Adaptive Human-Chatbot Interactions: Contextual Factors, Variability Design and Levels of Automation

by

Glaucia Melo dos Santos

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2023

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Hausi A. Müller Professor, Dept. of Computer Science, University of Victoria
Supervisor(s):	Paulo Alencar Adjunct Professor, Dept. of Computer Science University of Waterloo
	Daniel M. Berry Professor, Dept. of Computer Science, University of Waterloo
	Donald D. Cowan Distinguished Professor Emeritus, Dept. of Computer Science University of Waterloo
Internal Member(s):	Edith Law Associate Professor, Dept. of Computer Science University of Waterloo
	Meiyappan Nagappan Associate Professor, Dept. of Computer Science University of Waterloo
Internal-External Member:	Ladan Tahvildari Professor, Dept. of Electrical and Computer Engineering University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public."

Statement of Contributions

In this thesis, the main content is derived from a fusion of previously published papers. Several segments have been modified and repurposed from the list of publications as follows:

- Chapter 4 draws on a workshop paper. This paper is co-authored by my supervisors Alencar and Cowan and me. I developed the experimental methodology and the design of the literature review. I also carried out the execution of the literature review. Prof. Alencar provided feedback and comments throughout this process. Profs. Alencar and Cowan assisted also with the writing of the paper.
  - G. Melo, P. Alencar, D. Cowan, "Context-Augmented Software Development in Traditional and Big Data Projects: Literature Review and Preliminary Framework", 2019 IEEE International Conference on Big Data (Big Data), 2019, pp. 3449-3457.
- Chapter 5 draws on a workshop paper where Edith Law and I collaborated on the design of the user study. Law, as the PI of the study, undertook the procedures of the user study ethics application and partially funded the study. I designed the study with the support of Law. I executed the study, collecting and analyzing data. Alencar provided feedback and comments throughout this process, also having partially funded the study. Alencar and Cowan assisted also with the writing of the paper. Parts of this chapter were drawn from an ICSE Doctoral Symposium publication and another publication at the BotSE, an ICSE Co-located workshop, as follows:
  - G. Melo, E. Law, P. Alencar, D. Cowan, "Understanding User Understanding: What do Developers Expect from a Cognitive Assistant?", 2020 IEEE International Conference on Big Data (Big Data), Atlanta, GA, USA, 2020, pp. 3165-3172.
  - G. Melo, "Designing Adaptive Developer-Chatbot Interactions: Context Integration, Experimental Studies and Levels of Automation" 2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE '23). IEEE Press, Melbourne, Australia, pp. 235–239.
  - G. Melo, L.F. Lins, P. Alencar, D. Cowan, "Supporting Conversational Agent-Based Software Development" Proceedings of the 5th International Workshop on Bots in Software Engineering (BotSE), Melbourne, Australia, 2023 pp. 9-13.

- Chapter 6 draws on publications co-authored by Nascimento, Alencar, Cowan and me. I designed the study and planned the literature review. Nascimento assisted with the design of feature models and the execution of the literature review. Alencar provided feedback and comments throughout this process. Alencar and Cowan assisted with reviewing the writing of the paper. Parts of Chapter 6 draw also on the ICSE Doctoral Symposium publication mentioned above.
  - G. Melo, N. Nascimento, P. Alencar, D. Cowan, "Understanding Levels of Automation in Human-Machine Collaboration", 2022 IEEE International Conference on Big Data (Big Data), 2022, pp. 3952-3958.
  - G. Melo, N. Nascimento, P. Alencar, D. Cowan, "Identifying Factors that Impact Levels of Automation in Autonomous Systems", in IEEE Access, 2023, vol. 11, pp. 56437-56452.
- Section 6.5 draws from a workshop paper, co-authored Nascimento, Alencar, Cowan and me. Alencar, Nascimento and I worked together on conceptualizing the paper, and Dr. Nascimento contributed the example. I designed the framework proposed in this paper, based on conceptual discussions. Alencar provided feedback and comments throughout this process. Alencar and Cowan assisted with the writing of the paper.
  - G. Melo, N. Nascimento, P. Alencar, D. Cowan, "Variability-Aware Architecture for Human-Chatbot Interactions: Taming Levels of Automation", in 1st International Workshop on Artificial Intelligence for Autonomous Computing Systems. Co-located with 4th IEEE International Conference on Autonomic Computing and Self-Organizing Systems, 2023.
- Appendix E of this thesis draws on an ICSE New Idea and Emerging Results Track paper. This paper is co-authored by Alencar, Cowan me. Alencar and I developed the idea of the framework together. Alencar and Cowan assisted with the writing of the paper.
  - G. Melo, P. Alencar, D. Cowan, "A Cognitive and Machine Learning-Based Software Development Paradigm Supported by Context.", 2021 IEEE/ACM 43rd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER), 2021, pp. 11-15.

Abstract

The landscape of software development is undergoing a significant transformation characterized by various factors. A notable shift is the surging demand for software developers, driven by industries' increasing reliance on software solutions to support their operations. This increased demand is accompanied by an escalation in the complexity of software development projects. In this dynamic environment, modern software systems interact with numerous external systems, interfaces, data sources, and work practices. This complexity requires developers to navigate a complex environment while creating software.

Adding to this landscape is the emergence of AI-based conversational systems, a transformative trend that is shaping software development processes. These systems, powered by artificial intelligence and natural language processing, enable human-like interactions through chatbots and virtual assistants. Software developers are increasingly turning to AI-powered chatbots to support their work. These chatbots play diverse roles, ranging from technical query resolution or load testing to providing project management insights and automating routine tasks. By harnessing the capabilities of these AI-driven tools, developers can potentially enhance productivity, access pertinent information swiftly, and optimize their workflows.

However, amid these developments, many challenges arise due to the intricate web of contextual factors that influence software development processes, especially when chatbots come into play. These contextual factors act as distinct pieces of a puzzle, each altering how software development functions in the presence of chatbots. Unfortunately, the existing research landscape has a limited understanding of these contextual intricacies, resulting in insufficient design methods to adequately support developers using chatbots. Moreover, addressing the customization of automation levels in these interactions remains unexplored.

With the growing complexity of software development, coupled with the emergence of advanced, AI-based conversational systems, the integration of chatbots to support developers in their work has become prominent. There is a pressing need to address the challenges in human-chatbot interactions, particularly in leveraging conversational agents' advances to tailor interactions to developers' specific contexts and desired levels of automation. This research explores the design of context-based adaptive interactions between software developers and chatbots. By understanding and integrating the contextual factors that influence software development with chatbots, we aim to gain novel insights into developers' expectations regarding these interactions and the levels of automation involved and advance the design of human-chatbot adaptive applications.

First, I perform a user study to investigate the requirements of conversational agents in software development. I uncovered a vast list of desired requirements and insights from participants, including that they are interested in working with such tools, in various parts of the development lifecycle such as managing their tasks and version control. One of the insights of this study was that contrary to the authors' beliefs, not all developers were interested in automating all possible tasks. This insight led me to the next part of this thesis, which was the investigation of the factors that influence how much automation is desired in systems.

I then perform a literature review focused on studies about taxonomies of levels of automation. I aimed to uncover from these studies, the factors that influence systems switching from one level of automation to a different level. I identified these factors and composed a list of 61 factors, which we divided into five categories, system, task, human, environment, and quality. I propose feature model designs to represent these factors and their relationships and instantiate this model with use cases.

This research provides a roadmap for the design of adaptive chatbot interactions that align with developers' specific needs and workflows. Empirical studies are conducted to gain insights from developers' experiences and expectations, ultimately driving the design of context-aware chatbot interactions. Additionally, by examining the influence of varying levels of automation on these interactions, I sought to identify factors that shape the variability of automation levels, bridging the gap between human-system interactions and autonomous systems.

Acknowledgements

Many things had to work precisely right so I could come from being born in a poor city in a 3rd world country to becoming a doctor at a world-renowned university in Canada. Therefore, I have to start by thanking the unknown forces of this universe that made all these precise movements on my behalf.

I owe immense thanks to my supervisor, Prof. Paulo Alencar, whose support and guidance have been instrumental in shaping me as a researcher and an individual. Prof. Alencar's kindness, respect, and unwavering motivation have been a source of strength during both triumphs and challenges. Working with such a brilliant and amiable mentor has been a privilege, and I can only hope for more opportunities to collaborate in the future. I am deeply honoured and grateful to have worked with Donald Cowan and Daniel Berry as my co-supervisors. Don, your exceptional guidance has been invaluable, and I cannot express enough gratitude for having collaborated with you. Dan, I am incredibly thankful for your constant support and availability, despite your prominence in the field. The privilege of being supervised by all of you is truly amazing.

I extend my gratitude to my committee members for their exceptional contributions. Their support and guidance in shaping the scope of my work, and their insightful feedback during both the comprehensive exam and the defence, have been invaluable. The detailed and constructive feedback provided not only enriched the quality of my research but also fostered engaging and enjoyable discussions during the defence. I am truly grateful for your dedication, and I look back on our collaborative efforts with immense appreciation.

There are so many more people to give thanks to, as they were key to this accomplishment, this is another challenging section to write. I thank you all so much. Some of these people I'm happy to mention specifically.

Toacy Oliveira for having paved the opportunity and supporting my research since the very beginning, and for all the support I received when first arriving in Canada, thank you so, so much. I also thank Walter Magioli, who made this all a possibility in the first place. I am grateful to have had you as a boss and mentor.

To Damien Masson and Nils Lukas, my good friends since literally day one of our Ph.D., thank you for your support, fruitful conversations, and friendship. Your presence has been essential throughout the last five years, making me a better researcher and person. I am extremely proud of the path you built in these last 5 years we've known each other and I am so glad to have had the opportunity to meet you. I am also thankful for the collaboration with my dear friends and co-authors Ulisses Telemaco,

Luis Fernando Lins, and Nathalia Nascimento, whose brilliance and friendship have made this journey way more enjoyable and fruitful.

An army of strong and supportive women has made this journey not only easier but possible. I am immensely grateful for their assistance in various aspects of my life, and therefore also considering this very challenging last 5 years. My mom, Fatima, has been my constant pillar of strength, always encouraging me to persevere and explore the unknown with the perfect mix of grace, positivity and toughness. My sister Gisele, with her unparalleled wisdom and determination, has been a constant inspiration. The way you live your life and make your choices has inspired me so much. I am so proud of you and I learn so much from you. Being raised by these women could not have resulted in anything else for me other than big accomplishments such as this one. I deeply honour your contributions to my life. To my other sisters from other parents: Thaisa Melo, Paola Fernandes, Viviane Caldas, and Karla Cogo, your selfless support and willingness to help me have been immense, in ways that all others have lacked. Thank you for being who you are, and having truly supported me in crucial moments of this journey.

Some people not only brought in their own help to support me but also the most amazing families I could ever wish to meet in Canada. For the Oliveira family, who supported me when I had just moved to Canada in so many great ways, for Sarah and Sangram for making me feel so loved and welcome, for all the guidance and refuge. The McGill family, you all have welcomed me since day one as part of your family, I appreciate you all so much and will be forever grateful for that. During the times when my own idea of family was shaken by both the geographical distance and the passing of my father, you were the pillar that helped me remember I still had close support, love and strings attached to this world. I am immensely grateful to you all.

To my partner, Seth, thank you for supporting me throughout this journey. Your love and encouragement have been key. When I think back, there were many key moments where your laid-back personality, great sense of humour and kindness, to me and others, made a truly positive difference, and I will never forget how consistently you were by my side throughout these very challenging past few years. Thank you so much for still being by my side and for being with me through it all.

I would also like to express my deepest gratitude to my mental health support team, Emanuele Blasioli and Clarissa Guelves, for their professional help and guidance in maintaining my mental well-being during this very demanding period. Thank you so much.

To the friends I made in Waterloo, my local extended family, you hold a special place

in my heart, and I also want to give special thanks to you. To my dear friends Renato, John and the powerhouse women I am honoured to call friends: Cristina Herrera & Tavares, Elena, Ella, Karina, Luana, and Maria, thank you for your relentless support and presence in my life, especially in the very hard last three years. I appreciate and love you so much.

Lastly, I want to give thanks to my dad, who showed me that the world is vast, fascinating, and worth exploring. All this was a prospect in the first place because of the way you have raised me surrounded by encyclopedias, magazines about science, trips to museums, and your unremitting love for and excitement about seeking and consuming knowledge. I have you and only you to thank for that. Wherever you are, I love you.

Dedication

This thesis is dedicated to my dad.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AI Artificial Intelligence 1, 2, 6, 8, 12, 16, 18, 20–23, 25–27, 30, 99, 112, 113, 116, 125, 131, 132, 135, 136

CA Conversational Agent 57, 58, 71–73, 78, 90, 95

FODA Feature-Oriented Domain Analysis 116, 117

GQM Goal-Question-Metric 39

HCI Human-Computer Interaction 30

LOA Level of Automation 2, 3, 6–8, 24, 32, 33, 98–104, 107, 108, 110–134

LR Literature Review 35, 37–41, 44, 46, 49, 51–54, 102, 107, 132

NLP Natural Language Processing 19, 20, 23, 80

NLU Natural Language Understanding 19, 21, 22, 71, 72

PICO Population Intervention Comparison Outcome 41, 42

SD Software Development 18, 167, 168, 174

SE Software Engineering 30, 57, 58, 78

SPL Software Product Lines 33, 34

UAV Uncrewed Aerial Vehicles 31

WOZ Wizard-of-Oz 62

Chapter 1

Introduction

The landscape of software development is experiencing a significant transformation, driven by a convergence of multiple factors. One of the most prominent shifts is the growing demand for software developers [19]. As businesses and industries increasingly rely on software solutions to streamline operations [169], enhance customer experiences, and stay competitive, the need for skilled software professionals has skyrocketed. This surge in demand is accompanied by a corresponding rise in the complexity of software development projects. Modern software systems are intricate ecosystems that interact with numerous external systems, interfaces, data sources, and work practices [110, 158]. Developers must navigate this intricate web of dependencies, ensuring that their software functions seamlessly within this multifaceted environment [85].

Moreover, software development involves a multitude of tasks [110]. From coding and testing to debugging and deployment, developers must juggle various responsibilities throughout the software development lifecycle. Each task requires careful attention and expertise to ensure the final product meets quality and functionality standards. Adding to this complexity is the dynamic nature of software development [116]. Context changes frequently, often in response to shifting project requirements, technological advancements, or feedback from stakeholders. Developers must be agile and adaptive, ready to pivot their strategies and approaches to accommodate these alterations.

Amid this evolving landscape, there has been a notable emergence of Artificial Intelligence (AI)-based conversational systems. These sophisticated systems leverage artificial intelligence and natural language processing to facilitate human-like interactions through chatbots and virtual assistants. Their potential applications span a wide range of industries and use cases. Software developers are increasingly turning to these AI-

powered chatbots to support their work [1, 129]. These chatbots can assist developers in various ways, from answering technical queries (ChatGPT) and performing load tests [126] to offering project management insights and automating routine tasks [20]. By leveraging these AI-driven tools, developers can enhance their productivity, access relevant information more efficiently, and streamline their workflows.

1.1 Challenges

The complexity within software development becomes amplified due to diverse contextual factors influencing software development processes when chatbots are involved. These contextual factors act like various pieces of a puzzle, and these pieces can alter how software development functions when chatbots are in play. Moreover, there is limited research to understand the design methods to support developers using chatbots to perform their tasks and how to cope with customized LOAs in their human-chatbot interactions. There is limited published data on the methods to integrate context into developer-chatbot interactions [153, 123], and the notion of customized levels of automation in these interactions is yet to be comprehensively addressed [52, 131, 160]. It is important that chatbot users are supported, including with customization, to increase user satisfaction and avoid frustration[29], to address the increasing interest of software developers in creating and integrating chatbots into websites [2] and to uncover prominent features for chatbot tools, such as providing satisfactory responses is a key to success in the adoption of chatbots [132].

Assisting software developers in their context-based interaction with chatbots then becomes essential. In particular, understanding the context surrounding software development using chatbots and integrating this context into design models can lead to a novel understanding of what software developers expect concerning these human-chatbot interactions and their LOA. Any taxonomy of LOAs is a framework that can be used to describe the degrees to which a system is automated, each typically ranging from fully manual to fully automated, and thus, the corresponding degrees to which the system requires complementary manual action.

In response to these challenges, this research aims to investigate the design of context-based adaptive interactions between software developers and chatbots and foster solutions and knowledge to support software developers at work. By adopting a multi-faceted approach that draws insights from context-aware systems, conversational agents, variability approaches, and autonomous systems, this thesis aspires to understand the design of chatbot interactions between software developers and chatbots.

It is crucial to examine the impact of highly dynamic context in software development and consider the influence of developers' perspectives on designing context-based chatbot tools. Additionally, it is vital to consider the effect of varying LOAs on the design of these tools. Designing context-based chatbot tools that adequately support developers remains an area that requires further exploration. This thesis aims to bridge the gap in designing effective adaptive developer-chatbot interactions that seamlessly integrate context and consider varying LOAs. By enhancing the design of context-based chatbots, we seek to provide better support for software development and foster knowledge to advance these tools.

1.2 Research Questions

To achieve these goals, I focus on three primary research questions:

RQ1: What types of contexts have been identified by researchers in software development projects? Understanding the contextual factors that influence software development is the first crucial step for designing context-aware chatbots. I explore the various types of context identified by researchers in software development projects, enabling us to create adaptive chatbot interactions that align with developers' specific needs and workflows.

RQ2: How well can a chatbot support a software developer when executing a software development task? To address the limited research on design methods for context-based chatbot support and the absence of information from users (developers) to inform chatbot features, I conduct empirical user studies. These studies extract valuable insights and requirements from developers' experiences and expectations, guiding us in the design of context-aware chatbot interactions.

RQ3: Which factors impact the variability of levels of automation in autonomous systems? To understand better how to handle customized LOAs in human-chatbot interactions, I investigate factors that influence LOAs during these interactions. Analyzing the integration of context into software development and studying the adaptive interactions between developers and chatbots will shed light on effectively systemizing context in software engineering, allowing the proposal of model designs and fulfilling developers' expectations regarding context-based chatbot interactions and their LOA.

By addressing these research questions, we anticipate that our findings will contribute to the advancement of adaptive developer-chatbot interactions, fostering a new

era of context-aware and personalized support for software development. Specific contributions are detailed next.

1.3 Research Method Design

In this thesis, we use multiple empirical methods to answer the three research questions. More specifically, I follow a mixed-methods approach with a sequential explanatory strategy, combining literature reviews, and a chatbot prototype design with semi-structured interviews conducted with software developers, qualitative survey analysis, and other experimental studies. The research design comprises three phases and several complementary studies, as presented in Figure 1.1.

Preliminary Study: Identifying Contexts in Software Engineering. This phase consists of studies conducted during the definition of this thesis' scope and helped define my research motivation. This phase is described in Chapter 4 of this thesis. In this preliminary study (Study 1), I conducted a Literature Review to uncover the identified contexts in software engineering, and how research has suggested the management (capture, use) of these contexts. Directed by RQ1, I extracted information from papers in the literature and uncovered various definitions for context. The findings reveal diverse types of context classifications, encompassing project tasks, static software structure, dynamic system execution, historical artifact changes, developer activity, and team and organization activity. The proposed contexts in the literature serve distinct objectives, such as supplying developers with code artifacts, offering dynamic execution insights, accessing historical data, understanding developer work processes, and considering broader activities across value streams. Some papers touch upon the usage of context in different software development phases, like task allocation planning, bug-focused phases, and coding stages. Certain studies conducted evaluations of their context-aware tools, yielding varying levels of success. Some limitations and gaps were identified, such as the scarcity of tools for historical artifacts, potential duplicate bug reports from code clones, and relatively unexplored team and organization activity contexts. The benefits of context encompass heightened developer productivity, optimized task allocation, and enhanced interaction styles. However, the disadvantages were not explicitly outlined. Context instances are extracted to offer recommendations for task resolution, bug database queries, and artifact editing based on context history. Interestingly, none of the reviewed papers mention the use of model specification techniques or abstractions in their proposed contexts.

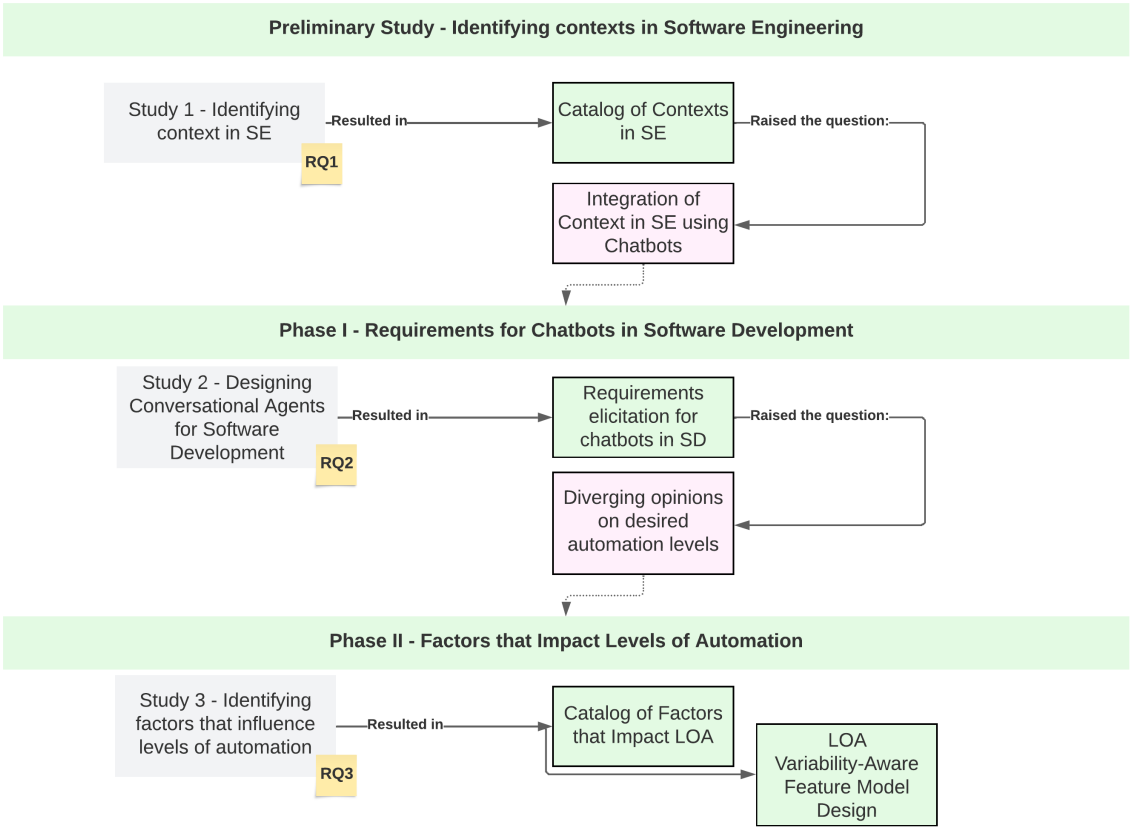

Figure 1.1: Overview of the Research Design.

After this first study, I wondered how to integrate context into development, as it was clear from the literature review the benefits of integrating contextual information into software development. With the emergence of AI and the use of such tools, and other benefits, I investigated the integration of context into development through conversational agents. I focused on uncovering the requirements and design of chatbots for software development, which led us to Phase I.

Phase I: Requirements for Chatbots in Software Development. To further understand the interaction characteristics of software developers with chatbots, I conducted a pilot study, followed by a mixed-methods study, in which participants interacted with a chatbot prototype and asked this prototype some questions. This phase is described in Chapter 5 of this thesis. I aimed to capture these questions and interview participants, to grasp requirements and design opportunities for chatbots in software development. I found that participants showed an interest in using chatbots to aid them in tasks related to task and repository management, reflecting a real demand for such support. However, preferences between guidance and automation showed variations among users, challenging the assumption that automation would be universally favored. This discrepancy suggests the need for in-depth investigations into the factors influencing users' automation preferences. Moreover, the study highlighted the desire for chatbots to possess a deeper contextual understanding, with participants expressing the need for these virtual assistants to be more personalized and capable of providing context-driven recommendations. Furthermore, participants with less experience indicated the chatbot's perceived helpfulness was lower, underlining the importance of tailoring chatbot responses to the user's skill level.

Phase II: Factors that Impact Levels of Automation. This phase comprises two studies to identify the factors that influence LOAs in systems, aiming at answering RQ3. This phase is described in Chapter 6 of this thesis. After phase I, where study participants pointed out distinct opinions on the use of automation, I decided to investigate what factors influence how much automation humans expect from systems. In this comprehensive study, the focus was on identifying and categorizing the factors that influence the selection of different LOAs in autonomous systems. A systematic literature review was conducted to uncover these factors, and they were then categorized into five main groups: Quality, Task, Agent/System, Human, and Environmental factors. Furthermore, the research delved into how these factors could be effectively captured and embedded into autonomous systems to enhance their adaptability and functionality, through modelling these factors as feature models (features and constraints).

In summary, this research provides contributions to the design of adaptive systems. Systematically categorizing and demonstrating the impact of influential factors, equips

engineers with the tools to create more versatile and effective autonomous systems, paving the way for enhanced performance and adaptability in various operational scenarios.

1.4 Problem Statement

In this thesis, I tackle the problem of integrating contextual software development information into development through context-based chatbots. Specifically by understanding what this context is, what software developers expect concerning context-based chatbot interactions, and the LOA desired. This way, I increase the body of knowledge on context-based chatbot approaches to support software development and foster the understanding of the design of context-based adaptive chatbots so that these tools, once built, can provide optimal support to software developers.

1.5 Research Contributions

A summary of the contributions of this thesis is described next. The contributions of this thesis can be categorized into three parts: *methodological*, which includes systematic literature reviews on context for software engineering and LOAs; *theoretical*, which includes formal techniques that advance methodological contributions; and *empirical*, which includes the understanding of human-chatbot interactions, preferences of LOAs, along with associated datasets.

- Preliminary literature review revealing contextual factors in software engineering
- Context model design
- Within-subjects user study with publicly available data and experimental evaluation
- Systematic literature review revealing factors that influence LOAs, categorization of these factors, and the results of the analysis of the relationships between these identified factors and LOAs
- Context-based chatbot prototype design

- Adaptive design model that demonstrates the variability of the factors and LOA using feature models and constraints
- Use cases demonstrating the applicability of the LOA variability feature design models in three different domains

The contributions of this thesis align with the evolving landscape of General AI (Gen-AI) and major industry players such as Microsoft Bot, Amazon Lex, and IBM Watson. As the industry witnesses a surge in AI-based conversational systems, this thesis contributes by addressing the pressing challenges in human-chatbot interactions. It aims to integrate advancements in conversational agents with the complexities of software development, fostering adaptive interactions tailored to developers' specific contexts and desired LOAs. Through an exploration of contextual factors influencing software development, this research aspires to offer valuable insights into developers' expectations and, in turn, propel the design of adaptive applications for human-chatbot interactions in the dynamic landscape of Gen-AI.

Based on the above contributions, this thesis makes the following thesis statement.

With the growing complexity of software development, coupled with the emergence of advanced AI-based conversational systems, the integration of chatbots to support developers in their work has become prominent. There is a pressing need to address the challenges in human-chatbot interactions, particularly in leveraging conversational agents' advancements to tailor interactions to developers' specific contexts and desired LOAs. This research explores the design of context-based adaptive interactions between software developers and chatbots. By understanding and integrating the contextual factors that influence software development with chatbots, we aspire to gain novel insights into developers' expectations regarding these interactions and the LOAs involved and advance the design of human chatbot adaptive applications.

1.5.1 Published Research

Throughout the Ph.D. program, I have actively contributed to the scholarly community by publishing multiple papers directly aligned with the thesis topic. Our principal findings have been disseminated through many venues, including IEEE Big Data in 2019 and 2020, BotSE (ICSE) 2023, Doctoral Symposium (ICSE) 2023, IEEE Access Journal, and NIER (ICSE) 2023. In the following list, you will find a consolidated list of references encompassing both the research originating from this Ph.D. program and publications spanning the entire duration of the program.

