bots?*) (section 3.2). Then we will go over the experiment design aspect of our survey instrument, which enabled us to operationalize the Autonomy and Persona constructs (section 3.3). Additionally, we will show how our design helped us to address **RQ-1.1** and **RQ-2.1** (*how developers' preference of bots' attributes differs conditioned on their background and expertise?*). Prior to data collection and processing that resulted in the compilation of a dataset (subsection 3.5.1), we will explain the deployment of our Custom Developed Survey Instrument (CDSI) (section 3.4), followed by a discussion of participant recruitment policies and strategies (section 3.5). Finally, we will conclude this chapter with a summary of the methodology and experimental setup (section 3.6).

3.2 Methodology

In this study, we employed a quantitative research methodology to examine the relationship between Autonomy, Persona, and the Perception of developers as they interact with each other. Quantitative research is a methodology that involves the collection and analysis of numerical data, and surveying is a common method of

collecting quantitative data. The benefits of quantitative research and surveying include objectivity, generalizability, precision, the ability to test hypotheses and identify cause-and-effect relationships, and ease of analysis and comparison[3].

A vignette-based survey instrument was used as the primary method of data collection. The vignette-based survey, also known as a scenario-based survey, is a method of collecting data[4] where participants are presented with a short fictional scenario and asked to respond to it. This method allows researchers to elicit responses on a particular topic in a more naturalistic and realistic way. For example, in this study, we used vignettes that represented different scenarios identical to GitHub pull request discussion pages, where each vignette imitates various types of interactions observed in the real world between a bot and a developer.

These interactions are categorized as **A)** written conversation: communication among bots and developers by exchanging messages through written text in the English language, **B)** emoji responses: colloquially known as Reaction or *Gitmoji* is a very concise way of communicating where participants of the conversation can reply or assign emojis to a message, conveying a specific meaning with respect to the context of the conversation, and **C)** administrative actions: these are decisions made by the participants of the conversation such as opening or closing a pull request, adding labels and tags, or assigning developers (or bots) to a particular task. Then, participants were asked to respond to each vignette, rating their perception of the bot as a developer described in the scenario. The participants were carefully selected from a pool of developers who possessed prior experience in open-source programming on GitHub and had previously interacted with bots.

3.2.1 Overview

3.2.1.1 Independent Variables

Before designing vignettes, we first began with the two Independent Variables (IV) of our experiment. Autonomy and Persona are the two constructs that we aim to map onto our desired variables. We specified two opposite poles (extremes) for each of our constructs, as discussed in section 1.3. For Autonomy, **Proactive** bots can act independently without human intervention, whereas **Reactive** bots respond to a prompt from a human developer and their actions are determined by a human moderator. For example, a Proactive bot may open an entire PR autonomously to patch a bug whereas a Reactive bot will be invoked by a developer in an existing PR

to carry out the exact function. Persona is also polarized in the same manner. In a pull request discussion, **Personable** bots appear more human-friendly. We defined personable bots by their ability to communicate similarly to humans, where emotions play an important role. In our case, emotions are displayed with the use of informal language, slang, and emojis, which can even be tailored to the community's cultural norms. **Factual** bots, on the other hand, display an emotionless portrait of pure logical entities where they communicate concisely in a formal manner regardless of cultural values. For example, a Personable bot may reject a PR by saying "*Hi dear Jack, Sorry but we are unable to accept your PR. Feel free to modify it in order to meet our Code Convention Policy (CCP) and then open another PR...*" whereas a Factual bot may say "*This PR has been rejected and closed due to violation of repository Code Convention Policy.*" "

3.2.1.2 Dependent Variables

In modelling our Dependent Variable (DV), i.e. Preference, we adopted a similar polarization approach with some modifications. Investigating the fluctuation in the independent variables (i.e., Autonomy and Persona) and their consequent influence on developers' preferences is pivotal for addressing Research **RQ-1** and **RQ-2**. In this context, it is crucial to quantify the impact of the constructs' extremities (Proactive/Reactive or Personable/Factual) on the developers' propensity to select a particular bot over the other. Initially, we associated Preference with the concept of acceptance, where participants could either accept or reject a given bot-human interaction. However, this approach did not align well with situations where participants could prefer both or neither of the options presented. Similar to how individuals may remain neutral in their preference between Coffee and Tea, we allow for neutrality as a valid response to our dependent variable, resulting in three poles for Preference. The tri-polarization of Preference is as follows: **A)** Prefer, which implies that participants accept some or all of the interactions presented in a scenario given a polarized construct; **B)** Avoid, which implies that participants do not accept some or all of the interactions presented in a scenario given a polarized construct; and **C)** Remain Neutral, which implies that participants either accept or reject both poles of a given construct. For instance, a developer may prefer Proactive bots over Reactive bots, or remain impartial toward both options.

3.2.1.3 Influential Factors

In addition to discerning whether Autonomy or Persona impacts the developers' Preference, our study also aims to delve into the factors that shape such decisions. **RQ-1.1** and **RQ-2.1** directly correlate with these determining elements. In the initial phase of our study, we conducted interviews and identified several themes, which we used to construct a causal graph[41] to model these relationships. A causal model is a Directed Acyclic Graph (DAG), where the arrows signify a causal (influence) relationship between variables. The causal graph, illustrated in Figure 3.1, indicates that factors like `BotExperience`, `GitHubActivity`, and `PRActivity` exert influence on preferences for the degree of Autonomy and Persona style. The specifics of these factors will be clarified in section 4.2.

A hidden node in our model represents unmodelled sources of variability. We encode Preference using the variables `VignetteChoice_Autonomy` and `VignetteChoice_Persona`. The causal factors or "treatment effects" inducing these choices are `AmountOfAutonomy` and `AmountofPersona` that are mirrored in poles of our constructs (*e.g.*, Proactive/Reactive for Autonomy). In other words, we aim to model the likelihood of a respondent's selection (*e.g.*, "Prefer Reactive","Neutral", or "Prefer Proactive") in our vignettes.

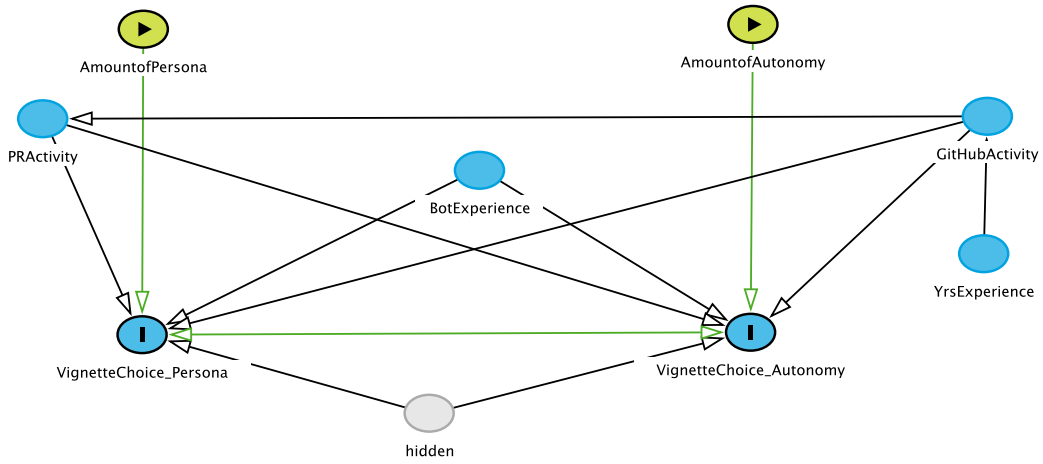

Figure 3.1: Causal model illustrating the influence of various factors on the developer's preference (codified with `VignetteChoice_Autonomy` and `VignetteChoice_Persona`)

3.2.2 Scenarios

A scenario can be defined as a unique combination of our constructs that portrays a single vignette. In the context of our study, we have two independent variables, each with two poles. As a result, four unique combinations can be generated by merging these two constructs. One of the immediate advantages of merging these constructs is that it allows us to control one variable while varying the other, thereby facilitating the isolation of a single construct and measurement of Preference (dependent variable) in the absence of the controlled variable. The combinations and corresponding scenarios resulting from this merging process are presented in Table 3.1

Fixed Construct	Fixed Value (pole)	Variable Construct	Scenario Number / ID
Persona	Factual	Proactive — Reactive	1 / SC1
	Personable	Proactive — Reactive	2 / SC2
Autonomy	Proactive	Personable — Factual	3 / SC3
	Reactive	Personable — Factual	4 / SC4

Table 3.1: The merging of Persona and Autonomy constructs leads to a set of scenario combinations. It is noteworthy that each scenario is characterized by one variable being fixed at a particular value while the other variable varies between its two poles.

Furthermore, this formulation is consistent with our vignette-based design, which allows us to present two contrasting values simultaneously on a single page and ask participants to report their preference between the two variables. To illustrate, in constructing Scenario 2 (SC2), we present a single page featuring a proactive bot on the left-hand side and a reactive bot on the right-hand side, while keeping Persona fixed at the Personable pole for both bots. Subsequently, participants are given the option to indicate their preference for either the left or right side or remain neutral (see Figure 3.2).

3.2.3 Vignettes

Having established our four scenarios, we can now proceed to map them onto a vignette that simulates a GitHub pull request discussion page. However, before doing so, it is necessary to explain the instrument layout that comprises the pages on which the vignettes will be presented. As shown in Figure 3.2, each page of the survey instrument is composed of three horizontal sections.

The first section, located at the top of the page, displays a simple text prompt that serves to draw respondents' attention to the appropriate question they are to answer. We utilized the prompt "*The two conversations differ by who starts them*" prominently to effectively guide participants' attention towards the autonomy of the bot and discourage them from being distracted by extraneous elements of the conversation. This prompt only appears in Scenario 1 and 2 as the varying construct is Autonomy. Similarly, we used the prompt "*The two conversations differ by the text used by*" in order to guide participants' attention towards the personality of the bot displayed in the conversation that we wished to highlight, and to discourage them from fixating on unimportant details. This prompt appeared in Scenario 3 and 4 since the varying construct was Persona.

In the middle section of the page, we present the scenario by showing two vignettes side-by-side. These vignettes have been designed to replicate the appearance and theme of a GitHub pull request discussion page. We utilized the HTML source code of the GitHub PR web page and compressed it to remove clickable links and navigation to other tabs. In addition, we replaced profile pictures with AI-generated images of humans and changed developer names and IDs to protect their privacy. We will discuss the process of scenario implementation in section 3.3 as we delve further into how we operationalized each construct and integrated them into a PR discussion to produce a believable vignette.

The bottom section of the page is reserved for collecting responses, and it contains three buttons indicating the respondent's preference. Respondents can indicate their preferred vignette by selecting "*I Prefer the Left Side*" located underneath the left vignette or "*I Prefer the Right Side*" located underneath the right vignette. They can also remain neutral by selecting the "*Neutral*" option. As the two end poles of the Preference (i.e., Prefer and Avoid) are mutually exclusive, we can omit one as it can be inferred by selecting the other pole. When two vignettes are presented side-by-side, respondents can indicate their preferred side, and we can automatically infer which side they would avoid. Before proceeding to the next scenario, respondents are required to provide a brief textual explanation of their choice. We utilize this open-ended response later to analyze their incentives and the logic behind their preference and also to support some of our claims in section 4.4.

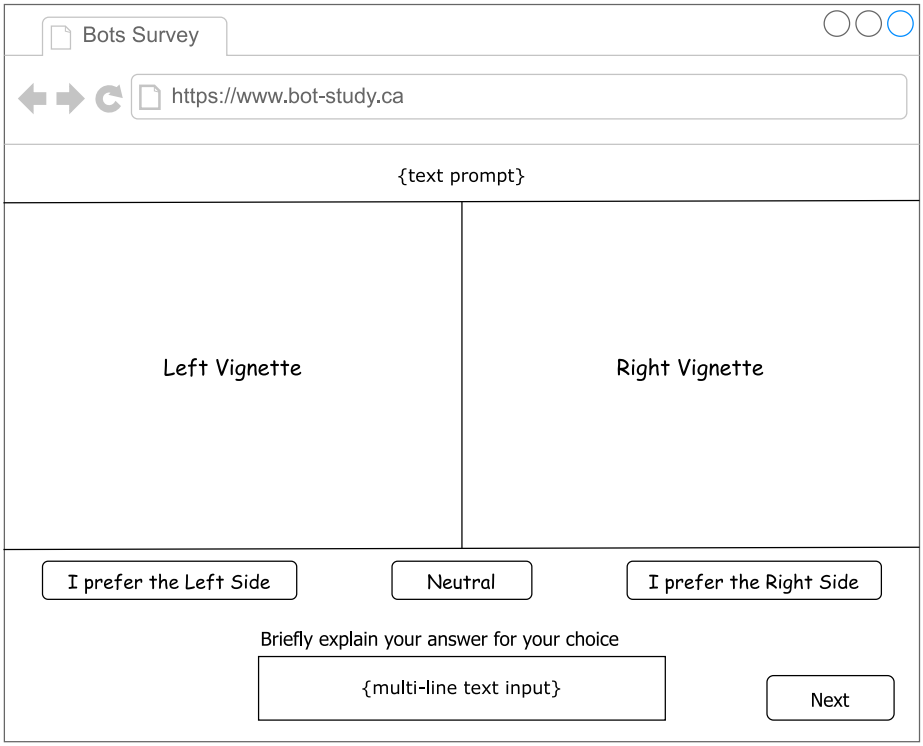

Figure 3.2: The survey instrument comprises a user interface (UI) sketch that includes three distinct sections. The topmost section features a textual prompt that explains the intended focus of attention. The middle section consists of two vignettes that present conversations in the style of GitHub pull request (PR) discussions. Finally, the bottom section comprises a multi-line text input field where participants can justify their preference or lack thereof .

3.3 Operationalization of the Constructs

In our study, we examine the interactions between bots and developers in open-source projects, with a particular focus on GitHub. This choice was motivated by GitHub's widespread popularity among developers, as well as the high volume of active projects hosted on the platform. The study primarily focuses on bot-developer interactions that occur within Pull Request (PR) discussions. To identify suitable repositories for analysis, we initially explored several highly active repositories, based on factors such as project complexity, GitHub stars, and total number of PRs. Notably, repositories such as Facebook's React, Google's Flutter, and VueJs were identified as potential candidates. From these repositories, we sampled over 20 PR discussions that displayed bots with varying levels of Autonomy and Persona. Ultimately, three discussions involving `bug-finder-bot`, `oca-clabot`, and `flutter-dash-bot` inter-

acting with developers were selected for analysis. To ensure that participants were not distracted, we deactivated links, restricted navigation to other tabs, and removed irrelevant parts of the UI such as sidebar navigation panels and footer sections, when designing the vignettes that simulated these GitHub PR discussions.

The final and the most challenging aspect of the vignette design involved ensuring that the conversations and interactions portrayed in the vignettes accurately reflected the definition of the poles in the constructs being studied. Specifically, it was important to ensure that the implementation of these constructs, for example in Scenario 4 (SC4) that Autonomy is fixed to Reactive and Persona varies between Personable and Factual, felt natural and believable, while also making it easy for respondents to distinguish between the two poles of the Persona. Additionally, there were challenges related to operationalizing each pole of the constructs. While Autonomy was relatively straightforward to operationalize by determining who was commanding the bot or whether the bot was acting without intervention (see subsection 3.3.1), operationalizing the Persona construct proved to be more difficult. In the absence of a proper framework, it was difficult to determine what types of behavior or conversation styles would aid respondents in distinguishing between a Personable bot and a Factual bot. As a result, we revised the original definition of the Persona construct several times to ensure that it encapsulated all of our operationalization goals (see subsection 3.3.2).

To facilitate better differentiation, in all cases, we assigned entirely distinct and neutral profile photos for bots as compared to humans. Furthermore, the names assigned to bots included the word `bot` at the end of their displayed name. Additionally, a "*bot*" tag was included right next to the displayed name, which only appears when the owner account is legitimately registered as a bot on GitHub.