- Related to Thesis:
  - G. Melo, N. Nascimento, P. Alencar, D. Cowan, "Variability-Aware Architecture for Human-Chatbot Interactions: Taming Levels of Automation", to appear in 4th IEEE International Conference on Autonomic Computing and Self-Organizing Systems (AI4AS Workshop), 2023.
  - G. Melo, N. Nascimento, P. Alencar, D. Cowan, "Identifying Factors that Impact Levels of Automation in Autonomous Systems", IEEE Access, vol. 11, 2023, pp. 56437-56452.
  - G. Melo, "Designing Adaptive Developer-Chatbot Interactions: Context Integration, Experimental Studies, and Levels of Automation", 2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE), 2023, pp. 235-239.
  - G. Melo, L. F. Lins, P. Alencar, D. Cowan, "Supporting Contextual Conversational Agent-Based Software Development", 2023 IEEE/ACM 5th International Workshop on Bots in Software Engineering (BotSE, ICSE), 2023, pp. 9-13.
  - G. Melo, N. Nascimento, P. Alencar, D. Cowan, "Understanding Levels of Automation in Human-Machine Collaboration", 2022 IEEE International Conference on Big Data (Big Data), Osaka, Japan, 2022, pp. 3952-3958.
  - G. Melo, P. Alencar, D. Cowan, "A Cognitive and Machine Learning-Based Software Development Paradigm Supported by Context", 2021 IEEE/ ACM 43rd International Conference on Software Engineering: New Ideas and Emerging Results (NIER, ICSE), 2021, pp. 11-15.
  - G. Melo, E. Law, P. Alencar, D. Cowan, "Understanding User Understanding: What do Developers Expect from a Cognitive Assistant?", 2020 IEEE International Conference on Big Data (Big Data), 2020, pp. 3165-3172.
  - G. Melo, P. Alencar, D. Cowan, "Context-Augmented Software Development in Traditional and Big Data Projects: Literature Review and Preliminary Framework", 2019 IEEE International Conference on Big Data (Big Data), 2019, pp. 3449-3457.
- Others:
  - U. Telemaco, T. Oliveira, R. Pillat, P. Alencar, D. Cowan, **G. Melo**, "Scaffolding Process-Aware Information Systems with the AKIP Platform", in Web

Information Systems and Technologies. WEBIST 2022. Lecture Notes in Business Information Processing, Springer, vol 494.

- D. Paulino, A. Correia, D. Guimaraes, R. Chaves, **G. Melo**, D. Schneider, J. Barroso and H. Paredes, "Stigmergy in Crowdsourcing and Task Fingerprinting: Study on Behavioral Traces of Weather Experts in Interaction Logs," 2023 26th International Conference on Computer Supported Cooperative Work in Design (CSCWD), 2023, pp. 1293-1299.
- U. Telemaco, T. Oliveira, R. Pillat, P. Alencar, D. Cowan, **G. Melo**, "AKIP Process Automation Platform: A Framework for the Development of Process-aware Web Applications" in Proceedings of the 18th International Conference on Web Information Systems & Technologies, WEBIST, 2022, pp. 64–74.
- L.F. Lins, **G. Melo**, T. Oliveira, P. Alencar, D. Cowan, "PACAs: Process-Aware Conversational Agents" in BPM 2021: International Workshops, Lecture Notes in Business Information Processing, 436 LNBIP, 2021, Revised Selected Papers 2022, pp. 312-318.
- R. Vital, **G. Melo**, T. Oliveira, F. Abreu, "Towards understanding quality-related characteristics in Knowledge-Intensive Processes - A Systematic Literature Review" in Quality of Information and Communications Technology: 14th International Conference, QUATIC 2021, pp. 197-207.
- **G. Melo**, T. Oliveira, P. Alencar, D. Cowan, "Knowledge reuse in software projects: Retrieving software development Q&A posts based on project task similarity". PLOS ONE 15(12): e0243852.
- R. Cohen, A. Parmentier, **G. Melo**, G.Sahu, et. al., 2020 "Digital Literacy for Secondary School Students: Using Computer Technology to Educate about Credibility of Content Online". In Creative Education Journal, Vol. 11, 674-692.
- **G. Melo**, T. Oliveira, P. Alencar, D. Cowan, "Retrieving Curated Stack Overflow Posts from Project Task Similarities" in 31st International Conference on Software Engineering and Knowledge Engineering, SEKE 2019, pp. 415–418.
- R. Vital, **G. Melo**, T. Oliveira, P. Alencar, D. Cowan, "AgileCritPath: Identifying Critical Tasks in Agile Environments" in 31st International Conference on Software Engineering and Knowledge Engineering, SEKE 2019, pp. 20–25.

1.6 Thesis Overview

This section provides a summary of the chapters in the thesis.

Chapter 2 provides an overview of the literature in the context of software engineering, chatbots to support software development, autonomous systems, and variability modelling. These are the pillars that support the theory in which I ground this thesis.

Chapter 3 adds a related work analysis, where I explore works in the overarching areas of context-aware software development systems, conversational agents to support software developers, levels of automation and variability in software systems.

Chapter 4 delves into the critical aspect of contexts in software engineering. Building upon a previously published workshop paper [102], this chapter contains the preliminary work that investigates the various contextual factors that influence software development projects. By understanding the different types of contexts that developers encounter, I can design chatbot interactions that adapt to specific project contexts, facilitating developers' decision-making processes.

Chapter 5 delves into the perspectives and requirements of software developers concerning conversational agents. Drawing from published conference papers [103, 101, 104], this chapter presents the design and results of the empirical study that extracted valuable insights from developers' experiences and expectations regarding chatbot interactions. By gaining a deeper understanding of developers' needs, I can refine the design of chatbot interactions to support better their tasks and workflows.

Chapter 6 explores the intricate interplay between levels of automation and human-chatbot interactions in autonomous systems. Drawing from a published journal paper [106] and two workshop papers [105, 107], this chapter investigates the factors that influence the variability of automation levels during interactions between developers and chatbots. By understanding how developers perceive and adapt to varying degrees of automation, I can tailor chatbot interactions to complement human decision-making and creativity effectively.

Chapter 7 presents the culmination of our research journey. Here, I synthesize the key findings from each chapter and provide a cohesive conclusion. Additionally, I highlight the implications of our research and discuss future research directions to further advance the field of adaptive developer-chatbot interactions. By identifying areas for future exploration, I aim to inspire continued progress and innovation in this dynamic domain.

Chapter 2

Background

This thesis is founded upon a synthesis of key concepts from the fields of software engineering and development. In this chapter, I provide a comprehensive overview of the fundamental background encompassing the concepts central to this thesis. Specifically, I delve into (1) the significance of context in software engineering, (2) the role of chatbots in supporting software development, (3) the dynamic interplay between automation and autonomy in software systems, and (4) software product lines. These foundational pillars serve as the bedrock upon which the subsequent chapters of this thesis are constructed, facilitating a deeper exploration and analysis of the research topic.

Here is how these pillars are linked to my thesis. I explain (1) why context matters in software engineering, (2) how chatbots are useful in helping with software work, (3) the relationship between automation and autonomy in software, and (4) software product lines. These basics create a strong base for what I explore in the following chapters. I show that understanding the context in software engineering (1) helps us see how software development and context are connected. Saying that chatbots are important for supporting software development (2) fits with the wider trend of using advanced AI chat systems. Looking at how automation and autonomy interact in software (3) gives a detailed view of the challenges in human-chatbot interactions and making experiences fit specific contexts. Talking about software product lines (4) brings in a way of thinking and designing for variability and changing needs in software development. These pillars are the main ideas that support my research in investigating applications for how humans and chatbots work together, dealing with the challenges as software development gets more complicated and we see more AI-based chat systems.

2.1 Context in Software Engineering

Software development is a complex and knowledge-intensive effort that involves various contexts and technologies [110]. In the field of software engineering, context refers to the information and data that surround and influence a specific software system or application [116]. These contextual factors can include user input, system configuration, and other data that the software relies on to function correctly. Additionally, context encompasses the current state of the system, the current user, and any other relevant information that affects the software's operation. Some examples of context in software engineering and their description are presented in Table 2.1.

Table 2.1: Examples of Context Factors in Software Development.

Context Factor	**Description**
User Input	Data or commands provided by the user
System Configuration	Settings and configurations of the software system
Current State	The state of the software system at a specific point in time
User Profile	Information about the user, such as their role or preferences
Development Environment	Tools, libraries, and frameworks used in development
Task Context	Information specific to the current task being performed
Collaboration Context	Contextual factors related to team collaboration
Tacit Knowledge	Unspoken, experiential, and context-specific expertise

Understanding and managing context is crucial in software engineering as it significantly influences the behavior and functionality of a software system [8, 118, 116, 172]. The context affects how the software processes input, generates output and interacts with other systems and users, including developers. By considering and leveraging context effectively, software developers can build reliable, efficient, and user-friendly software solutions.

Software development is a human-centred task, influenced by diverse practices shaped by individual expertise, personal interests, gender, stress management, and other user-related contexts [170, 137, 88, 69, 150]. Each developer's technique when executing tasks is heavily influenced by these various traits and the specific work environment in which they operate.

Extensive research has been conducted to address the context of software development [77, 63, 67, 110]. The significance of treating context as a first-class construct in software development has been highlighted [116], as it can lead to significant changes in how developers approach their work.

Despite the substantial amount of research, the context in software development is often not explicitly captured or presented as a comprehensive framework that broadly supports the situations faced by software developers [116]. Consequently, the ability to reuse this rich context across software development projects is severely limited. Furthermore, this context can be implicit and reside solely in the developers' minds (tacit knowledge) or be dispersed throughout extensive documentation [43]. As developers work on software projects, they must maintain mental models of various tasks and information [85]. However, context can easily be lost or forgotten in the absence of explicit mechanisms to capture and preserve it. Without historical context, developers may not be adequately supported by their current environment and struggle to make informed decisions [38].

To address these challenges, one potential approach is to leverage chatbots in software development, enabling developers to communicate with a system designed to support them during the development process. By engaging in conversations, developers can interact with a chatbot that is aware of the context. This context-aware chatbot should be capable of understanding and capturing the various contexts relevant to software development, such as the developer's repository, projects, and commands. By incorporating context into the chatbot's functionality, developers can receive tailored assistance and guidance throughout their software development endeavors. A conceptual representation of this chatbot is presented in Figure 2.1.

The proposal to use chatbots as a means of supporting software development aligns with the need to address context explicitly and facilitate its effective utilization. By capturing and utilizing context within a chatbot framework, developers can benefit from a system that supports them in leveraging contextual information to enhance their productivity and decision-making. This motivation drives the continuation of research in this area, as described in the subsequent sections.

2.2 Chatbots to Support Software Development

Software development is a complex and demanding field that requires developers to spend considerable time on routine activities, which can detract from addressing more challenging problems. To alleviate this issue, researchers have explored the integration of context-aware intelligent assistants, such as chatbots, to support software development teams. By leveraging chatbots, developers can offload repetitive tasks, streamline communication, and enhance collaboration within their teams. Chatbots in software development serve a wide range of activities, from automating tedious chores to bridging

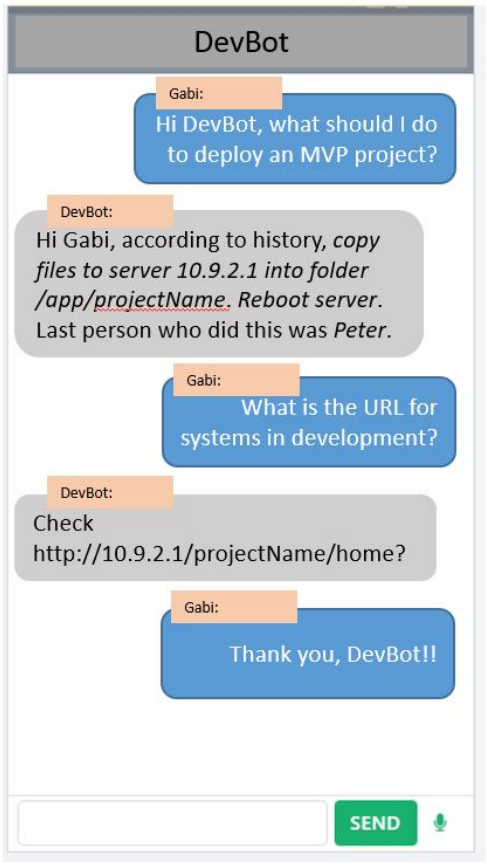

Figure 2.1: Conceptual Representation of a Context-Aware Chatbot in Software Development.

knowledge and communication gaps within software teams [162]. For instance, these chatbots can assist in automating code formatting, generating documentation, managing software builds, and conducting code reviews, among other tasks. This automation not only saves time but also reduces the likelihood of human error, thereby improving code quality.

Additionally, chatbots have been deployed in specific contexts to address different challenges in software development. One area of focus has been on detecting code conflicts when multiple developers work on the same codebase simultaneously. By analyzing code changes in real-time, these chatbots can identify potential conflicts and notify the relevant developers to resolve them collaboratively [128]. In open-source projects,

chatbots have been used to recommend the right person to contact based on their expertise and prior contributions. These chatbots leverage data from version control systems and issue trackers to assist developers in finding the most suitable person to seek guidance or collaborate with on specific tasks [28].

The integration of chatbots in software development has seen substantial adoption in open-source projects, with approximately 26% of projects on GitHub utilizing software bots for various activities [179]. One late example is Microsoft Copilot[1] is an AI-powered code completion tool developed by OpenAI in collaboration with GitHub. It is integrated into the Visual Studio Code (VS Code) editor and assists developers by providing intelligent code suggestions and autocompletion as they write code. Microsoft Copilot is built on OpenAI's Codex, which is a powerful language model capable of understanding and generating code snippets in multiple programming languages. Copilot aims to boost developers' productivity by offering contextual code suggestions, helping with repetitive tasks, and speeding up the coding process. However, despite the increasing prevalence of chatbots, there remains a gap in research that explores developers' perspectives on context-based chatbots within real-world scenarios [11]. Understanding software developers' perceptions of using chatbots in their workflow is crucial for improving the design and implementation of such tools. This entails empirical exploration of the benefits and challenges posed by context-aware chatbots, and the identification of design opportunities to enhance their effectiveness in supporting software development teams [11, 1].

To address this research gap, several studies have sought to evaluate the impact of chatbots in software development settings. One such study by Assavakamhaenghan and colleagues [11] conducted a qualitative analysis by interviewing software developers who had experience using context-based chatbots. The study found that developers appreciated the automation of routine tasks by chatbots, as it allowed them to concentrate on more critical aspects of their work. However, some developers expressed concerns about chatbots misunderstanding context or providing inaccurate suggestions, which could lead to potential issues in the codebase. In another study, Abdellatif and Smith [1] surveyed to gauge the perceptions of software developers regarding the use of chatbots in their development workflow. The survey results revealed that developers found chatbots to be valuable for managing repetitive tasks and enhancing team collaboration. Developers also highlighted the need for better integration of chatbots with existing development tools and emphasized the importance of customizable bot behavior to align with different team preferences and project requirements.

[1] https://copilot.microsoft.com/

Matthies et al. [99] discuss the potential of chatbots in the field of software development, particularly in supporting analyses and measurements of teams' project data. They emphasize that recent advancements in natural language processing and data analysis enable software bots to act as virtual team members, offering additional insights and automation to support teamwork. The focus is on using chatbots to analyze software project artifacts, such as commits in version control systems, to gain valuable information about team collaboration and work patterns. This data analysis is particularly relevant for agile retrospective meetings, where teams discuss process improvements. By employing chatbots as a user interface, developers can conveniently interact with the outcomes of retrospectives and use associated measurements to track improvement actions over development iterations.

Okanović et al. [126] discuss the limited use and questionable effectiveness of load testing in assessing load-related quality properties of software systems. To address this, the authors propose using chatbots to provide suitable support for load testing. They introduce "PerformoBot," a chatbot designed to guide developers through the process of configuring and running load tests. Using natural language conversation, PerformoBot helps developers specify load test parameters, which it then automatically executes using a state-of-the-art load testing tool. After the execution, PerformoBot generates a report with relevant information. The authors conducted a user study involving 47 participants to assess PerformoBot's acceptance and effectiveness. They found that participants, especially those with less expertise in performance engineering, viewed PerformoBot positively.

More recently, Farrah et al. [53] discuss the growing use of chatbots for educational purposes, particularly in social coding platforms, where automated agents support software developers with tasks like code reviews. Challenges like steep learning curves and privacy concerns hinder their adoption in software engineering education. To address this gap, the authors developed an online learning application that simulates code review features using chatbot identities for instructors to interact with students. They conducted a controlled in-class experiment to examine the impact of explaining content via chatbot identities on students' perceived usability of the lesson, engagement with the code review process, and learning gains. While the quantitative analysis did not yield significant differences, qualitative results suggest that students expect explicit feedback and could benefit from automated replies provided by interactive chatbots. The authors propose further exploration of this research area in future work. For reference, "bot" is a general term for automated software, "chatbot" specifically focuses on conversation (voice or text), "automated agent" is a broader term for any automated system, and "context-aware chatbot" emphasizes the ability to understand and use con-

text in conversations. The usage and distinctions between these terms can vary, and the field of conversational agents continues to evolve with advancements in artificial intelligence and natural language processing.

The manual process of eliciting and classifying requirements can be time-consuming and error-prone, especially when dealing with many requirements. To address this, Surana et al. [141] propose an innovative approach to automate Requirements Elicitation and Classification using an intelligent conversational chatbot. Leveraging Machine Learning and Artificial Intelligence, the chatbot engages in Natural Language conversations with stakeholders to elicit formal system requirements. Subsequently, it classifies the elicited requirements into Functional and Non-functional system requirements, streamlining the process and improving efficiency.

Based on these studies findings and other related research, designers and developers can make informed decisions while developing and implementing context-aware chatbots for software development. By addressing developers' concerns and providing customizable options, these chatbots can become valuable assets in modern software development teams, contributing to increased productivity, code quality, and team collaboration.

Chatbots have shown great promise in supporting software development teams by automating tasks, improving collaboration, and enhancing code quality. Through empirical evaluation and understanding of developers' perspectives, these chatbots can be designed and optimized to cater to the specific needs of development projects. The successful implementation of chatbots in various software development scenarios underscores the potential for these AI-powered tools to play a crucial role in the future of software engineering. Continued research and innovation in this field will further unlock the benefits of chatbots for developers and contribute to more efficient and successful software development processes.

With the research framework in place, providing valuable insights into the landscape of chatbots in software development, it is now time to shift our focus to the industry and practice. Having laid the foundation for understanding research efforts for chatbots in Software Development (SD), we can now delve into how chatbots are being utilized in real-world scenarios and the practical implications they bring to the software development process. By exploring their adoption, impact, and success stories in various industries, we can gain a deeper understanding of the tangible benefits that chatbots offer to developers and organizations alike.

Chatbots for software development have revolutionized the way developers interact with tools and streamline their workflows. The current landscape of chatbots developed

specifically to support the process of software development includes a range of voice and text-based applications. These chatbots leverage Natural Language Understanding (NLU) and Natural Language Processing (NLP) capabilities to enable developers to interact with them through voice or text, enhancing productivity and collaboration. This section provides an overview of the current landscape of chatbots developed to support the process of software development in voice and text-based interactions.

Voice-Based Chatbots: Voice-based chatbots in software development aim to provide developers with hands-free and voice-activated interactions to access information, perform tasks, and receive real-time assistance. These chatbots utilize speech recognition and synthesis technologies to understand and respond to developers' voice commands and queries. Examples of voice-based chatbots in software development include:

- Microsoft's CodeTalk: CodeTalk[2] is a voice assistant for software development integrated with Visual Studio. It enables developers to perform coding tasks, such as navigating code, debugging, and running tests, through voice commands.
- Amazon Lex: Lex[3] provides voice-based chatbot capabilities for software development. Developers can use voice commands to interact with Lex and perform actions like triggering builds, deploying applications, or retrieving information from development platforms.

Text-Based Chatbots: Text-based chatbots are widely employed in software development to provide real-time support, automate tasks, and facilitate collaboration through text-based conversations. These chatbots can be accessed through messaging platforms, integrated development environments (IDEs), or standalone applications. Examples of text-based chatbots in software development include:

- Microsoft Copilot: Copilot, developed by GitHub in collaboration with OpenAI, offers text-based code completion and suggestions. It assists developers by providing intelligent code snippets and suggestions as they type, accelerating the coding process.

2.2.1 Collaboration and Communication Chatbots:

Collaboration and communication chatbots focus on improving team interactions, facilitating knowledge sharing, and streamlining communication within development

[2]https://www.microsoft.com/en-us/research/project/codetalk/
[3]https://aws.amazon.com/lex/

teams. These chatbots typically integrate with messaging platforms and support both voice and text-based interactions. Examples of existing collaboration and communication chatbots in software development include:

- Slackbot: Slackbot[4] is an assistant within the Slack messaging platform that enables developers to perform various tasks, such as scheduling reminders, managing notifications, and accessing information.
- Pull Reminders: Pull Reminders is a Slack-based chatbot that helps developers manage code reviews and collaboration. It sends reminders about pending pull requests and facilitates smoother communication within development teams.

The current landscape of chatbots developed to support the process of software development encompasses both voice and text-based applications. These chatbots aim to enhance productivity, automate tasks, improve collaboration, and provide real-time support to developers. The integration of voice-based chatbots allows for hands-free and efficient interactions, while text-based chatbots offer flexibility and accessibility across multiple platforms. As AI and NLP technologies advance, we can expect further innovations and advancements in chatbots tailored specifically to support software development, ultimately improving the efficiency and effectiveness of development workflows.

2.2.2 The Current Landscape of Platforms and Frameworks to Develop Chatbot Applications

The development of chatbot applications has witnessed significant growth and innovation over the past few years. As the demand for AI-powered conversational interfaces increases across various industries, numerous platforms and frameworks have emerged to simplify the creation and deployment of chatbots. This section provides an overview of the current landscape of platforms and frameworks used to develop chatbot applications.

Chatbot Development Platforms: Several chatbot development platforms offer comprehensive tools and services that facilitate the entire chatbot development lifecycle, from design and training to deployment and maintenance. These platforms cater to both text-based and voice-based chatbot applications and often support integration

[4]slack.com

with popular messaging platforms, voice assistants, and websites. Some of the leading chatbot development platforms include:

- Dialogflow (2016): Powered by Google Cloud, Dialogflow[5] provides a NLU engine that allows developers to build AI-driven conversational interfaces for various platforms like Google Assistant, Facebook Messenger, and more.
- Microsoft Bot Framework (2016): Microsoft's Bot Framework[6] enables developers to create intelligent bots for Microsoft Teams, Skype, and other channels. It supports multiple programming languages and provides built-in NLU capabilities.
- Amazon Lex (2016): Part of Amazon Web Services (AWS), Lex[7] is a service for building conversational interfaces for applications using voice and text. It is the technology behind Amazon's Alexa.
- IBM Watson Assistant (2010): IBM Watson Assistant[8] offers a robust platform for developing AI chatbots that can be deployed across multiple channels, including web, mobile apps, and messaging platforms.

Open-Source Chatbot Frameworks: Open-source chatbot frameworks provide developers with the freedom to customize and extend their chatbot applications. These frameworks typically come with pre-built components, machine learning libraries, and APIs to accelerate the development process. Some prominent open-source chatbot frameworks include:

- Rasa (2016): Rasa[9] is an open-source conversational AI platform that offers tools for natural language processing and dialogue management. It empowers developers to create sophisticated and context-aware chatbots.
- Botpress (2017): Botpress[10] is an open-source chatbot development framework that focuses on scalability and customizability. It offers a visual flow editor, built-in NLU, and integrations with various messaging platforms.

[5]https://cloud.google.com/dialogflow
[6]https://www.botframework.com/
[7]https://aws.amazon.com/lex/
[8]https://www.ibm.com/products/watson-assistant
[9]https://rasa.com/
[10]https://botpress.com/

- ChatterBot (2014): ChatterBot[11] is a Python-based library for building chatbots that can engage in conversation with users. It uses a machine learning algorithm to generate responses based on training data.

Cloud-Based AI Services: Cloud service providers offer AI services that include chatbot functionalities as part of their offerings. These services leverage the providers' advanced AI technologies and infrastructure, allowing developers to integrate chatbots seamlessly into their applications. Examples of cloud-based AI services for chatbot development include:

- Google Cloud AI (2008): Google Cloud AI[12] provides a wide range of AI services, including Dialogflow for creating chatbots, as well as Cloud Natural Language and Cloud Translation for language processing and translation tasks.
- Azure Cognitive Services (2016): Microsoft's Azure Cognitive Services[13] include Language Understanding (LUIS) for NLU tasks and QnA Maker for building question-and-answer-style chatbots.
- AWS AI Services (2006): Amazon Web Services[14] offers AI services like Amazon Lex for building chatbots, Amazon Polly for text-to-speech synthesis, and Amazon Comprehend for language understanding.

Chatbot-Building Tools for Non-Developers: In addition to platforms and frameworks targeted at developers, there are also chatbot-building tools designed for non-technical users. These tools employ visual interfaces and no-code, low-code approaches, enabling business users to create simple chatbots without extensive coding knowledge. Examples of such tools include:

- Chatfuel (2015): Chatfuel[15] is a popular no-code chatbot platform that allows users to build Facebook Messenger chatbots without writing any code.
- ManyChat (2015): ManyChat[16] is another no-code chatbot platform designed specifically for creating chatbots on Facebook Messenger.

[11] https://chatterbot.readthedocs.io/
[12] https://cloud.google.com/products/ai
[13] https://azure.microsoft.com/en-us/products/ai-services?activetab=pivot:azureopenaiservicetab
[14] https://aws.amazon.com/machine-learning/ai-services/
[15] https://chatfuel.com/
[16] https://manychat.com/

- Landbot (2017): Landbot[17] is a chatbot-building tool that supports text-based and visually engaging chatbots, with drag-and-drop functionality for easy customization.

The current landscape (including GPT-3 and GPT-4) of platforms and frameworks for chatbot development offers developers and businesses a wide array of options to create sophisticated and intelligent conversational interfaces. Depending on the specific use case, development expertise, and budget considerations, stakeholders can choose the most suitable platform or framework to build chatbot applications that enhance user experiences and streamline interactions across various channels. As the field of AI and NLP continues to advance, we can expect further innovations in chatbot development tools and services, making it even more accessible for businesses of all sizes to leverage the power of conversational AI.

2.3 Automation and Autonomy of Software Systems

The rise of machines and artificial intelligence (AI) has revolutionized various industries, leading to increased automation of tasks that were once solely performed by humans [59, 112]. Numerous fields, such as manufacturing, data entry, bookkeeping, administrative tasks, and customer service, have witnessed a significant shift toward computerization. For instance, in manufacturing, robots now handle tasks like welding, painting, and packaging on assembly lines, enhancing productivity and precision.

The benefits of automation are evident, as it improves productivity, ensures consistent quality, reduces errors, and streamlines processes [165]. AI advancements have further accelerated this trend, allowing machines to take over human responsibilities across various domains. However, the question of whether complete automation is always advantageous remains a subject of debate [57].

In this context, it is essential to distinguish between automation and autonomy in software systems. Automation refers to the process of automating specific tasks or functions, where machines execute predefined actions based on predefined rules and algorithms. On the other hand, autonomy refers to the capability of software systems to operate independently, making decisions without direct human intervention.

To strike a balance between human involvement and machine automation, many researchers propose the concept of "levels of automation" (LOA) [165, 50]. This approach

[17] https://landbot.io/

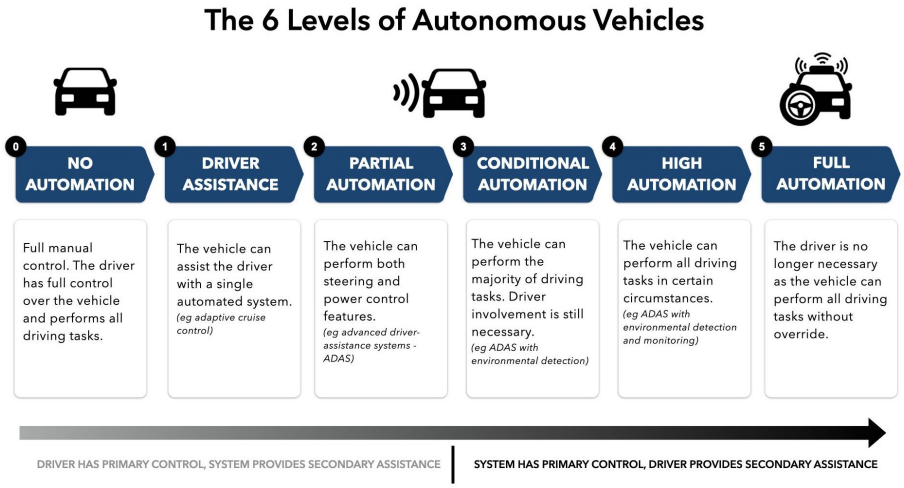

Figure 2.2: Example of Levels of Automation in Autonomous Vehicles [140].

suggests that different tasks or jobs can be assigned distinct levels of automation based on their complexity, criticality, and the required human-machine collaboration. Figure 2.2 shows the levels of automation in autonomous vehicles.

For example, in autonomous vehicles, certain driving tasks can be automated (e.g., lane keeping, adaptive cruise control) while still requiring human supervision and intervention for more complex situations (e.g., unexpected obstacles or adverse weather conditions). This approach aims to leverage the strengths of both humans and machines to achieve optimal performance and safety.

Consider, for instance, a situation where performance deteriorates at a higher level of automation due to various factors. In such cases, it is crucial to have mechanisms in place for re-evaluating the system's operating LOA and ideally modifying it. This adaptive approach allows for a potential transition to a lower automation level and/or a transfer of control to the human operator until the issue is resolved. Conversely, circumstances might arise where high human workload or inadequate human responses prompt the system to elevate its automation level, reducing its reliance on human intervention. The application of adaptive automation can potentially alleviate human performance costs associated with high-level decision automation, such as imbalanced mental workload, reduced situational awareness, complacency, and skill degradation. Traditionally, the allocation of functions between humans and computers has been rooted in stereotypical characteristics of their capabilities, yielding limited success [165].

In the realm of software development, the concept of autonomy and adaptive allo-

cation of tasks becomes increasingly relevant. With the rapid advancements in AI and machine learning, software systems can exhibit a certain level of autonomy, such as automatically optimizing code or making decisions in response to user inputs. However, ensuring effective collaboration between humans and autonomous systems in software development remains a challenge [52, 131, 160].

Research in manufacturing and human-robot interaction has shed some light on the factors that influence effective collaboration between humans and autonomous systems [177]. However, there is a need for further investigation into adaptive task allocation and decision-making in different levels of automation in the context of software development.

To address these challenges, developing intelligent conversational agents or chatbots can play a crucial role. These chatbots can act as intermediaries between humans and autonomous software systems, facilitating communication and understanding between the two. For instance, in software testing, chatbots can elicit requirements from developers, automatically generate test cases, and provide test results, while still allowing developers to intervene and make critical decisions when needed. To illustrate this concept, consider an AI-driven chatbot called "TestBot" integrated into a software development team. TestBot autonomously generates test cases based on the software requirements but consults with developers through natural language conversations before executing critical tests that require human judgment. This symbiotic collaboration ensures that the testing process benefits from automation while leveraging human expertise to handle complex scenarios effectively.

In conclusion, the distinction between automation and autonomy is crucial in modern software development. By adopting a "levels of automation" approach and leveraging intelligent chatbots, we can reach a balance between human and machine capabilities, optimizing the efficiency and effectiveness of software systems. Future research in this area should focus on enhancing the collaboration between humans and autonomous systems to harness the full potential of automation while maintaining human oversight and intervention when necessary.

2.4 Variability Modelling

Software development is a field inherently characterized by its variability, wherein solutions are crafted to accommodate diverse and evolving requirements [36, 62]. Variability is the ability of software systems or artifacts to be adjusted for different contexts

[166, 61]. The significance of variability in software development is further accentuated when considering the context of human-chatbot interactions. In this landscape, the ability to customize interactions and adapt to the unique demands of developers becomes paramount [32].

Variability modelling, a well-established concept in software engineering, offers a promising avenue for addressing the intricacies of human-chatbot interactions. It provides a structured approach to capturing and managing diverse features and requirements within a software system [35]. These models enable software engineers to create flexible and customizable solutions that can be tailored to specific user needs, ensuring that the software can adapt to changing circumstances [35, 36]. When applied to the realm of AI-driven chatbot interactions, variability modelling can empower developers with tools and methodologies to create chatbot systems that are not only efficient but also highly adaptable. Such adaptability is particularly crucial when considering the evolving nature of chatbot technologies and the ever-changing requirements of developers. Research in this domain has shown that embracing variability modelling in software engineering can lead to significant benefits. It allows for the systematic representation of different interaction scenarios and user preferences, enabling the development of chatbot systems that can seamlessly switch between various modes of interaction. Moreover, variability models facilitate the rapid development of chatbot features by reusing existing components, reducing development time and effort [35].

As the field of AI-driven chatbots continues to advance, variability modelling offers a structured and efficient approach to designing and implementing interactions that are responsive to the individual needs of users. By embracing variability modelling in this context, we can harness its potential to create chatbot systems that are not only intelligent but also highly adaptable, catering to the unique requirements of software developers as users of these tools.

In conclusion, this chapter serves as the foundational background in this thesis, synthesizing key concepts from software engineering. The significance of context in software engineering, the role of chatbots in supporting software development, the dynamic interplay between automation in software systems, and the concept of software product lines are integral pillars of this research. These pillars are linked to the thesis's overarching goals: understanding the interconnectedness of software development and context, recognizing the importance of advanced AI chatbots, exploring the challenges in human-chatbot interactions within the context of LOAs, and embracing a design-thinking approach to address evolving needs in software development. As the chapters progress, these foundational concepts will be further dissected and applied to shed light on human-chatbot collaboration in the evolving landscape of software development.

Chapter 3

Related Work

Several related areas play pivotal roles in shaping the future of developer-chatbot interactions. Context-aware systems form a crucial foundation, as they enable software to adapt and respond to situational cues, enhancing the user experience. Conversational agents, empowered by natural language processing advancements, offer the potential to bridge the gap between developers and automated assistance. Variability approaches, essential in managing the diverse tasks and challenges developers face, provide valuable insights into tailoring solutions to specific contexts.

Moreover, the realm of autonomous systems, marked by systems that can operate without (or with minimal) human intervention, offers a tantalizing glimpse into the future of software development support. However, the adoption of such systems needs to be grounded in a deep understanding of the levels of automation (LOAs) appropriate for different contexts. The implications of employing LOAs within human-chatbot interactions are critical to providing seamless and effective developer support.

In this Chapter, we introduce the related works of the major topics discussed in this thesis, namely (1) context-aware systems, (2) conversational agents, (3) variability approaches, and (4) autonomous systems and levels of automation.

3.1 Context-Aware Software Development Systems

The significance of context in software development cannot be understated, as it greatly influences the approach and decisions made by software developers while performing

their tasks. Researchers have extensively explored the concept of context in software engineering, aiming to understand its implications and harness its potential benefits.

One area of research in this domain focuses on leveraging context in software engineering to recommend the most suitable developer for specific tasks. Lin et al. [92] delved into this aspect, examining how context-aware techniques can be employed to assign appropriate developers to particular project tasks. By considering the individual expertise, experience, and availability of developers within the context of the software project, this approach aims to enhance productivity and task allocation efficiency.

Another line of research has explored context-aware systems for online documentation to aid developers in fixing software bugs. Ashok et al. [10] conducted studies to develop systems that intelligently recommend relevant documentation to programmers when they encounter bug-related challenges. Such context-aware documentation recommendations aim to streamline the debugging process, reducing development time and enhancing software quality.

Additionally, researchers have investigated context-awareness for software artifacts. Cubranic and Murphy [181] proposed an innovative approach called Hipikat, which suggests related artifacts that should be accessed or edited when developers interact with a particular software artifact. This context-aware recommendation system aims to facilitate code comprehension, collaboration, and maintenance, thus improving software development productivity.

The integration of context in software development has the potential to revolutionize the way developers work, as observed by Murphy et al. [116] and Murphy [119]. It can lead to a more efficient and informed decision-making process, fostering better collaboration and code quality. However, despite its recognized benefits, context awareness in software development is still not fully supported [20, 116, 110]. The very recent integration of advanced technologies, such as Microsoft Copilot, holds the potential to significantly enhance context awareness. Challenges and gaps remain in providing comprehensive context support during various software development tasks.

Understanding and using context in software engineering can significantly impact the efficiency, productivity, and quality of software development processes. Research efforts in this area have explored context-aware developer task assignments, documentation recommendations, and artifact interactions, showcasing the potential of context-aware approaches to transform software development practices. Nevertheless, further advancements are needed to embrace context awareness in software development and bridge the existing gaps to create a more contextually intelligent and adaptive development environment.

3.2 Conversational Agents in Software Development

Conversational agents (or chatbots), a subset of AI technology, have significantly transformed the way humans interact with systems. Their application in software development scenarios introduces unique challenges and opportunities. Through an in-depth analysis of conversational agents' role in supporting developers, this thesis seeks to establish requirements for optimizing their design and implementation.

Conversational agents (CAs) are software tools that interact with people while capturing information, interpreting and responding in natural language, like conversing with another human being, [100]. Researchers argue Human-Computer Interaction (HCI) may transition from graphic to conversational interfaces with chatbots [58] with 68% of participants of a study [21] reporting that a key goal of this form of interaction is a likely increase in productivity. Embedding context into workflows and providing development information to workers was investigated by Bradley et al. [20], to leverage versioning information to developers and other software development workflows through a voice-activated agent. Other research uses chatbots to detect code conflicts [128] and provide expert recommendations [28]. Chatbots are interesting to investigate, as these tools enable software developers to engage in development operations without needing technical knowledge [87]. Other studies have identified other benefits of using chatbots in Software Engineering (SE), such as the benefit novice developers can acquire [126]. In conclusion, researchers have been interested in studying chatbots due to the many benefits this tool can bring to software development. One evidence of this interest is the increasing number of publications and workshops in the area. Scopus database reported seven publications with the keywords chatbots or "conversational agent" and "software engineering" in 2019, and this number doubled in 2022.

Several papers have reported the experience of software developers being supported by chatbots. The work of Okanović and colleagues [126] presents PerformoBot, a chatbot that supports developers when performing load testing. This research has investigated how developers interact and perceive PerformoBot, showing that novice users especially could benefit from the use of the proposed tool, as well as some aggregated educational effects promoted by the tool.

The work of Liu et al. [94] presents an in-depth analysis of the interaction and user experience of developers and bots, as reported in the paper "... these bots may have consequences on the user experience of existing and prospective project contributors." They introduce a mental model to facilitate the understanding of the human-bot interaction, as well as suggest seven principles for bot developers, which can be used as

guidelines to evaluate the user experience of bots and their interaction with developers. Such an emerging topic in software engineering has sparked the interest of researchers and there are now workshops in major conferences in the field targeting bots in software engineering. One example is BotSE [botse.org], which presents the behavioral science concept of choice architecture and the impacts of human decisions [22]. Another example presented in this workshop addresses the importance of the integration of bots with humans in software development workflows [168]. Given chatbots have been explored as tools to support software developers in their work, we also propose to use these communication tools to recommend their workflow to developers, given a specific context. We believe this tool leverages the contribution of the human-in-the-loop, having the developers as users. Additionally, the advances in chatbot development and research have shown that there is potential to use chatbots with their natural language processing techniques to lower the learning curve to use the tool. Other than the demonstrated advantages and advances in the related work, the possibility of having a semi-ubiquitous tool in the development IDE, sending developers reminders, engaging with simple or complex issues, collecting information and also providing information to developers, appears promising.

Chatbots have already been explored as tools to support software developers in their work. Our work supports the idea that conversational agents can be used to provide contextual-relevant information to software developers, given the massive amount of context developers must deal with daily. We then investigate, from software developers, the requirements desired in conversational agents through user studies.

3.3 Autonomous and Adaptive Systems & Levels of Automation

In the past few years, the development of autonomous systems has increased, with an increasing focus on integrating automation within these systems. For example, in Uncrewed Aerial Vehicless (UAVs), researchers have explored various methods to automate the control of these vehicles [33]. These methods include using deep learning to detect and avoid obstacles for uncrewed vehicles [12], using reinforcement learning to optimize autonomous driving agents [80] and developing specific software stacks to support the advancement of self-driving cars [138]. Moreover, Lorenz et al. [95] have investigated how variations influence trust in an autonomous system in system speed, accuracy, and uncertainty. This study demonstrated that humans are likelier to miss

system errors when highly trusting the system. The level of self-correction with which an automated system produces responses can also impact human trust, according to the authors.

On adaptive systems, increasing the awareness of feedback loops and monitoring are vital for creating adjustment systems in systems that need to deal with changing needs in a flexible way [164]. Depending on the system's adaptation goal (namely optimization, recovery, repairing, configuration and others), these systems will have different properties [174]. Previous research has delved into the assessment of adaptive software systems. Researchers have introduced a framework rooted in control theory, focusing on the evaluation of stability and robustness as fundamental properties of adaptive systems [109]. These properties were dissected in relation to programming paradigms, architectural styles, modelling paradigms, and software engineering principles. Salehie and Tahvildari [148] conducted a comprehensive survey of various projects dealing with software system adaptation, addressing concerns regarding the 'how,' 'what,' 'when,' and 'where' of adaptation. They also introduced a hierarchical perspective exploring their connection with software quality factors. Our research suggests a system adaptation determined by a combination of factors influencing LOA.

3.3.1 Levels of Automation

Levels of automation (LOA) have been widely used to describe the degree to which a system is automated, ranging from fully manual to fully autonomous. Several taxonomies have been proposed to categorize different levels of automation, such as the ones from Parasuraman and Sheridan [131] and Riley et al. [144]. In recent years, researchers have extended these taxonomies to specific domains. The work of Machado et al. [97] focuses on the heavy-duty mobile machinery industry and presents a two-dimensional 6x6 matrix. The work of Kugele et al. [83] presents a four-level taxonomy that provides a foundation for describing future systems, including robotic and drone taxi systems.

Within levels of automation, there is the potential for the level and nature of automation to remain flexible and capable of real-time adjustment during the operation of a system. This concept denoted as adaptive automation, entails the dynamic modification of autonomy levels during system operation, as highlighted by studies such as Moray et al. [115], Parasuraman et al. [130] and Scerbo [152]. This adaptive automation paradigm aligns with the concept of dynamic task or allocation, where the distribution of tasks between human and machine agents is not predetermined but rather adaptable,

responsive, and context-sensitive.

Adaptive automation stands in contrast to the "automate as much as possible" philosophy, as elaborated in Fereidunian et al. [56]. Hence, an effective automation solution should possess the capability to adjust dynamically the LOA in response to peripheral changes. A multitude of research has delved into investigating human performance in the context of adaptive automation, often through simulations involving domains such as flight, air traffic control, driving tasks, and process control [115, 70, 130, 152].

Our research investigates the factors that induce changes in automation levels within autonomous or semi-autonomous systems. Instead of concentrating solely on a specific use case or domain, our objective is to compile a comprehensive list of these factors. This compilation serves the purpose of facilitating a deeper understanding of the design elements within autonomous systems.

3.4 Variability in Software Systems

Research has mainly looked at variability in Software Product Liness (SPLs), but variability is a big part of most software systems, not only related to SPL [60]. Many different types of software systems are made with variability in mind. For example, systems that can change themselves, platforms that can be customized, or systems that put services together in different ways while they are running. There are situations where differences need to be managed. This can include setting up systems, changing parts of a system to make it fit better, choosing different features, or changing how a software service works while it is running. Since these differences are everywhere, software makers need to understand them well. They also need good ways to deal with them, like methods and tools that help them understand, manage, and figure out what to do about differences. This is especially important when many different people are involved, like the people who use the software and want it to work in different ways, the people who write the code and need to know where the differences are, or the people who test the software and need to check all the different possibilities [61]. To this end, Galster et al. [61] have examined research on variability in software systems, proposing a vast literature review. They concluded that software quality attributes have not received much attention in the context of variability and that research designs on variability are vaguely described.

Another review was conducted by Chen et al. [30], this time focusing on the variability management in software product lines. They observed that a significant portion

of the existing work revolves around variability concerning features, assets, or decisions. Chen and Babar [31] have also evaluated variability management approaches in software product lines. Also rooted in the SPL domain, Kontogogos and Avgeriou [82] conducted a review of variability in service-based systems. Their study identified approaches that incorporate integrated variability modelling, which introduces new representations of variability specifically for service-based systems. Similarly, Kazhamiakin et al. [74] conducted an informal review that explored adaptation in service-based systems.

Furthermore, Alves et al. [7] examined variability in the realm of requirements engineering for software product lines. Their study aimed to identify requirements artifacts addressed by contemporary product line approaches, the related requirements engineering activities, and the strategies for product line adoption they employ. In contrast, our study takes a broader perspective that encompasses variability beyond the scope of product lines and requirements engineering. In essence, our study seeks to contribute to the understanding and design of context-based chatbots, that can adapt to these contexts, while similarly to the related work, we use SPL to represent and model part of our proposed design in Chapter 6.

This chapter explores various domains crucial to the evolution of developer-chatbot interactions. It begins by highlighting the importance of context-aware systems in software development, emphasizing their potential to enhance user experience by adapting to situational cues. The discussion extends to conversational agents, powered by natural language processing, and their role in bridging the gap between developers and automated assistance. Variability approaches, crucial for managing diverse tasks, and the realm of autonomous systems, operating with minimal human intervention, are also discussed.

Chapter 4

Preliminary Study: Understanding Context in Software Engineering

> "The consideration of context as a first-class construct opens up new opportunities to take a substantial step forward in providing tools for developers that enable the developer to use their cognitive abilities to attack the problems only a human can address." – Gail Murphy [116]

Before investigating solutions and ways to enhance software development with context, there is a need to understand context in software engineering, how it has been used, what researchers have called it, and other characteristics. We propose a Literature Review (LR) in contexts in software engineering to answer such questions. This chapter[1] presents the first step of our approach, i.e., to understand the context in software engineering. It includes the details of the planning and execution of the LR, as well as the observed results.

Our objective in this preliminary study is to enhance the current state-of-the-art by incorporating the latest insights into contexts within software engineering. In doing so, we have compiled a comprehensive repository of contextual information. Through this study, we have recognized the multifaceted nature of context within software engineering and its impact on the daily tasks of software developers. This discovery has ignited our research efforts and deepened our commitment to addressing the challenge of managing this context to provide valuable support for software development. As an extension of our ongoing efforts, beyond the examination conducted in this chapter,

[1] This chapter is reprinted in modified form from [102].

we have taken a moment to reflect and introduce a paradigm in the realm of software development, considering context, chatbots and machine learning. This paradigm is presented in Appendix E.

4.1 Overview and Motivation

Software development is a multifaceted and knowledge-intensive endeavor [43] [110]. It involves various technologies, and the documentation for these technologies is scattered across multiple sources, including tutorials, Stack Overflow, GitHub, project wikis, and API tutorials. Moreover, being a human-centred task [170][137], software development is influenced by diverse practices, driven by each developer's expertise, personal interests, gender [88] [69], stress management approaches [150], and many other factors. These individual characteristics and the context in which developers operate heavily influence their approach to performing tasks. Recognizing the significance of this context in software development, researchers have emphasized the need to treat it as a first-class construct [116], as it can lead to transformative changes in developers' work processes.

The information generated or handled by software developers is vast, and the problem domains they work in are highly diverse. Moreover, the software development context is presented in various formats, such as code repositories like GitHub for code and issue tracking, natural language communication through code comments, GitHub comments, emails, meeting reports, and other media. Despite the growing recognition of the importance of software development context, there is still a lack of comprehensive models explicitly capturing this variable context in a unified adaptive software development framework capable of providing context-driven recommendations to developers throughout the project lifecycle.

Context holds a central and indispensable role in empirical software engineering, distinguishing it as a unique discipline. It permeates software practice, playing a crucial part in shaping and influencing software development processes and outcomes [48].

Defining context involves understanding its etymological origins in the Latin term *contextus*, which denotes weaving together or forming connections [146]. Various philosophical and practical perspectives offer different insights into context, but in software engineering, the predominant approach has been to treat context as a set of variables [48]. Context can be perceived as an integral part of an environment that can be sensed and considered [147]. Context has also been assumed with a more specific and focused definition: it refers to the information encompassing the system under development, as

well as the environment and process in which the system is being created [116]. This definition underscores the importance of understanding the software's broader context, including the dynamic factors, the development environment, and the human processes involved.

Given the mutable nature of various contextual aspects, such as the domain, process, technologies, and people, it becomes evident that software systems that can adapt and respond to these changes have a considerable advantage over those that are ill-prepared for contextual variations [121]. Embracing context-aware methods in software engineering enables systems to accommodate and leverage these changing scenarios, thereby improving the effectiveness and relevance of the developed software.

In summary, context stands as a fundamental concept in empirical software engineering, profoundly influencing software practices and research. Its definition may vary, but in the context of software development, it involves understanding the interplay between the system, the environment, and the development process. Acknowledging and incorporating context awareness in software engineering can lead to more flexible, adaptive, and robust systems capable of addressing the complexities posed by the dynamic and diverse contexts in which they operate.

4.1.1 Objectives

Our primary objective is to understand the context of software development projects, and where and how it has been used and managed. By having access to relevant contextual information, developers can focus on creative tasks rather than being preoccupied with executing procedural tasks or searching for specific information. Furthermore, considering the variability of context formats is essential. Relevant contextual information may include the next artifact to edit or read, an API tutorial, a code snippet, or insights from other developers. To establish a foundation for this research, we propose conducting a LR in the context of software development. Based on the findings, we also present a preliminary context-augmented framework for software development projects (in Appendix B).

To achieve the goal of managing context in software development projects through an adaptive context framework, we outline the following steps:

1. **Identify the Context**: We conduct a comprehensive LR of context in software development to explore the types of contexts researchers have identified in this domain. Presented in this introductory Chapter.

2. **Model the Context**: Propose a context model based on the information gathered from the LR. Presented in Appendix B.

3. **Preliminary Framework**: Develop a preliminary framework capable of capturing software development context and dynamically adapting to contextual changes. Presented in Appendix B.

In this study, we aim to identify the types of software development contexts through an LR to lay the groundwork for our research. We structure our investigation of the research question as follows: ***RQ1: What types of context have been identified by researchers in software development projects?*** This research question branches into nine sub-questions, as detailed in Section 4.2. The gathered information from the LR is organized and presented in Tables A.1 and A.2. Table 4.1 presents the nine sub-questions that supported the LR. These nine questions relate specifically to the main RQ1 in this thesis, and are sub-questions of this main RQ1 broader question. Table 4.2 presents the summary of the findings of each question. The complete table with the findings is presented in the Appendix, in Table 2.1.

4.1.2 Literature Review and Preliminary Framework Proposal

The LR plays a crucial role in understanding the different facets of the software development context. By exploring existing research on context in software development projects, we can identify the various contextual factors that influence developers' actions, decisions, and interactions. The preliminary framework we propose in Appendix B is a step furthering this investigation, aiming at representing the knowledge acquired from the LR into a context-aware adaptive system that empowers developers with relevant contextual information and recommendations. The framework is designed to adapt dynamically to changes in the development context, providing a more efficient and supportive environment for developers throughout the software development lifecycle.

By systematically exploring the research questions and summarizing the findings, we gain valuable insights into the existing knowledge of the software development context. These insights will serve as the building blocks for our research presented in this thesis, paving the way for further research and the development of adaptive systems to support software developers in their challenging and dynamic work environments. Adapting to contextual changes will enable developers to focus on creative problem-solving and improve their overall productivity and efficiency during the software development process. Ultimately, our research goal is to contribute to the advancement of

Table 4.1: Research (sub)Questions in the Literature Review.

RQ1.1: What are the types or classifications of context?
RQ1.2: Is there a model specification technique used?
RQ1.3: What are the goals or purposes of context?
RQ1.4: On what step or phase of the software development does the context focus?
RQ1.5: Are there any evaluations performed?
RQ1.6: Are there identified limitations or gaps when using context?
RQ1.7: What are the advantages or disadvantages of this context?
RQ1.8: How are the context instances mined?
RQ1.9: Are there any proposed abstractions?

software development practices by harnessing the power of context-aware systems to empower developers and promote seamless collaboration within software development projects.

4.2 Literature Review

We present the LR that attempts to identify articles that depict contextual information in software development projects. The research method for the LR is divided into two steps. First, we performed a search using a search string, presented in Section 4.2.1, and then we performed the execution of the review, presented in Section 4.2.2. We performed a backwards snowballing approach [71] within the retrieved papers, presented in Section 4.2.4. Last, we perform the analysis of the retrieved articles, described in section 4.2.3. This section ends with a discussion of the findings of the LR.

4.2.1 Planning Phase

Literature Reviews are a standard method of obtaining evidence on a particular subject and provide categorized results that have been published in a specific research area [133] [14]. The LR reported in this chapter was conducted to assess the state-of-the-art of current articles that propose research in contextual information for software development, to understand better the use and variability of diverse contexts in software engineering.

Table 4.2: Summary of Findings from the Literature Review. Full table available in Appendix.

Research Question	Summary of Findings
RQ1.1	Contextual factors include task context, software structure, team, person, file version, etc.
RQ1.2	Interactions with IDEs, RSEE systems development, etc.
RQ1.3	IDE provide code artifact, task allocations, project memory from artifacts, etc.
RQ1.4	Planning, when a bug occurs, coding, etc.
RQ1.5	Improvements comparing to baseline, useful results, etc.
RQ1.6	Duplicate bug reports, few tools available to practitioners, etc.
RQ1.7	Provides meaningful information x Missing evaluations on performance of developers, etc.
RQ1.8	Recommends who shouldresolve a task, etc.
RQ1.9	Contextual factors abstracted to represent artifacts, tasks abstracted into high-level tasks, etc.

The protocol suggested by Petersen et al. [133] uses the Goal-Question-Metric (GQM) approach [167] to define a goal for a LR. According to the GQM approach, the goal of this LR is to:

analyze software development

with the purpose of developing a characterization

regarding software developers' context

from the point of view of researchers

in the context of software projects

Emerging from the defined objective, the research question this LR aims to answer is ***RQ1: What types of context have been identified by researchers in software development projects?***

This question aims to determine the state of practice and lay the foundations for this research. Specifically, we pursue the following characteristics of software development context in the literature as outlined by the following research questions:

RQ1.1: *What are the types or classifications of context?* Are there any types or classifications of the context subject of the article retrieved from the literature?

RQ1.2: *Is there a model specification technique used?* Are there model specifications for the proposed context, for example, an ontology, a model extension, or other types of models of context?

RQ1.3: *What are the goals or purposes of context?* In this research question, we aim to retrieve the purpose or the goals of the context subject of the retrieved article.

RQ1.4: *On what step or phase of the software development does the context focus?* In traditional software development, there are different development phases, such as analysis (requirements), coding, testing, and deploying. If a paper identifies the phase where the context can be applied, we want to capture this information and make it explicit.

RQ1.5: *Are there any evaluations performed?* With this specific research question, we are exploring if any evaluation was performed on the proposed context or the context's purpose.

RQ1.6: *Are there identified limitations or gaps when using context?* With this question, we wish to investigate if there are any identified limitations provided within the context proposal or utilization, we also aim to make this information explicit in this LR.