3.3.1 Autonomy

The operationalization of the two poles of Autonomy was a straightforward process as it was defined by the level of independence exhibited when taking an action. Specifically, the degree to which human developers moderated the actions of the bot determined the level of Autonomy. In the case of Proactive bots, they are capable of performing a range of tasks, such as closing a pull request at their discretion, adding or removing tags and labels for a particular discussion, assigning human developers to tasks like code review, resolving conflicts, or merging into a branch, as well as

generating code for automatic bug fixes and opening a pull request to merge it into the main branch. It is important to note that these interactions do not require human intervention to initiate. In contrast, Reactive bots only respond to human developer requests and perform functions as directed. Reactive bots may also assign labels and tags to specific comments in a discussion. Essentially, Reactive bots have the same capabilities as Proactive bots, with the key distinction being that they require human developer invocation, or their actions trigger the bot to perform a specific task.

3.3.2 Persona

Due to the absence of objective measures for evaluating personality and human-seemingness in non-sentient entities, and the uncertainty around what is considered personable and human-friendly behavior for bots and where to draw the line, we employed **anthropomorphism** to make bots more human-like. Anthropomorphism is the attribution of human-like qualities, emotions, or behaviors to non-human entities, such as animals, objects, or even robots. This is often done to make these entities more relatable or understandable to humans. An example of a robot that uses anthropomorphism is the Pepper[6] robot, which is designed to interact with humans in a social and engaging manner. Pepper has a humanoid appearance, with a head, arms, and a torso, and is capable of recognizing human faces and emotions, speaking multiple languages, and providing information and assistance.

Given that software bots in in our study can only communicate with developers via text in a PR and cannot have a physical representation, we identified a set of easily recognizable anthropomorphic traits to model our Personable bots. Informal language usage that includes emojis and slang is a common pattern observed in human communication via text. However, the main aim of Personable bots is to convey emotions to appear human-friendly, in contrast to Factual bots that view the English language as a set of grammatical rules and words for conveying precise information with no additional context. To elaborate further, Personable bots are characterized by their ability to engage in direct and interactive conversations, which facilitates the identification of the conversation's participants and simulates a sense of personalized attention from the bot towards its human interlocutors. In contrast, Factual bots mainly focus on broadcasting information and do not place any particular individual or group in the spotlight.

3.3.3 Design Breakdown

In Figure 3.3, we present Scenario 1, which involves a conversation between `BugFinderBot` and one of the repository maintainers. As detailed in Table 3.1, Scenario 1 employs a fixed Persona of Factual, with the left vignette depicting a Proactive bot and the right vignette portraying a Reactive bot. While the manner in which the PR is initiated and the conversation begins differs between the two vignettes, the remaining aspects of the two vignettes remain consistent. From the left vignette, it is evident that `BugFinderBot` has identified a bug and created a corresponding fix, and promptly initiated a PR to resolve the issue. On the right side, we see the maintainer directing the bot to generate a patch against a particular commit. In both vignettes, the maintainer expresses gratitude towards the bot, despite the incorrect patch produced for the bug, and closes the PR with a message that conveys to other developers that the patch is not useful. The factual nature of the bot is reflected in its use of concise and formal language in its messages.

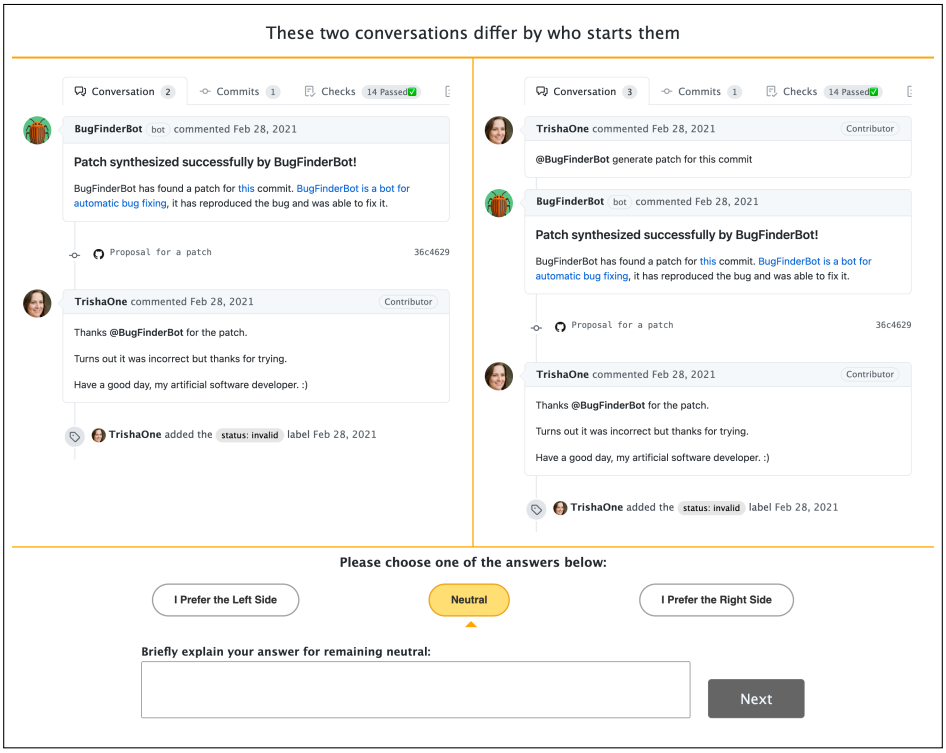

Figure 3.3: Scenario 1.

Scenario 2, as illustrated in Figure 3.4, features `BugFinderBot` once again, tasked with generating a patch for a possible bug. In this scenario, the right vignette depicts

the bot acting autonomously, while the left side portrays a Reactive bot. To achieve a Persona that is fixed at the Personable pole, we permitted the bot to use slangs and emojis and created an energetic character for the bot. In this scenario, a different maintainer responds to the generated patch, pointing out the bot's flaw in addressing the bug and closes the PR. Notably, certain adjustments were made in this scenario as opposed to Scenario 1 to mitigate any potential learning bias on the participants.

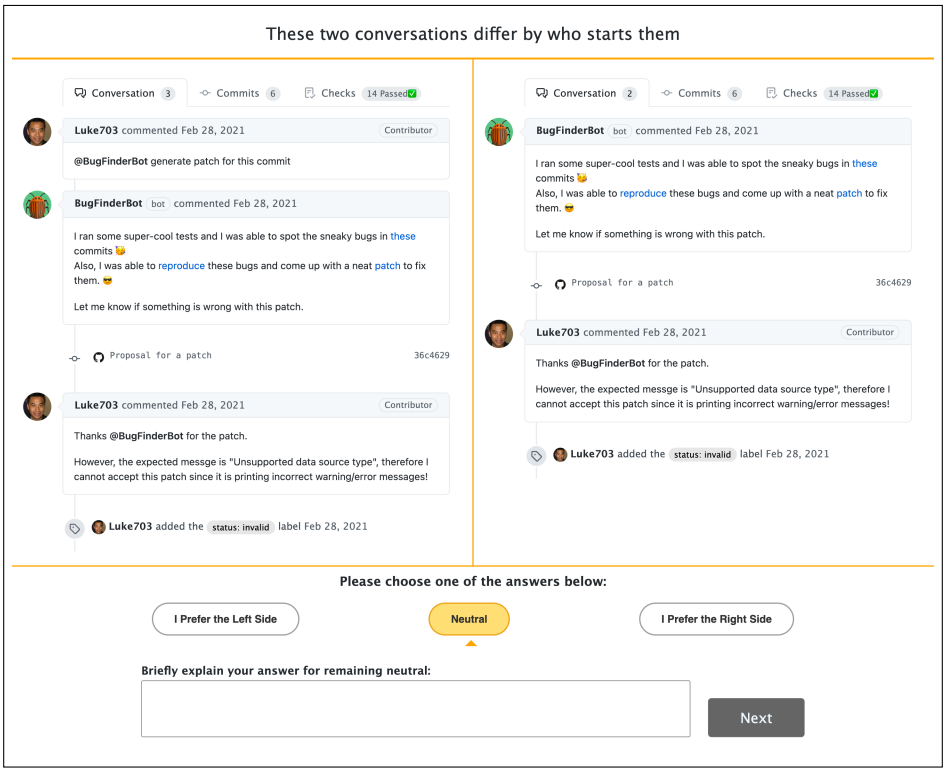

Figure 3.4: Scenario 2.

Scenario 3, depicted in Figure 3.5, features both vignettes set to the Proactive pole for their Autonomy construct, with the left vignette featuring a Personable bot, in contrast to the Factual bot in the right vignette. In this scenario, a contributor initiates a PR, but the `flutter-dash-bot` prevents them from merging due to non-compliance with repository standards. Despite the bot's response to the contributor, adding an "INVALID" label to the conversation and assigning a maintainer for further investigation is done proactively. In the left vignette of Scenario 3, the bot communicates directly with the contributor, using pronouns (you/your) to address them and kindly apologizing for any inconvenience caused, in contrast to the Factual bot in the right vignette, which broadcasts a message in the discussion, stating simply

why the PR is considered invalid.

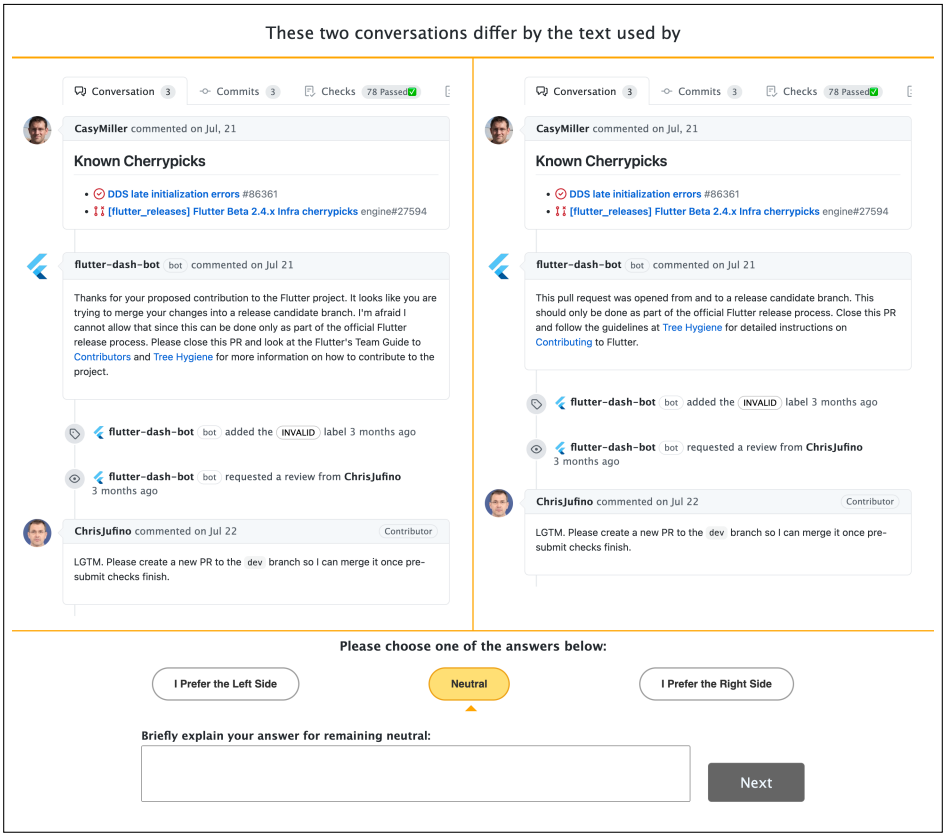

Figure 3.5: Scenario 3.

In the fourth and final scenario, depicted in Figure 3.6, both vignettes feature a Reactive bot, with a Factual bot in the left vignette and a Personable bot in the right vignette. In Scenario 4, the `oca-clabot` responds to a PR, stating that the contributor has not signed the Contributor License Agreement (CLA), and asks them to sign the CLA before proceeding. After the developer signs the CLA a day later, the bot responds again and adds a "CLA Signed: Yes" label to the discussion. In the left vignette, the bot presents the problem succinctly as a Factual bot would do and directs the contributor to sign the CLA by providing a link to a web page. In the right vignette, the `oca-clabot` appears Personable by addressing the contributor and expressing appreciation for their contribution, while kindly directing them to a web page to sign the CLA.

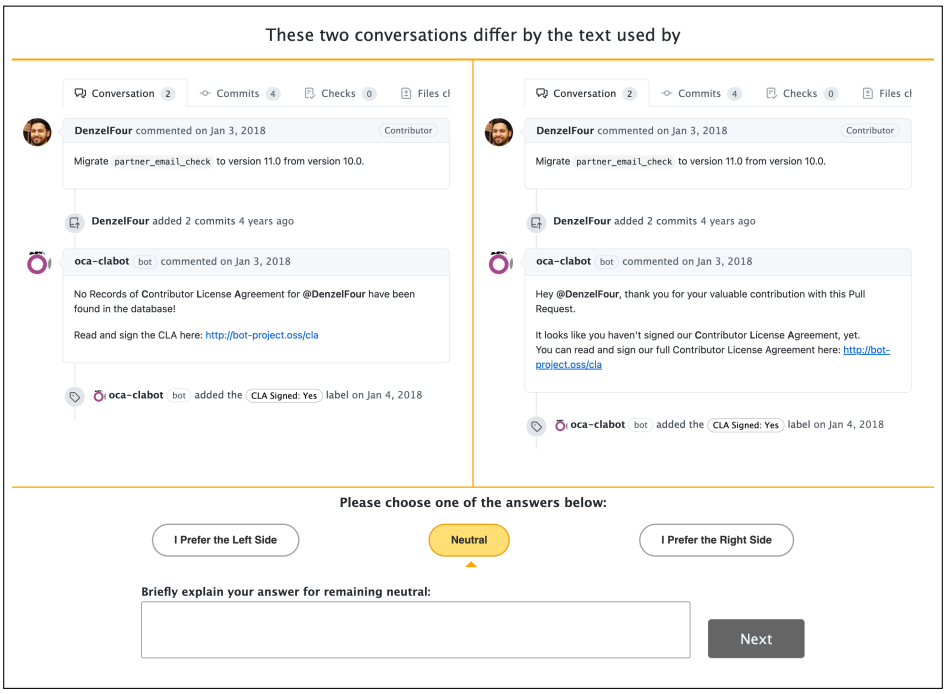

Figure 3.6: Scenario 4.

3.4 Survey Deployment

The scenarios were not the only components of our survey instrument. Conceptually, we divided our survey instrument into three stages. Figure 3.7 shows the general design of the survey instrument and how participants progressed through each stage. Regarding the technical aspects, we obtained a computing server from the IT Department of the University of Victoria and developed a simple HTTPS server utilizing NodeJS as our server application, ExpressJS as our web framework and static file server, and MySQL for our database. We ensured that the entirety of the survey instrument, comprising both stored and transient data, remained securely within the server. For the purpose of our study, we intentionally divided our participants into two distinct groups: the on-site group and the online group, based on how they were introduced to the research instrument. The on-site population received a concise introduction to our research and were physically present at the research site when they completed the survey. Conversely, the online population consisted of participants recruited from external platforms and accessed the survey instrument through third-party screening tools. Further information on the recruitment process for participants can be found in Section 3.5.

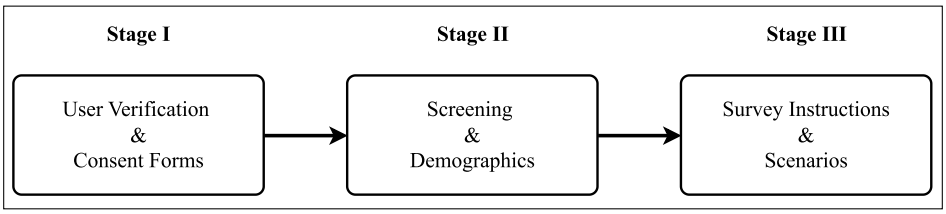

Figure 3.7: Stages of survey instrument in the order which participants go through them.

3.4.1 Stage I: User Verification and Consent Forms

In the first stage of our survey instrument, we required participants to register by providing a valid email address and their name. Before proceeding to the next stage, participants had to read and agree to the consent forms provided. The email address and name entered by participants were sent to our back-end server for verification, after which a unique stateful session was generated for each participant and stored in our self-maintained SQL database. Stateful sessions enabled participants to save their progress as they went and continue answering questions even if there were network issues or errors that prevented them from accessing the instrument. We implemented this approach in our beta testing phase, during which we encountered various network issues and code bugs that caused testers to lose their progress.

During the first login, participants were assigned a sequence of randomly generated scenario IDs for their session, such as `(SC2, SC3, SC1, SC4)`. However, this sequence was not entirely random, as it included a strict constraint: scenarios with the same fixed construct could not be adjacent to each other. This constraint was implemented to mitigate the learning effect that occurred when participants responded to two consecutive scenarios of the same construct. Our beta testing revealed that testers frequently referred to previously seen scenarios to justify their preferences and viewed the scenario with the same fixed construct presented immediately before the current scenario as a continuation of the same construct. Figure 3.8 shows the JavaScript code that was used to produce these random sequences. Applying this constraint left us with only eight combinations of scenario sequences that we could sample randomly and assign to each session.

```javascript
function generateRandomScenarioSequence() {

  // grouping secanrios with the same fixed construct
  var fixed_persona = shuffle(["sc1", "sc2"]);
  var fixed_autonomy = shuffle(["sc3", "sc4"]);

  var sequence = [];
  if (Math.random() < 0.5) {
    // start the sequence with a fixed Persona construct
    for (var i = 0; i < fixed_persona.length; i++) {
      sequence.push(fixed_persona[i], fixed_autonomy[i]);
    }
  } else {
    // start the sequence with a fixed Autnomy construct
    for (var i = 0; i < fixed_autonomy.length; i++) {
      sequence.push(fixed_autonomy[i], fixed_persona[i]);
    }
  }

  return sequence;
}

```

SC3→SC2→SC4→SC1

SC1→SC3→SC2→SC4

SC4→SC2→SC3→SC1

SC2→SC4→SC1→SC3

SC1→SC4→SC2→SC3

SC3→SC1→SC4→SC2

SC4→SC1→SC3→SC2

SC2→SC3→SC1→SC4

(a) (b)

Figure 3.8: On the left side (a), we have a JavaScript function used to generate a random sequence of scenario IDs. We first group the scenarios that cannot appear after one another and then randomly insert a different scenario in between. On the right side (b), we listed all 8 possible sequences generated by this function. Scenarios with fixed Persona construct are colored blue and red for Autonomy. Note that how no two scenarios with the same color are adjacent to each other

3.4.2 Stage II: Demographics and Screening Questionnaire

Upon successful registration and session creation, participants transitioned to the second stage of the survey instrument. In this stage, we presented two sets of questionnaires to collect demographic data and assess their suitability for our research using screening questions. This stage consisted of a total of 17 questions that had to be answered. The demographic questionnaire contained 9 questions listed in Table 3.2, which allowed us to identify and analyze the diversity of the population under study. Furthermore, demographic data was utilized in our modeling approach to distinguish between experienced and novice developers, which was crucial for addressing **RQ-1.1** and **RQ-2.1**. This enabled us to identify and analyze the differences in behavior, skills, and performance between these two groups, providing valuable insights into their respective development processes.

The remaining questions in the second stage were utilized as a means of screening out participants who were considered unsuitable for our study. In the context of our research, screening is defined as a process of filtering out individuals who lack the necessary background or knowledge required to participate in our survey. In order to ensure consistency between our on-site and online populations, we stipulated that participants must possess a reasonable level of programming knowledge and have had prior interaction with a bot on GitHub. Following the recommendations of Danilova *et al.* and Ebert *et al.* [16, 17], we employed a screening protocol consisting of eight questions, which were divided into two categories: Programming Knowledge (PLK) and GitHub Experience (GHE). The details of these screening questions can be found in Table 3.3, although it should be noted that the full description of questions except *Q1*, *Q6*, and Q10 are too lengthy to be fully included in the table. We included factual questions (*Q2, Q3, Q5*, and *Q11*) to test the basic knowledge of our participants, an analogy question (*Q10*) to assess their pattern recognition and critical thinking abilities, and an attention question (*Q4*) to evaluate their ability to focus during the survey.

In parallel to our own survey instrument, we recreated the second stage of the instrument on *SurveyMonkey*, which was hosted inside Canada[1]. SurveyMonkey is an online survey tool that allows individuals and organizations to create and distribute

[1]In Canada, there are specific laws and regulations in place to safeguard personal information, and using servers hosted in Canada can help ensure compliance with these requirements, which can help to mitigate legal and reputational risks associated with data breaches or mishandling of personal information. (`https://www.justice.gc.ca/eng/csj-sjc/pl/charter-charte/c27_1.html`)

Number	Question	Answer options
7	Have you ever submitted a pull request to any open-source project?	One of *Yes, No.*
8	Have you ever reviewed pull requests on any open-source project?	One of: *Yes, No.*
9	How often do you collaborate with repositories on GitHub, Gitlab, or Bitbucket?	One of: *At least once a day, At least once a week, At least once a month, At least once every six months, At least once a year, Never.*
12	Have you ever interacted with a bot in an open-source project? if yes, please enter the name of at least one of these bots.	*Open Input*
13	What is a "bot"? Briefly describe or provide a definition.	*Open Input*
14	To which gender identity do you most identify?	One of: *Male, Female, Non-binary, Other, Prefer not to say.*
15	How old are you?	One of: *Under 18, 18-24, 25-34, 35-44, 45-54, 55-64, 65+*
16	What is your profession?	*Open Input*
17	How many years of experience do you have in your profession?	*One of: Less than one year, one to three years, three to five years, more than five years.*

Table 3.2: demographic questions presented in the second stage of the survey instrument.

Number	Category	Question	Answer Options	Correct Answer
1	PLK	Have you ever done programming before?	One of: Yes, No.	Yes
2	PLK	What is the purpose of the "break" statement and where is it used?	Four answer options	Inside of a loop to stop the current loop from iterating
3	PLK	Select all the parameters of the function below?	A, B, C, D, K, P	A, B, C
4	PLK	According to Wikipedia on Integer bit overflow ...	Four answer options	Options 1 or 3
5	PLK	Choose the answer that best fits the definition of a recursive function.	Four answer options	A function that calls itself
6	GHE	Do you have a GitHub or Gitlab account?	One of: Yes, No.	Yes
10	GHE	An engine is to an airplane as _____ is to GitHub/Gitlab.	One of Terminal, Git, Browser, Algorithm	Git
11	GHE	Choose the answer that best fits the definition of a pull request	Four answer options	A software change that others have to review, then pull into the main branch

Table 3.3: Screening questions.

surveys and collect responses from survey participants. It is a popular and secure platform that offers a variety of survey question types, customization options, and data analysis tools. Given our goal of publishing the survey online, particularly on participant recruitment platforms (as detailed in section 3.5) to increase the size of our study population, we made the decision to conceal our server and survey instrument from public view. This allowed us to transfer participants from the online population directly to the third stage of the survey, bypassing the registration and screening in the first two stages, only after they successfully completed the screening test on SurveyMonkey. Despite the fact that eligible participants from an external source were directly transitioned into the third stage, a session that included a random sequence of scenario IDs was still generated. In addition to security considerations, outsourcing the screening task to SurveyMonkey provided an additional benefit by reducing the load on our server, which helped to streamline the survey process and enhance the user experience.

3.4.3 Stage III: Survey Instructions and Scenarios

Participants who met the eligibility criteria were presented with an instruction page in the third stage that outlined how to respond to the upcoming scenarios. They were also informed that once they submitted an answer for a scenario, they could not go back and make changes. In the next step, the survey instrument presented scenarios to each participant in the order determined by their randomly generated scenario ID sequence, until they completed all four scenarios. The overall design of the survey instrument and the flow of on-site and online participants in the system were depicted in Figure 3.9.

3.5 Participant Recruitment

Following the receipt of the Ethics Review Board (ERB) approval letter, we proceeded to deploy the survey instrument online. To recruit our on-site population, we invited third-year undergraduate software engineering students at the University of Victoria who had already gained sufficient experience in open-source collaboration and programming on GitHub to participate in the survey. The instructor of the class ensured that participation in the survey was optional and left the room while students took the survey to minimize pressure on them. 26 students participated in the survey and

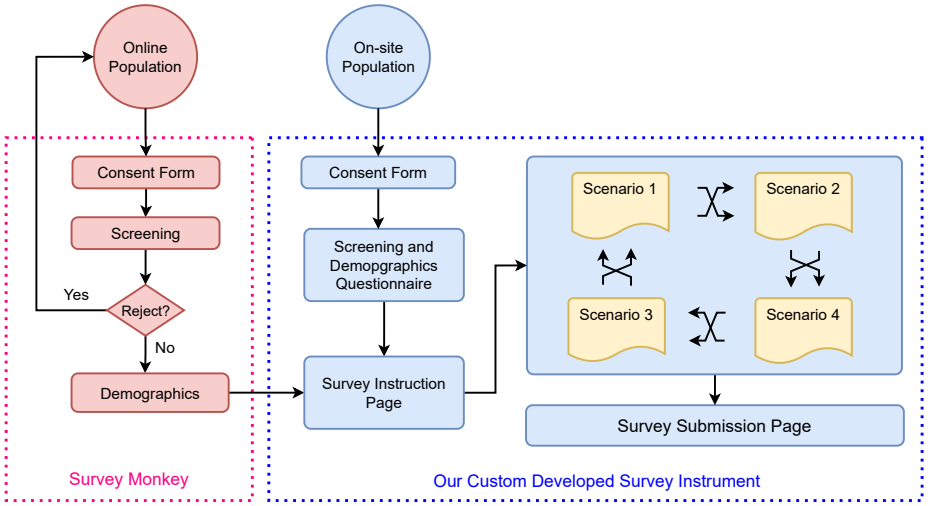

Figure 3.9: Overall flow of participants in the system.

their responses were collected anonymously.

To enhance the diversity and increase the population, we opted to employ a Participant Recruitment Platform (PRP) and recruit professional participants. Here, "*professional*" refers to developers who work in the industry and are not students. After evaluating various options, we selected Prolific[2] due to its lower drop-out rates in the screening process for software engineering tasks[16]. Prolific is an online platform that enables researchers to recruit and pay participants for studies and surveys. It offers a large and diverse participant pool, as well as tools for managing participant recruitment, screening, and payments, making it a popular and convenient option for researchers in various disciplines.

3.5.1 Data Collection

We conducted the recruitment of 300 participants in four rounds on the Prolific platform. We employed the internal screening criteria available within the Prolific Control Panel to ensure the balance of male and female participants and that they have computer programming as a skill listed in their profile. Furthermore, we selected participants from different countries in North America[3] and Europe[4] and offered a

[2] `https://www.prolific.co`

[3] Canada and United State

[4] Ireland, Germany, France, Spain, Belgium, Denmark, Finland, Greece, Italy, Netherlands, Norway, Poland, Portugal, Sweden, and Switzerland

compensation rate of 7.5 GBP[5] per hour even to participants who were screened-out. The dates and times of each recruitment round were carefully chosen to ensure that most working professionals were available to participate in the survey. Table 3.4 list all the hiring iteration on Prolific.

Iteration	Date	Location	Total number of participants	Eligible participants	Recruitment rate
1	Feb 23, 2022	NA	50	9	18%
2	Mar 15, 2022	EU	50	4	8%
3	Mar 29, 2022	NA	100	9	9%
4	Apr 4, 2022	EU	100	8	8%
		Total	**300**	**30**	**10%**

Table 3.4: Hiring iteration on Prolific and their respective recruitment rate. For all iterations, sex is balanced, and the only mandatory screening criteria applied on the Prolific control panel is Computer Programming.

Compared to our on-site population with a recruitment rate of 37.6%, we achieved a low recruitment rate of only 10% on Prolific. Research has shown that recruitment rates tend to suffer when requirements become more specific, and tasks require expertise in certain domains[17, 42, 40]. Another contributing factor could be that some participants may misrepresent their skills to participate in surveys for financial reasons. Out of 270 participants who were removed from the study on Prolific, 196 (72.6%) were screened out by programming knowledge questions and other 74 (27.4%) participants were screened out by GitHub experience questions.

3.5.2 Demographics

The study collected a total of 56 responses, with 30 from participants hired on Prolific and 26 from software engineering students at the University of Victoria. Although the intention was to hire professionals on Prolific, 4 out of the 30 participants had their job status listed as "student" on their public Prolific profile. Among the participants, 19 identified as female, 34 as male, 2 as non-binary, and 1 did not disclose their gender (*Q14*). In terms of age, 34 respondents were aged 18-24, 18 were 25-34, and 4 were 35 or older (*Q15*). Regarding experience in software development, 27 had 1-3 years of experience, 19 had more than 3 years, and 10 had less than a year (*Q17*). Additionally, 10 participants had previous bot interactions, and only 6 could not

[5]Prolific pays participants in GBP (British Pound Sterling)

define a software bot (*Q13*). Figure 3.10 shows the age, gender, and professional experience distribution in our dataset.

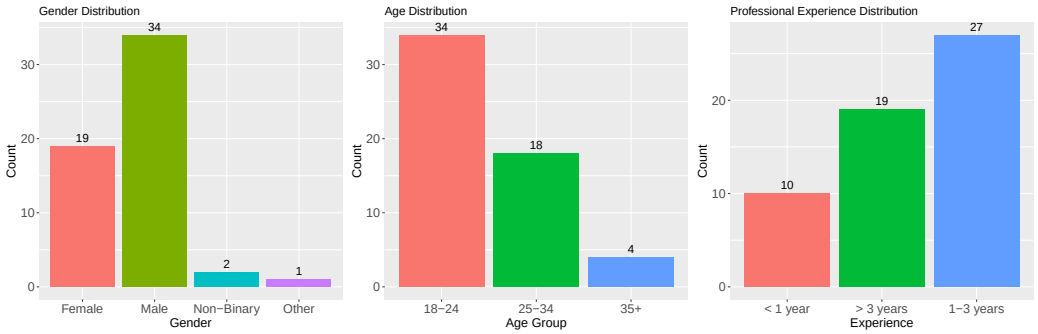

Figure 3.10: Gender, Age, and Professional Experience distribution in our dataset.

3.6 Chapter Summary

The paper initially introduces the research methodology in section 3.1, designed to examine developers' preferences for distinct categories of bots: reactive/proactive and personable/factual, as further detailed in section 3.2. The methodology involves a vignette-based survey instrument simulating realistic bot-developer interactions, which are evaluated through written dialogues, emoji responses, and administrative actions.

In the experimental design, subsection 3.2.1, Autonomy and Persona are defined as two independent variables, each possessing two opposing characteristics. Bots can exhibit Proactive (autonomous) or Reactive (responding to human triggers) behavior under Autonomy, and demonstrate either a Personable (human-like) or Factual (strictly informative) persona under Persona. Participants' preferences form the dependent variable, allowing for three potential outcomes: Prefer, Avoid, or Remain Neutral.

Subsection 3.2.2 outlines the unique combinations of constructs in each scenario, enabling single-construct isolation and preference measurement. These scenarios are subsequently represented through vignettes simulating GitHub PR discussions in subsection 3.2.3. Participant responses are collected in a structured format, allowing them to show preference for either vignette or remain neutral, with the "Avoid" preference inferred from the remaining choices.

Operationalizing the constructs of bot-developer interactions within the GitHub context is described in section 3.3. This involves selecting suitable GitHub repositories (React, Flutter, and VueJs), identifying suitable bots for our investigations , and recreating their PR discussions as vignettes. Distinct identifiers are used to distinguish bots, and the constructs of Autonomy and Persona are reflected, based on our definitions, within these vignettes. For example, Proactive bots perform tasks autonomously, while Reactive bots respond to human triggers. Personable bots use informal language, and Factual bots stick to strictly informative dialogues.

Subsections 3.3.1 and 3.3.2 elucidate how Autonomy and Persona were operationalized, respectively. The design of the four scenarios representing various constructs of bot-developer interactions is further detailed in subsection 3.3.3. Lastly, section 3.4 discusses the survey deployment strategy involving a three-stage design and the use of a secure computing server provided by the University of Victoria IT Department.

Chapter 4

Developer Perception and Preferences

4.1 Introduction

In this chapter, we focus on exploring, modeling, and analyzing the data that we collected. Our primary objective is to gain a better understanding of how Autonomy and Persona influence the preferences of developers. To accomplish this, we begin by employing descriptive analysis techniques, as outlined in section 4.2. This approach enables us to identify patterns and summarize various features within the dataset. Furthermore, in response to **RQ-1** and **RQ-2** descriptive analysis allows us to formulate hypotheses that we will test later using statistical modelling (*i.e.*, Linear Modelling and Generalized Linear Modelling) tools, as described in section 4.3.