RQ1.7: *What are the advantages or disadvantages of this context?* We want to explore the pros and cons of the context retrieved from the literature.

RQ1.8: *How are the context instances mined?* With this research question, we are looking for the uses of the context and if they were mined to retrieve other processed information such as a recommendation.

RQ1.9: *Are there any proposed abstractions?* With this research question, we are looking for abstractions of the proposed context within the article retrieved from the literature.

To create the search string, we have used the PICO (**P**opulation, **I**ntervention, **C**omparison, **O**utcome) strategy, proposed by Pai et al. [127]. The PICO definition is presented in Table 4.3.

Having defined PICO, the search string for each database is

(Programmer OR (software AND (developer OR tester)) OR ("software development project" OR "software development environment")
AND (Context OR "event based" OR "self adapt" OR skill OR "team size" OR "organizational structure" OR "organizational structure" OR situational OR "application type" OR "type of

ht!

Table 4.3: Search string creation process with PICO [127].

(P)opulation: Software developer in software development
Keywords: (Programmer OR (software AND (developer OR tester)) OR (”software development project” OR ”software development environment”)
(I)ntervention Control: Context
Keywords: (context OR ”event based” OR ”self adapt”)
(C)omparison: None
(O)utcome Measure: Methodology
Keywords: tool* OR system* OR recommend*

application”)
AND (tool OR system* OR recommend*).*

In terms of article selection, inclusion and exclusion criteria were proposed. These criteria consider the articles:

- Written in English
- Within a Software Engineering scope
- Involving software development
- That discusses or involves software development projects
- Presenting studies of context in software development
- That is NOT about IoT (Internet of Things) or hardware

4.2.2 Execution Phase

The initial set of articles was retrieved from the ACM Digital Library on August 9th, 2019. The execution phase returned 135 articles. After reading the title and abstract (two researchers), 18 articles were selected for full reading, according to inclusion and exclusion criteria definitions. The complete list of selected articles is presented next. The list shows the year of publication, authors and publication title.

1. Keith Marzullo, Douglas Wiebe. "Jasmine: A software system modelling facility." *ACM SIGPLAN Notices*, 22(1):121–130, 1987. [98]

2. Bowen Alpern, Alan Carle, Barry Rosen, Peter Sweeney, Kenneth Zadeck. "Graph attribution as a specification paradigm." *ACM SIGPlan Notices*, 24(2):121–129, 1988. [6]

3. Allen Goldberg. "Reusing software developments." *ACM SIGSOFT Software Engineering Notes*, 15(6):107–119, 1990. [64]

4. Paul L Baker. "Ada as a preprocessor language." *ACM SIGAda Ada Letters*, 10(1):83–91, 1990. [13]

5. Davor ČubraniĆ, Gail C. Murphy, Janice Singer, Kellogg S. Booth. "Learning from project history: A case study for software development." In *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work, CSCW '04*, pages 82–91, 2004. ACM. [171]

6. Vesna Mikulovic, Michael Heiss. "How do I Know What I Have To Do?" The Role of the Inquiry Culture in Requirements Communication for Distributed Software Development Projects. In *Proceedings of the 28th International Conference on Software Engineering*, pages 921–925, 2006. [113]

7. Vincent Rosener, Denis Avrilionis. "Elements for the definition of a model of software engineering." In *Proceedings of the 2006 International Workshop on Global Integrated Model Management*, pages 29–34, 2006. [145]

8. Kleber Rocha de Oliveira, Mauro de Mesquita Spinola. "Porei: Patterns-Oriented Requirements Elicitation Integrated - Proposal of a Metamodel Patterns-Oriented for Integration of the Requirement Elicitation Process." In *Proceedings of the 2007 Euro American Conference on Telematics and Information Systems*, pages 1–8, 2007. [39]

9. B. Ashok, Joseph Joy, Hongkang Liang, Sriram K. Rajamani, Gopal Srinivasa, Vipindeep Vangala. "Debugadvisor: A recommender system for debugging." In *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, ESEC/FSE '09*, pages 373–382, 2009. [10]

10. Marcelo Cataldo, James D Herbsleb. "End-to-end features as meta-entities for enabling coordination in geographically distributed software development." In *2009 ICSE Workshop on Software Development Governance*, pages 21–26, 2009. [27]

11. Hossein Tajalli, Nenad Medvidovic. "A reference architecture for integrated development and run-time environments." In *2012 Second International Workshop on Developing Tools as Plug-Ins (TOPI)*, pages 19–24. IEEE, 2012. [163]

12. Nicolas Devos, Christophe Ponsard, Jean-Christophe Deprez, Renaud Bauvin, Benedicte Moriau, Guy Anckaerts. "Efficient reuse of domain-specific test knowledge: An industrial case in the smart card domain." In *2012 34th International Conference on Software Engineering (ICSE)*, pages 1123–1132. IEEE, 2012. [42]

13. Juliana Saraiva. "A roadmap for software maintainability measurement." In *2013 35th International Conference on Software Engineering (ICSE)*, pages 1453–1455. IEEE, 2013. [149]

14. Jun Lin. "Context-aware Task Allocation for Distributed Agile Team." In *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering, ASE'13*, pages 758–761, 2013. [92]

15. Patrick Wagstrom, Subhajit Datta. "Does latitude hurt while longitude kills? geographical and temporal separation in a large-scale software development project." In *Proceedings of the 36th International Conference on Software Engineering*, pages 199–210, 2014. [175]

16. Gail C. Murphy. "Getting to flow in software development." In *Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, Onward! 2014*, pages 269–281, 2014. [117]

17. A. M. Lima, R. Q. Reis, C. A. L. Reis. "Empirical evidence of factors influencing project context in distributed software projects." In *2015 IEEE/ACM 2nd International Workshop on Context for Software Development*, pages 6–7, 2015. [90]

18. Gail Murphy. "Beyond integrated development environments: adding context to software development." In *Proceedings of the 41st International Conference on Software Engineering*, pages 73–76. IEEE Press, 2019. [116]

After fully reading the 18 articles, the five articles that had content that satisfied the defined research questions and inclusion and exclusion criteria are presented in Table 4.4. The analysis of the LR is presented in the next section.

Table 4.4: Articles selected for consideration after full read.

ID	Publication Year	Authors	Title
A1	2019	Murphy, Gail C. [116]	Beyond Integrated Development Environments: Adding Context to Software Development
A2	2014	Murphy, Gail C. [117]	Getting to Flow in Software Development
A3	2013	Lin, Jun [92]	Context-aware Task Allocation for Distributed Agile Team
A4	2009	Ashok, B.; Joy, Joseph; Liang, Hongkang; Rajamani, Sriram K.; Srinivasa, Gopal; Vangala, Vipindeep [10]	DebugAdvisor: A Recommender System for Debugging
A5	2004	ČubraniĆ, Davor; Murphy, Gail C.; Singer, Janice; Booth, Kellogg S. [171]	Learning from Project History: A Case Study for Software Development

4.2.3 Analysis Phase

In this section, we present the findings of the articles retrieved by the LR using the search string. Each of the nine sub-research questions is discussed. We also present the results in a condensed table of contexts, which we included in Appendix A.

RQ1.1: ***What are the types or classifications of the context?***

Regarding the first research question, the types or classifications of context vary. Papers A2, A3 and A5 explicitly mention project task context. These articles define task context as the information around a project task or the relationships in an information space that are relevant to a software developer as they work on a particular task. A1 mentions context in a more broad and integrative perspective, by listing the following existing contexts: static software structure, dynamic system execution, historical artifact changes, developer activity, and team and organization activity. The historical artifact changes proposed by A1 can also relate to the context suggested by A5, in a way that both articles mention the provenance of information. A4 proposes context around the error scenario, meaning that the context proposed is natural language text, textual rendering of core dumps or the debugger output of errors that might occur when developing software.

RQ1.2: ***Is there a model specification technique used?***

For RQ1.2, none of the papers mention whether there are model specification techniques used for the proposed contexts.

RQ1.3: ***What are the goals or purposes of context?***

Regarding RQ1.3, the objectives of the contexts are clarified. For article A1, each proposed context has a specific goal. The context type is listed below, followed by its purpose.

- Static software structure: IDEs provide static source code artifacts as context to tools hosted in the environment.
- Dynamic system execution: Context in the form of dynamic execution information about a system under development.
- Historical artifact changes: Tools that access historical information about a system's static artifacts.

- Developer activity: Context about how humans work to produce the system, and not necessarily what was generated during the system's production. An example is Mylyn's degree of interest [116].
- Team and organization activity: Treating the activities across a value stream as context.

Paper A2 states that the use of task context can approximate task context by either capturing developers' interactions or using data from repositories. The context is then used to determine if the information captured or used is relevant to new tasks that will be performed.

Paper A3 applies task-related information to produce task allocation recommendations. Paper A4 employs error texts to create a query which could be kilobytes of structured and unstructured data containing all contextual data for the issue being debugged. This query allows users to search through all available software repositories such as version control, bug database and logs of debugger sessions. Finally, paper A5 explains that storing context information (Person, Message, Document, Change Task and File version) can be used to create a project memory from the artifacts and communications created during a software development project's history. Using this context information can facilitate knowledge transfer from experienced to novice developers.

RQ1.4: ***On what step or phase of the software development does the context focus?***

Regarding RQ1.4, Papers A3, A4 and A5 mention where the context should be used during software development. A3 explains that its proposed approach should be used during the planning when tasks are being allocated to software developers. A4 states that their proposed context focuses on the occurrence of bugs and A5 explains their proposed context can be manipulated during coding or when bugs occur.

RQ1.5: ***Are there any evaluations performed?***

Regarding evaluation, the subject of RQ1.5, papers A1 and A2 do not present performed evaluation. A3 mentions that a tool was built and evaluated, presenting better results than the tool being used as a comparison. A4 explains the performance evaluations returned useful results (bug resolutions) for 75% of the cases tried. Finally, A5 performed a qualitative evaluation regarding the effective use of history information by newcomers within the developed tool that implements the contexts. Results, when tasks are considered complex, are very prominent as:

> "The examples of previous changes provided by Hipikat were helpful to newcomers working on the two change tasks. The recommendations were

used as pointers to snippets of code that could be reused in the new tasks and as indicators of starting points from which to explore and understand the system. Without such help, it is hard for a newcomer to a project even to know where to begin" [171].

RQ1.6: ***Are there identified limitations or gaps when using context?***

Regarding gaps found, answering RQ1.6, paper A1 mentions that for the historical artifact changes context type, many research tools have been proposed that use historical information, but few tools are available to practicing developers. A4 mentions that duplicate bug reports can occur because of code clones, which can hamper evaluation results. The other articles do not mention gaps.

RQ1.7: ***What are the advantages or disadvantages of this context?***

Regarding RQ1.7, paper A1 mentions as advantages of the historical artifact changes in the fact that task contexts enable developers to be more productive by making it easy to recall the source code associated with a given task and by allowing other tools, such as content assist, to order information based on work performed as part of the task. As for the team and organization activity context, the author of the same article explains that this context enables the correlation of downstream effects with upstream choices and would open new opportunities for feedback to be provided to developers as development is undertaken. As a disadvantage of this type of context, it is mentioned that this context is still unexplored. A2 mentions as an advantage the fact that a task context can be used to support an interaction style with the increased flow that reduces the information shown to a software developer and enables parts of different information spaces to be related automatically. A3 mentions as an advantage the fact that the tool built to work based on context helps to alleviate a common problem, that is, that tasks were allocated more often to experienced developers, while the less experienced developers received fewer tasks to perform. The other papers do not mention the advantages or disadvantages.

RQ1.8: ***How are the context instances mined?***

Regarding RQ1.8, paper A3 presents instances as recommendations of who should resolve a task through a tool. A4 suggests as instances the recommendations of information from bug databases considering queries of contextual information about an issue. Finally, A5 describes as instances of the recommendation of artifacts that should be edited according to the captured context history.

RQ1.9: ***Are there any abstractions?***

For RQ1.9, no papers mention abstractions. A summary of these findings is presented in Tables A.1 and A.2.

4.2.4 Snowballing Search

For the area of software engineering context, the term ”context” is broad and there are variations in the nomenclature in the literature. Therefore, to mitigate this problem, we also implemented snowballing [71] search from the five papers analyzed in the LR. We retrieved seven more papers to analyze according to our defined research questions.

Table 4.5: Articles retrieved from Snowballing Search Literature Review.

ID	Year	Authors	Title
SBA1	2017	M. Gasparic, G. C. Murphy, and F. Ricci [63]	A context model for IDE-based recommendation systems
SBA2	2018	N. C. Bradley, T. Fritz, and R. Holmes [20]	Context-aware Conversational Developer Assistants
SBA3	2003	D. Čubranić and G. C. Murphy [181]	Hipikat: Recommending Pertinent Software Development Artifacts
SBA4	2014	L. Ponzanelli, G. Bavota, M. D. Penta, R. Oliveto, and M. Lanza [136]	Prompter: A Self-Confident Recommender System
SBA5	2007	F. W. Warr and M. P. Robillard [178]	Suade: Topology-Based Searches for Software Investigation
SBA6	2005	R. Holmes and G. C. Murphy [67]	Using Structural Context to Recommend Source Code Examples
SBA7	2006	M. Kersten and G. C. Murphy [77]	Using Task Context to Improve Programmer Productivity

The papers retrieved from the LR were used as a seed for the snowballing literature study. From the seed papers, we performed a backward snowballing search step [71], in that we looked at all their references, going backward in the citation graph. We stopped the process with the first set of collected papers. After we selected the papers, we also collected information regarding the research questions from these papers.

For the backward snowballing (reference search), 87 papers were extracted from the references from the five seed papers. After deleting the duplicates, 81 papers were left. The title and abstract of each paper were read by two researchers, who were looking for papers according to the objectives set in section 4.2.1. After the title and abstract exclusion, 14 papers were selected for a full reading. Duplicate publications about the same solution in different proceedings or transactions were excluded. After fully reading the articles, 7 papers were selected for analysis, according to the same inclusion and exclusion criteria used during the Search String Search step. These papers are presented in Table 4.5. These papers were fully read, and relevant information about each of them was included in the Appendices Chapter A, in Table A.1 and Table A.2. These articles are identified by the ID "SBAx" in these tables.

4.3 Discussion

In this research, we explore the context within the domain of software engineering. Our journey begins with this preliminary study, aiming to lay the foundation for an enhanced understanding of this critical yet multifaceted element. We align ourselves with the sentiments expressed by Prof. Gail Murphy in emphasizing context as a pivotal construct, one that holds the potential to empower developers to tackle uniquely human challenges in software development.

Our primary motivation is rooted in the recognition of the intricate nature of software development. This knowledge-intensive endeavor involves a diverse array of technologies and relies on many information sources, from tutorials to online forums [43]. Furthermore, the human-centric aspect of software development, influenced by individual characteristics, personal interests, and a range of external factors, underscores the immense impact of context on developers' work methods. Therefore, understanding and effectively managing this context can prompt transformative changes in how developers approach their tasks.

Software development context encompasses a vast expanse of information, spanning various formats and sources, including code repositories, natural language com-

munication, and many tools and artifacts. Despite its acknowledged importance, there remains a gap in developing comprehensive models that capture this dynamic context and utilize it to provide context-driven recommendations to developers throughout the project lifecycle. Context, in this context (no pun intended), extends beyond a mere set of variables; it embodies the very essence of the development environment, the dynamic elements at play, and the human processes involved.

By embracing context-aware methodologies, software engineering can stride towards adaptability, enabling systems to respond to changing scenarios flexibly. This adaptability not only enhances the effectiveness and relevance of software but also equips it to navigate the complexities posed by the ever-evolving and diverse contexts. In essence, our research recognizes context as an intrinsic and influential aspect of empirical software engineering, one that permeates software practices and research, and which brings complexities to developing software. Our study sets the stage for a more context-aware and adaptable software development landscape.

The findings from our LR revealed a diverse landscape of context types, ranging from project task context to static software structure and historical artifact changes. Notably, these contexts served specific purposes, such as improving developer productivity or aiding in task allocation. However, we observed that there was a notable absence of discussions on model specification techniques for context. Moreover, our analysis shed light on the phases of software development where context could be applied effectively, such as during task allocation or bug resolution. Some papers in our review conducted evaluations, which generally yielded positive outcomes, while others highlighted limitations, like the scarcity of tools for certain context types.

In our LR, the first round of articles returned by the search had articles from 1977 to 2019. However, the selected articles ranged from 2003 to 2019. The majority of the articles use context as the information around project tasks and the information about the tasks. One article also considers the information from errors raised during software development as contexts. All the instances of the contexts mentioned were aimed at recommending information to software developers. The advantages and disadvantages identified in the papers analyzed were very specific to each proposal. We could also note that the contexts of the articles analyzed are within different steps of the software development cycle, although using the same information (e.g., project task context), from which it can be inferred that a platform can be built for more than one step of the software development exploring multiple purposes of the same context information. Running a search using the same string on the Scopus database, there are 2547 papers listed between 2019 and November 2023. This demonstrates how much interest the topic has gained in the research community, and that updating this LR is a great open

research opportunity.

This research also resulted in a proposed adaptive context-augmented framework for software development. This proposal (presented in Appendix B), addresses a fundamental need within the software development landscape: the ability to adapt to the dynamic and multifaceted nature of context. This framework's premise, rooted in the recognition of context as mutable, aligns with the findings from the LR, which highlighted the diverse dimensions of context, including environment, people, domain, and technology. Indeed, software systems that can respond to these contextual variations hold a distinct advantage, as they can better cater to the unique demands of each project, and address the complexity of dealing with the multitude of contexts in SD. The framework's modules - the software development project, a reconfigurable context model, and an adaptive engine - form a promising foundation for context-aware software development. By providing a flexible context model, the framework acknowledges the varying context types identified in the LR. This adaptability is crucial, as it enables the model to evolve in response to different project contexts, from MVP-focused to mature software projects. The extended Context Model, as illustrated in the Appendix, serves as a practical example of how the framework can effectively capture and integrate contextual attributes, including project identification, team expertise, technology stack, and more. This proposed framework has the potential to enhance software development practices by empowering developers with context-specific knowledge and recommendations. It represents a forward-looking approach that recognizes the intricate interplay between context and software development, aligning with the evolving demands of the field. However, it is essential to consider the challenges of implementation, such as data collection and model adaptation, as well as the need for ongoing refinement to ensure the framework's effectiveness in real-world development scenarios.

4.4 Threats to Validity

We present a candid analysis of the LR, describing the threats to the validity. The following potential threats should be considered:

Publication Bias: The LR may be influenced by publication bias as it only includes articles published up to 2019. More recent articles on the topic might have been published after this date, and their exclusion could limit the comprehensiveness of the review. This section of this thesis reports the LR that started the Ph.D. work in 2019.

Selection and Sampling Bias: The choice to focus on articles that use context for

recommending information to software developers might overlook studies that explore the context for other purposes or in different contexts within software engineering.

Limited Scope: The LR primarily focuses on context as information around project tasks and the tasks themselves, which could neglect other valuable aspects of context within software development, potentially leading to a narrow perspective.

Potential Overlooked Studies: Despite the comprehensive analysis of the selected articles, there remains a possibility that some relevant studies discussing contexts in software engineering were inadvertently overlooked during the search and selection process. Conducting a more exhaustive search and ensuring unbiased selection criteria can help provide a more comprehensive understanding of the topic.

4.5 Conclusion

The purpose of the current study is to determine how the software development context is presented in the literature. We also propose a preliminary context model that can serve as a foundation to support the identified context and its possible variations. These are the first steps to a solution that explicitly considers software development context to provide in-depth contextual knowledge to software projects throughout a project life cycle. Prior studies have noted the importance of the presence or absence of context information, and how it can influence recommendations during software development [108]. The proposed framework should then be able to be adapted to the contextual factors that are available and provide recommendations accordingly.

We presented the LR performed to understand contexts in software engineering, how they are used, in which phase of the software development life cycle, and other characteristics. Understanding context is relevant to our thesis, as we aim to study ways to integrate this context into software development. Thus, developers can receive support from this context, and be more aware of their dynamic environment, aiding their cognitive load and the complexity of developing software in our current scenario. Moreover, knowing these contexts can help us to implement tools that consider this information to support the abilities of software developers when working.

Results show that various types or classifications of context exist, including project task context, static software structure, dynamic system execution, historical artifact changes, developer activity, and team and organization activity. The contexts proposed in the papers found in the literature search serve specific goals, such as providing developers with static source code artifacts, offering dynamic execution information, access-

ing historical information about static artifacts, understanding how developers work, and treating activities across a value stream as context. The use of context in software development phases is mentioned in some papers. For instance, it is recommended for planning task allocation, focusing on the occurrence of bugs, or being manipulated during coding or when bugs occur. Some papers conducted evaluations of their proposed context-aware tools or models, with varying degrees of success and effectiveness. A few papers identified limitations or gaps, such as the limited availability of tools for historical artifact changes, the potential for duplicate bug reports due to code clones, and the relatively unexplored nature of team and organization activity context. Advantages of context include improved developer productivity, better allocation of tasks to developers, and enhanced support for specific interaction styles. Disadvantages are not explicitly mentioned in the reviewed papers. Context instances are mined as recommendations for task resolution, information retrieval from bug databases, and artifact editing recommendations based on captured context history. None of the papers mention the use of model specification techniques for the proposed contexts, as well as none mention abstractions in the context they propose.

These findings provide insights into the diverse nature and potential applications of context in software development, as well as the challenges and advantages associated with its use. Further research and development in this area could help address the identified limitations and gaps, ultimately enhancing software development processes. We discuss our considerations for providing this support to developers through a conversational agent in Appendix E, where we propose a reflection on the open questions to realize a paradigm where developers receive contextual support during development through recommendations and using a conversational agent as the main tool in this approach. From this reflection, we started to consider conversational agents as the aid to assist developers with context, which led us to our next set of studies that compose this thesis and that are presented in the next Chapter.

Chapter 5

User Studies to Inform the Design of Human-Chatbot Interactions

"We encounter the deep questions of design when we recognize that in designing tools we are designing ways of being." – Terry Winograd and Fernando Flores

5.1 Introduction

Developing software is a challenging task. For example, developers tipically work on multiple projects, comply with company processes, demonstrate technical knowledge and soft skills, attend meetings, and train newcomers. In this very dynamic and rich environment [20, 110], proposing solutions to support software development has been a topic of increasing interest both in research and industry, as developers often search and require more knowledge than they have at their immediate disposal [89]. Moreover, as companies become more technology-centric[1], facilitating the job of software developers is becoming a major issue.

Many different efforts to support software developers in their work have been investigated, ranging from adopting new processes and complex tools to complying with domain-specific processes. Approaches have been proposed to support developers in multiple ways, including finding artifacts [151], aiding software maintenance [154] and automating tasks [91]. One of the big challenges developers face is remembering the

[1] https://www.wsj.com/articles/every-company-is-now-a-tech-company-1543901207

details of what is to be done, and dealing with the many context changes developers endure daily [110]. Solutions to support developers in some of the tasks they must execute are thought to be beneficial. A developer does not need to rely only on memory or page through large volumes of documentation, thus aiding the software development process and potentially resulting in more productivity. The automation of many of these tasks has been pursued, although this often leads to other challenges [20]. For example, it is hard to determine what is to be executed when given a certain process context. Moreover, developers would still have to remember the existence and operation of specific scripts to automate their tasks.

In SE, some activities are hard to automate fully, such as selecting the best code to reuse from the many threads on Stack Overflow, managing exactly which files to commit while creating a version of the system and commenting on branches. Research shows that having the human-in-the-loop can be beneficial in software engineering [108] when curating external documentation to support software development. When considering all these automation challenges, text-based Conversational Agents (CAs) or chatbots are one tool that could mitigate some of these challenges. The conversation with the chatbot happens using natural language, thus excluding the need to learn how to execute commands. Chatbots can be seamlessly integrated into IDEs and customized and integrated with existing tools while keeping histories of conversations and interactions if necessary. Chatbots have already been used as the chosen tools to support software developers in different scenarios, with reported case studies [126, 1, 20]. However, a tool that captures the current context (artifacts, team members, project information) and guides developers in the development process has not yet been proposed. Given the complexities of developing such a tool, and the considerations highlighted in our reflection represented in Appendix E, we proposed a set of studies with software developers, aiming at investigating: (1) The extent to which developers are willing to use a chatbot to support the execution of their tasks, (2) extract requirements from software developers for such tools, and (3) support the design of CAs to assist software development. This investigation provides insights into the perceptions of how developers would be interested in receiving contextual support, and for what within what context. Moreover, this research informs the system design of context-aware conversational tools for software developers, as well as supports the tailoring of systems to developers' preferences.

We conducted a set of exploratory studies to understand the preferences and requirements of software developers when being supported in their work by a text-based CA. Our purpose is to investigate the developer-chatbot interaction in terms of developer preferences and gather requirements to inform the development of such tools. Note that we are not evaluating tools or their features; instead, we are assessing and aim-

ing to capture the interaction aspects and developers' preferences when using a text-based chatbot. We performed the investigation in two parts. In the first part, we used a Wizard-of-Oz methodology, with 5 participants in a classroom environment (Section 5.2). We gathered user study data and perceptions of the participants into the study design. Then, in the second part, we incorporated relevant feedback on the study design and designed a complementary study, this time with 29 participants, all software developers, and used a chatbot prototype. An overview of the study methodology design is presented in Figure 5.1. This figure contributes to the comprehension of this set of studies. In Step 1 (Section 5.2), we start by conducting a pilot study using the Wizard-of-Oz methodology with the support of a scenario. In Step 2, we delineate the data collection and analysis procedures. Moving on to Step 3, we integrate the feedback gathered during the pilot study into the main study's design (Section 5.3), concurrently developing a chatbot prototype for deployment in this investigation. In Step 4, data is gathered through interviews, questionnaires, and participants' interactions with the chatbot. Finally, in Step 5, we analyze the study's outcomes using both quantitative and qualitative analyses.

After running the pilot study and getting feedback from participants, we performed exploratory user investigation using a developed and deployed chatbot prototype – DevBot (see Figure 5.2).

By interacting with DevBot, developers had the experience of interacting with a real tool and reporting their experience through questionnaires and interviews. Spoiler alert: results indicate developers are interested in having chatbot tools to support the many aspects of the software development pipeline, managing their tasks and supporting version control.

Our specific goals with this study are to:

- Collect and reflect on the perceptions and experiences of software developers concerning interacting with chatbots to support their work during software development
- Report the perceived opportunities and challenges described by developers of using chatbots
- Gather requirements for the development of CAs in SE
- Identify the potential areas for enhancement and improvement of chatbots as a possible tool to connect the software developer to any other environment in a

Figure 5.1: User studies methodology.

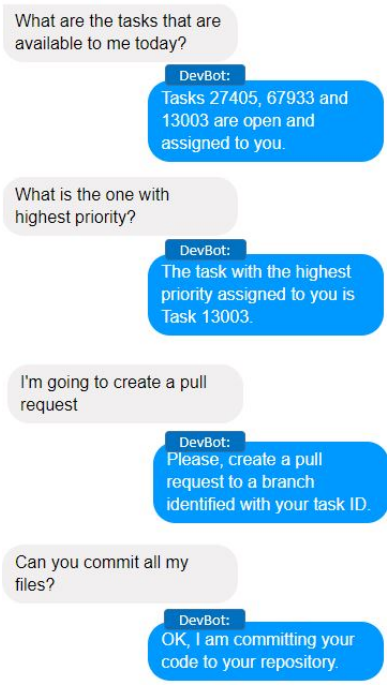

Figure 5.2: Chat with the prototype DevBot.

natural way, i.e., without the need to learn how to run scripts or search for documentation

- Provide a qualitative study design addressing chatbots as a way to support developers in their work
- Present a discussion of the study results, highlighting the potential methods to improve the user experience of developers during software development

This set of studies does not evaluate the tool DevBot itself. Rather, it uses this prototype to gather requirements and participants' views on this type of tool (chatbots) for work and to provide insights on how developers ask work-related questions of chatbots. This study methodology:

- Describes the design and results of a user study that aims at collecting information from the experience of developers with chatbots that support them during software development

- Describes the design and deployment of a chatbot prototype
- Identifies the areas that need to be addressed during chatbot design and implementation to support software developers (requirements for chatbots to support software development)
- Reports the perceptions of developers about their experience with interacting with a chatbot for software development
- Analyzes the questions developers asked the chatbot and draws conclusions on how developers interact with such tools
- Discusses the perception of developers on the controllability of their work in terms of levels of automation (automated execution vs. manual execution)
- And proposes a conceptual design model based on the reports provided by participants in the study

  Next, we describe the studies' methodology designs and results in detail.