We then proceed to compare models for our independent variable of preference, based on different formulations of possible predictors, to address **RQ-1.1** and **RQ-2.1**. We also consider proxies for programming experience. There are several different ways of operationalizing programming experience[47], and we consider multiple proxies that might provide complementary insights into how experience impacts preferences. Specifically, we examine GitHub activity level, pull request activity level, and bot interaction experience as a potential predictors.

4.2 Descriptive Analysis

Our analysis of the data using simple data aggregation techniques revealed that respondents preferred Reactive and Personable bots over Proactive and Factual bots. This simple counting approach gives us a preliminary response for **RQ-1** (*i.e.*, **majority of developers prefer Reactive bots**) and **RQ-2** (*i.e.*, **majority of developers prefer Personable bots)**. Figure 4.1 presents an overview of the preferences of our respondents with regards to Autonomy and Persona. While analyzing the preferences of developers provides a solid foundation and a frame of reference for asking more specific questions, we also need to understand how their preferences may change under different conditions. Our respondents come from a diverse range of backgrounds, programming experiences, and levels of exposure to open-source development. Therefore, it is essential to investigate whether these factors contribute to their preferences or not. By doing so, we can gain a more comprehensive understanding of the factors that influence developer preferences and how they vary across different conditions. To gain a more nuanced understanding of the factors that influence developer pref-

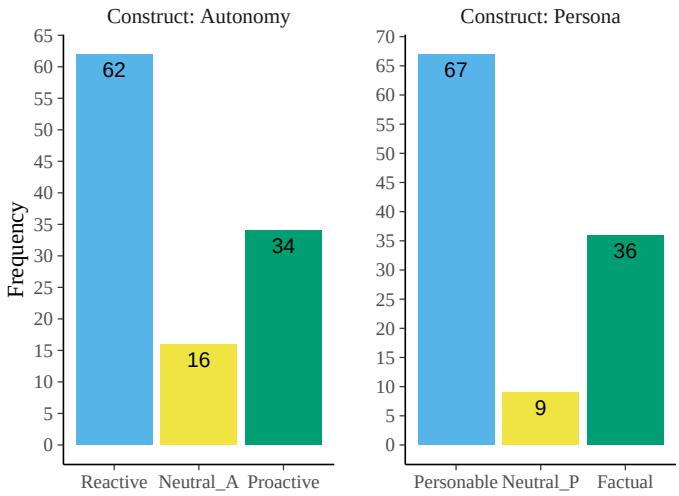

Figure 4.1: Summary of respondent choices across both vignettes, per construct. Left plot aggregates responses for scenarios 1 & 2 while the plot on the right side aggregates responses for scenarios 3 & 4.

erences, we divided the responses into individual scenarios. Interestingly, a slightly different picture emerges when we examine the responses in this way, as shown in Figure 4.2. For Autonomy, the preference for Reactive over Proactive bots persists in scenarios 1 and 2 with noticeable gaps. However, the results for Persona show

a different trend when comparing scenarios 3 and 4. In scenario 3, the majority of respondents prefer a Personable bot, while in scenario 4, they do not display a significant inclination towards Personable bots anymore. These findings suggest that the context in which the bot operates may play an important role in shaping developer preferences, particularly with regards to Persona. To investigate how preference is influence by the context in which bots and developers interact, considering the the fact that not all developers have the same background and expectations, we introduce a categorization element to our analysis in order to account for variable other than Autonomy and Persona such as GitHub Activity, PR Activity, and Bot Experience. We derived these variables directly from the causal graph (see Figure 3.1) that we introduced in sub-section 3.2.1.3. This conditional analysis opens a direct path for us to address **RQ-1.1** and **RQ-2.1**.

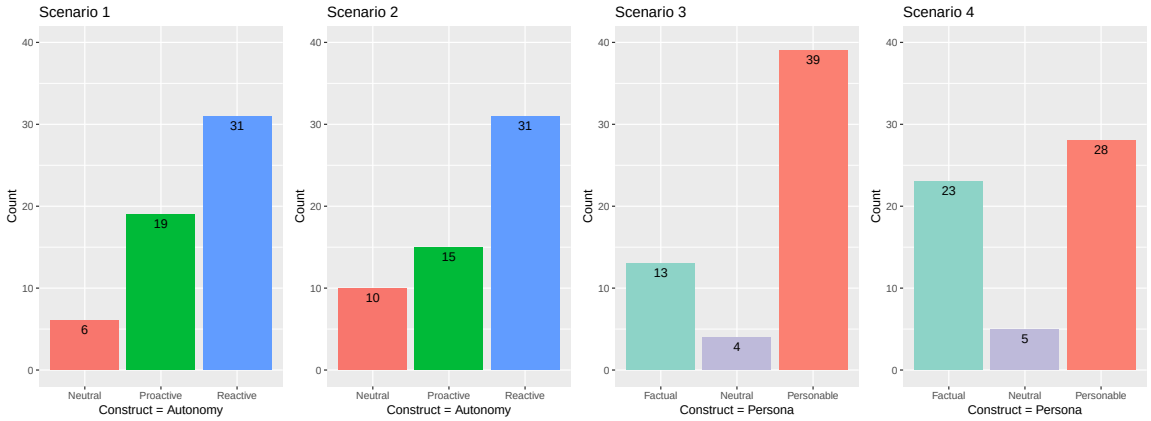

Figure 4.2: Responses broken down into individual scenarios.

4.2.1 GitHub and PR Activity

Our first two conditional variables are levels of GitHub and PR activity. We divide participants into three groups of *Experienced*, *Intermediate*, and *Inexperienced* developers. To do this, we refer to the survey demographic questionnaire (see Table 3.2), where we categorize respondents against Q9 for GitHub activity level. We define and measure GitHub activity by how frequently our respondents collaborate on GitHub. Additionally, we categorize respondents based on their history of PR submission and review processes, as indicated by their responses to Q7 and Q8, respectively.

To categorize developers based on GitHub activity, we mark them as Experienced when they collaborate more than once a month on GitHub. This is done by select-

ing either "*At least once a day*" or "*At least once a week*" in response to Q9 in the survey demographic questionnaire. We categorize developers as Inexperienced when they have never collaborated on GitHub, as indicated by their selection of "*Never*" in response to Q9. Developers who fall between these two extremes are categorized as Intermediate. To categorize developers based on PR activity, we classify them as Experienced when they have both submitted and reviewed at least one PR, as indicated by their selection of "*Yes*" in response to both Q7 and Q8 in the survey demographic questionnaire. Developers who have selected "*No*" for both these questions are classified as Inexperienced. Intermediate developers are those who fall between these two categories, having either submitted or reviewed a PR but not both. Table 4.1 presents the frequency of developers in each category based on their GitHub and PR activity levels. Figure 4.3 and Figure 4.4 show the frequency of each category across scenarios.

Activity Level	**GitHub** (freq.)	**PR** (freq.)
Experienced	20	11
Intermediate	31	44
Inexperienced	5	1

Table 4.1: Distribution of 56 respondents based on GitHub and PR activity.

Notably, upon merging the outcomes for Scenario 1 and 2, which aimed to examine the Autonomy construct, our results reveal that Reactive and Proactive bots are equally preferred by Experienced developers, while Intermediate and Inexperienced developers strongly favour Reactive bots. Our statistical analysis, specifically a chi-square test, reveals a significant association between the level of experience (GitHub Activity) and the type of Autonomy ($X^2 = 5.5977; p = 0.01798, \alpha = 0.05$). Furthermore, Fisher's exact test provides supplementary evidence to support this association ($p = 0.01535; \alpha = 0.05$). These findings appear to contradict the conclusions drawn by Schaffer *et al.* and Liao *et al.*, who discovered that expert users are less inclined to accept the input of automated agents and are wary of proactive automated agents due to the possibility of interruptions [43, 31]. Upon conducting a chi-square test to assess the association between the Persona construct and levels of experience in GitHub activity, by combining the outcomes of Scenario 3 and 4, we found no significant correlation between these variables ($X^2 = 0.094679; p = 0.7583$). Consequently, we are unable to confidently assert that the activity level on GitHub impacts developers'

preference towards bots with specific personalities.

We conducted a similar analysis for PR activity level, which demonstrated that there was no statistically significant association between either the Autonomy or Persona constructs and the level of PR activity ($p > 0.97; \alpha = 0.05$). Therefore, we cannot assert with confidence that there is a relationship between these constructs and the level of PR activity.

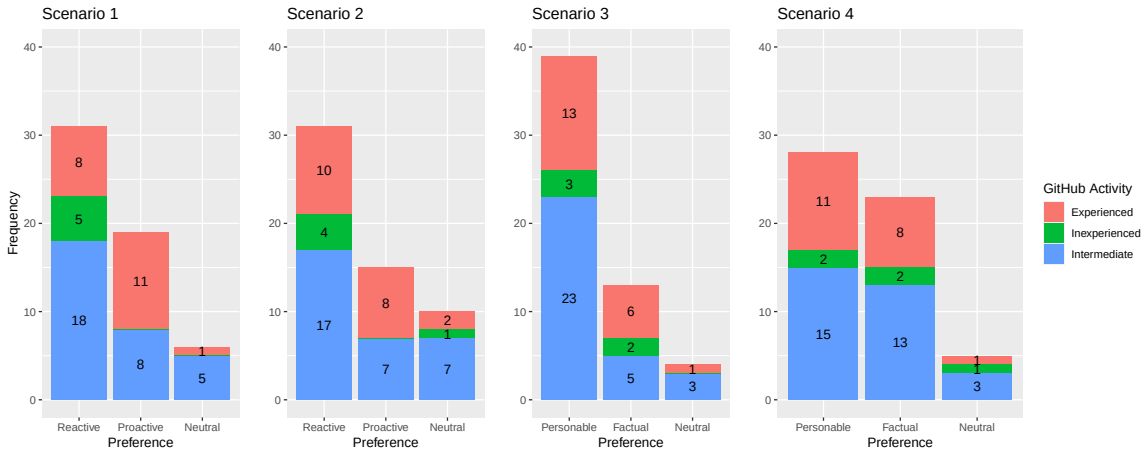

Figure 4.3: Preference of developers conditioned on GitHub activity level

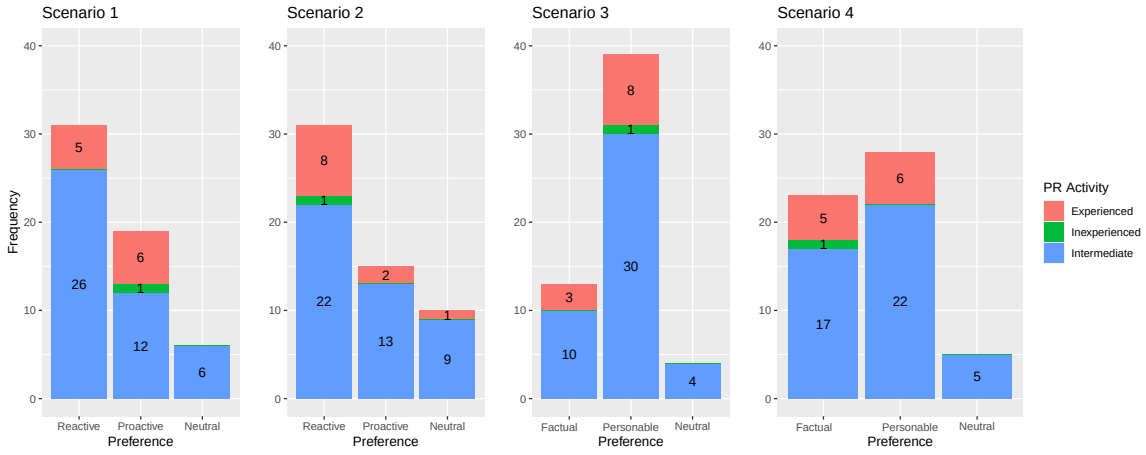

Figure 4.4: Preference of developers conditioned on PR activity level

4.2.2 Bot Experience

In addition to the GitHub and PR activity levels, we also classified our survey participants into three distinct groups of *Expert*, *Intermediate*, and *Novice* based on their

expertise in bot interaction. We categorized respondents using Q12 (history of bot interaction) and Q13 (bot definition) in the demographic questionnaire (refer to Table 3.2). Out of the 56 responses, 10 respondents reported having previous interaction with at least one bot on GitHub, while the remaining 46 participants reported having no interaction with any bot. With regard to bot definition, we followed the definitions proposed by Erlenhov *et al.* and categorized the definitions provided by participants into four labels and coded responses to Q13 as following: `SmartBot`(35/56), `AutonomousBot`(6/56), `ChatBot`(9/56), and `DevTool`(1/56) [19, 18]. We also included an `Unsure` (5/56) label in our coding guideline to signify incorrect or incomplete definitions. Next, we assigned a single label to each respondent's definition of a bot. By quantifying bot definitions and prior bot interaction, we were able to categorize respondents based on the following criteria: individuals who reported prior bot interaction were identified as *Experts* (10/56). Respondents who had no prior bot interaction experience and whose definition of a bot included only one of the `SmartBot`, `AutonomousBot`, or `ChatBot` labels were identified as *Intermediate* (40/56). Finally, respondents who had no prior bot interaction and whose definition of a bot included only one of the `DevTool` or `Unsure` labels were identified as *Novice* (6/56). Figure 4.5 shows the frequency and distribution of each expertise level across all scenarios.

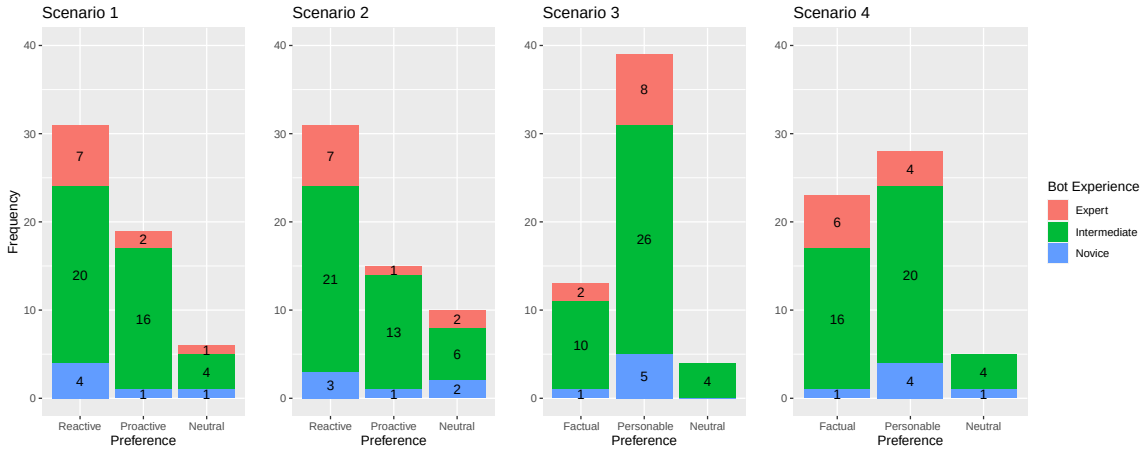

Figure 4.5: Preference of developers conditioned on Bot Experience

Similar to our findings regarding PR activity, we did not observe any discernible patterns among the Expert and Non-expert (Intermediates + Novices) groups when conditioned on bot experience. A chi-square test revealed no statistically significant association between bot expertise and developers' preference for the Autonomy ($X^2 = 1.9859; p = 0.1588; \alpha = 0.05$) and Persona ($X^2 = 0.070904; p = 0.79; \alpha = 0.05$)

constructs. Therefore, we cannot confidently assert that bot experience plays a role in determining developers' preferences for bots with certain Autonomy or Persona traits.

4.3 Statistical Modelling

To further validate our descriptive analysis results, we utilized statistical modelling tools such as Linear Models (LM) and Generalized Linear Models (GLM)[35] found in `lm` and `glm` functions in R programming language. Linear models are a class of statistical models that assume a linear relationship between the independent variables and the dependent variable. They are widely used in various fields, including statistics, economics, and machine learning. Linear models aim to estimate the coefficients of the independent variables to predict or explain the dependent variable. The relationship is represented by a linear equation, typically in the form of $y = \beta_0+\beta_1 x_1+\beta_2 x_2+\ldots+\beta_n x_n$, where y is the dependent variable, β_0 is the intercept, $\beta_1, \beta_2, \ldots, \beta_n$ are the coefficients of the independent variables $x_1, x_2, \ldots, x_n$.

A Generalized Linear Model (GLM) broadens the scope of linear models by accommodating various types of response variable distributions. It's comprised of three parts: the random component (specifying the distribution of the response variable Y, such as binomial or Gaussian), the systematic component ($\eta = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n$, a linear combination of predictor variables), and the link function g, which forms the bridge between the two. This function, $g(E(Y)) = \eta$, enables the transformation of the expected value of the response variable, allowing us to model diverse data types like binary outcomes in logistic regression. The optimal parameters for the model $(\beta_0, \beta_1, ..., \beta_n)$ are found using maximum likelihood estimation, which strives to find the values that yield the highest likelihood for the observed data under the given model.

Such modeling strategies allow us to apprehend the intrinsic distribution and relationships among variables, thereby facilitating a more intricate inquiry into our data. To comprehend the factors influencing developers' perceptions of bots with varying degrees of Autonomy and Persona, we modeled participants' choice (*i.e.*, preference) for each Autonomy (or Persona) scenarios. Subsequently, we established three model descriptions based on the causal graph, previously presented in subsection 3.2.1.3, as depicted in Table 4.2. In the table, a tilde symbol ($\sim$) implies "the dependent variable (left side) is explicated by the interactions of independent variables

(right side)". We devised three models for each of the four scenarios, resulting in a total of twelve models. Although the table specifically articulates models for the Autonomy construct (scenarios 1 and 2), similar formulations were also applied to Persona construct (scenarios 3 and 4).

Model $\mathcal{M}_1$ is the most straightforward, proposing that the extent of GitHubActivity independently serves as an adequate predictor in each vignette. Model $\mathcal{M}_2$ incorporates additional variables, such as expertise with PRs, bots, and general software development experience, to assess whether these enhance the strength of explanation by the model. Finally, Model $\mathcal{M}_3$ examines the potential correlation of responses across scenarios for independent constructs, such as whether a participant's preference for Autonomy influences their choice for Persona, and vice-versa.

Model Number	Statistical Model
$\mathcal{M}_1$	Scenario_Choice_Autonomy $\sim$ GitHubActivity
$\mathcal{M}_2$	Scenario_Choice_Autonomy $\sim$ BotExperience + GitHubActivity + PRActivity + YearsOfExperience
$\mathcal{M}_3$	Scenario_Choice_Autonomy $\sim$ Scenario_Choice_Persona

Table 4.2: Candidate models for Autonomy preferences

Before introducing the data to our models for evaluation, we initially transformed some of the variables in our dataset into suitable data types. Given that our construct variables (such as Autonomy) were categorical (for instance, Reactive, Neutral, or Proactive), we strived to maintain this structure within the data while converting it into numerical form. Thus, we employed the one-hot encoding technique [49] to map each value of a particular construct into a vector. For example, a Reactive value would be transposed into $[1, 0, 0]$, while a Proactive value would be transformed into $[0, 0, 1]$. Regarding our internal variables (conditional variables like GitHub activity or PR activity), we converted them into ordinal numerical values, as these variables inherently possess an ordinal structure (for instance, Inexperienced, Intermediate, Experienced correspond to 1, 2, and 3, respectively).

A crucial aspect to consider during the data modeling process is the alignment of our research questions, insights derived from the literature review, and our observations from the descriptive analysis concerning our constructs Autonomy and Persona (*i.e.*, a majority of developers prefer Reactive and Personable bots). In order to more effectively address our research questions, we place greater emphasis on modeling developers' preferences towards the two extremes of each construct (such as

Personable and Factual), leaving the Neutral option for inference. Put simply, while neutrality is a valid response in our scenarios, our objective in this section is not to model bot-developer interactions resulting in a neutral bot preference. This does not imply exclusion of neutral data from our dataset; instead, we modify the perspective through which we interpret the models and data. This is accomplished with our research question acting as a spotlight, guiding where and how to deconstruct our models to comprehend why developers favour a Reactive bot over a Proactive bot (or similarly, a Personable bot over a Factual bot).

4.3.1 Linear Models

In our research, we implemented the pre-existing `lm()` function from a common R setup, using it to analyze our dataset alongside the specifications for our three models (refer to Table 4.2). With the help of the `summary()` function, we aggregated several key evaluation metrics, as illustrated in Table 4.3. The table disassembles the models' performance, considering their outcomes across different construct values.

	Variable	*Autonomy*				*Persona*			
	Scenario	1		2		3		4	
Models	**Value**	*Reactive*	*Proactive*	*Reactive*	*Proactive*	*Factual*	*Personable*	*Factual*	*Personable*
$\mathcal{M}_1$	`r-squared`	0.59687	0.6326	0.61827	0.56882	0.00128	0.0008023	0.0000095	0.0076643
	`std error`	0.28113	0.149	0.1916	0.2352	0.4297	0.468	0.501	0.5072
	`p-value`	0.01955	0.00579	0.0205	0.0481	0.7935	0.8358	0.9444	0.5213
$\mathcal{M}_2$	`r-squared`	0.1182	0.1678	0.03298	0.07722	0.04489	0.02375	0.05605	0.03269
	`std error`	0.4892	0.4526	0.5123	0.4458	0.4414	0.4812	0.5009	0.5153
	`p-value`	0.1624	0.0488	0.7828	0.3825	0.9937	0.9982	0.558	0.7857
$\mathcal{M}_3$	`r-squared`	0.05694	0.02163	0.02359	0.00246	0.06829	0.06759	0.01999	0.02094
	`std error`	0.4962	0.4814	0.5049	0.4546	0.4189	0.4564	0.5006	0.5085
	`p-value`	0.2115	0.5602	0.5312	0.9368	0.1535	0.1565	0.5857	0.5708

Table 4.3: Comparison of standard error (`std error`), R-squared (`r-squared`), and `p-value` of the three models. The performance of each model for each pole of a given construct (*e.g.*, Reactive for Autonomy) is detailed in the respective column.

The evaluation metrics include `r-squared`, `std error` (standard error), and `p-value`. The `r-squared` values, ranging from 0 to 1, serve to express the proportion of the variability in the dependent variable that can be explained by the independent variable(s); closer to 1 signifies a higher proportion of explained variability. The `std error` approximates the statistical dispersion or variability in a set of data; smaller standard

error values indicate more precise measurements. The `p-value`, which ranges between 0 and 1, reflects the probability that any observed result occurred by chance. The threshold for significance is set at 0.05, where a `p-value` less than 0.05 suggests that the observed result is statistically significant and not likely due to random chance.

As depicted in Table 4.3, $\mathcal{M}_1$ yielded statistically significant results (p-values $<$ 0.05) across the Autonomy construct, employing only GitHub activity as the conditional variable. However, this same model fell short in reflecting the impact of GitHub activity on developers' preference for the Persona construct (p-values > 0.05). Despite offering increased granularity by augmenting $\mathcal{M}_1$ with PR activity, bot experience, and total years of programming experience, $\mathcal{M}_2$ was less informative due to the broad error margins and p-values significantly exceeding 0.05. Similarly, $\mathcal{M}_3$ failed to produce any statistically significant findings when exploring whether a developer's choice of bot autonomy influenced their selection of bot personality, and vice-versa.

Variable	**Scenarios**	**1**		**2**	
		Reactive	**Proactive**	**Reactive**	**Proactive**
Estimate	intercept	1.1268	-0.29957	0.8026	-0.1626
	coefficient	-0.2528	0.28170	-0.1791	0.2097
Std. Error	intercept	0.2467	0.23028	0.2572	0.2232
	coefficient	0.1050	0.09803	0.1095	0.0950

Table 4.4: Coefficients and Intercept values for $\mathcal{M}_1$ on Autonomy construct (scenario 1 and 2)

Given that $\mathcal{M}_1$ is the sole model exhibiting statistical validity and acceptability, we disregard the other models. A more detailed examination of $\mathcal{M}_1$ is provided in Table 4.4. Figure 4.6 and Figure 4.7, derived from Table 4.4, graphically represent the $\mathcal{M}_1$ prediction functions (depicted as lines) for the Autonomy construct. Both plots consistently indicate that the prediction value for the Amount of Autonomy (*i.e.*, the probability of choosing Reactive bots, denoted on the vertical axis) decreases as GitHub Activity (represented on the horizontal axis) increases. This inverse relationship implies that Inexperienced developers exhibit a preference for Reactive bots, with this preference diminishing as GitHub users become more experienced. In contrast, the propensity for developers to select Proactive bots heightens with increased GitHub activity. The intersection of these two lines defines a boundary, signifying the level of GitHub Activity at which this shift in preference takes place. According to $\mathcal{M}_1$, for the Autonomy construct, GitHub levels 1 and 2 (corresponding to Inex-

perienced and Intermediate, respectively) fall to the left of this boundary, indicating a predominant preference for Reactive bots. Conversely, Experienced GitHub users (GitHub activity = 3) demonstrate a preference for Proactive bots, as they fall to the right of the boundary line. Put shortly, in response to **RQ-1.1**, we state that **GitHub activity level is a major contributing factor affecting developers' choice of Autonomy. Inexperienced and Intermediate GitHub developers prefer Reactive bots while Experienced developers prefer Proactive bots.**

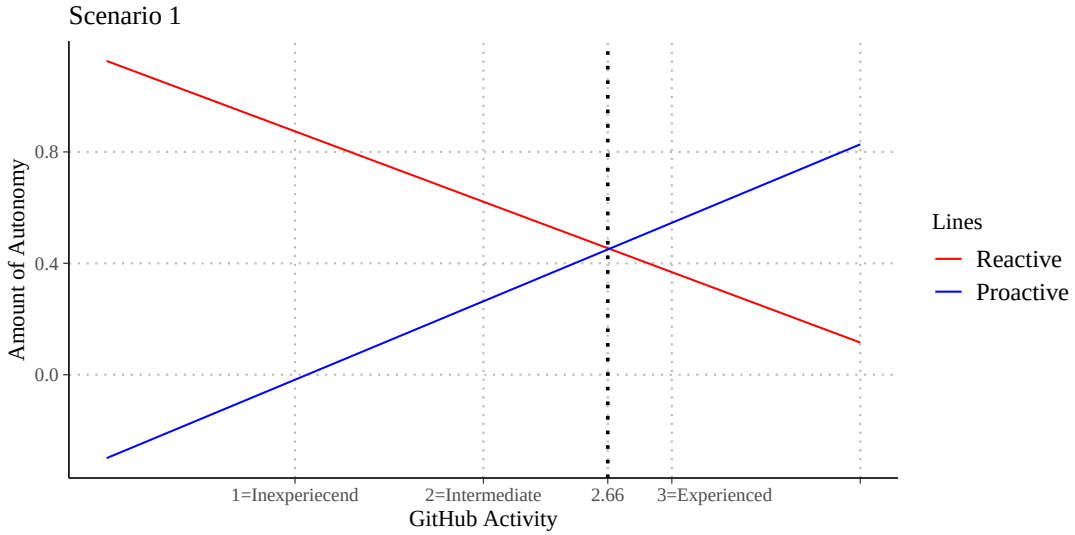

Figure 4.6: $\mathcal{M}_1$ visualization on Scenario 1

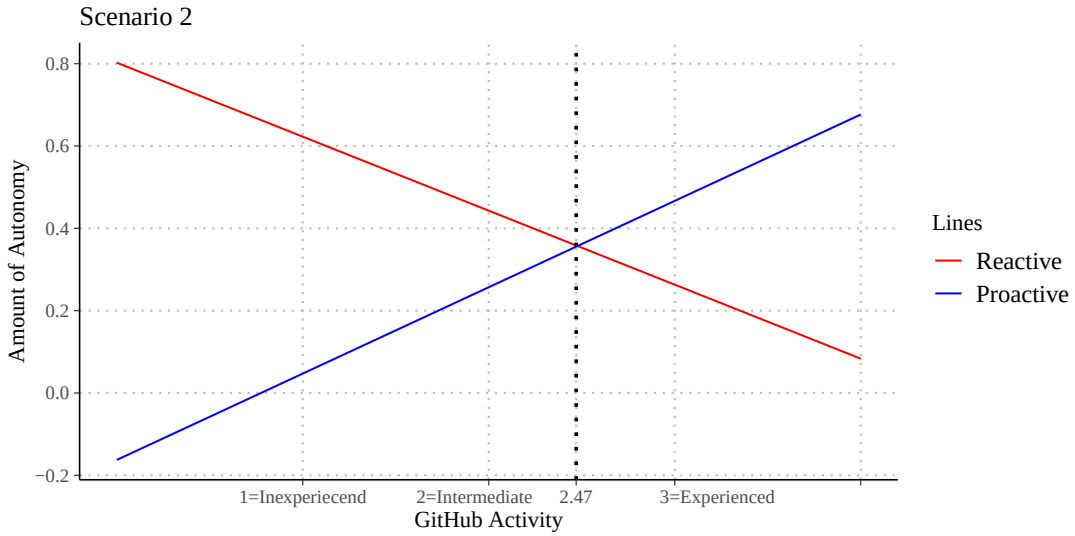

Figure 4.7: $\mathcal{M}_1$ visualization on Scenario 2

4.3.2 Generalized Linear Models

In addition to employing Linear Models, we further substantiated our findings using a Generalized Linear Model (GLM). GLMs enhance traditional Linear Models by incorporating two pivotal elements, namely the Link Function and the Response Distribution. The Link Function connects the linear predictor with the mean of the distribution function, essentially delineating how the expected value of the response interrelates with the linear amalgamation of the predictors. In conventional Linear Models, the identity function serves as the link function, suggesting that the expected value of the response is predicted directly from a linear combination of parameters. In contrast, GLMs offer the flexibility to utilize alternative link functions, such as the `logit` function which we employed for our binary data (*e.g.*, Reactive or not?). Furthermore, Response Distribution is another key distinction. Standard Linear Models assume that the residuals or error terms adhere to a normal distribution. However, GLMs allow for the response variable to exhibit any distribution from the exponential family of distributions. We tested Gaussian, binomial, Poisson, gamma, and inverse Gaussian distributions. This flexibility renders GLMs capable of managing response variables with error distributions beyond just normal, thus extending their range of applicability.

The identical model specification delineated in Table 4.2 was also employed for GLMs, utilizing the standard `glm()` function available in the default R setup. As a reminder, model $\mathcal{M}_1$ suggests GitHubActivity alone predicts outcomes in each case. Model $\mathcal{M}_2$ includes extra variables like PR expertise, bot experience, and software development experience to enhance the model's explanation. Model $\mathcal{M}_3$ explores how preferences for Autonomy and Persona might be linked across scenarios.

Multiple response distribution strategies were scrutinized, yet only the binomial distribution consistently produced dependable results across the same constructs and scenarios. The GLM aligned with the LM on $\mathcal{M}_1$ regarding the Autonomy construct, but failed to ascertain the interaction between GitHub activity and the Persona construct. This agreement was substantiated by lower Akaike Information Criterion (AIC)[1] values, presented in Table 4.5, relative to other models for the Autonomy

[1]The Akaike Information Criterion is a statistical parameter used for model selection and comparison. It primarily aims to achieve a balance between the model's goodness of fit and its complexity, thereby preventing overfitting. The AIC takes into account both the model's explanatory power with respect to the data and the number of parameters employed in the model. It penalizes models incorporating more parameters, thereby discouraging the inclusion of redundant variables that offer minimal contributions to the model's performance[37].

construct. Similar to LM, $\mathcal{M}_2$ failed to clarify the interaction of its independent variables with either the Autonomy or Persona construct by producing higher AIC in both constructs and p-values exceeding 0.05 (lowest p-value recorded across scenarios was 0.23). The notable insufficiency of both LM and GLM in modeling $\mathcal{M}_2$ is reflected in the absence of any statistically significant correlation between PR activity and Bot experience with either the Autonomy or Persona constructs (see subsection 4.2.1 and subsection 4.2.2).

$\mathcal{M}_3$ (*i.e.*, effects of Autonomy on Persona and vice-versa), tailored to the Persona construct, posits that a developer's choice of autonomy impacts their perception and consequently their preference for bot personality. Although LM fell short for $\mathcal{M}_3$, the GLM introduced a unique perspective for the Persona construct (scenarios 3 and 4). It revealed that among all GLM variations, modeling Persona, a binomial response distribution was able to capture the interaction of Autonomy and Persona in such a manner that the AIC score was consistently lower than all other models for the same construct (and same scenario).