5.2 Pilot Study

The goal of our experiment is to understand how software developers would use a chatbot to facilitate daily software engineering tasks and processes. For that, we aim to provide a chatbot interface to developers and use an example scenario of what developers might perform in one day and the related questions. Based on this scenario, developers would ask questions to the chatbot. We would then capture and analyze the questions developers asked the chatbot (utterances) and interview the participants. In effect, our goal is to investigate how are developers willing to use chatbots at work and for what types of tasks. We intend to determine if there is one part of the scenario participants ask more or fewer questions about (interest) and the types of words and questions used (vocabulary). We begin by experimenting with five graduate students as participants and the Wizard-of-Oz methodology. First, we give participants a scenario of a day in the life of a software developer. The scenario aims at delimiting the scope of questions that could be asked. Then, we ask participants to write questions regarding any step of this scenario to the chatbot. Next, we present details of the study design.

5.2.1 Procedure Setup

In this study, we use a mixed-method approach, combining Wizard-of-Oz (WOZ) [65, 75] and a structured questionnaire. We decided to perform the pilot study using WOZ to verify user engagement and understanding of the study before investing time in developing a more complex study. This pilot is organized in three different steps.

Step 1: Participants fill out a questionnaire (Google Forms) to collect demographic data (current degree, program, length of experience with software development, knowledge regarding software repositories such as Git and Jira), and were informed that they would be reading a scenario and asking any questions they could think of to the chatbot. When they click "Next" in Google Forms, they see the scenario text. The scenario can be read at this moment or at any point while interacting with the chatbot.

Step 2: We use Slack as our interface for the WOZ study. After understanding the scenario, participants were supposed to start writing their questions on Slack, to the user called DevBot. As this was a WOZ study, DevBot was the author of this thesis, answering participants according to the preconfigured answers in Table 5.1[2]. When participants ask the chatbot to execute tasks repetitively, the chatbot responds with "Done." and utters the answer configured in "No answer", to stimulate the participant to move forward with other questions.

Step 3: After interacting for about 10 minutes with the DevBot slack bot, participants were presented with a new questionnaire. This new questionnaire aims to gather participants' opinions regarding their experience with the chatbot. Some questions about participants' experiences with the Chatbot followed a Likert scale, while others were open-ended questions. Below is the list of questions asked after the interaction with the chatbot, and their reasoning.

- Do you think that the chatbot was helpful for the questions you asked? (0 = Helped zero tasks to 5 = Helped in all tasks): We aim to investigate the perception of participants regarding the usefulness of the questions asked during the study.
- Do you think the chatbot was useful for the presented scenario? (0 = Not useful to 5 = Very useful): We aim to investigate the perception of participants regarding the usefulness of the simple answers planned to be answered during the study.

[2] "What can the bot do" answer was presented either when the participant asked explicitly what the bot could do, or after the third time the bot had to fallback.

- Speed of answer perception? (0= Very slow to 5 = Very fast): We aim to investigate how the users perceive the speed of response of the bot.
- What other steps do you think this chatbot could cover? (Open question): We aim to investigate the participant's suggestions of topics that can be covered by such chatbots.
- What did you like about the chatbot? (Open question): We aim to investigate the positive aspects of the chatbot interaction with the participants.
- What can be improved in the chatbot? (Open question): We aim to gather constructive criticism from participants.
- Do you think this solution adds value? Why / why not? (Open question): We aim to investigate if participants think the idea of having software development supported by a smart chatbot is interesting and worthwhile.
- Do you have any general comments? (Open question): We aim to investigate if participants have other general comments regarding the interaction with the chatbot.

None of the participants should know which questions the bot is trained to answer, to guarantee they use their vocabulary when interacting with the bot. However, during the study, participants asked what the bot could do, or when they asked many questions the bot did not know how to answer (questions out of the scope of the scenario), they were given a list of possibilities based on the current scope of the scenario.

5.2.2 Scenario

Once demographic data collection was completed, participants were presented with a scenario. This scenario describes a typical day in the life of a software developer. This scenario was inspired by the study of Meyer et al. [111], and validated by two experienced software developers. The goal of having this scenario is so that participants have an idea of the scope of the questions they are supposed to ask. They could also ask questions that were out of the scope of the scenario, and although the chatbot was not ready to answer these, we also analyzed and computed the topics that arose that were not covered by the scenario. The scenario is presented next.

***Scenario**: You arrive at your office at 8 a.m., sit on your chair and open the task manager to check what tasks are assigned to you. You look for the one task with the highest priority and read the text of the task. You understand what you are supposed to do for this task. You have a stand-up meeting and redefine task priority. You go to the cafeteria and grab a coffee. You're back to your desk and open Eclipse. You sync your local code repository to get the newest version of code for that specific task. You look for the artifacts that you might have to edit to complete this task. You ask your colleague what he thinks about the task and the artifacts you decided to edit. You start coding the test case for that new implementation and then edit the .java class that is supposed to receive the edits. You get another coffee. You save your edits, commit your code and create a pull request, inserting a comment in the pull request. It's 4 pm. You update the task assigned to you to status Done and with the time you invested in solving the task. You leave the office.*

After reading and understanding the given scenario, participants were invited to interact with the chatbot. The chatbot operator (Wizard-of-Oz) was prepared to answer a few questions related to the scenario and handle greetings and exceptions. When participants asked questions related to certain topics in the scenario, the chatbot operator would answer according to the answers presented in Table 5.1.

5.2.3 Participants

For the pilot study, the participants recruited were students from the Human-Computer Interaction and Conversational Agents Winter 2020 course (David R. Cheriton School of Computer Science, University of Waterloo). Five Computer Science students participated, four master's students and one undergraduate. In their experience with software development, only one participant reported having less than one year of experience, as three participants have between one and five years, and one participant had more than five years of experience. Participants reported either some knowledge of software repositories (60%) or good knowledge (40%).

5.2.4 Study Results

After being presented with the scenario and completing the first part of the questionnaire (demographics), participants were supposed to interact with the chatbot. The answers were provided according to keywords (Table 5.1) in the participants' questions, and if no expected keyword was present, a default response (Sorry, I can't help you with

Table 5.1: Answers that Wizard-of-Oz was prepared to respond, according to the context of the question asked.

Greetings	
Greet	Hey! What do you need to know regarding your scenario today?
Goodbye	Bye!!
Scenario	
Artifacts	You need to edit class client.java and class diagram OrderMain.
Commit	OK, I am committing your code to your repository.
Tasks	Tasks 27405, 67933 and 13003 are open and assigned to you.
Priority	The task with the highest priority assigned to you is Task 13003.
Pull Request	Please, create a pull request to a branch identified with your task ID.
Code Push	Please, download or clone your repository to your local machine. Keep it synchronized.
Sync	You should synchronize your repo. Click on the sync repo button on your version control page.
Tool	You have to open Eclipse or VSCode and synchronize your code. After, assign a test task to the test team.
Exceptions	
No answer	Did that help you? What else can I help you with?
What can the bot do	I can help you discover which tasks are assigned to you, their priorities, how to do a pull request and you can ask me to commit your code.
Not on the list	Sorry, I can't help you with that. Is there anything else I can help with?

that. Is there anything else I can help with?) was presented. Participants asked an average of 15 questions. Based on the analysis of the questions, we highlight the following interactions from each participant in Table 5.2. The "x" in the table represents that this participant, in particular, has asked questions regarding what the experiment was ready to answer. The table also shows that all participants asked unexpected questions.

Table 5.2: Mapping of expected interactions for each participant.

Expected Intention	**P1**	**P2**	**P3**	**P4**	**P5**
Greet	x	x	x	x	x
Goodbye	x	x	x	x	x
Artifacts	x	x		x	x
Commit	x	x		x	x
Priority	x	x	x	x	x
Pull Request		x		x	x
Code Push					
Sync		x			x
Tasks	x	x	x	x	x
Tool	x		x		
Others/fallback	x	x	x	x	x
What can the bot do	x	x			x

We have also extracted entities through semantic analysis of participants' questions using the KH Coder Tool, filtering out stop words[3]. We extracted the word frequency of the questions, and the results of the top 10 words are presented in Table 5.3. We have also analyzed the question in clusters, using the Jaccard Distance. The Jaccard Distance is a measure of dissimilarity or distance between two sets. It is used to quantify the dissimilarity between two sets by measuring the proportion of elements that are different between them. In the context of clustering or similarity analysis, it's often used to determine how similar or dissimilar two sets are based on their elements. The Jaccard Distance is calculated using the formula: Jaccard Distance = (Number of Elements in Both Sets) / (Total Number of Unique Elements in Both Sets). The clusters created relate to the questions of task (18 documents), pull request (9 documents), code (9 documents), greeting (3 documents) and others (40 documents). Nine documents were not included in any cluster.

Word frequency and Jaccard distance are two distinct measures that can be used to

[3] https://gist.github.com/sebleier/554280

Table 5.3: Word Frequency in Participants' Questions, using KH Coder.

Word	Frequency	Word	Frequency
task	20	need	7
pull	9	artifact	5
request	9	code	5
DevBot	8	commit	5
help	8	create	5

find clusters of words, and when used together, they can offer a more comprehensive approach to identifying meaningful word groups. They complement each other because word frequency refers to how often a specific word appears in a text or dataset. It provides information about the relative importance or prominence of words within the context of the dataset. By analyzing word frequency, you can identify common words that are frequently used and may carry essential semantic meaning or contextual relevance. This approach is particularly useful for identifying frequently occurring words that might indicate thematic focus or topics within the data. Jaccard distance measures the dissimilarity between sets based on the proportion of differing elements. In the context of text analysis, sets can represent word occurrences within documents. When applied to text, the Jaccard distance can help identify how similar or dissimilar documents are in terms of their word content. It is effective for capturing the overlap or shared vocabulary between documents. When used together, word frequency and Jaccard distance can provide a more nuanced and comprehensive understanding of word clusters.

In summary, on the one hand, word frequency helps identify words that occur frequently, which might be common across documents or specific to certain subsets. These frequently occurring words can serve as anchor words that contribute to clustering. On the other hand, the Jaccard distance, when applied to sets of words within documents, can capture the similarity of word usage between documents. This can help detect contextual relevance and identify documents that share similar themes or content.

After interacting with the chatbot, participants were asked a few questions regarding their experience. Most participants reported having helpful answers to the questions they asked (4 out of 5), and useful tools to answer questions related to the scenario (4 out of 5).

Participants have offered valuable insights into enhancing the functionality of the chatbot. They have emphasized the importance of incorporating best practices for coding, which involves giving developers detailed explanations of each task and providing

step-by-step guidance on how to perform them. This approach empowers developers to execute tasks themselves, fostering their growth and understanding.

Participants have also highlighted the potential of the chatbot to aid developers in various ways. This includes suggesting similar tasks to a given one, locating experts for specific tasks, and facilitating the comparison of file versions across multiple historical commits. Furthermore, participants have emphasized the significance of context awareness for the chatbot. According to participants, being able to comprehend the current repository and utilize pull request comment templates can enhance the chatbot's effectiveness. This context-sensitive approach ensures that the chatbot's interactions are tailored to the specific project, increasing its value and usability.

Positive Feedback on the Chatbot: Participants praised the chatbot for its ability to provide clarity about its usage. It was highlighted that the chatbot was particularly beneficial for scheduling and day planning, as mentioned by one participant: *"helpful in terms of finding my schedule and how my day looks like so I can plan ahead"*. Another participant expressed that DevBot holds promise as a valuable resource for those less familiar with software development workflows.

Suggestions for Improvement: Participants offered several suggestions for enhancing the chatbot's capabilities, including:

- Handling a broader range of questions
- Introducing its purpose clearly at the outset
- Enabling code analysis and debugging
- Providing coding best practices
- Incorporating more specific details about the developer's ongoing work
- Suggesting similar tasks
- Integrating webpages for crowd-sourced knowledge support
- Enhancing its ability to provide alternative steps or suggestions when unable to assist

Notably, one participant provided an insightful suggestion: *"The leading prompts were helpful but it would have been better if the DevBot could lead with the tasks it can complete and how a user can interact with it. Perhaps it could have different modes depending on what git*

tools, and IDEs a user is working with (i.e., Eclipse, VScode, vim XD) that could give specific advice for each type of software. A user could select the level of help they require, ie. if a user is brand new, maybe they'd prefer lots of prompts and extra resources for reading about how version control works etc. for an advanced user, maybe they'd prefer just the tasks and a reminder of the workflow..."

Perceived Value and General Comments: Regarding the chatbot's value, the majority of participants acknowledged its worth and stated that solutions like DevBot could significantly benefit software developers. Some participants envisioned integrating the chatbot as a personal assistant within software development workflows, along with suggesting the incorporation of a more distinct personality to the chatbot's interactions.

5.2.5 Discussion

The results of the study provide valuable insights into participants' perceptions and interactions with the chatbot, shedding light on several key points. The discussions on these findings reveal significant implications for chatbot design, user expectations, and the potential to enhance developer productivity.

One noticeable trend in the participant interactions was the predominant focus on tasks and development steps rather than task execution. This observation underscores that developers are more interested in receiving guidance and information about tasks rather than seeking a chatbot that performs tasks on their behalf. This discrepancy between the expectations of chatbot designers and users emphasizes the need for chatbot functionality that aligns with developers' actual requirements and preferences.

Contextual understanding emerged as a prominent theme from participants' feedback. Many participants expressed interest in the chatbot having context awareness, including knowledge about repository details, branches, and commit message patterns. This finding suggests that the inclusion of contextual information could greatly enhance the chatbot's utility and personalization. The concept of incorporating context discovery and recommendation mechanisms based on interaction history emerges as a potential avenue for future chatbot development.

A noteworthy discovery pertains to the lower engagement score provided by a less experienced developer (P1) in terms of finding the chatbot helpful for their questions. This potentially indicates that the chatbot may need to provide more detailed explanations to cater to less experienced users. Conversely, participants with more than one year of experience generally found the chatbot to be helpful, underscoring the potential for this tool to benefit developers with varying levels of expertise.

The varying nature of the participants' questions is also intriguing. Participants who posed questions about the tool itself, rather than pull requests, might suggest a relative lack of experience with version control repositories and software development processes. The study's data also highlights that participants interacted with the chatbot efficiently, with an average of 15 questions asked in around 10.8 minutes of active chat time. This insight offers valuable information for designing chatbot interfaces that accommodate quick and productive interactions. Moreover, the interactions participants had with the chatbot also included greetings at the start and end of the conversation. Additionally, it is interesting to note that one participant identified the experiment as a Wizard-of-Oz scenario, suggesting that a chatbot prototype specifically for the goal of this study should be designed to handle interactions seamlessly and that users may not always recognize the underlying experimental setup, but some users might. Therefore, an experiment with a more realistic design could be advantageous to avoid bias. Furthermore, the high frequency of questions related to pull requests, a common aspect of software development among the more experienced developers, indicates that developers are more concerned with tasks directly linked to collaborative development.

In summary, the study's outcomes underscore the importance of aligning chatbot functionality with users' actual needs and preferences. Incorporating contextual awareness, enhancing explanations for less experienced users, and focusing on tasks relevant to collaborative development appear to be pivotal considerations for designing effective and beneficial chatbots for software development support. Having these results at hand, and incorporating the feedback received from participants, we decided to expand this pilot study.

5.3 Main Study Design

Following the pilot study, and having incorporated the feedback gathered in the pilot, we worked on expanding the investigation. To capture and understand the characteristics of the interaction of software developers with chatbots, we created a prototype to support the user study. We aim to provide a simple chatbot interface to developers and use a scenario (the same scenario used in the pilot study) to support the scope of the study. Then, developers would ask the chatbot any questions that they would see fit for the chatbot prototype, and we interviewed participants to collect information on their experience. We would then analyze these questions, as well as the questionnaire answers and the interviews. In effect, we want to investigate five topics, namely (1) if developers are given a chatbot are they willing to use it to support their daily work?;

(2) what types of tasks should this chatbot support?; (3) what types of questions should chatbots be able to answer?; (4) are there unexpected questions/requests that were not anticipated?; and finally (5) the overall opinion of the developers regarding the use of a chatbot.

We conducted this experiment with 29 software developers. We provided participants with the scenario, which comprised a description of what a common day in the life of a software developer is. Then, we asked participants to write questions regarding any step of this scenario to Devbot (chatbot prototype), or any other question the developer could think of. Here, we intend to determine if there is one part of the scenario participants ask more or fewer questions about (interest), the types of words and questions used (vocabulary), and if there are any other areas of interest, not necessarily portrayed in the scenario. Moreover, the experiment also aims to identify the perception of developers when a chatbot knows the context in which they are working. Since this study involves human participants, it has been reviewed and received ethics clearance through the University of Waterloo Research Ethics Committee (ORE#42126). Each of the resources used in the study (call for participation, information letter and feedback form) is in Appendix D. The Google Forms used can be accessed in https://forms.gle/GsscMeWzub8zWqVi8. The prototype implementation details can be found in Appendix C.

5.3.1 Prototype Design

To gain insights into the preferences of software developers, we developed a CA prototype named DevBot. In its initial stage, DevBot is designed to interact with developers through one-on-one text-based chats, responding to inquiries related to software development. The structure of these interactions is outlined in the aforementioned scenario (Section 5.2.2), and the questions the chatbot is ready to answer (the utterances it recognizes) were also configured based on the question structure we have in the pilot study. These questions are demonstrated in Table 5.1.

Our implementation of DevBot is built upon the open-source Rasa chatbot platform (Rasa.com). Rasa offers a platform for users to create and deploy chatbots tailored to specific tasks. Rasa presents an architecture that is designed for scalability and adaptability. Refer to Figure 5.3 for an overarching view of the Rasa architecture. The architecture is centred around two primary elements: NLU and dialogue management. The NLU component takes charge of tasks like classifying intentions, extracting entities, and retrieving appropriate responses. This is depicted as the NLU Pipeline, as it employs an

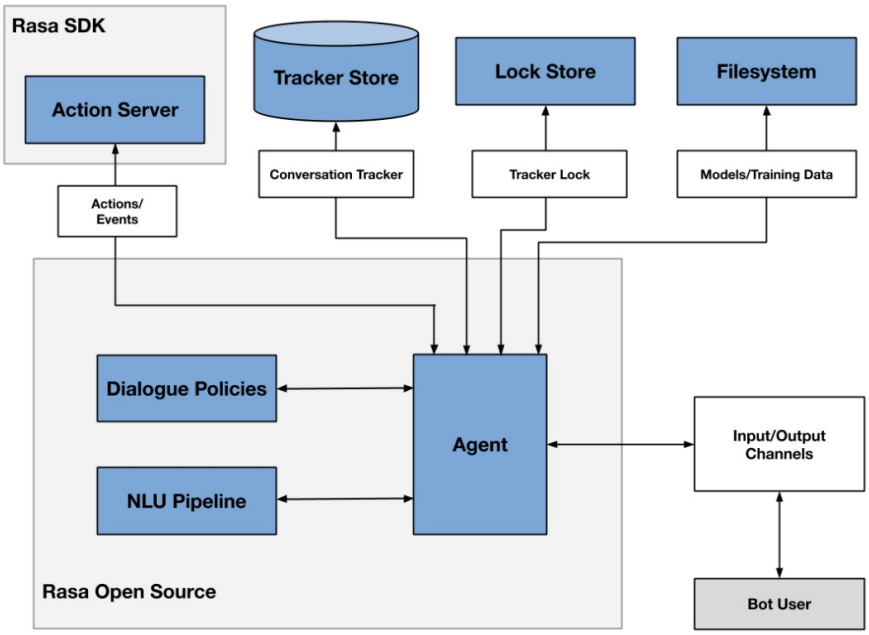

Figure 5.3: Rasa Architecture. https://rasa.com/docs/rasa/arch-overview

NLU model generated by the trained pipeline to process user inputs.

On the other hand, the dialogue management component plays a pivotal role in determining the subsequent action within a conversation, based on the ongoing context. This functionality is illustrated as Dialogue Policies in the diagram, signifying its role in steering the conversation flow. We opted to use Rasa for several reasons. First, its open-source nature fosters a community, coupled with comprehensive online documentation. Second, it seamlessly integrates with popular platforms such as Facebook, Telegram, and Slack. This choice aligns with our goal of deploying the chatbot on a familiar platform to users, ensuring a smooth and natural experience. This way, users can access the chatbot through platforms they already commonly use.

Our design centers around utilizing Rasa to govern the CA's functions, such as question recognition and response generation. We deployed DevBot on Facebook (developers.facebook.com), requiring participants to access the chatbot through Facebook Messenger. Rasa allows the seamless deployment of the same chatbot in different messengers, such as Whatsapp, Telegram, Slack and others. This deployment was facilitated by running the chatbot on a server and leveraging the RASA webhook to manage conversation processing on the server, thereby delivering responses through the Facebook app.

A screenshot of a real conversation is available in Figure 5.2.

Participants initiated the interaction by visiting facebook.com/agentdevbot and clicking "Send Message." This action triggered the Facebook messenger callback, which was redirected to the deployed chatbot server address. To expose the local server to the internet, we used ngrok[4], an API that allows us to expose to the internet any service deployed in a local host. The complete set of DevBot files can be accessed in this repository github.com/glauciams/devbot, and the files are also appended to this thesis, in Appendix C.

5.3.2 Participants

Participants were recruited through university graduate student mailing lists, social media posts and participant databases. The interested participants would be screened according to the participation requirements such as "be a software developer for over one year" and would receive a remuneration of CAD 15 after the completion of their participation. We collected the following demographic information from the participants:

1. Gender
2. Age
3. Highest degree or level of school completed
4. Experience with software development (in years)
5. Knowledge regarding software repositories (Git/Jira)
6. Have you interacted with CAs before and which ones?
7. How interested are you in CAs?

We recruited participants with an open call, using social media and email to the graduate students mailing list of the University of Waterloo. 31 participants responded to our call, but only 29 participants could be part of the study. The two participants were excluded as they did not have enough software development experience as specified in the study design (at least 1 year). We stopped recruiting participants once our research had reached a level of saturation, i.e., no new information was being added.

[4] ngrok.com

Demographic data was collected from all participants while maintaining anonymity. Out of the 29 software developers interviewed, 26 reported previous interactions with text-based or voice-based chatbots. Only one participant lacked familiarity with task management tools like Jira or Trello. The interview pool included 4 self-declared females and 25 males, aged between 22 and 41 years. Zoom was the platform used for all interviews conducted between September 2020 and February 2021. Participants were enlisted through an open call distributed via email and social media, and they were chosen based on possessing a minimum of 1-year experience in software development within the industry.

5.3.3 Procedure Setup

As participants connected in the Zoom call, they were presented with the demographics form. After filling out the demographics, they were presented with the scenario, which consists of "a day in the life of a software developer". This was the same scenario used in the pilot study. To aid in reading this thesis, we are presenting the scenario again.

***Scenario**: You arrive at your office at 8 a.m., sit on your chair and open the task manager to check what tasks are assigned to you. You look for the one task with the highest priority and read the text of the task. You understand what you are supposed to do for this task. You have a stand-up meeting and redefine task priority. You go to the cafeteria and grab a coffee. You're back to your desk and open Eclipse. You sync your local code repository to get the newest version of code for that specific task. You look for the artifacts that you might have to edit to complete this task. You ask your colleague what he thinks about the task and the artifacts you decided to edit. You start coding the test case for that new implementation and then edit the .java class that is supposed to receive the edits. You get another coffee. You save your edits, commit your code and create a pull request, inserting a comment in the pull request. It's 4 pm. You update the task assigned to you to status Done and with the time you invested in solving the task. You leave the office.*

After reading the scenario, participants were directed to the prototype DevBot, deployed on the Facebook Messenger platform. Participants were instructed to interact with the participants for about 10 minutes, or for as long as they had questions to ask. We ask each participant the following question: "What questions would you like to ask DevBot, that would help you go about your day?"

Then, after interacting for about 10 minutes (or for as long as they could think of questions), they were presented with a post-study questionnaire, which comprised of questions to gather participants' opinions on the interaction. The questions about the

participant's experience with the chatbot prototype follow a Likert scale from one to five or are open-ended questions. Likert scales ranged from 1-5, where 1 meant "Not at all" and 5 "Very Much".

1. How much did you like interacting with DevBot?, Likert scale and open text field to explain the answer.
2. How much would you be interested in using DevBot in your company?, Likert scale and open text field to explain the answer.
3. Do you think that chatbot was helpful for the questions you asked?, Likert scale.
4. In general, did the bot answer what you were expecting?, Likert scale.
5. Do you think the chatbot was useful for the presented scenario?, Likert scale.
6. What was your perception of the time DevBot took to answer your questions?, Likert scale.
7. What other steps not covered in the scenario do you think the chatbot could be useful for?, Open text field.
8. What did you like about the chatbot?, Open text field.
9. What can be improved in the chatbot? Is there anything you would change about DevBot?, Open text field.
10. Do you think this solution adds value to software development? Why / Why not?, Open text field.
11. Do you have any general comments?, Open text field

Finally, when participants completed the questionnaire, we conducted a semi-structured interview where we again asked about their experience with DevBot. The question asked of all participants was: "Tell me how your interaction was, and what did you think about DevBot". Extra clarifying questions could arise depending on participants' answers. This last part aimed to capture opinions and takeaways from participants that they might not have written in the questionnaire. We asked participants to turn off their cameras while recording the interviews on Zoom and kept the audio file only, also deleting the video file as an extra step to guarantee the participants' privacy.

None of the participants should know which questions the bot is trained to answer, to guarantee they use their vocabulary when interacting with the bot. However, during the study, if participants asked what the bot could do, or when they were asked many questions the bot did not know how to answer, DevBot was ready to answer with a list of possibilities based on the current scenario. Every participant was presented with the same scenario. The next section describes the study results.

5.4 Study Results

Participants asked a total of 619 questions (or issued commands) to DevBot, with an average of 21.3 questions per participant. Each interaction averaged 12 minutes, and the total time for interaction was 5:56 hours. We collected all questions and transcribed the interviews using Whisper https://openai.com/blog/whisper/. Whisper is a pre-trained model for automatic speech recognition (ASR) and speech translation. Trained on 680k hours of labelled data, Whisper models demonstrate a strong ability to generalize to many datasets and domains without the need for fine-tuning.

5.4.1 Questionnaire Data Analysis

In this Section, we present the results extracted from the analysis of the data of the post-study questionnaire.

Experience interacting with DevBot: Most of the respondents claimed they felt neutral (Likert scale = 3) when asked about how much they liked interacting with DevBot. Only 4 participants (out of 29) claimed to have enjoyed interacting with the tool (Likert scale = 4). In the questionnaire, participants questioned the ability of the chatbot to reply to the questions they had asked, as P1 mentioned *"The Devbot was quick in responding but was not able to help me with correct answers to the questions that I had. Had it been able to answer my questions, it would have been more fun."* . Participants claimed DevBot was not helpful for some of the specific questions asked, and pointed out that understanding the purpose of the chatbot would result in a more valuable experience overall. One participant mentioned *"If it can have better intelligence, I'm definitely interested in using it"*. Another participant claimed *".. I did like the ability for the chatbot to improve the usability of some repetitive tasks mainly: pushing code, committing code, creating branches, pulling/updating code, and creating PRs."*.