	Scenarios			
Models	1	2	3	4
$\mathcal{M}_1$	63.047	58.91	62.417	74.083
$\mathcal{M}_2$	67.291	64.571	68.282	77.705
$\mathcal{M}_3$	70.578	64.058	59.207	71.973
Construct	*Autonomy*		*Persona*	

Table 4.5: AIC scores for GLMs with binomial response distribution and `logit` link function. For each scenario (column) the lowest AIC among GLM models with various different distribution strategies is marked with a green background colour.

As discernible from Table 4.6, the positive coefficient associated with Reactive responses (and correspondingly, the negative coefficient for Proactive responses) indicates that developers' inclination towards Personable bots in Scenario 3 is amplified if they express preference for Reactive bots in Scenario 1 and 2 (p-value = 0.0472). The same relationship is observable in Scenario 4, although such an interpretation should be viewed with due caution given the produced p-values slightly greater than 0.05. The intercept or baseline for Scenario 3 implies that in the absence of information regarding developers' choice of Autonomy, the model predicts a dominant preference for Personable bots as the chosen bot personality that is also observed in Figure 4.1.

Based on Scenario 3, we can assert that developers' selection of Persona is in-

Scenario	Autonomy Choice	intercept	coefficient	std error	p-value
3	Reactive	6.8145	2.2387	0.3429	0.0419
	Proactive	0.8301	-1.8373	0.2718	0.0381
4	Reactive	7.9182	1.3428	0.4518	0.06493
	Proactive	2.3671	-0.872	0.4918	0.08745

Table 4.6: The computed values of the intercept and coefficient for $\mathcal{M}_3$ as it models Scenario 3 and 4. The *'Autonomy Choice'* columns present the preference of developers for a specific Autonomy value, which is either Reactive or Proactive.

fluenced by their prior responses or biases toward a bot's Autonomy. With regard to **RQ-2.1**, although we were unable to identify any of the pre-defined factors (*e.g.,* GitHub activity, PR activity, Bot experience) as influential, we have demonstrated that a **developer's preference towards a bot's Autonomy influences their perception of bot Personality. Developers who favour Reactive bots tend to show a preference for Personable bots. Conversely, developers who lean towards Proactive bots have a preference for Factual bots.**

4.4 Open-ended Responses

In addition to choosing a side in response to a scenario (e.g., Reactive or Proactive for Autonomy construct) or maintaining neutrality, we asked our participants to provide a brief explanation for their choice, using a maximum of 512 characters in a specified field (for further details, refer to subsection 3.2.3). To gain a more comprehensive understanding of how respondents interpret autonomy and persona, we open-coded their reasons for their preferences. The analysis of open-ended responses also served to validate that our constructs accurately captured participants' reactions. These responses echoed the themes identified during the Phase 1 interviews, reinforcing the validity and relevance of our constructs.

Three researchers, including the author of this thesis, independently coded the open responses from the participants in three rounds. In order to eliminate researchers' bias while coding the responses, we detached any association between the respondent's choice of Autonomy or Persona and the specific scenario number that the researcher was coding. This process involved including only the open answers and linking them to a unique response identifier. By implementing this approach, the researchers were kept unaware of the respondent's preferences in any given scenario

while coding. This measure ensured a more objective analysis, minimizing the potential for unconscious biases to influence the interpretation of the responses, and thus maintaining the integrity and reliability of the research findings.

Following each round, the coders addressed any disagreements to enhance the collective comprehension of the coding task and improve the coding guide. A preliminary coding guide was established, incorporating themes discovered in the literature and the Phase I interviews. This guide was refined through the coding rounds, where it was narrowed down and ultimately resulted in five high-level codes. Each of these codes contained two sub-codes, representing opposing polarities, as shown in Table 4.7. We also included a code for comments stating that they have no preference, or for noise/unusable data.

We had substantial Kappa Fleiss inter-rater agreement of $\kappa = 0.723$ by the conclusion of the final round. Kappa Fleiss inter-rater coding score, also known as Fleiss' kappa, is a statistical measure used to assess the level of agreement among multiple raters or coders when assigning categorical ratings to items or subjects. It quantifies the degree of agreement beyond what could be expected by chance alone. The value of Fleiss' kappa ranges from -1 to 1, with 1 indicating perfect agreement, 0 suggesting agreement no better than chance, and negative values representing worse-than-chance agreement[22].

A significant advantage of open-ended responses for preference justification is the process of model-response alignment. We define model-response alignment as the process of using our model predictions of developers' preferences on Autonomy and Persona, and aligning them with respondents' justifications, coded according to the guide in Table 4.7. Alongside developer categorization, as introduced in subsection 4.2.1 and subsection 4.2.2, we can offer further qualitative insights into our quantitative analysis and the explanation of our models. The following subsections will discuss two distinct facets of model-response alignment: subsection 4.4.1 will focus on the Autonomy constructs (scenarios 1 and 2) and extract open responses given by respondents to verify and elucidate model $\mathcal{M}_1$. Similarly, subsection 4.4.2 will aim to explain model $\mathcal{M}_3$ using comments given in response to the Persona construct (scenarios 3 and 4). We will also attempt to clarify the reasoning behind respondents' choice of remaining neutral in both constructs.

Code	**Polarity** (freq.)	**Definition**
Noisiness	Noisy (4) / Clear (41)	The bot clutters the conversation by sending too many messages or pinging developers frequently, OR messages are clear, short and easy to understand.
Persona	Polite (40) / Rude (7)	The bot is friendly and the language used by this bot is similar to human conversation OR The bot does not care about human feelings in their messages and actions.
Control	Bot-initiated (30) / Human-initiated (41)	The bot is in control, initiates actions, and is accountable for these actions OR The developer is in control of the bot, initiates it, and is responsible for the bot's actions
Productivity	Boosts (2) / Hurts (1)	The bot is efficient and improves the workflow by saving time and resources OR The bot is inefficient, consuming resources and wasting time
Information Content	Rich (19) / Poor (1)	The bot is configured to provide enough information for the assigned task OR The bot does not produce useful information

Table 4.7: Open coding results for the open-ended question on rationale. Frequency of observed codes in the entire dataset is presented after each polarity value.

4.4.1 Model $\mathcal{M}_1$ Alignment

In model $\mathcal{M}_1$, we discerned a pattern indicating that the likelihood of selecting a Reactive bot declines as the level of GitHub activity increases. Specifically, within the context of the Autonomy construct (see Figure 3.3 and Figure 3.4), the model elucidated that Inexperienced and Intermediate developers exhibit a preference for Reactive bots, whereas Experienced developers are inclined towards Proactive bots. To further investigate this relationship, we proceeded to analyze the open responses, employing GitHub activity and Autonomy as selective filters.

4.4.1.1 Experienced GitHub Developers

A majority of responses from Experienced GitHub developers favouring a Proactive bot were classified under the *Control* code, comprising 72.1% of such responses. Though there is almost an even split among Experienced GitHub developers who favoured Proactive bots (47.5%) and Reactive bots (45%), with a minority remaining Neutral (7.5%), the underlying reasoning for their preferences unveils a distinct viewpoint. The principal rationale provided was the belief that bots should undertake actions or instigate conversations autonomously, thereby minimizing human intervention. For instance, Respondent P30 expressed that, "*A user should not need to invoke the bot; the bot should be automatically triggered by some relevant user action,* e.g. *a pull request being created. It's a waste of users' time and focus to need to remember to invoke this kind of functionality.*" Similarly, P13 asserted, "*When bot actions are triggered for every pull request, it can find bugs that programmers are not aware of.*" P41 expressed a preference for automatic test runs, stating, "*I prefer automatic test runs over rather me having to initiate the bot to run the tests.*" Offering a unique viewpoint, P19 succinctly highlighted the appeal of Proactive bots by noting, "*Less interaction required.*"

Alongside the reduction of developer-bot interaction, there were recurring references to secondary factors such as bot configuration and the context of conversation as influential in the preference for Proactive bots among Experienced GitHub developers as P20 states"*It seems like this bot is designed to automatically take action on commits, and therefore it should not require human interaction to initiate it.*"

Conversely, the reasoning provided by proponents of Reactive bots among Experienced GitHub developers takes on a different dimension. In their responses, 59.4% were tagged with the *Noisiness* code, 36.3% with the *Control* code, and 4.3% with

other codes. Those in favour of Reactive bots emphasized concerns that fully autonomous bots can create unnecessary noise and clutter conversations. To many, such a high degree of autonomy seems superfluous. For instance, P16 noted, "*feels weird the bot starting something on its own. i feel like it would quickly clog up the conversation.*" Similarly, P25 expressed, "*I think that any action that could potentially add needless clutter to the repo (like for instance, as in the example if the bot doesn't actually contribute useful code), it shouldn't be triggered automatically.*" P9 specifically highlighted concerns about spam, saying, "*I believe that bots that automatically interact on their own can result in a lot of spam or false calls. In this case, we see that the bot's suggested fix was not an actual fix to a problem, and thus a bot capable of automatically generating patches and suggesting them could result in hundreds upon hundreds of patches if the bot malfunctions.*"

In addition to the issue of clutter, some respondents were apprehensive about bearing the responsibility for a bot's incorrect or breaking changes. P2 revealed this sentiment by stating, "*I prefer the right side because the person requests the bot to try a patch. Even though both attempts by the bot are incorrect, I don't want the possibility of the bot going out on its own to try to do fixes if it may consistently get them wrong. If that were the case, there could be a lot of invalid patches to go through.*" This perspective was supported by P18, who similarly articulated, "*If there is a chance of it being wrong I want to call on it myself.*" P12 highlights the importance of information content the clarity of conversation by stating "*I rather see everything that contributors write, even bots commands*".

4.4.1.2 Inexperienced and Intermediate GitHub Developers

Among Inexperienced and Intermediate (non-expert) GitHub developers, a breakdown of their preferences shows that 61% voted for Reactive bots, 20.8% for Proactive bots, and 18.2% of votes remained Neutral. The rationale behind these choices was primarily divided between the *Noisiness* (64.1%) and *Control* (29%) codes, with other codes accounting for the remaining 6.9%.

Similar to Experienced GitHub developers, non-expert developers expressed concerns about bots generating noise and potentially spamming conversations. They saw these issues as primary reasons for preferring either Reactive or Proactive bots. For instance, P3 posited, "*Actions that could potentially add needless clutter to a repo should only be triggered by a human, and not automatically.*" However, some

responses from this group revealed a slightly different perspective, highlighting that interactions with a Reactive bot appeared more clear and straightforward. For example, P52 suggested that, "*The flow is more detailed,*" a sentiment supported by P14, who added, "*It's more clarifying in its response.*" These comments reflect a nuanced view of the autonomy construct among non-expert developers, emphasizing not only the potential drawbacks of automated action but also the communicative benefits of a more controlled and human-triggered interaction.

Non-expert developers also exhibited apprehension regarding the implications of an autonomous bot and advocated the idea that a human developer should invoke bots to execute actions. P7 expressed this by stating, "*I prefer the option for a real life human to call the bot before it acts. Here it makes sense that a human should have to call the bot for bug checking, but it doesn't if it is a small error.*" This perspective was reinforced by P10, who elucidated, "*Given that the bot identified a false positive I would imagine that if the bot was to run automatically on every commit and then produce a large number of false positives developers might get frustrated and ignore its messages completely. It can be very time consuming for them to go through each error to validate its legitimacy. It's best then if the developer can choose which commits they want the bot to run on - maybe more sensitive ones - and dedicate less time validating the bot's results on trivial commits.*" This observation sheds light on the negative impacts of Proactive bots on developers' productivity. P32 encapsulates numerous viewpoints favouring Reactive bots by proclaiming, "*The programmer must call the bot, giving the programmer more control.*"

Advocates for Proactive bots within the non-expert GitHub developer group contend that an autonomous bot can enhance their productivity by detecting and rectifying mistakes that might otherwise go unnoticed. P55 articulates this view by stating, "*automatic bug fixing does sound good as the user wouldn't necessarily know if there were any bugs to check.*" Similarly, P38 perceives Proactive bots as beneficial for the developer experience, asserting, "*The bot can generate a patch without needing to be asked so makes life easier.*" Furthermore, P21 emphasizes the significance of the bot's ability to act autonomously to mitigate the potential risk of faulty code, remarking, "*When bot actions are triggered for every pull request, it can find bugs that programmers are not aware of.*" Certain respondents maintain that Proactive bots foster conversations that are quicker to read due to reduced interaction and more succinct dialogue. P1 encapsulated this idea by stating, "*Less communication between bot and user but information requested is concise*", implying that the discussion be-

comes more accessible with fewer instances of human-bot interaction. Though they are a minority within this demographic, non-experts who favour Proactive bots offer cogent arguments for their preferences, arguments that resonate with the reasoning employed by expert GitHub developers.

4.4.1.3 Neutral Participants

A significant portion of feedback from respondents who held a neutral position regarding the Autonomy construct did not furnish in-depth explanations. Many employed variations of phrases[2] such as "*No preference*", "*No strong opinion*", or "*No difference*" to elucidate their neutrality. Nevertheless, insights from two Experienced GitHub developers shed light on the rationale for their neutrality. They propose that the bot's degree of autonomy is contingent upon the specific objectives of the repository and the prevailing community standards, and they recognize that the context in which these bots are deployed may also influence their level of autonomy. P31 clarified their neutral stance concerning bot autonomy by articulating, "*I feel that having the bot be called versus it running automatically is a matter of purpose in the repository and either is valid.*" Similarly, P47 commented, "*I think either automatically running or running on call is a matter of usefulness in the repository.*"

4.4.2 Model $\mathcal{M}_3$ Alignment

Model $\mathcal{M}_3$ (refer to Table 4.2) demonstrated the highest explanatory power for the Persona construct (see Figure 3.5 and Figure 3.6), confirmed by its lowest AIC score (see Table 4.5). This model implies that the preference for a specific bot personality (Factual or Personable) is contingent on developers' choices regarding bot Autonomy. Building upon our observed phenomena that a majority of developers prefer Personable bots (see Figure 4.1) and considering the findings from model $\mathcal{M}_1$ that described Autonomy, we postulate that developers who opt for Reactive bots would lean towards a preference for Personable bots, while those who select Proactive bots would favour Factual bots. To elucidate the relationship between the two constructs, we selectively analyzed respondents' justifications, first based on Autonomy and then on Persona.

[2]The precise phrasings are not delineated here, but they have been distilled into three representative statements that encapsulate their essence

4.4.2.1 Reactive Personable Bots

Through the alignment of responses with model $\mathcal{M}_1$, it has come to light that developers who selected Reactive bots perceive conversations as human-driven and attribute the responsibility of invoking bot actions to humans. This human-centric design is similarly evident when these developers express a preference for Personable bots. Proponents of the Reactive bot put forth the argument that a Personable bot exhibits a greater degree of friendliness and human-likeness, resulting in more enjoyable and pleasant conversations. Moreover, these developers asserted that Personable bots tend to be less intimidating and condescending, particularly in instances where a bot is declining a human developer's request. The rationale of developers (anonymized using PID) for selecting both Reactive and Personable bots is presented in Table 4.8.

Irrespective of Autonomy preferences, developers who expressed a preference for Personable bots universally emphasized the necessity for bot-human interactions to adhere to principles of human-centric design. As indicated by Table 4.7, 40 responses were categorized under the label *Polite*, with 38 instances specifically characterizing a Personable bot within the Persona construct (scenario 3 or scenario 4). Terms such as *Friendly*, *Human-like*, *Pleasant*, *Approachable*, and *Polite* featured prominently in the justifications for choosing Personable bots. Certain respondents also portrayed the Factual bot as displaying a *Rude* demeanor towards human developers, with instances such as P6 from Table 4.8 or P17, who selected a Personable bot and remarked, "*This bot is friendlier (less scolding).*" These observations highlight that the Factual bot's responses were perceived as abrasive in comparison to a Personable bot.

4.4.2.2 Proactive Factual Bots

Contrary to the developers who favoured Reactive and Personable bots, we formulated a hypothesis suggesting that proponents of Proactive bots are more likely to lean towards Factual bots. Our data substantiates this hypothesis, as 63.1% of developers who expressed a preference for Proactive bots also opted for Factual bots. Upon investigating the open-ended responses, a sampling of which is listed in Table 4.9, it became apparent that developers in this category are inclined towards receiving critical information with minimal interaction, leading to succinct conversations. In support of the Factual bot, P56 indicated that they are "*More professional. Left side*[3] *is too wordy.*" An intriguing comment was made by P25, who acknowledged

[3]Personable bot in scenario 3

PID	Rationale for Reactive bot	Rationale for Personable bot
P39	It seems like this bot is designed to interact with people, and therefore I think a human should start the interaction with the bot (not the bot itself).	This bot is designed to interact with people, and I therefore think it is very important that it output text in a way that is human-centered and friendly.
P18	If there is a chance of it being wrong I want to call on it myself.	Is more friendly and approachable.
P43	I prefer that users call upon bots to do work, not have bots do work automatically.	Bots that interact more humanely and not logically can result in more pleasant interactions overall.
P6	I prefer when bots, especially for pull requests are explicitly called. They may be called every time but the interaction of calling them generates an easy to follow conversation rather than just having a message appear on a PR.	More human response that is less mechanical and abrasive than the straight to the point message on the right side. The left side is generally nicer (please close this) and is less attacking. The right side made me feel defensive about the pull request compared to the left side.
P16	feels weird the bot starting something on its own. i feel like it would quickly clog up the conversation.	feels more polite and more human to read. and honestly, its more understandable.

Table 4.8: Examples of human-centric design rationales provided by developers who preferred Reactive and Personable bots

the friendly nature of the Personable bot but still favoured the Factual bot, stating, "*Now this is interesting, though the left was more friendly, it was way too wordy. The right got the point across far better.*"

Furthermore, as demonstrated by Table 4.9, these developers perceive bots as tools configured to autonomously execute tasks, thus saving time and effort on explicit commands. This perspective aligns with the concept of *robot-ness*, suggesting that bots should behave distinctly as bots and avoid human-like behavior. This distinction between bots and humans is emphasized by these developers.

4.4.2.3 Neutral Participants

Upon examining the open-ended responses from participants who held a neutral stance regarding the Persona construct, no novel insights were uncovered. Their responses closely resembled those of other developers with strong preferences, listing the pros and cons of both Personable and Factual bots, yet unable to prefer one over the other. Alternatively, some neutral participants did not provide a distinct explanation for their neutral position.

Participant number	Rationale for Proactive bot	Rationale for Factual bot
P28	The bot should automatically run without the need of a request from a developer.	I prefer the bot to be as straight forward as possible in its instruction. It doesn't need to pretend to be a human with being more polite, I know its a bot.
P11	I would rather a bot fail after them suggesting a change and not after I specifically asked for help.	succinct and to the point.
P19	Less interaction required.	I do not expect courtesy from a bot. I strongly prefer more straight-to-the point communication from bots.
P41	I prefer automatic test runs over me having to initiate the bot to run the tests.	I prefer reading concise feedback that clearly states the issue and solution.
P22	Less intervention.	Prefer bot messages to read like bot messages.

Table 4.9: Example rationales provided by developers who preferred Proactive and Factual bots

4.5 Chapter Summary

In this chapter, we analyzed data from 56 respondents. In response to **RQ-1** and **RQ-2** (i.e., developers' preferences for Autonomy and Persona), our initial descriptive analysis in section 4.2 revealed that most developers favor Reactive and Personable bots for PR discussions. This finding guided the formulation of hypotheses and a testing framework for further data examination.

To address **RQ-1.1** and **RQ-2.1** (i.e., factors influencing extreme preference), we established three criteria from screening and demographic questionnaires: GitHub activity level, Bot expertise, and PR experience. Among these, only GitHub activity level exhibited a statistically significant correlation with Autonomy.

To delve deeper into the data and test our hypotheses, we employed two modeling approaches using LMs (refer to subsection 4.3.1) and GLMs (refer to subsection 4.3.2), using our three criteria as inputs. Three model descriptions (refer to Table 4.2) were formulated, each involving different combinations of contributing factors.

LMs could only explain $\mathcal{M}_1$, which accounted for GitHub activity's influence on developers' preference for Autonomy. $\mathcal{M}_1$ indicated that as GitHub activity level rises, developers tend to prefer Proactive bots. Furthermore, the plots in $\mathcal{M}_1$ (see Figure 4.6 and Figure 4.7) showed that Inexperienced and Intermediate developers prefer Reactive bots, while Experienced developers favor Proactive bots. GLMs, while confirming our Autonomy conclusions from LMs, shed light on the Persona construct. The lowest AIC scores (see Table 4.5) were consistently found for $\mathcal{M}_3$ in scenarios 3 and 4. However, since p-values slightly deviated from 0.05 (see Table 4.6), caution is advised when interpreting these results. $\mathcal{M}_3$ suggested that Autonomy choice influences Persona preference. Our analysis revealed that developers who preferred Reactive bots tended to prefer more personable bots.

In section 4.4, we detailed the analysis of open-ended responses. Respondents provided justifications for their choices on both constructs in all four scenarios. Three researchers iteratively labeled these responses and achieved an inter-rater agreement of $\kappa = 0.723$. We used these responses and associated codes to align our models and support our modeling results (see subsection 4.4.1 and subsection 4.4.2). Despite aligning with our models, open-ended responses did not definitively determine the preferable extreme of Autonomy or Persona. Instead, they illustrated the intricate interactions and subjectivity within the open-source community's views on bots.

Chapter 5

Discussion, Future Work & Conclusion

5.1 Discussion

5.1.1 Bot Autonomy and Developer's Perception

Numerous research studies have highlighted that both users and professionals exhibit apprehension toward autonomous agents, extending beyond the GitHub pull request environment [31, 45, 43]. Wessel *et al.* have also observed that autonomous bots can introduce noise and disruptions to developers and project workflows [56]. In the context of our study, we have identified a pattern where more experienced developers tend to favour proactive bots in comparison to their less experienced counterparts, who often lean towards reactive bots. While participants in our study also cited *noise* as a justification for their preferences, aligning with prior research findings, we contend that the confidence or tolerance for proactive bots increases as GitHub users gain more experience. Conversely, less experienced developers might perceive proactive bots as either too imposing or overly assertive. Erlenhov *et al.* demonstrated that newcomers may feel overwhelmed or intimidated by bots [19]. Building upon this insight, our study contributes to this line of research by demonstrating that the autonomy of bots directly impacts the perceptions of new developers and users, with fully autonomous bots exerting a negative influence in this context.

Within the rationales provided by our participants, we identified a multitude of contradictory viewpoints pertaining to the Autonomy construct. Respondents seemed to cluster into distinct groups, each holding opposing beliefs about the appropriate

degree of autonomy a bot should exhibit. Some advocated for bots to be invoked, while others held the contrary perspective. Similarly, some participants perceived proactive bots as sources of noise, while others recognized their potential to enhance productivity. Furthermore, opinions diverged on whether clear bot-human interactions were essential when a bot generated results relied upon by developers, with some considering such interactions vital and others viewing them as wasteful distractions.

Since determining the superiority of these conflicting viewpoints is challenging, we argue that all these opinions carry equal significance. Each perspective unveils a facet of the intricate and dynamic interaction at play. Particularly within the context of Autonomy, we refrain from making claims based on objective measurements. Instead, we align our argument with our two Neutral respondents (P31 and P47), who argued that the degree of a bot's autonomy should hinge upon the objectives of the repository and the norms of the community. Consequently, we propose that bot designers consider adjusting the levels of autonomy based on developers' experience levels. This approach acknowledges the complexity of the issue and tailors autonomy to the specific context.

> **Recommendation #1**: Developers should be given options to scale the autonomy of the bots involved in their development process. For instance, they could choose between *Proactive* and *Reactive* Or something in between those extremes.

Developers could also be provided with the option to select bots they find comfortable to interact with, as well as the choice to mute bots that they might prefer to minimize interactions with. Nonetheless, this customization should be rooted in developer preferences, as highlighted by both our survey and the study by Erlenhov *et al.* [19]. One viable approach could involve capturing developer responses to bot comments, possibly through mechanisms such as using a set of emojis or incorporating a hyperlink to a dedicated feedback questionnaire page created by bot developers. This user-friendly interface can be seamlessly integrated into bot messages, ensuring easy accessibility for developers to share their thoughts and tailor their interactions with bots according to their individual preferences. Therefore,

> **Recommendation #2**: In order to tune bot behaviors, projects must have a better way of getting feedback on bot behavior and developer preferences.

5.1.2 Bot Personality and Developer's Perception

The manner in which bots communicate significantly impacts how users perceive them, as demonstrated in prior research by Clark et al. [14]. Volkel et al. also observed that users of bots may be more inclined to embrace bots that mirror their own personalities [50]. Our study reveals that developer preferences for bot personas are contingent on the specific context presented in a vignette. This underscores the practical significance of persona and emphasizes that it should not be an afterthought in bot design.

In light of these findings, it becomes imperative to offer developers options to shape the persona of bots involved in their development endeavors. This is particularly relevant given the established notion that bots exhibiting higher levels of anthropomorphism tend to cultivate greater trust with developers [45, 30, 39]. Indeed, our research findings align with the studies mentioned, as a significant majority of respondents favored Personable bots, valuing qualities like *friendliness*, *human-centeredness*, and *politeness*. These participants reported positive and pleasant interactions, thereby corroborating the alignment between our findings and the previously cited research studies. However, it's important to acknowledge that contrasting viewpoints also exist within the Persona construct. Some participants regarded the Personable bot as *wordy*, perceiving its attempt to emulate human conversational style as futile for a bot. They highlighted the expectation for bots to communicate succinctly. This dual perspective underscores the complex nature of the Persona construct and the diverse expectations that users hold when engaging with bots. Therefore,

> **Recommendation #3:** Developers should be able to select and change the persona of a bot based on their own preferences.

One approach could involve allowing developers to choose a persona from an assortment of predefined personas and communication styles. For instance, some respondents expressed a preference for bots that strictly convey information without non-informative utterances.

5.2 Threats to Validity

Trustworthiness: The trustworthiness of our qualitative analyses and conclusions was assessed through the lenses of credibility and confirmability. To minimize the

potential bias stemming from a single labeler, we employed robust labeling practices involving multiple authors and clear guidelines. Furthermore, we previously reported on rater reliability concerning the coding of survey responses.

In terms of transferability, we believe that our study adheres to the transferability standards by demonstrating the applicability of our results to other comparable contexts, particularly within free and open source communities. Additionally, we enhanced transferability by providing comprehensive descriptions of the study's scope, participants' diverse experiences, methodology execution, and quantitative outcomes.

Internal Validity: To uphold internal validity, we piloted the survey with members of our lab. Designing the survey involved careful trade-offs. We randomized the order of vignettes for each participant and maintained simplicity in certain questions to minimize potential experimental bias, such as fatigue. Altering the context of each vignette aimed to prevent a learning effect. Furthermore, we balanced the text length on both sides of a vignette.

While it's true that both Vignette 1 and 2 showcased a bot's rejected pull request, potentially influencing preferences or perceptions of the acceptability of an autonomous bot, we observed explicit mentions of the bot's failure in the justifications provided by respondents who favored the reactive bot.

Construct Validity: The complexity and personal nature of constructs like Persona and Autonomy necessitated operationalizations that, while capturing essential aspects, inevitably excluded certain nuances. The open-ended answers aimed to enable respondents to express this complexity more comprehensively. We measured Preference by asking respondents to indicate their favored scenario within a vignette. This approach allowed respondents to compare and evaluate the two poles of the construct presented in the scenarios. Our open-ended answers ensured we grasped the underlying rationale for their preferences, further establishing alignment with the constructs under examination.

However, the need for circumspection in interpreting the results stemming from our GLM model (see subsection 4.3.2) within the framework of scenarios 3 and 4 is underscored. This caution is necessitated by the modest divergence in the AIC scores (see Table 4.5) and the subtle departure of p-values from the customary significance threshold of 0.05. Moreover, it is important to acknowledge that the intricacies associated with the implementation of a bot's personality, as elucidated in section 3.3, and its intricate interplay with autonomy, may serve as a potential source of ambiguity for survey respondents. Consequently, this can give rise to an elevated degree

of variability and distinctive patterns observed from scenario 3 to scenario 4.

External Validity: Our survey respondents were carefully screened for basic software development knowledge and pull request use. We sampled from a mix of self-identified software industry professionals via Prolific, as well as third-year software engineering students at various times. The employment of screening questions and the use of multiple Prolific iterations aligns with current recommended practices [17, 16].

While there could be a potential bias in individuals opting to participate in Prolific studies, and our dataset lacked participants over the age of 35 or with more than 5 years of experience, we didn't observe any notable distinctions based on age or experience within our data. This observation suggests that our study's external validity is more resilient to potential biases related to participant selection.

5.3 Future Work

Our study primarily centered on exploring the pivotal themes of autonomy and persona, suggesting the necessity for increased bot configurability and awareness of developer perceptions. Both maintainers and developers should possess the ability to extensively tailor the communication styles employed by bots. However, it's important to acknowledge that there are additional dimensions to developer preferences concerning bots. Themes such as trust, past experiences with bots, bot appearance, and identity emerged during Phase I interviews and survey responses. To comprehensively comprehend the roles that bots play in software development, these themes necessitate further comprehensive investigation.

As the interactions between software bots and developers continue to grow in complexity, a broader array of theories and findings related to behavior, trust, and language will become essential. Concepts such as social norms from Bicchieri [8], Searle's speech acts [44], or Politeness Theory from Brown and Levinson [10] serve as illustrative examples, underlining the potential for fruitful trans-disciplinary research in this field. The dynamic interplay between these factors presents a rich avenue for exploration, offering insights into how software bots can be optimally designed to enhance collaboration and productivity in software development environments.

In future research endeavors, it would be valuable to delve into the influence of other factors on developers' preferences for bot personas. For instance, the unique attributes of open source projects could guide the design of predefined personas that align with community preferences. Alami *et al.* [1, 2] identified three distinct but

not mutually exclusive styles of governance in open source communities: lenient, equitable, and protective. These styles pertain to PR (pull request) governance processes, with lenient valuing inclusiveness, equitable prioritizing community rules, and protective emphasizing trust and relationships [1]. Exploring various bot persona styles that resonate with these community styles could potentially lead to default choices that cater to contributors' inclinations.

Moreover, our study has not fully uncovered why developers' persona preferences are tied to the specific task at hand. Future investigations could delve deeper into the intricate interplay between bot personas, software development tasks, and the personalities of developers. Understanding these dynamics could provide nuanced insights into how bot personas can be optimally tailored to enhance developers' engagement and efficiency in diverse software development scenarios.

Ultimately, a method to investigate the spectrum of diversity in persona expression involves the utilization of Large Language Models (LLMs) as the principal linguistic interface for bots. Contemporary LLMs such as ChatGPT, Llama, or BERT possess the capacity not only to maintain a persistent personality archetype (*e.g.*, ENFJ) consistent with trait theory but also to undergo alterations and adjustments in specific facets of their personality traits [26].

5.4 Conclusion

In this thesis, we embarked on a comprehensive exploration of the factors that shape developers' perceptions of GitHub bots. Our journey commenced with an interview study, which served as a platform to extract key themes characterizing the utilization and perception of software bots among projects and developers. Through this process, two crucial constructs emerged as pivotal influencers of perceptions: the degree of Autonomy exhibited by a bot and its Persona. Subsequently, employing a vignette-based survey, we harnessed controlled scenarios to delve deeper into respondents' reactions to bots within a simulated pull request (PR) discussion setting. The insights drawn from this survey enriched our understanding of developers' preferences and reactions, especially concerning the interplay between autonomy and persona.

Our study uncovered a noteworthy trend: participants with limited software experience leaned towards preferring reactive, less autonomous bots, yet were more inclined to tolerate personas characterized by friendliness. This finding underscores the significance of tailoring bot development to account for the preferences and tol-

erance levels of developers with varying degrees of expertise. It serves as a reminder that in the endeavor to design effective bots, the intricate balance between autonomy and persona must be a focal consideration.

The implications of our research stretch beyond the immediate context of software development. The degree to which a bot operates autonomously and the persona it portrays influence not only how bots are perceived, but also the efficacy of their interactions. As the world of software development evolves and becomes more reliant on automation and collaboration, understanding these nuances is paramount for harnessing bots to their fullest potential.

In light of our findings, we advocate for a more configurable approach to bot development within software projects. Empowering both maintainers and developers to fine-tune the communication styles and autonomy levels of bots aligns with the diversity of preferences and perceptions we observed. This enhanced configurability can create a harmonious interaction between bots and developers, enhancing the collaborative atmosphere and efficiency within software development processes.