Willingness to use DevBot while at work: When asked about how interested participants were in using DevBot for work, the answers were mixed. 11 participants selected options 4 or 5 on the Likert scale, while all others selected options 1 to 3. As one participant points out *"... it would be very nice to use, given a bit more access to task details"*. Another participant expressed the need to improve the capabilities of the prototype. They mentioned *"I think a tool like this would be super helpful in training and onboarding newer developers. I can see it being useful in reminding developers what the software process is like in a company, and it could answer fairly generic questions like branching strategy. For more experienced developers, I guess it could be used as a faster alternative to going on JIRA/their task managing site themselves to look at their tasks."*

Interest to have contextual information incorporated into DevBot: Participants have recognized the importance that using such tools could provide contextual answers once integrated with task management tools and with an enabled history of conversations. *"It would also be helpful to have contextual answers, e.g. 'Start task X', and then when I'm finished I could say 'Set task completed', instead of having to remind the right task. Or even ask it to remind me of the current task I'm working on after context switching, like coming back from a meeting or lunch."*

Identified design opportunities: To analyze this topic, we have listed all possible identified design opportunities that were mentioned by the participants and organized them into keywords. Then, we grouped similar keywords into more general groups and summarized the occurrences. For this question, developers expressed an interest in having a chatbot that can support and leverage git functions (10), and manage their tasks (9) and their schedule (9).

Participants suggested that process-related information would be of most interest to implement in a chatbot. This information is connecting with teammates, and managing their tasks and their schedule. More specific development information should also be part of such solutions, such as auto-complete commands, integration with tools such as git, and supporting and leveraging git functions. One participant mentioned: *"I think the chatbot could be used as a developer's help tool since it clearly seemed to understand what my questions were. For example, if I had issues with merge conflicts or started my IDE, it could respond with common causes or more resources."*

Participants have also pointed out they want to see the capabilities of the chatbot tool beforehand, and they differ in their opinions regarding how much controllability in terms of the level of automation the chatbot should have, and provide suggestions for how features that allow automated executions through the chatbot should run. Other topics mentioned are features already available in tools such as RASA or require mini-

mal configurations, such as keeping conversation history, improving answers, providing positive answers when a topic's solution is unknown and, as mentioned, informing users of the chatbot's capabilities at the beginning of the interaction. Other suggestions require implementation efforts, such as managing tasks, and knowing the environment, such as where the code goes after commits or which test environments are used. Participants also pointed out it is not always desirable to have tools and bots execute tasks on their behalf. Because of the number of comments we received regarding this aspect of controllability, we clarified this issue in the interview after analyzing the study's preliminary results with 8 participants. From this point on, we asked participants their opinions on controllability in the semi-structured interview.

Positive feedback: Many participants mentioned the fast response of the chatbot as a characteristic of the solution they liked (15 occurrences). Participants have mentioned being interested in having features already inherent to text-based chatbot applications, such as asking many questions about one item that was already mentioned (chatbot understanding the history of the conversation) and the speed of answers. Others mentioned being interested in features that would need to be implemented, such as integration with task portals and calendar management tools.

Participants suggested a mix of features they liked that would need to be implemented and features that are already built-in to chatbot solutions such as RASA, which would have minimal complexity in their implementation. Built-in features such as the ability to formulate questions freely and provide fast responses are already inherent to textual CA solutions. Other ideas such as managing schedules, automating repetitive tasks, and giving task details should be implemented and leveraged by the solution's model training and integration abilities. One participant mentioned, *"The whole idea of having an assistant to help developers manage their tasks has great potential."*

Requirements for development of chatbots in SE: Most participants pointed out that the solution adds value and added comments to their answers, some providing examples of where a chatbot like DevBot would help them, others providing requirements that would allow them to use such tools. The topics that arose are summarized and presented next.

- Needs to be extremely well done, otherwise frustrating
- Great way to teach beginners
- Great to integrate fragmented information/tools in SE
- Good to add structure to work, i.e., breaking down tasks

- Great to manage tasks, as *"Jira can get really boring and complicated to navigate"*
- Great if it does the repetitive tasks that programmers have to do
- Great to serve as a team database/communication partner, instead of asking colleagues questions
- Has to be "smart"
- Has to work for the proposed purpose
- Can also have voice commands
- Use in low-level tasks such as *"communicating, managing meetings, fetching to-do lists"*
- Helps to save time when not at work and work-related subjects arise

One participant has mentioned *"I think it does if the bot can answer some questions., I think it is a great way to teach a junior/beginner developer good practices. I remember as a junior developer joining a new team having tons of questions because I really wanted to do things the right way. But I could not ask my colleagues over and over."*

When asked if they had **any general comments**, participants mentioned that the chatbot is a good idea, but it would need improvements to be efficient. One of the participants mentioned that a voice-controlled chatbot might also be interesting to investigate. Another participant mentioned that knowing what the chatbot can do in advance would be advantageous.

5.4.2 Semantic Analysis of Questions

In this Section, we perform an analysis of the questions that participants asked DevBot. We extracted entities through semantic analysis (KH Coder tool) of participants' interactions. We have loaded stopwords from https://gist.github.com/sebleier/554280. For the word frequency of the questions asked by the participants, the results of the top 10 words are presented in Table 5.4.

We analyzed the question in clusters, using Jaccard distance and selecting five clusters. The clusters created relate to the questions of task (24 documents), pull request / merge / branch (17 documents), code (14 documents), classes/java (14 documents),

Table 5.4: Word Frequency in Participants' Interactions, using KH Coder.

Word	Frequency	Word	Frequency
task	46	pull (noun)	12
request	18	need	11
code	17	pull (verb)	8
create	15	git	7
branch	12	priority	7

greeting (3 documents) and others (50 documents). Thirteen documents were not included in any cluster. To analyze the questions participants asked Devbot, we captured the word frequency, which organized the questions into clusters. Before transforming the documents into clusters or running the analysis, we use NLP techniques such as removing stop words that carry little meaning, such as "on" or "at" of every set of questions. Then, we ran an algorithm to collect the frequency of words. We hypothesize that the words with higher frequency can be the most interesting subjects to developers.

5.4.3 Interview Analysis

During the semi-structured interview sessions, we collected valuable insights regarding participants' interactions with DevBot, their expectations, and new suggestions. These inputs were consistent with the feedback previously gathered from the questionnaire. The interviews were transcribed, and participants' remarks were categorized and grouped into several key themes. The following sections outline the main topics of interest, listed in descending order of occurrence:

Software Development Environment Awareness: Participants expressed interest in a tool that could provide insights into their development environment. They indicated a desire for information such as which specific artifact to modify based on a given task or identifying the last person who made changes to a file. Additionally, participants mentioned that knowing colleagues' statuses (e.g., in a meeting or out of the office) and access to documentation used by the team for specific tasks would be highly beneficial.

Process Awareness: Process awareness emerged as another prominent theme. Participants expressed a need for guidance on tasks like branch naming, task status management, and accessing task-related information directly through the chatbot. They appreciated the idea of streamlining these processes without navigating external task portals and dealing with filters.

Chatbot Capabilities: A recurring topic was participants' desire to understand the capabilities of the chatbot before utilizing it. This underscores the importance of clarifying the chatbot's functionalities upfront, a sentiment aligned with participants' survey responses.

Controllability: Participants presented varying opinions regarding the level of control they preferred over the chatbot's actions versus the automation of tasks. From the interview transcripts, it was apparent that while some participants were open to the chatbot executing automated actions, others preferred its role to remain informative and guiding, rather than running scripts automatically. Among the participants who expressed their views on controllability, 9 out of 27 participants favored automation, while 18 out of 27 preferred guidance or automation limited to non-disruptive tasks such as status updates.

Chatbot Built-In Features: Participants indicated a desire for features inherent to chatbot tools. For instance, they expressed interest in features like conversation history tracking. For example, if a task is mentioned and later the participant refers to "closing the task," the chatbot would recognize the context and the specific task being referred to, enhancing the conversation's continuity.

5.4.4 Design Opportunities - From Questionnaire

Based on the answers provided in the questionnaire, the additional steps where respondents believe the chatbot could be useful are as follows:

1. Task Description and Task Portal: Participants mentioned that the chatbot could describe tasks or open the task portal when asked for task details.
2. Code Review and Suggestions: Participants suggested that the chatbot could act as a second set of eyes for code review or provide suggestions before pushing to the repository.
3. Task Management and Prioritization: Participants expressed a need for the chatbot to provide information about tasks on their task manager, including background information to help them think about solutions or present tasks during standup meetings. They also mentioned commands to change schedules, remove tasks, and prioritize tasks based on priority.

4. Meeting and Appointment Scheduling: Participants mentioned that the chatbot could help schedule meetings, make appointments, and provide reminders for meetings and standups.

5. Developer Support: Participants highlighted the chatbot's potential to assist with developer-related tasks such as resolving merge conflicts, providing information on IDE-related issues, and notifying about comments and events related to merge requests.

6. Accessing External Tools and Information: Some participants suggested that the chatbot could integrate with tools like Jira or Trello to provide task updates or status reports. They also mentioned the chatbot's ability to search for software development terms, provide documentation about commands, and auto-complete commands.

7. Notifications and Aggregation: Participants mentioned the chatbot's potential to aggregate notifications from various sources such as design documents, code reviews, and comments in tracking tools.

8. Reminders and Planning: Participants mentioned that the chatbot could remind them of meetings, and deadlines, and provide news updates, software updates, and printing/scanning functionalities.

9. Task and Project Management: Participants suggested that the chatbot could help create tasks, projects, features, tickets, test cases, and provide centralized access to related documents and stakeholders.

10. PR (Pull Request) Management: The chatbot could provide information about PR reviews, comments, and status.

11. Repository and File Navigation: Participants mentioned that the chatbot could help locate repositories or specific files related to features or bugs.

12. Automating Software Development Processes: The chatbot could automate various aspects of the software development process, such as running tests, dealing with merge conflicts, creating new repositories or cloud instances, and checking server statuses in production.

13. Calendar and Meeting Management: Participants suggested that the chatbot could handle calendar management, schedule meetings, remind about meetings, and provide information about participants' availability.

14. Collaborative Communication and Information Sharing: Participants mentioned that the chatbot could update them about their coworkers' problems, provide search history or implemented solutions, and facilitate communication with colleagues.

15. Integration with External Tools and Data: Participants mentioned the chatbot's potential to integrate with external tools, know general data, and complete tasks based on that integration.

These responses highlight the diverse range of tasks and functionalities that users believe the chatbot could assist with, covering areas such as task management, code-related support, meeting scheduling, information retrieval, notifications, and collaboration.

5.4.5 Design Opportunities - From Questions

Based on the provided list of questions that participants asked Devbot, we analyzed the most prominent themes of questions asked, obtaining further insights and design opportunities. Here are some potential categories and design opportunities for chatbots in the context of software development:

1. Task Management:

    - Participants frequently asked about their assigned tasks, priorities, deadlines, and task details.
    - Design opportunity: A chatbot can provide users with real-time information about their tasks, including task names, descriptions, deadlines, priorities, and associated artifacts. It can also allow users to update task status, redefine priorities, and perform actions like marking tasks as complete or creating pull requests. An integration with project management or issue management tools is needed.

2. Source Code and Version Control:

    - Participants had questions related to code repositories, branches, commits, pull requests, and merges.

- Design opportunity: A chatbot can assist users with tasks like pulling code, pushing code, creating branches, making commits, creating pull requests, merging branches, and resolving merge conflicts. It can also provide information about the status of code builds and deployments. To develop this, the implementation of integration with version control tools such as git is needed.

3. Issue Tracking and Bug Fixes:

    - Participants inquired DevBot about issues, tickets, and bug fixes associated with specific tasks.
    - Design opportunity: A chatbot can help users find associated issues or tickets in platforms like GitHub or Jira. It can also provide information about the status of bug fixes, approvals from QA, and whether a build was successful.

4. IDE and Development Environment:

    - Participants sought assistance with development environments, IDEs, debugging, and specific programming languages like Java.
    - Design opportunity: A chatbot can guide IDE-related tasks, such as opening IDEs like Eclipse, helping with debugging code, suggesting solutions for common programming issues, providing syntax examples, and answering language-specific questions.

5. Meeting and Schedule Management:

    - Participants asked about stand-up meetings, schedules, and meeting attendees.
    - Design opportunity: A chatbot can provide users with information about their schedules, upcoming meetings, attendees, agendas, and meeting locations. It can also help schedule new meetings, notify participants, and set reminders.

6. Documentation and Artifacts:

    - Participants requested information about documentation, requirements, class diagrams, and artifacts related to their tasks.
    - Design opportunity: A chatbot can assist users in finding relevant documentation, requirements, diagrams, and other artifacts associated with their tasks. It can provide links or access to these resources for easy reference.

7. General Assistance and Miscellaneous:

    - Participants had various general questions, such as asking for help with coding, asking about the weather, checking the time, requesting assistance with specific programming languages or concepts, and verifying information.
    - Design opportunity: A chatbot can serve as a general assistant, providing help, answering questions about programming languages, explaining concepts, offering command-line examples, and performing basic tasks like checking the time.

These categories and design opportunities can help guide the development of chatbots for software development, focusing on addressing the specific needs and workflows of developers, streamlining their tasks, and providing quick access to relevant information and resources.

5.4.6 Design Opportunities - From Interviews

Based on what participants described in the interview, the following features are desired in a chatbot.

1. Participants interested in a chatbot that guides tasks and provides control:

    (a) Participant 2: Appreciated quick responses, prefers chatbot that guides rather than automates tasks, likes to be in control.

    (b) Participant 21: Positive experience, interested in a chatbot that automates tasks and provides guidance, desires control over assigned tasks.

    (c) Participant 22: Positive experience, prefers granular control over code commits, comfortable with automated commits if clear representation is provided.

2. Participants struggling with chatbot usability and understanding:

    (a) Participant 3: Had difficulty using the chatbot, confused about communication, prefers direct communication with task assigners.

    (b) Participant 4: Found it challenging to get helpful responses, uncertain about keywords/commands, prefers chatbot that helps with specific tasks.

    (c) Participant 23: Frustrated with limited capabilities and understanding of the chatbot, desires more guidance, prefers automation with safety nets.

3. Participants providing general suggestions for improvement:

   (a) Participant 5: Recommends training the algorithm, suggests saving task IDs and conversation context, desires automation for specific tasks.

   (b) Participant 19: Highlights areas for improvement, suggests handling missed tasks, understanding terminology, ontological understanding, providing task guidance, office environment integration, and task priority management.

4. Participants expressing positive feedback and interest in chatbot:

   (a) Participant 10: Finds the chatbot useful, desires automation and voice input, and suggests integration with other systems.

   (b) Participant 12: Finds the idea of a chatbot appealing and timesaving, interested in intelligence, desires guidance and automation, and asks about voice input.

   (c) Participant 14: Interested in a comprehensive question-answering chatbot, prefers assistant role, desires information retrieval and code-related actions.

5.4.7 Demographics and Post-Survey Correlations

We investigated possible correlations of demographic data (age, experience) with the opinions of participants in the study. The Pearson correlation coefficient [134] ranges from -1 to 1, where -1 indicates a perfect negative correlation, 1 indicates a perfect positive correlation, and 0 indicates no correlation.

Correlation Calculation: The correlation coefficient is a statistical measure used to quantify the strength and direction of the linear relationship between two variables. It indicates how closely the data points of these variables cluster around a linear trend. The correlation coefficient typically ranges between -1 and 1.

The formula to calculate the correlation coefficient (often denoted as "r") between two variables, let's call them X and Y, is as follows:

$$r = \frac{n(\sum XY) - (\sum X)(\sum Y)}{\sqrt{[n\sum X^2 - (\sum X)^2][n\sum Y^2 - (\sum Y)^2]}}$$

where:

- n is the number of data points
- $\sum$ represents the summation symbol
- X represents the values of the first variable
- Y represents the values of the second variable
- XY represents the product of the corresponding values of X and Y
- $\sum X^2$ represents the sum of the squares of the values of X
- $\sum Y^2$ represents the sum of the squares of the values of Y

The correlation coefficient value r provides information about the relationship between the two variables:

- $r = 1$: Perfect positive correlation, meaning that as one variable increases, the other variable also increases linearly.
- $r = -1$: Perfect negative correlation, indicating that as one variable increases, the other variable decreases linearly.
- $r \approx 0$: Little to no linear correlation between the variables.

The absolute value of r also indicates the strength of the correlation. Closer to 1 (positive or negative) implies a stronger linear relationship, while closer to 0 indicates a weaker or no linear relationship. It is important to note that the correlation coefficient measures only linear relationships. It might not capture complex relationships, outliers, or nonlinear patterns between variables. Additionally, correlation does not imply causation; a strong correlation does not necessarily mean one variable causes the other to change.

Age x Likes DevBot: To determine the correlation between age and the likeness of interacting with DevBot, we can calculate the correlation coefficient using the study data. The correlation coefficient measures the strength and direction of the linear relationship between two variables. When comparing age and how much participants liked to interact with Devbot, we found a correlation coefficient of -0.022. Based on this coefficient, there is a very weak negative correlation (-0.022) between age and the likeness of interacting with DevBot. In fact, one might argue that the correlation is so weak we cannot claim there is a correlation at all.

A negative correlation means that as age increases, there is a tendency for liking DevBot to decrease slightly. However, the correlation coefficient of -0.022 indicates that this relationship is not very strong.

However, the correlation is close to zero, indicating that there is no substantial relationship between age and the likability of the chatbot. Other factors not considered in this analysis may have a stronger influence on the likability ratings.

Years of Experience x Likes DevBot: Based on the calculated correlation coefficient, there is a weak negative correlation (-0.138) between years of experience in software development and the likeness of interacting with DevBot. However, the correlation is close to zero, indicating that there is no substantial relationship between years of experience and the likability of the chatbot. Other factors not considered in this analysis may have a stronger influence on the likability ratings.

Age x Automation Preference: Based on the provided data and the correlation analysis, the correlation coefficient between automation preference and age is approximately -0.15. This value suggests a weak negative correlation between the two variables.

A negative correlation means that as age increases, there is a tendency for the preference for automation to decrease slightly. However, the correlation coefficient of -0.15 indicates that this relationship is not very strong.

It is important to note that correlation does not imply causation, and this analysis is based on a limited set of data. Therefore, it is crucial to interpret the results with caution. Other factors beyond age may also influence the preference for automation, and additional data or a more comprehensive study would be required for a more accurate analysis.

Gender x Automation Preference: Based on the given data, we cannot determine the correlation between automation preference and gender. Further analysis with an appropriate statistical method using a more extensive dataset would be required to assess any potential relationship between the two variables. The cross-tabulation of gender x automation preference is presented in Table 5.5. There are only 28 responses in the table because we were unable to gather the preferences of one of the participants. *Yes* in the table means (prefers automation), *No* means prefers guidance and *Mix* means prefers a mix between automation and guidance.

Experience x Automation Preference: When analyzing the correlation between years of experience and automation preference, results also show a preference for a mix between automation and guidance, in all three experience ranges. We collected ranges of experience of $\geq 1 year \leq 5 years (juniors), \geq 5 years \leq 10 years (mid-level), and \geq 10 years, seniors$.

Table 5.5: Gender x Automation Preference.

	Male	**Female**
Yes	3	1
No	5	2
Mix	14	3

The three categories (juniors, mid-levels and seniors) prefer a mix between automation and guidance. Of juniors, 46% prefer a mix of automation, while 34% prefer no automation and only guidance. Of mid-levels, 50% prefer a mix, while 33% prefer automation. Of seniors, 100% of them responded they prefer a mix of full automation and guidance.

Sentiment Analysis: In mixed-methods research, which combines both qualitative and quantitative approaches, conducting sentiment analysis on interview results can provide valuable insights and enhance the overall depth of understanding. Sentiment analysis involves the automated process of determining the sentiment or emotional tone expressed in a piece of text, whether it's positive, negative, or neutral. By incorporating sentiment analysis, researchers can gain a deeper understanding of the emotional responses, attitudes, and perceptions of participants. This adds a layer of richness to the analysis beyond just extracting themes and patterns. Moreover, adding sentiment analysis to qualitative data analysis aids in interpretation. It helps researchers identify the tone and sentiment of participants' statements, making it easier to distinguish between strongly positive, mildly positive, neutral, mildly negative, and strongly negative sentiments. This nuanced interpretation contributes to a deeper understanding of participants' viewpoints. To analyze sentiment, we used the GPT-3.5 model. This model is not specifically dedicated to sentiment analysis like some specialized sentiment analysis models, such as the VADER (Valence Aware Dictionary and sEntiment Reasoner) model, or BERT (Bidirectional Encoder Representations from Transformers). However, it can still perform sentiment analysis by analyzing the overall tone and context of the text.

According to this model, the overall sentiment analysis is presented in Table 5.6.

5.5 Discussion

Next, we discuss some of the key implications raised when analyzing the reported results and the experience of the user study.

Table 5.6: Sentiment Count of the User Study.

Sentiment	Count
Positive	10
Negative	3
Neutral	15

Topics of Interest. Our results indicate that most of the questions asked by participants were about tasks and repository management. Of course, those were the two main topics in the scenario. However, it still shows developers are interested in receiving such support, accessing information from their tasks, or pushing code through CAs, mostly for task management.

Guidance or Automation? There are varying opinions regarding preferences between guidance and automation. This contradicts a hypothesis we had that automation would be an almost unanimous preference. Further investigations on what influences the automation preferences of users must be undertaken.

Need for more context. Respondents have also indicated that adding more context to the chatbot would be desired. For example, the chatbot should know the repository address, branch or patterns expected to commit messages. However, having a simple chatbot with a few expected questions and giving participants a simple scenario was also indicated as being helpful by most of the participants. Based on feedback from participants, adding context to the chatbot could immensely improve the chatbot's capabilities, as it would be personalized. The chatbot could also incorporate the discovery of context or recommendations based on the history of interactions. With ChatGPT, many of the features that participants claimed to desire in this study have been covered, however, adding context is not one of them. Further discussion and investigations on how to add context to LLMs must be undertaken.

Need for domain-specific knowledge. On a similar spectrum, results also indicate that the least experienced developers account for a low score when asked if the chatbot was helpful for the questions asked. This can indicate that the chatbot has to be prepared to give more details about the process to less experienced users. Most experienced participants, who have more than five years of experience, indicated that the chatbot was more helpful.

Chatbot acceptance. A rather surprising result was that some participants asked questions beyond the scenario presented, which can indicate a real interest in using the chatbot in further contexts as well, not only to the ones limited to the presented scenario.

Some examples include:

- More general questions such as ”What’s the best language to develop Machine learning models?
- Meeting Schedules and Agendas: Where is my next meeting? or What is the agenda for the meeting today or When is the stand-up meeting?
- Code Debugging and Assistance: Can you help me in debugging code in Eclipse?
- Others:
  - Are we using Github or Gitlabs? or When is lunch?
  - Are there any updates on the Github repo while I was gone?
  - Can you send me the doc string for function np.argmax()?
  - Are there any updates from the test team?
  - Please print this document. [File attached]
  - Can you send a copy of my current edit to team X?

Chatbot understanding. One negative result reported is related to how the chatbot understands the questions. Because only around 7 intentions were mapped to answers, when the developers asked questions that the chatbot did not understand but were still related to the scenario, some disappointment was reported. Although the purpose of the study was not to evaluate the tool, therefore our prototype was very limited in that sense, having a chatbot that could understand several other intents might have generated a more positive reaction from the developers after the chatbot interaction.

Scenario. Presenting participants of a mixed-methods user study with a scenario can have both advantages and disadvantages. Here’s an overview of these points:

- Advantages:
  - Contextual Understanding: A scenario provides participants with a clear context and background for the study. This helps them understand the purpose, scope, and objectives of the research more effectively.
  - Engagement: Scenarios can make the study more engaging by presenting a relatable situation. Participants can connect better with the study and its goals, which might increase their involvement and motivation.

  - Realism: By presenting a scenario, you create a realistic setting that participants can relate to. This can lead to more authentic responses and insights, as participants engage with the scenario as they would in a real-world context.
  - Consistency: When all participants start with the same scenario, it ensures a consistent starting point for the study. This reduces variability in participants' initial understanding and sets the stage for more meaningful comparisons.
  - Guided Exploration: Scenarios guide participants toward specific topics or aspects of interest. This helps ensure that all participants explore the same core concepts, making the study's findings more focused and relevant.

- Disadvantages:

  - Bias and Stereotyping: Scenarios could unintentionally introduce bias or stereotyping, influencing participants' perceptions or responses. Careful crafting of scenarios is needed to avoid any unintended implications. We have crafted the scenario with software developers and with the support of relevant literature [111].
  - Limited Flexibility: Some participants might feel constrained by the scenario and may not be able to express their genuine thoughts that fall outside its scope.
  - Artificiality: Depending on the complexity of the scenario, or the experience of participants, they might feel that the scenario is artificial or detached from their real experiences. This could affect the authenticity of their responses.
  - Misinterpretation: There is a risk that participants might misunderstand the scenario or its intent. This can lead to participants providing irrelevant or inaccurate responses that don't align with the study's goals.
  - Limited Generalization: While scenarios provide a focused context, the findings might not be as generalizable to broader contexts, as participants are responding to a specific situation.

Potential follow-up studies. We believe that these preliminary results, although promising, still require further development. Additional work is required to involve a larger population of software developers and to mitigate potential generalization biases. Further investigation should examine the specific support that such a tool should provide to developers, and how to integrate a chatbot with a context model that understands the software development workflow. In addition, further qualitative and

quantitative studies will be needed to demonstrate the quality of a chatbot's recommendations.

We believe that understanding the interaction of developers with the systems as chatbot users is key to improving developers' experience and advancing software engineering practices, providing the needed timely support for developers. Nonetheless, it seems that even having a chatbot that is limited in terms of what it can do and how it can help, participants were keen on the idea of using a chatbot. These empirical findings are aligned with the discussions presented in [161]. It is worth mentioning this analysis was executed in 2021, an era before ChatGPT was available to the public.

5.6 Threats to Validity

This section addresses potential threats to the validity of the user study conducted to investigate software developers' interactions with the DevBot chatbot prototype.

- Construct Validity:
  - Scenario Realism: The study scenario, depicting a typical day in the life of a software developer, may not fully represent the diversity of tasks and situations developers encounter. This could affect the realism of participants' interactions with the chatbot. To mitigate this, we have validated the scenario with three experienced software developers, and we also got inspiration from Meyer et al. [111].
  - Question Prompting: Participants were asked to generate questions for the chatbot. The framing of this request may have influenced the types of questions asked, potentially leading to biased or limited question sets.
  - Question Clarity: The clarity of participants' questions may vary, potentially affecting the quality of interactions and the chatbot's ability to provide relevant responses.
- Internal Validity:
  - Limited Interaction Time: Participants were allotted a fixed interaction time of approximately 10 minutes with the chatbot. This constraint might not fully capture the potential benefits or limitations of the chatbot over extended periods of use. Most participants, though, indicated they had run out of questions

to ask after around 10 minutes of interaction with the chatbot. Anyone who could think of more questions was encouraged to continue to ask.

  - Prior Chatbot Experience: Participants had varying degrees of experience with chatbots. This prior experience might have influenced their expectations and assessments of the chatbot's performance.

- External Validity:

  - Participant Pool: The study recruited participants primarily from university graduate student mailing lists, potentially limiting the generalizability of the findings to a broader population of software developers. The study involved 29 participants, which may not fully capture the diversity of software developers in terms of experience, expertise, and preferences.
  - Platform Dependency: The study deployed the chatbot on the Facebook Messenger platform. This choice of platform might not represent all the platforms and tools developers use in their daily work. However, it is possible to deploy the same Rasa chatbot using other platforms, such as Whatsapp, Telegram, Slack, Twilio, Google Hangouts, Cisco Webex, and others.