However, our study is but a stepping stone in a broader realm of research that warrants exploration. Factors such as trust, past bot experiences, appearance, and identity also play roles in shaping developer preferences towards bots. These dimensions merit further comprehensive investigation to unravel their implications for software development environments. The complexities woven into the interaction between developers and software bots beckon for a broader set of theories and findings centered around behavior, trust, and language. The concept of social norms, speech acts, and politeness theory are but a few avenues ripe for exploration, offering prospects for transdisciplinary research that can illuminate the intricate dynamics of this evolving landscape.

In conclusion, this study delved into the intricate fabric of developer perceptions regarding GitHub bots. The constructs of autonomy and persona emerged as potent influencers, guiding preferences and reactions. Our findings not only enrich our comprehension of software development practices but also shed light on the broader spectrum of human-bot interactions. As technology advances and collaboration becomes increasingly reliant on automation, the lessons learned from this exploration will undoubtedly prove invaluable in shaping the future of software development.

Bibliography

[1] Adam Alami, Marisa Leavitt Cohn, and Andrzej Wasowski. How do FOSS communities decide to accept pull requests? In *Proceedings of the Evaluation and Assessment in Software Engineering*. ACM, apr 2020. URL: https://doi.org/10.1145%2F3383219.3383242, doi:10.1145/3383219.3383242.

[2] Adam Alami, Raul Pardo, Marisa Leavitt Cohn, and Andrzej Wasowski. Pull request governance in open source communities. *IEEE Transactions on Software Engineering*, 2021.

[3] Fernando Almeida André Queirós, Daniel Faria. Strengths and limitations of qualitative and quantitative research methods. *European Journal of Education Studies*, 3(9), 2017. doi:10.5281/zenodo.887089.

[4] Christiane Atzmüller and Peter M. Steiner. Experimental vignette studies in survey research. *Methodology*, 6(3):128–138, 2010. arXiv:https://doi.org/10.1027/1614-2241/a000014, doi:10.1027/1614-2241/a000014.

[5] Aaron Baird and Likoebe M Maruping. The next generation of research on is use: A theoretical framework of delegation to and from agentic is artifacts. *MIS quarterly*, 45(1), 2021. doi:10.25300/MISQ/2021/15882.

[6] Z. Al barakeh, S. alkork, A. S. Karar, S. Said, and T. Beyrouthy. Pepper humanoid robot as a service robot: a customer approach. In *2019 3rd International Conference on Bio-engineering for Smart Technologies (BioSMART)*, pages 1–4, 2019. doi:10.1109/BIOSMART.2019.8734250.

[7] Caio Barbosa, Anderson Uchôa, Daniel Coutinho, Filipe Falcão, Hyago Brito, Guilherme Amaral, Vinicius Soares, Alessandro Garcia, Baldoino Fonseca, Marcio Ribeiro, and Leonardo Sousa. Revealing the social aspects of design decay: A

retrospective study of pull requests. In *Proceedings of the XXXIV Brazilian Symposium on Software Engineering*, SBES '20, page 364–373, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3422392.3422443.

[8] Cristina Bicchieri. *Norms in the Wild: how to Diagnose, Measure and Change Social Norms*. Oxford University Press, 2016.

[9] Chris Brown and Chris Parnin. Sorry to bother you: Designing bots for effective recommendations. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, pages 54–58, 2019. doi:10.1109/BotSE.2019.00021.

[10] Penelope E. Brown and S. Levinson. *Politeness: some universals in language usage*. Cambridge University Press, 1987.

[11] Ana Paula Chaves, Jesse Egbert, Toby Hocking, Eck Doerry, and Marco Aurelio Gerosa. Chatbots language design: The influence of language variation on user experience with tourist assistant chatbots. *ACM Transactions on Computer-Human Interaction*, 29:1–38, 4 2022. doi:10.1145/3487193.

[12] Ana Paula Chaves and Marco Aurelio Gerosa. How should my chatbot interact? a survey on social characteristics in human–chatbot interaction design. *International Journal of Human–Computer Interaction*, pages 1–30, November 2020. doi:10.1080/10447318.2020.1841438.

[13] Ashish Chopra, Morgan Mo, Samuel Dodson, Ivan Beschastnikh, Sidney S. Fels, and Dongwook Yoon. "@alex, this fixes #9": Analysis of referencing patterns in pull request discussions. *Proc. ACM Hum.-Comput. Interact.*, 5(CSCW2), oct 2021. doi:10.1145/3479529.

[14] Leigh Clark, Nadia Pantidi, Orla Cooney, Philip Doyle, Diego Garaialde, Justin Edwards, Brendan Spillane, Emer Gilmartin, Christine Murad, Cosmin Munteanu, Vincent Wade, and Benjamin R. Cowan. What makes a good conversation? In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–12. ACM, 2019. doi:10.1145/3290605.3300705.

[15] John W. Creswell and Vicki L. Plano Clark. *Designing and Conducting Mixed Methods Research*. SAGE Publications, 3rd edition, 2017.

[16] Anastasia Danilova, Alena Naiakshina, Stefan Horstmann, and Matthew Smith. Do you really code? designing and evaluating screening questions for online surveys with programmers. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 537–548. IEEE, 5 2021. doi:10.1109/ICSE43902.2021.00057.

[17] Felipe Ebert, Alexander Serebrenik, Christoph Treude, Nicole Novielli, and Fernando Castor. On recruiting experienced github contributors for interviews and surveys on prolific. In *1st International Workshop on Recruiting Participants for Empirical Software Engineering (RoPES'22)*, 2022.

[18] Linda Erlenhov, Francisco Gomes de Oliveira Neto, Riccardo Scandariato, and Philipp Leitner. Current and future bots in software development. In *Proceedings of the 1st International Workshop on Bots in Software Engineering*, pages 7—-11. IEEE Press, 2019. doi:10.1109/BotSE.2019.00009.

[19] Linda Erlenhov, Francisco Gomes de Oliveira Neto, and Philipp Leitner. An empirical study of bots in software development: Characteristics and challenges from a practitioner's perspective. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 445–455, 2020.

[20] Juan Carlos Farah, Vandit Sharma, Sandy Ingram, and Denis Gillet. Conveying the perception of humor arising from ambiguous grammatical constructs in human-chatbot interaction. In *Proceedings of the 9th International Conference on Human-Agent Interaction*, pages 257–262, 2021.

[21] Juan Carlos Farah, Basile Spaenlehauer, Xinyang Lu, Sandy Ingram, and Denis Gillet. An exploratory study of reactions to bot comments on github. In *Proceedings of the Fourth International Workshop on Bots in Software Engineering*, BotSE '22, page 18–22, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3528228.3528409.

[22] Joseph L Fleiss. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 76(5):378, 1971. doi:10.1037/h0031619.

[23] Amir Ghorbani, Nathan W. Cassee, Derek Robinson, Adam Alami, Neil Ernst, Alexander Serebrenik, and Andrzej Wasowski. Autonomy is an acquired taste:

Exploring developer preferences for github bots. In *45th IEEE/ACM International Conference on Software Engineering*, pages 1409–1421, May 2023.

[24] Mehdi Golzadeh, Alexandre Decan, Eleni Constantinou, and Tom Mens. Identifying bot activity in github pull request and issue comments. In *2021 IEEE/ACM Third International Workshop on Bots in Software Engineering (BotSE)*, pages 21–25, 2021. doi:10.1109/BotSE52550.2021.00012.

[25] Zhewei Hu and Edward F. Gehringer. Improving feedback on github pull requests: A bots approach. In *2019 IEEE Frontiers in Education Conference (FIE)*, pages 1–9, 2019. doi:10.1109/FIE43999.2019.9028685.

[26] Jen-tse Huang, Wenxuan Wang, Man Ho Lam, Eric John Li, Wenxiang Jiao, and Michael R Lyu. Chatgpt an enfj, bard an istj: Empirical study on personalities of large language models. *arXiv preprint arXiv:2305.19926*, 2023.

[27] Mohit Jain, Pratyush Kumar, Ramachandra Kota, and Shwetak N. Patel. Evaluating and informing the design of chatbots. In *Proceedings of the 2018 Designing Interactive Systems Conference*, pages 895–906. ACM, 6 2018. doi:10.1145/3196709.3196735.

[28] Carlene Lebeuf, Margaret-Anne Storey, and Alexey Zagalsky. Software bots. *IEEE Software*, 35(1):18–23, 2017.

[29] Carlene Lebeuf, Alexey Zagalsky, Matthieu Foucault, and Margaret-Anne Storey. Defining and classifying software bots: A faceted taxonomy. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*. IEEE, may 2019. URL: https://doi.org/10.1109%2Fbotse.2019.00008, doi:10.1109/botse.2019.00008.

[30] Mengjun Li and Ayoung Suh. Machinelike or humanlike? a literature review of anthropomorphism in ai-enabled technology. In *Proceedings of the 54th Hawaii International Conference on System Sciences*, page 4053, 2021.

[31] Q. Vera Liao, Matthew Davis, Werner Geyer, Michael Muller, and N. Sadat Shami. What can you do? In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems*, pages 264–275. ACM, 6 2016. doi:10.1145/2901790.2901842.

[32] Chandra Maddila, Chetan Bansal, and Nachiappan Nagappan. Predicting pull request completion time: A case study on large scale cloud services. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019, page 874–882, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3338906.3340457.

[33] Samim Mirhosseini and Chris Parnin. Can automated pull requests encourage software developers to upgrade out-of-date dependencies? *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, pages 84–94, 2017. doi:10.1109/ASE.2017.8115621.

[34] Clifford Nass, Jonathan Steuer, and Ellen R. Tauber. Computers are social actors. In *Conference companion on Human factors in computing systems - CHI '94*, page 204. ACM Press, 1994. doi:10.1145/259963.260288.

[35] J. A. Nelder and R. W. M. Wedderburn. Generalized linear models. *Journal of the Royal Statistical Society. Series A (General)*, 135(3):370–384, 1972. URL: http://www.jstor.org/stable/2344614.

[36] Zhenhui Peng and Xiaojuan Ma. Exploring how software developers work with mention bot in GitHub. *CCF Transactions on Pervasive Computing and Interaction*, 1(3):190–203, sep 2019. URL: https://doi.org/10.1007%2Fs42486-019-00013-2, doi:10.1007/s42486-019-00013-2.

[37] William D Penny. Comparing dynamic causal models using aic, bic and free energy. *Neuroimage*, 59(1):319–330, 2012. doi:10.1016/j.neuroimage.2011.07.039.

[38] Yasset Perez-Riverol, Laurent Gatto, Rui Wang, Timo Sachsenberg, Julian Uszkoreit, Felipe da Veiga Leprevost, Christian Fufezan, Tobias Ternent, Stephen J. Eglen, Daniel S. Katz, Tom J. Pollard, Alexander Konovalov, Robert M. Flight, Kai Blin, and Juan Antonio Vizcaíno. Ten simple rules for taking advantage of git and github. *PLOS Computational Biology*, 12(7):1–11, 07 2016. doi:10.1371/journal.pcbi.1004947.

[39] Nicolas Pfeuffer, Alexander Benlian, Henner Gimpel, and Oliver Hinz. Anthropomorphic information systems. *Business & Information Systems Engineering*, 61(4):523–533, 2019.

[40] Brittany Reid, Markus Wagner, Marcelo d'Amorim, and Christoph Treude. Software engineering user study recruitment on prolific: An experience report, 2022. URL: https://arxiv.org/abs/2201.05348, doi:10.48550/ARXIV.2201.05348.

[41] Julia M. Rohrer. Thinking clearly about correlations and causation: Graphical causal models for observational data. *Advances in Methods and Practices in Psychological Science*, 1(1):27–42, January 2018. doi:10.1177/2515245917745629.

[42] Daniel Russo. Recruiting software engineers on prolific. 2022. URL: https://arxiv.org/abs/2203.14695, doi:10.48550/ARXIV.2203.14695.

[43] James Schaffer, John O'Donovan, James Michaelis, Adrienne Raglin, and Tobias Höllerer. I can do better than your AI. In *Proceedings of the 24th International Conference on Intelligent User Interfaces*, pages 240–251. ACM, 3 2019. doi:10.1145/3301275.3302308.

[44] John R. Searle. Speech acts: An essay in the philosophy of language. *Language*, 46:217, 1969.

[45] Anja Seiffer, Ulrich Gnewuch, and Alexander Maedche. Understanding employee responses to software robots: A systematic literature. In *International Conference on Information Systems (ICIS)*, volume 2021, 2021.

[46] Ben Sheehan, Hyun Seung Jin, and Udo Gottlieb. Customer service chatbots: Anthropomorphism and adoption. *Journal of Business Research*, 115:14–24, 2020. URL: https://www.sciencedirect.com/science/article/pii/S0148296320302484, doi:https://doi.org/10.1016/j.jbusres.2020.04.030.

[47] Janet Siegmund, Christian Kästner, Jörg Liebig, Sven Apel, and Stefan Hanenberg. Measuring and modeling programming experience. *Empir. Softw. Eng.*, 19(5):1299–1334, 2014. doi:10.1007/s10664-013-9286-4.

[48] Margaret-Anne Storey and Alexey Zagalsky. Disrupting developer productivity one bot at a time. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 928–931, 2016.

[49] Ikram Ul Haq, Iqbal Gondal, Peter Vamplew, and Simon Brown. Categorical features transformation with compact one-hot encoder for fraud detection in distributed environment. In *Data Mining: 16th Australasian Conference, AusDM 2018, Bahrurst, NSW, Australia, November 28–30, 2018, Revised Selected Papers 16*, pages 69–80. Springer, 2019. doi:10.1007/978-981-13-6661-1_6.

[50] Sarah Theres Völkel, Daniel Buschek, Jelena Pranjic, and Heinrich Hussmann. Understanding emoji interpretation through user personality and message context. In *Proceedings of the 21st International Conference on Human-Computer Interaction with Mobile Devices and Services*, pages 1–12. ACM, 10 2019. URL: https://dl.acm.org/doi/10.1145/3338286.3340114, doi:10.1145/3338286.3340114.

[51] Mairieli Wessel, Ahmad Abdellatif, Igor Wiese, Tayana Conte, Emad Shihab, Marco A. Gerosa, and Igor Steinmacher. Bots for pull requests: The good, the bad, and the promising. In *Proceedings of the 44th International Conference on Software Engineering*, pages 274–286. ACM, 5 2022. URL: https://dl.acm.org/doi/10.1145/3510003.3512765, doi:10.1145/3510003.3512765.

[52] Mairieli Wessel, Bruno Mendes De Souza, Igor Steinmacher, Igor S Wiese, Ivanilton Polato, Ana Paula Chaves, and Marco A Gerosa. The power of bots: Understanding bots in oss projects. *Proceedings of the ACM on Human-Computer Interaction*, 2(CSCW):1–19, 2018.

[53] Mairieli Wessel, Alexander Serebrenik, Igor Wiese, Igor Steinmacher, and Marco A. Gerosa. Effects of adopting code review bots on pull requests to oss projects. 2020. URL: https://www.win.tue.nl/~aserebre/ICSME2020Mairieli.pdf.

[54] Mairieli Wessel, Alexander Serebrenik, Igor Wiese, Igor Steinmacher, and Marco A Gerosa. What to expect from code review bots on github? a survey with oss maintainers. In *Proceedings of the 34th Brazilian Symposium on Software Engineering*, pages 457–462, 2020.

[55] Mairieli Wessel and Igor Steinmacher. The inconvenient side of software bots on pull requests. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*. ACM, jun 2020. URL: https://doi.org/10.1145%2F3387940.3391504, doi:10.1145/3387940.3391504.

[56] Mairieli Wessel, Igor Wiese, Igor Steinmacher, and Marco Aurelio Gerosa. Don't disturb me: Challenges of interacting with software bots on open source software projects. *Proceedings of the ACM on Human-Computer Interaction*, 5(CSCW2):1–21, 2021.

[57] Marvin Wyrich, Raoul Ghit, Tobias Haller, and Christian Müller. Bots don't mind waiting, do they? comparing the interaction with automatically and manually created pull requests. In *2021 IEEE/ACM Third International Workshop on Bots in Software Engineering (BotSE)*, pages 6–10. IEEE, 2021.