- Conclusion Validity:

  - Questionnaire Subjectivity: The Likert scale questions in the post-study questionnaire and open-ended responses are subject to participants' subjectivity and potential bias in their evaluations.
  - Interviewer Bias: The semi-structured interviews were conducted by the study organizers, introducing the possibility of interviewer bias in the interpretation of participants' responses.

5.7 Conclusion

We have a user study that explores the preferences of software developers when interacting with a chatbot. Developers usually work with various tools in a very dynamic situation, and promoting ways to support them is critical for the quality of their work. As well, a chatbot could improve productivity, lessen training time and make implicit preferences explicit through capture processes. We collected many design opportunities

for chatbots in the software engineering domain. Our results also demonstrate developers are willing to work with such tools and have found this solution to be interesting while providing ideas for the future on how chatbots could behave and connect.

In this user study, we delved into the interaction of software developers with chatbots, unearthing intriguing insights that have significant implications for the realm of human-machine interaction in software development. The study revealed that the majority of questions posed by participants gravitated towards tasks and repository management, aligning with the primary themes of the provided scenario. This highlights developers' inherent interest in receiving support in these areas through CAs, especially in the realm of task management. The use of a scenario in the study offered advantages such as contextual understanding, engagement, and guided exploration. However, it also introduced potential biases, limited flexibility, and artificiality, warranting careful scenario design and consideration of its impact.

A notable divergence emerged regarding preferences between guidance and automation. Contrary to initial expectations, automation did not uniformly triumph as the preferred approach. This finding underscores the need for deeper investigations into the factors influencing users' automation preferences. Participants expressed a desire for chatbots to possess deeper contextual awareness, including repository details and interaction history. The potential for personalized interactions and context-driven recommendations emerged as a key avenue for improvement. The incorporation of context discovery mechanisms and historical interaction analysis is suggested for enhancing chatbot capabilities.

The study exposed that less experienced developers perceived the chatbot as less helpful, indicating a need for tailored support for novice users. Experienced participants, with over five years of experience, found the chatbot more beneficial. This finding underscores the importance of catering chatbot responses to the user's expertise level. Intriguingly, participants ventured beyond the scenario's boundaries, posing questions that extended beyond its scope. This suggests a genuine interest in using chatbots in broader contexts, beyond the confines of the presented scenario. This diverse range of additional queries demonstrates the potential versatility of chatbot applications in the developer's toolkit.

Participants expressed disappointment when the chatbot failed to comprehend questions that were related to the scenario but fell outside the predefined intents. The study's limited scope in this regard highlighted the importance of chatbots having a more comprehensive understanding of user queries to elicit a more positive user response.

These preliminary findings set the stage for future research endeavors. Further in-

vestigations should involve a larger and more diverse population of software developers to mitigate potential biases. Researchers must delve into the specific support mechanisms chatbots should offer to developers, incorporating context models to enhance workflow understanding. Qualitative and quantitative studies should assess the quality of chatbot recommendations.

In conclusion, this study serves as a foundational exploration of developers' interactions with chatbots. It underscores the significance of understanding developer needs and preferences to enhance their experiences and software engineering practices. The era of ChatGPT and advanced language models has allowed the prospects for enriched human-machine collaboration in software development to appear promising, warranting continued research in this domain.

Next, intrigued by the fact that not all developers wish to automate all of their tasks, we conducted an investigation into the factors that influence automation levels in systems.

Chapter 6

Variability Design and Levels of Automation in Human-Chatbot Interactions

"The recent phenomenal advances in the foundational areas of cognitive computing systems are poised to usher in even more sophisticated systems that will rival and perhaps even surpass human performance." – Gudivada VN, et al.

6.1 Overview

Quality, productivity, accuracy, precision, and other metrics are usually improved when machines perform tasks previously assigned to humans [165]. What is the best choice to execute a specific task, a human or a machine? Many debate the advantages and disadvantages of fully automating tasks as opposed to keeping humans involved [57]. However, rather than automation being an all-or-nothing proposition, many levels of automation can be used, ranging from entirely manual to fully autonomous [165, 54].

So-called autonomous systems such as self-driving automobiles or trucks, autopilots on airplanes, robots, machine tools, chatbots and 'smart' buildings still need human intervention under various conditions such as sudden changes in traffic, weather conditions, environment, or materials. Thus, these so-called autonomous systems need to operate independently to achieve the highest degree of automation possible while

they need to be designed to accept human intervention when necessary or appropriate. Although autonomous systems use machine learning, which recognizes long-term patterns, sometimes such systems must recognize short-term situations, a task at which humans are particularly competent. We are interested to understand the principles that should be used to design such systems. Also, we want to understand if we can develop a software engineering discipline that addresses autonomous system design. In this study, we identify factors that are common to autonomous systems and should be considered when developing a software design approach.

Few methods or resources are available to support the flexible and scalable assignment of tasks between humans and machines within autonomous systems [124, 4]. The relationship and responsibility for distributing tasks between humans and machines within autonomous systems are not clearly defined, and the LOA is far from uniform across various contexts. Thus, modelling techniques that can systematically support task distribution across a wide range of automation levels are needed.

There are inherent challenges in developing and researching the automation variability levels of these systems. When developing autonomous systems, the computational complexity and memory footprint of algorithms play a crucial role in the design and implementation of such, as these systems must be developed with computation times that satisfy real-time responses [156].

Moreover, these systems differ by quality standards, such as in the application or the agent responsible for automating the task. These applications can also change if the perspective or feedback of the person interacting with the agent differs and depends on human resources and the system's ability. To design systems properly that support different LOAs, developers must be provided with a clear understanding of the relevant factors that influence LOAs and with design criteria for addressing them [49].

We address this gap by identifying, refining, and representing the factors that can influence a LOA decision. We also introduce an approach to capture the factors that influence LOAs in autonomous systems. The approach presents several changes inherent in developing these autonomous systems, including those related to systems and humans. This approach aims at answering the following research question: **RQ3: Which factors affect the variance of levels of automation in autonomous systems?**

As specific contributions, this study:

- Presents an approach to identify the factors that influence LOAs
- Provides a list and categorization of the factors that influence LOAs

- Refines the identified factors by demonstrating how systems can capture these factors
- Introduces a representation of the variability of the factors and their relationships with LOAs in a feature diagram
- Demonstrates the feature diagram and approach with illustrative examples

6.2 Research Methodology

We conducted a systematic literature review (SLR) to answer the research question posed in the Introduction. In this section, we describe the methodology used to select the papers from where we should extract the factors that influence LOAs. To answer the proposed research question and investigate which factors impact the LOA variability in systems that support human-machine interactions, we have followed the approach illustrated in Figure 6.1. This figure shows how we present and organize the contributions and results of our work, in sequence. Therefore, the remainder of this chapter is organized as follows. First, we describe the Literature Search that we used to identify the factors that influence LOAs. Then, we list and categorize these factors and the interactions among them. Following, we refine the identified factors, by using examples from the literature to portray the factors as features and constraints, which will be a seed for the next step, the Feature Model creation. Lastly, we demonstrate the feature model in three scenarios.

This study aims to identify the factors that influence LOAs. Moreover, we represent the variability of the factors and their relationships with LOAs in a feature model [86, 122] and illustrate these variable factors with examples in different domains. The advantages of identifying the factors that govern human and autonomous systems interactions are manifold. Intelligent automation is one of the current emerging technologies. The ultimate purpose is to build autonomous systems that can handle edge cases and achieve the highest degree of automation possible. Building systems that consider human interaction and how this interaction impacts the system's behaviour leads to better system designs in terms of accuracy. As humans are better at spotting patterns in small data sets, combining human and artificial intelligence could provide highly accurate systems. Rule-based automation can sometimes be more precise than AI-based intelligent automation, while AI models are only partially correct. After all, no matter how perfectly you design a fully automated system with all possible outcomes, the reality is frequently complicated. Human-free end-to-end process automation is attractive

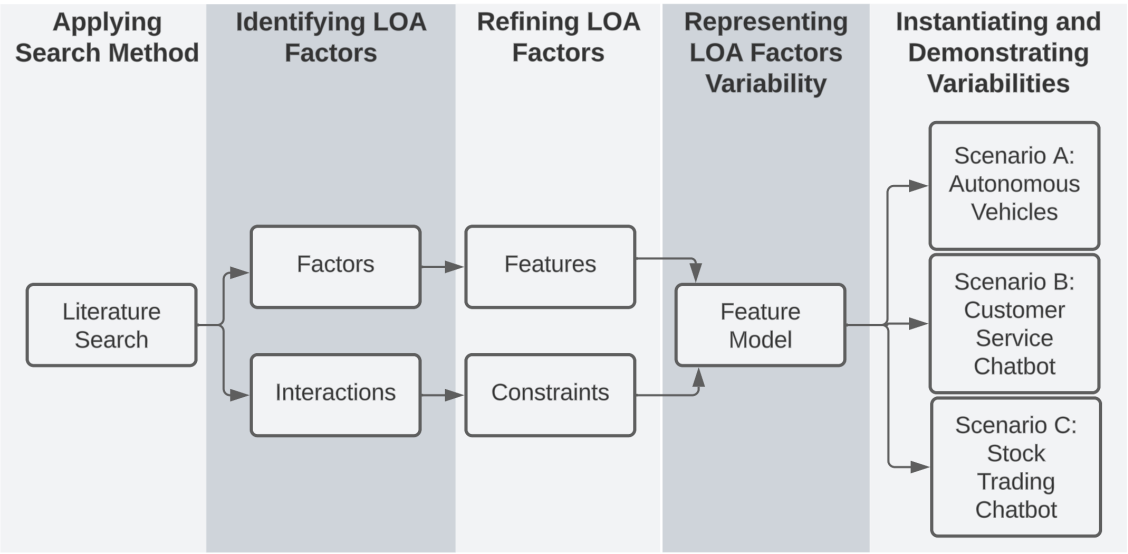

Figure 6.1: LOA Study Approach Overview.

because it is significantly easier to implement than systems that require human input. Our work mitigates these issues and difficulties by bringing to light factors, a model and illustrative examples to support the design and implementation of autonomous systems interacting with humans. The approach to supporting the design of autonomous systems is described next.

6.2.1 Applying Search Method

To identify the relevant studies, we searched several databases using keywords related to LOAs and factors influencing automation. We also manually searched the reference lists of relevant articles to identify additional studies. Our inclusion criteria were studies investigating factors influencing automation levels in systems supporting human-machine interactions. We excluded studies that did not meet our inclusion criteria, were not written in English, or were published before 2000 (including only papers published within the last 23 years). Two researchers participated in the selection of the included articles.

We started the research by examining secondary studies summarizing taxonomies for levels or degrees of automation in autonomous systems published since 2000 [165, 157]. Secondary studies synthesize or analyze published research. Secondary studies

can help researchers identify the most relevant and high-quality primary studies on a particular topic. They can also help them identify gaps or inconsistencies in the existing literature [81]. These studies are recent, and they follow systematic review protocols and report extensive results. Together they provide an important baseline for the topic. From these secondary studies, we performed backward and forward snowballing (one level in each secondary study paper). This step aims to identify proposed LOAs and the factors (contexts, characteristics) that influence the LOAs in these studies.

As exclusion criteria [81], we excluded papers not written in English and those that do not have information on factors that affect automation levels. All other articles were considered.

6.2.2 Identifying LOA Factors

In Section 6.3, for each LOA factor identified, we kept a record of the information in a spreadsheet. If the same information was found in another paper, we recorded the citation with the already listed factor. Each citation provides a scientific basis for the factors we identified. We then categorize each factor into more abstract concepts. For example, if the factors were Role and Cognitive Ability, these factors would be abstracted into ”Human” factors. If the factors were maintainability and reliability, these factors would be categorized into ”Quality” (Non-Functional Requirements) factors.

Two researchers reviewed over 150 papers and extracted and categorized the factors together. We used the inter-rater reliability measure with Cohen’s kappa statistical coefficient to measure the degree of agreement between the two researchers (judges) categorizing the identified factors [84]. The two researchers almost always agreed when categorizing the factors (k between 0.81 – 1.00). This research methodology aims to contribute to the state-of-the-art by illustrating the factors impacting automation levels and representing these factors and their relationships with LOAs. The ultimate goal of our work is to support the design and implementation of autonomous systems interacting with humans. To achieve this, we will provide models, refinements, use cases and discussions on our findings. By highlighting the factors influencing LOAs and their variability, we aim to improve the effectiveness and usefulness of autonomous systems and ultimately facilitate their widespread adoption.

6.2.3 Refining LOA Factors

Next, in Section 6.4, we organize these factors, categorize them into a table and provide examples of how these factors may be implemented in systems.

6.2.4 Representing LOA Factors Variability

Before discussing the representation of the factors, we propose a representation of this systematic approach in Section 6.5. Then, in Section 6.6, we represent the factors and the relationship between the identified factors and LOAs using a feature diagram.

6.2.5 Instantiating and Demonstrating Variabilities

Last, in subsections 6.6.2, 6.6.3 and 6.6.4, we present the instantiations of the features and constraints in three distinct scenarios. Three authors met after the last iteration to review the instantiated feature models, our final contribution to this study.

6.2.6 Methodology Highlights and Challenges

Assessing the research methodology proposed in our work, we list our method's highlights (strengths) and challenges (weaknesses).

Strengths:

- The methodology is clearly outlined and easy to follow, making it easier for other researchers and the software engineering community to replicate or build upon the study.
- The research question and objectives are explicitly stated, which helps to maintain focus throughout the study.
- A LR conducted by two authors, providing a rigorous and comprehensive approach to selecting relevant papers and extracting data from them.
- The researchers used online collaboration tools to control their versions of files and control file edits, enhancing the findings' reliability and validity.

- The use of inter-rater reliability measures to ensure consistency in categorizing the identified factors adds to the robustness of the study.
- The researchers clearly state the significance of their work and how it contributes to the field, which helps to justify the research and its outcomes.

Weaknesses:

- The study only includes papers published in English and after 2000, which may limit the generalizability of the findings. However, we do include a comprehensive review of the related works and background.
- The exclusion of non-English papers and those published before 2000 may have resulted in the omission of relevant studies or factors influencing LOAs.
- The use of backward and forward snowballing may have also limited the review's scope and missed important papers or factors that were not cited in the selected articles.
- The methodology does not clearly justify categorizing the identified factors into more abstract concepts, which may lead to potential biases or subjectivity in the analysis. We mitigated this by having two researchers categorize the factors and using Cohen's kappa statistical measure.

The following section presents the identified LOA factors based on the approach just described.

6.3 Identifying LOA Factors

In this section, answering the **RQ** proposed in this study, we present the factors that influence LOAs. We categorize these factors and present each of the categories in a subsection. The result of our analysis and categorization is presented in Table 6.1. This table illustrates the factors that influence the autonomy level decision, as identified in the literature. This table has four columns related to the factors identified in the literature review and one column related to the authors that cite each factor in the literature. Working across the table, each factor, if applicable, is further divided into subfactors. In other words, the factor information in the column to the right is related to the last row of the cell on the left. Then, we describe the classification of the factors, provide examples, and discuss how factors can influence LOAs.

6.3.1 Identifying Factors

Based on the literature review and after categorizing the factors, the result is five main factors that can influence the decision to adopt a specific LOA: Quality, Agent (System), Human, Task and Environment. These factors are assembled and categorized in Table 6.1, and their descriptions are provided next.

Quality Sheridan and Verplank [155], Khuat et al. [78], Proud et al. [139], and Beer et al. [15] identified quality criteria in systems that support some sort of autonomy. These authors describe factors such as Trust, Reliability, Fairness, Transparency and Accessibility. For instance, a system should provide a higher LOA in the tasks that this system is expected to carry out if this system is reliable (success in testing). These authors also mention Explainability, Understandability, Maintainability, Usability, Safety, Ethics, Legal compliance, and System Adaptability of the system as quality factors that influence LOA.

Task. There are factors specifically related to the task's characteristics that influence a system's LOA. For example, the result of the task (failure/success) and quality factors specific to the task, such as Performance, Complexity, Risk and Accessibility [49, 50, 155, 144, 33]. Other factors related to the task, such as Workload [173, 4], Frequency of a task [49] and Interaction Type [124] also influence LOAs.

Agent. One central factor affecting automation levels in autonomous systems is the Communication [155, 40, 33] between the human and the system. In the context of teams, a human (team member) can interact with the automated system as a team member or in the form of supervisory control [40]. Another identified factor is the cost of this system, namely the equipment's operating cost and the implementation cost of the agent [5, 139]. Other factors we have identified are related to the capabilities of this system, including Reactiveness, Situation Awareness, Decision Capability and Feedback capabilities [33, 24, 120, 49, 51, 124]. Other capabilities of the agent influence the desired LOA. For example, the Transparency of the agent regarding its procedures and goals [157, 15], or how the agent acquires and analyzes information [131]. Ability is explicitly also mentioned as one factor, as well as the Authority of the agent executing work autonomously [57].

Human. Many factors related to the person interacting with the system can influence the decision of LOAs. Authors refer to the age of the person interacting with a system [173], the recognition of the time to acquire control or the time to give up the control [125], the person's cognitive ability [125, 173] and other factors. Many authors point to factors related to how humans interacting with the system perceive the system [144, 15],

Table 6.1: (a)Levels of Automation Factors and Authors Citing Each Factor.

Factors		Authors
Quality		
Trust		[139][15]
Reliability		[155][78]
Fairness		[78]
Transparency		[78]
Accessibility		[78]
Explainability		[78]
Understandability		[78]
Maintainability		[78]
Usability		[78]
Safety		[78]
Ethic		[78]
Law Compliance		[78]
Adaptability		[78]
Task		
Result		[157][142][4]
Quality		[157][142][4]
	Performance	[49][50]
	Complexity	[155][144][33]
	Risk	[144]
Workload		[173][4]
Frequency		[49]
Interaction Type		[124]
Agent		
Cost		[5][139]
Communication (verbal)		[155][40] [33]
Capability		
	Reactive	[33][24][120]
	Situation Awareness	[49][51][120]
	Decision Capability	[51][124][120][131]
	Feedback	[155]
	Recovery Ability	[125][24][120]
	Team Cooperation (push/pull)	[40]
	Context (domain)	[24][124][4]
	Adaptability	[157][120][15]
	Systematic process	[120]
	Safety	[142]
	Transparency	[157][15]
	Intelligence	[15]
	Information Acquisition and Analysis	[131]
	Action implementation	[131]
	Architecture	[26]
	Ability	[57]
	Authority	[57]

Table 6.2: (b)Levels of Automation Factors and Authors Citing Each Factor.

Factors			Authors
Human			
Age			[173]
Control timing			[125]
Cognitive Ability			[125][173][26]
Situation Awareness			[50][114][33][55] [24][15][73][78]
Performance			[40] [173][73]
Role			[114]
Workload			[155][49][144][15][73]
System Acceptance			[49][15]
Knowledge			[173][26]
Skill			[144][26][24][173]
Attention Demand			[124][78]
Engagement			[24]
Social Skills			[15]
Perception			
	System		
		Reliability (trust)	[144][15][78]
	Task		
		Workload (heavy/light)	[50][96][24]
	Self		
		Tension (tense/calm)	[96][24][173]
		Fatigue (tired/rested)	[96][24][173]
		Confidence (high/low)	[96][24][173]
Environment			[157]
Variability			[47][173][157]
	Unchanging/Highly Dynamic		[47]
Competing tasks			[78]
Demands			[78]

the task the system is supposed to execute [50, 96, 24] and the humans themselves [96, 24, 173].

Environment. Our research shows that the environment can also impact a system's LOA. Authors discuss this variability in terms of the environment as either dynamic or static [47, 157]. Khuat et al. [78] also describe factors such as the Competing tasks or Demands of the environment as aspects that influence the LOA of systems.

We have answered the Research Question proposed in this study by identifying in the literature the factors that impact the LOA of systems that interact with humans. Next, we explain how these factors can influence LOA.

6.3.2 Identifying How Factors Impact LOAs

To demonstrate the relationship between the identified factors and a LOA, we have also extracted from the same papers in the LR how a combination of factors can influence a LOA. It should be highlighted that the factors and their relationships with LOAs are intended to be "reasonable" hypotheses to examine the possibilities for a formal treatment of qualitative factors.

Riley et al. [144] give strong examples of the relationship between one or many factors and LOA. They claim that if there is an error with the system, humans are less likely to trust the system, and the LOA tends to have high levels of human control. As humans trust the system (high reliability, high trust), LOAs can be more autonomous. According to their hypothesis, these authors then claim that "trust takes longer to be rebuilt than to be destroyed" and that humans tend not to change their opinion even as their experience with the system increases.

High system reliability = more automation
Low system trust = more manual work

Proud et al. [139] discuss the autonomous system/agent cost. They claim that increased autonomy levels throughout the design phase are expensive (high cost) and time-consuming. However, if properly implemented, they raise operational safety and effectiveness, which could lower total system lifespan costs. It is essential for designers of autonomous systems to then weigh the advantages of effectiveness and operational safety of autonomous systems over their cost.

High cost to design system = more automation = cost mitigated over system lifespan

Factors such as trust and costs (or design tradeoffs) are still abstract. To systemize LOA, factors such as reliability, perceived risk, and many others should become more concrete so that system specifications can capture them. Next, we provide a refined view of the factors we have identified. We present the factors as features a system can capture and describe how these features can influence LOA once captured.

6.4 Refining LOA Factors

Many of the factors identified in the literature are abstract. In other words, capturing these factors through a system or a feature is not straightforward. For example, the factor of "reliability." How can we capture the reliability of a system? However, our goal is to allow an autonomous system capable of identifying these factors to assign a LOA to that task. Therefore, this section shows how to relate abstract factors to concrete ones. In other words, factors that can be captured by a system or features that can be used to build systems. We will refine the meaning of these factors as stated by the authors and illustrate the factors with examples. We extract one factor from each category to show how to transform an abstract factor into a concrete one that can be used in a system capture. We are calling these concrete factors the "Features" and the interaction between the features we call "Constraints." Features and constraints are described next.

6.4.1 Capturing Factors as Features

This research program extracts factors that influence LOAs. To apply these factors systematically, we discuss in this section how to convert these factors into features or concrete factors. The features, their meaning, quotes from the papers where those features were extracted, and an example of the use of the feature are demonstrated in detail in the following sections. We present one or more features from each factor category (Quality, Task, Agent, Human and Environment).

Quality Transparency [78]. Meaning: the system can show the reasoning behind its results. Quote from paper [78]: "The reluctance by people to use results they cannot understand or explain can be frustrating for simple business applications, but it is completely warranted in high-stakes contexts, including medical diagnosis, financial investment and criminal justice. To do otherwise could be disastrous."

Explanation: Can the system show the reasoning behind its code/algorithms? If yes: the system is transparent. If not: the system is not transparent.

Task Frequency [49]. Meaning: how many times a task is executed. Quote from the paper [49]: "In addition to reducing workload, expert systems can further augment the user by providing new capabilities never possessed before. Bearing this in mind and armed with an understanding of the user's needs, the actual selection of expert system applications can proceed with additional inputs in task frequency, task criticality, technological capabilities, and user acceptance."

Explanation: Depending on how often one task is executed (frequency), this task can be a suitable candidate for full or more automation, if the frequency is high.

Agent Adaptability [157]. Meaning: Capacity of the system to adapt and improve its performance in a particular environment without human intervention. Quote from the paper [157]: "Adaptability is usually considered crucial for technical autonomy. Being autonomous requires learning and adapting behavior to a changing environment. A machine of this kind can process information, expanding the knowledge implemented by programmers and changing how the system responds. This allows the system to adapt and improve its performance within an environment without human intervention. Thus, adaptable systems can alter their behavior, making them more unpredictable and independent of human operators. Adaptability, therefore, shapes technical autonomy."

Explanation: Is the system adaptable? If yes, provide more autonomy. If not, allow for manual/human intervention.

Human Workload [155, 49, 144, 15, 73]. Meaning: the amount of work a human must perform in interacting with the system. The workload can be measured by the number of hours of each task that is currently performed by a person. We can then classify the result as high, medium, or low workload. Quote from the paper [144]: "Other characteristics of the operator that are of interest are his perceptions of risk and own workload, his skill level, his performance level (decision accuracy), and his level of self-confidence."

Explanation: One person has a workload of 36 hours (high), while another has a workload of 4 hours (low). Higher workloads could demand higher automation to expedite work.

Environment Variability [47]. Meaning: Variability refers to how the environment changes with time and ranges from unchanging (low variability or highly predictable) to highly dynamic (high variability or unpredictable). Quote from the paper [47]: "This

dimension determines whether automation is applicable: automated systems cannot function well in dynamic environments, but humans can."

Explanation: If the environment is highly predictable, full automation is preferred. Where the environment is highly unpredictable, less automation and more human involvement are preferred.

6.4.2 Capturing Constraints

As we depict the known factors and their relationship with LOAs, we also consider how different factors (two or more factors combined) can affect LOAs. To investigate the effects of the combination of factors, we turn to the analysis of taxonomy proposed by Simmler et al. [157]. This taxonomy does not offer a level for "manual execution" as it is only intended to classify human-machine collaboration. The taxonomy proposed by Simmler et al. includes the following levels:

- Level 1: Offers decisions. The technical component suggests options, and the human decides
- Level 2: Executes with human approval. Technical component acts after human approves
- Level 3: Executes if no human vetoes. Technical component acts unless human vetoes
- Level 4: Executes and then informs. Technical component acts independently, and human is informed about the actions carried out
- Level 5: Executes fully automated. Technical component carries out actions independently without informing human

In this taxonomy, if the first and lowest level of autonomy (Level 1 - Offers Decisions) is selected, the agent must have the capability of making recommendations and receiving feedback. In addition, transparency, traceability and predictability are requirements for system quality. The last and highest LOA (Level 5 - Executes fully automatically) can be selected if the system has nontransparent and undetermined quality features. Data gathering, interaction with other agents, and adaptability are some of the agent's

capability features that must exist to meet the requirements of level 5. The authors consider the ability to learn through machine learning algorithms and connecting to the Internet as optional features for this level. The following constraints are examples that we captured from these rules and that we should consider if Simmler's taxonomy is selected.

In Level 1, according to the authors, a given input should always lead to a specified output. There should be complete transparency in how the system reaches that output. The system is fully traceable and predictable, with no ability to learn. For example, calculators work on this level. Therefore, we can say that with the highest transparency and highest predictability, an automation level is set to 1. In the following representations the symbols $\Rightarrow$, $\wedge$, $\vee$ and $\neg$ mean implication (if...then), conjunction (and), disjunction (or) and negation (not) logical connectives, respectively.

$$\textit{High Transparency} \wedge \textit{High Traceability} \wedge \textit{High Predictability} \Rightarrow \textit{Level 1}$$

If a system is not transparent, this means not every step is predefined and traceable. The system holds back information and moves from the input to the output, altering its manners and impacting the observer's perception. However, the output can still be determined. An example is a system that weighs many parameters before deciding. Therefore, the authors define this combination of factors as leading to level 2 automation.

$$\textit{Low Transparency} \wedge \textit{Medium Traceability} \wedge \textit{Medium Predictability} \Rightarrow \textit{Level 2}$$

A system classified at level 5 is not transparent. An input might not lead to the same output every time, and the human cannot access how the system has reached that specific output. Systems based on machine learning algorithms and connected to data sources on the internet are examples of such systems. These systems have very low transparency and predictability while having high interaction with multiple data sources and high adaptability.

$$\textit{Low Transparency} \wedge \textit{Low Predictability} \wedge \textit{High Integration} \wedge \textit{High Adaptability} \Rightarrow \textit{Level 5}$$

Next, we describe the systematic representation of this design, the representation of the identified factors and how LOAs can vary according to these factors.

6.5 Variability-Aware Human-Chatbot Interactions: Taming Levels of Automation

Autonomous systems are an integral part of the technological landscape, playing a pivotal role in shaping human-computer interactions. Just as chatbots have become increasingly prevalent across various domains, autonomous systems are evolving to provide intelligent and automated solutions in fields ranging from self-driving cars to smart home devices. The rise of these AI-driven systems underscores the growing importance of technology's ability to operate independently and make decisions without constant human intervention. In parallel with the advancements in chatbot technology, autonomous systems are also expected to adapt and cater to the diverse needs and preferences of users, making the design and development of adaptable and user-centric AI systems a shared imperative. Consequently, enhancing the interactions between humans and both chatbots and autonomous systems alike is vital in ensuring that these technologies meet user expectations and deliver a satisfactory user experience. Chatbots have become increasingly prevalent in various domains, providing automated conversational interfaces to assist users in accomplishing tasks, obtaining information, or engaging in interactive conversations [159]. In recent years, chatbots have gained significant popularity as a means of human-computer interaction. These AI-powered conversational agents offer a wide range of applications, from customer support to personal assistants [66]. However, the success of chatbots heavily relies on their ability to adapt and cater to the diverse needs and preferences of users [32, 103]. Therefore, there is a growing need to address designs that can handle the variability inherent in human-chatbot interactions. Enhancing human-chatbot interactions is a critical aspect of designing and developing chatbot systems that meet user expectations and deliver a satisfactory user experience [32].

For instance, some users may prefer a high LOA, where the chatbot takes initiative and performs tasks autonomously, while others may prefer a lower LOA, with the chatbot providing options and requiring user input for decision-making [103]. Therefore, when designing such tools, we must be aware not only that users might have different preferences, but also of the factors that influence the variations in those preferences. To address this challenge, we propose a feature-oriented design for enhancing human-chatbot interactions. This design leverages variability-aware feature-oriented design methods, enabling the systematic identification, capture, and representation of features that influence the LOA in chatbot systems. By considering various factors that influence LOAs and the systems requirements, this design allows for the design and configuration of chatbot systems with different LOAs, tailored to specific user requirements and

contexts.

We present a feature-oriented design, outlining the steps involved in its application and instantiation. Continuing our work on capturing contextual factors that influence LOA, we now present a design to enable the systematic identification of these factors that influence LOA in systems.

Within this representation, we address the challenges of human-chatbot interactions within autonomous computing systems. The increasing complexity and dynamics of these interactions must consider the contextual factors of these interactions, such as individual preferences, environmental conditions, and system quality attributes. Our proposal of a variability-aware feature-oriented design accommodates the dynamic and varying requirements of autonomous computing systems, with a specific emphasis on chatbots. This approach is crucial for enabling chatbots to provide personalized and contextually appropriate instances, thus contributing to the overall efficiency and effectiveness of autonomous AI and ML-powered systems. We delve into the integration of design techniques to enhance systems' adaptability and decision-making.

The design and development of chatbot systems that effectively interact with humans require the consideration of various factors. In this section, we propose a feature-oriented design aimed at enhancing human-chatbot interactions. This design focuses on variability-aware feature-oriented design methods, which enable the systematic identification, capture, and representation of features that influence the LOAs in chatbot systems.

6.5.1 Variability-Aware Feature-Oriented Design

Variability-aware feature-oriented design methods provide a structured approach to handle the complex dependencies and relationships among features in a system. These methods allow for the identification and modelling of the variability within the system, enabling the development of flexible and adaptable designs. In the context of human-chatbot interactions, these methods are particularly valuable as they support the design of chatbot systems with varying degrees of automation, tailored to specific user requirements and contexts.

The process using variability-aware feature-oriented design involves the following steps, in Figure 6.2:

1. **Identifying LOA Requirements:** The first step is to identify the requirements of the chatbot system that will influence the LOA. This involves understanding user

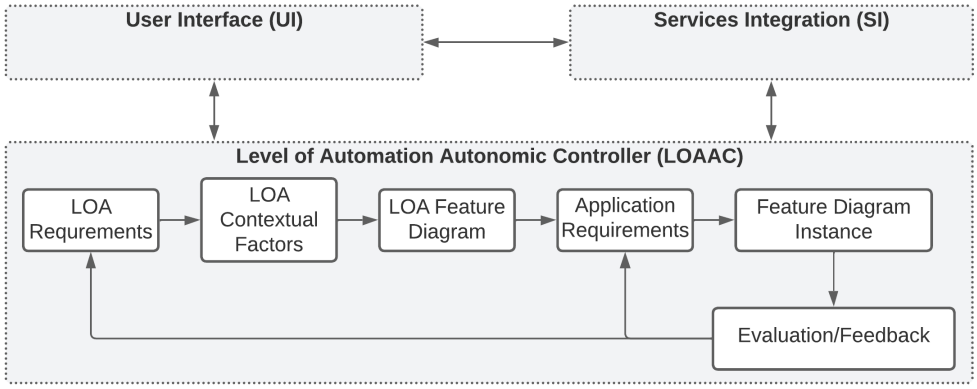

Figure 6.2: Variability-Aware Feature-Oriented Design for Enhancing Human-Chatbot Interactions.

preferences or contexts that can influence the extent of automation that will be desired. By considering factors such as the user's context, task complexity, and system quality, the requirements can be defined concerning the desired LOA.

2. **LOA Contextual Factors:** In this step, we establish the LOA factors that will be used (i.e., captured and used to define the LOA) by one instance. These factors can be categorized into different dimensions, such as quality, task, agent, human, and environment. Examples of factors include safety, usability, workload, communication capabilities, user skill, and environmental variability. By capturing these factors as features, their impact on the LOA can be systematically analyzed and modelled. For reference, the factors identified in Chapter 6 can be used.

3. **LOA Feature Diagram:** Having identified the factors that will influence the LOA, you can then design a "master" feature model, considering the constraints and dependencies among factors. This involves understanding how the absence or presence of certain features can influence the LOA. For example, the availability of real-time insights to agents may increase the LOA, while the need for user approval may decrease it. By representing these dependencies in a feature model, the design space for chatbot systems with different LOA can be explored.

4. **Application Requirements:** The application requirements can play a role and add to your LOA Feature Diagram some factors to consider that were not considered

before. Moreover, they can generate more constraints or dependencies between factors, as well as influence how you build your User Interface and Service Integrations.

5. **Feature Diagram Instance:** To illustrate the effectiveness of the feature-oriented design, it is important to instantiate and demonstrate the variabilities within the system. This demonstrates the application of the LOA Feature Diagram in a specific domain. By instantiating the LOA factors model in specific contexts, such as automated vehicles or customer service chatbots, the complexity of the interaction between features can be demonstrated. This step helps clarify the purpose and rationale of the model and provides insights into the design decisions.

6. **Evaluation/Feedback:** The design process is an iterative one, involving continuous refinement and improvement based on feedback and evaluation. By gathering feedback from users, designers, and stakeholders, the design can be refined to meet better the requirements and expectations. Iterative refinement ensures that the chatbot system evolves to provide optimal human-chatbot interactions and adaptability to changing needs. Depending on the evaluation or feedback received, new additions can be made to both the application and LOA requirements.

The feature-oriented design for enhancing human-chatbot interactions, as illustrated in Figure 6.2, consists of the following main components:

1. **User Interface (UI):** This component represents the interface through which users interact with the chatbot. It includes features related to natural language understanding, speech recognition, and dialogue management. The UI component provides the necessary capabilities for understanding user intents, extracting relevant information, and managing the dialogue flow.

2. **Service Integration (SI):** The SI component represents the integration of the chatbot with external services, such as calendars, databases, or appointment management systems. It includes features related to data retrieval, data processing, and service coordination. The SI component ensures that the chatbot can access and update the necessary information and consider its context accurately.

3. **Level of Automation Autonomic Controller (LOAAC):** This component acts as the decision-making entity that determines the appropriate LOA based on the identified LOA Contextual Factors and other components. As part of the design, there is a process that needs to happen, for chatbots to integrate the LOAs. The

LOA controller component considers contextual factors such as human factors (age, cognitive ability), task (complexity), and system capabilities (decision and feedback capabilities), system requirements and has a feedback look to determine and adapt to the appropriate LOA for each interaction. While UI and SI may not change if you switch domains, the LOAAC is the only step that is instantiated for each solution.

By combining and configuring different feature combinations within these components, varying LOAs can be achieved in the chatbot. For example, in an appointment-scheduling chatbot, a high LOA may involve the chatbot autonomously suggesting available time slots based on user preferences and confirming the appointment without user intervention. On the other hand, a lower LOA may involve the chatbot presenting available options to the user for manual selection and confirmation. The feature-oriented design provides a flexible and adaptable framework for designing chatbot systems that cater to different user requirements, contexts, and LOAs. By capturing the variabilities as features and modelling their constraints, the design enables the systematic exploration and configuration of chatbot systems with enhanced human-chatbot interactions.

From this systematic process, we now conclude our research by presenting the design representations of the LOA factors variability, and we also present three instances of this representation in specific scenarios, using feature models.

6.6 Representing LOA Factors Variability

The first step to achieving variability in a system is understanding and representing variability in its application domain. Given the diversity of current intelligent systems, our goal is to propose a flexible solution that may be used for several situations rather than one unique problem. Our approach incorporates Feature-Oriented Domain Analysis (FODA) [135] to represent the system variability using a feature model (FM). FMs are primarily used in domain engineering to represent common and variable characteristics to maximize the reuse of software features or components [176]. This tree-like notation (FODA) is typically used in software variability management to provide a visual hierarchy of features [86, 122], and has also been used to compare the design space of technologies such as model transformations [37], conversational AI systems [9], and asset management in Machine Learning [68].

Our study adds to the body of knowledge by providing a feature-model-based depiction of the factors affecting the degree of automation in autonomous systems. This paradigm can be used to develop autonomous systems that interact with people. Our goal is to represent and model the variability of factors that influence LOAs and how the interaction of these factors influences LOAs. Although other notations to represent variability exist, such as the Cardinality-based Feature Model (CBFM) or Common Variability Language (CVL), the original FODA's notation can effectively express commonality and variability. FODA notation represents a feature, mandatory, optional, AND, XOR, and constraints [176]. As a result of this factor mapping, we propose a model diagram represented in Figure 6.3.

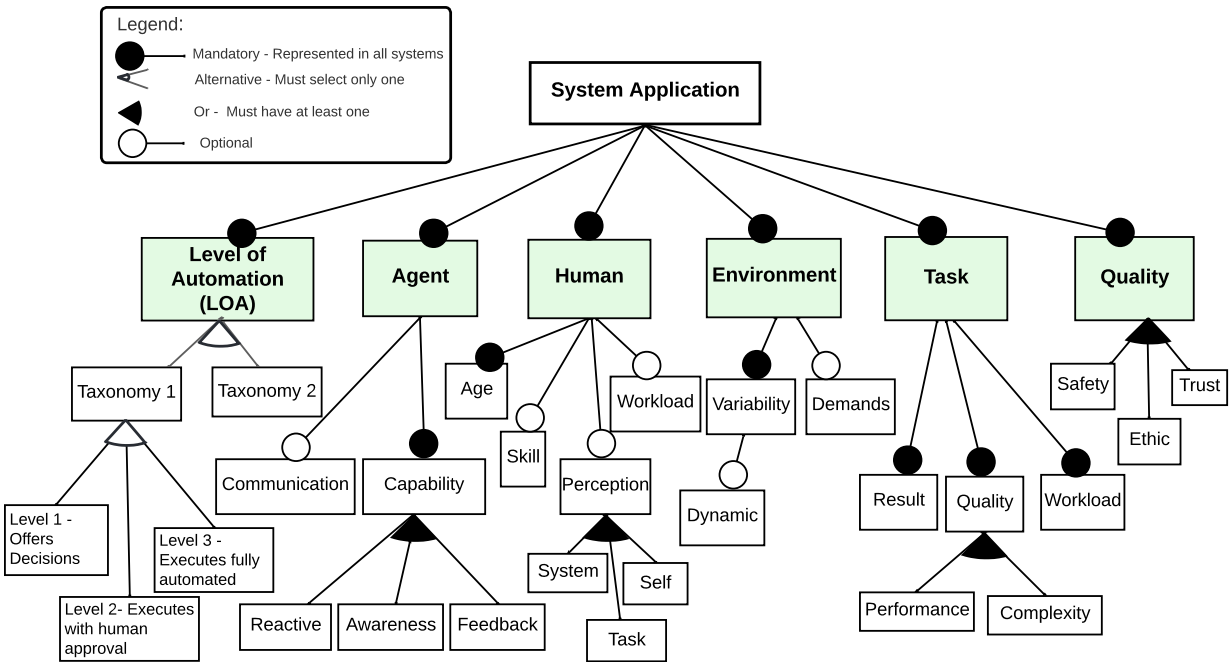

Figure 6.3: LOA Variability Model in Autonomous Systems: An Adaptable Feature Diagram.

The factors that were identified in our literature review are shown in this diagram. For the sake of simplicity, not every identified factor is presented in the diagram. We illustrate the primary factors: Agent, Task, Human, Environment and Quality. We added the representation of the LOA taxonomy, which is also affected by the variability of LOAs. The LOA and the primary factors are mandatory features, making them required to design adaptive automation systems. As shown in Figure 6.3, our feature

diagram specifies some rules that must be respected independently of the application and the taxonomy, such as 1) for every task, it is necessary to specify at least one quality criterion; 2) every agent must have at least one capability, such as reactivity and awareness; and 3) every human must have a role in the system. In this approach, researchers can use our model regardless of the LOAs taxonomy. Additional rules to control the relationship between the factors will be dynamically loaded based on the selected taxonomy.

Feature diagrams can have constraints associated with them. In our case, we can use at least three types of constraints. The first type of constraint is like the ones we have shown in the previous section. This type of constraint represents how the features (or factors) can impact the LOA and can be represented in general as expressions of the form:

$$\Sigma \Rightarrow \textit{Level X}$$

where Σ is an expression involving one or more features. An example of the constraints represented in the previous section is:

$$\textit{High Transparency} \wedge \textit{High Traceability} \wedge \textit{High Predictability} \Rightarrow \textit{Level 1}$$

The second type of constraint represents how the LOA can impact the agent behavior, that is, given an LOA, specific agent capabilities can be provided. This constraint can be represented in general by:

$$\textit{Level X} \Rightarrow \Theta$$

where Θ is an expression involving one or more features. An example of this type of constraint is:

$$\textit{Level 3} \Rightarrow \textit{Detect warning signs} \wedge \textit{Inform warning signs}$$

The third type of constraint represents how some features can impact other features and can be represented by:

$$\Sigma \Rightarrow \Theta$$

where Σ and Θ are each expressions involving one or more features.

6.6.1 Instantiating and Demonstrating Variabilities

This section presents examples of the proposed LOA factors model's instantiation in three domains. The purpose of instantiation is to clarify the model's purpose and rationale and the complexity of the interaction between factors. The first example presents a scenario in the development of automated vehicles, the second is an example of customer service chatbots, and the third is an instantiation of a stock trading chatbot.

6.6.2 Scenario A: Automated Vehicles

In the development of Autonomous Vehicles (AV), vehicles can execute a broad range of tasks without human intervention or partial intervention, such as controlling the car's speed and switching lanes [143]. There has yet to be a consensus about the complete autonomy of these vehicles, as some researchers have proposed strategies for controlling their LOA according to their location (e.g., highway, commercial street, residential street), application concerns (e.g., safety, security, improvement in fuel economy); or even to the driving style (e.g., aggressive, normal, calm). Recent research has investigated enhancing the control and decision-making capabilities of autonomous or semi-autonomous vehicles, enabling them to navigate their trajectories efficiently while avoiding obstacles [16, 46, 34].

Ribeiro et al. [143] present a literature review about the requirements involved in the development of AVs, identifying different types of autonomous vehicles with varying levels of autonomy. Based on this literature review, we identified and classified the factors that can make AVs assume different degrees of autonomy. Further, we represent these factors as features, making it possible to investigate and model their dependencies.

- Quality
  - Safety: Because of the auditing and certification process, there are a set of safety-related ISO standards usually addressed by AVs, such as ISO 26262, which handles possible hazards caused by the malfunctioning behavior of electrical or electronic systems.
  - Security: protection against cyber-attacks which can expose personal information on other connected devices.

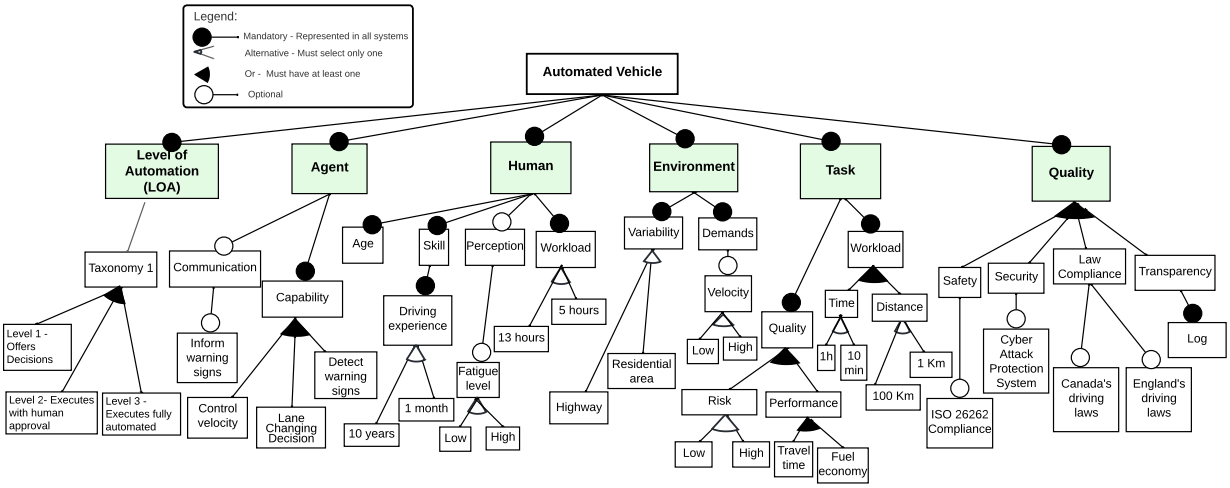

Figure 6.4: Scenario A: LOA Variability Model in Autonomous Vehicles.

  - Usability: functions that facilitate the interaction of the user with automated functions, autonomous taxis, or family vehicles (for children, the elderly, or people with disabilities).
  - Accessibility: A car that can be operated independently, even by those who cannot drive a conventional vehicle.
  - Law Compliance: complying with federal and state laws in a specific region.
  - Transparency: the system requires transparency in the process owing to the possibility of an accident or similar situation that needs verification.
  - Trust: Society and government that create and monitor regulations, as well as the driver or passenger, must be able to trust a vehicle.
  - Environmental impact: the driving style can impact vehicle emissions and energy consumption.

- Task
  - Quality:
    * Risk: driving has some eminent risks that vary according to the environment (e.g., the risks involved in a residential area differ from the risks on the highway; in some regions, the risks vary with the weather).

    - Performance: the performance of the task may be measured based on a reduction in travel time, traffic deaths, or exhaust emissions, or an improvement in fuel economy.
  - Workload: the driving activity may be associated with some workers, such as taxi and truck drivers. Thus, the task's workload may vary according to the distance and local information, such as local traffic or weather.

- Agent
  - Communication: some AVs have heads-up displays to show information to the driver. Vehicles should be able to communicate with other vehicles.
  - Safety: protection against faults at the system level, including hardware and software.
  - Capability: AVs may have many behaviors, such as controlling the car's velocity, lane change warnings, and obstacle avoidance.

- Human
  - Skill: time of driving experience measured through the driving license years.
  - Workload: The hours a driver can drive in a day vary according to local laws. In Canada, for example, a driver can only drive up to 13 hours daily.
  - Perception (Reliability): Fatigue level.

- Environment
  - Variability: The environment has dynamic properties related to mobile objects, such as pedestrians and other cars, and static properties related to roads, traffic signs, and weather [3]. Therefore, the vehicle needs to be aware of its environment.
  - Demands: Federal and state laws demand different safety requirements, and the weather, road type, and warning signs demand different driving behavior. For example, highways demand high speed, while residential areas require low speed. However, even on a highway, a crossing sign warns drivers to slow down and be prepared to stop.

After identifying the factors impacting a vehicle's levels of autonomy, we present a feature model in Figure 6.4 to explore designing AVs with different LOAs. Based on

this figure, we show below some examples of how these factors can impact the LOA and how the LOA can impact the agent behavior:

Residential area $\Rightarrow$ *Level 1 (Offers Decisions)*

Highway $\wedge$ *Low risk* $\Rightarrow$ *Level 3 (Fully Automated)*

Highway $\wedge$ *High risk* $\Rightarrow$ *Level 2 (Executes with Human Approval)*

Highway $\wedge$ *High risk* $\wedge$ *High fatigue* $\Rightarrow$ *Level 1 (Offers Decisions)*

Level 1 $\Rightarrow$ *Detect warning signs* $\wedge$ *Inform warning signs*

Level 3 $\Rightarrow$ *Control speed* $\wedge$ *Decide about lane changing*

Based on these dependencies, we can consider vehicles that can assume more than one degree of autonomy, selecting the most appropriate level for each situation. For example, a car controls the velocity on highways but returns control to the driver when it approaches residential areas. In such an example, for residential areas, the vehicle will have a lower LOA, making the driver responsible for the speed control. At this lower level, the car cannot control its speed, but it can provide information about warning signs, such as school crossing and speed limit warning signs.

6.6.3 Scenario B: Customer Service Chatbots

Chatbots can significantly support business operations. For example, in interactions with customers, 24/7 availability and machine learning capabilities can provide customers with automatically generated personalized responses based on their needs and hopefully resolve issues faster [44]. Customer service chatbots can replace FAQs, provide extensive information about a product, schedule appointments automatically and perform many other useful functions. Chatbots can set up and change customer appointments for all business types, from healthcare organizations to home maintenance companies. Chatbots are connected to the company's calendar and can educate customers about personnel availability and available timeslots, enabling them to make ap-

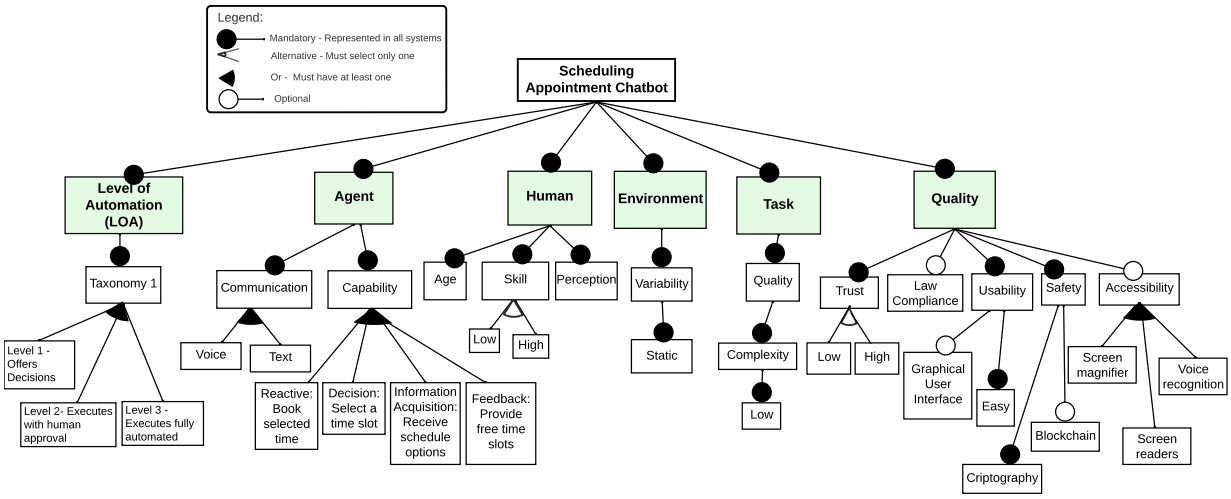

Figure 6.5: Scenario B: LOA Variability Model in Customer Services Chatbots.

pointments without contacting humans at "front desks." According to a 2019 survey[41], customer service chatbots must be equipped with the following:

- Ability to provide personalized responses to each customer regardless of whether it is a FAQ.
- Understand the customer's context.
- Provide real-time insights to agents to resolve inquiries quickly.
- Understand the value of the customer and their history of transactions/interactions with the company.
- Identify actions based on customer responses.
- Lead users through an automated dialogue to clarify the intent.

Considering a scheduling appointment chatbot as our second use case and automatically booking an appointment as the fully autonomous response of these systems, the following factors would influence the LOA of such systems:

- Quality

  - Trust: Users must trust the systems, however not as much as health care system patients.
  - Ethics and Law Compliance: No law compliance is needed to book appointments, however, patient data should be secure in case the system is scheduling medical appointments, for example.
  - Usability: Interfaces must be comprehensive and easy to use, as this system will be used by non-technical users.
  - Safety: Safety of the system is important; however, it does not need to be the reason for high investments.
- Task
  - Complexity: This system does not deal with complex tasks.
- Agent
  - Communication: text- or voice-based chatbots.
  - Safety: standard data protection suffices.
  - Capability: to understand the client's schedule request.
  - Domain: no domain-specific requirement to book appointments.
- Human
  - Age: the system might need adaptation for the elderly, accessibility.
  - Skill: Users do not need technical skills to interact with this system.
  - Perception (Reliability): Users must rely on the system.
- Environment
  - Variability: The system does not need to be aware of environmental changes and, therefore, can potentially be static.

We mapped these factors as features in Figure 6.5. Handling different feature combinations, we can explore some relations between the LOA and the behavior of the scheduling appointment chatbots, as follows:

Level 1 $\Rightarrow$ Provide free time slots to the user $\wedge$ Book the selected time.

Level 2 ⇒ Select a time slot for the user ∧ Book a time slot after the user's approval.

Level 3 ⇒ Select a time slot ∧ Book a time slot for the user.

In the same way, we can also explore some of the characteristics that the system must have to accomplish the different LOAs:

Feedback (provide free time slots) ∧ Book selected time ∧ Graphical User Interface ⇒ Level 1

Decision (Select a time slot) ∧ Receive user's approval ∧ Graphical User Interface ⇒ Level 2

Decision (Select a time slot) ∧ Book selected time ∧ Trust (High) ⇒ Level 3

As shown, if the system operates at levels one or two, the interaction between the agent and the human is higher, so the system must provide a graphical user interface to meet the mandatory requirement of easy usability. In the case of having a chatbot operating at the highest level, a robust interface is unnecessary (e.g., a command line interface), as the chatbot can make decisions and select the best time for the user autonomously. On the other hand, the level of trust in the system needs to be higher.

6.6.4 Scenario C: Stock Trading Chatbot

This example serves as another tangible demonstration of the proposed integration of the proposed feature model into a specific domain, showcasing its practical applicability. Chatbots are gaining ground in the financial industry, aiding in various sectors such as banking, insurance, and stock trading [72][180]. Mastercard's AI Assist, Bank of America's Erica, and KB's Liiv TalkTalk are some examples, of performing tasks from balance inquiries and transaction management to insurance assessments and stock consultations. Zhang et al. [180] introduce a trader chatbot that can not only manage the user's portfolio but also employ natural language to engage in negotiations with external traders.

However, the interaction of chatbots and humans faces obstacles. User acceptance, heavily influenced by age, is critical, with millennials favouring digital interactions and

older users leaning toward human interfaces [72]. Addressing security concerns is essential given the sensitivity of financial data, making it harder to develop advanced chatbots that can do more than just provide guidance. Moreover, compliance with local regulations is crucial, underlining the need for adaptable chatbot designs. Therefore, developing a chatbot solution that can be customized according to different factors - user acceptance, age demographics, cybersecurity requirements, and regulatory stipulations - is critical for successful implementation in the financial industry.

We base the identified factors in this example on the feature model of LOA factors proposed this chapter, as presented in Figure 6.3.

This illustrative example centers on a stock trading chatbot. Not only does it provide traditional services such as financial recommendations, historic data, and stock counselling, as suggested by Jang et al. [72], but it also includes advanced features similar to Zhang et al. [180]. These features include proactive portfolio management, trader negotiations, and market predictions.

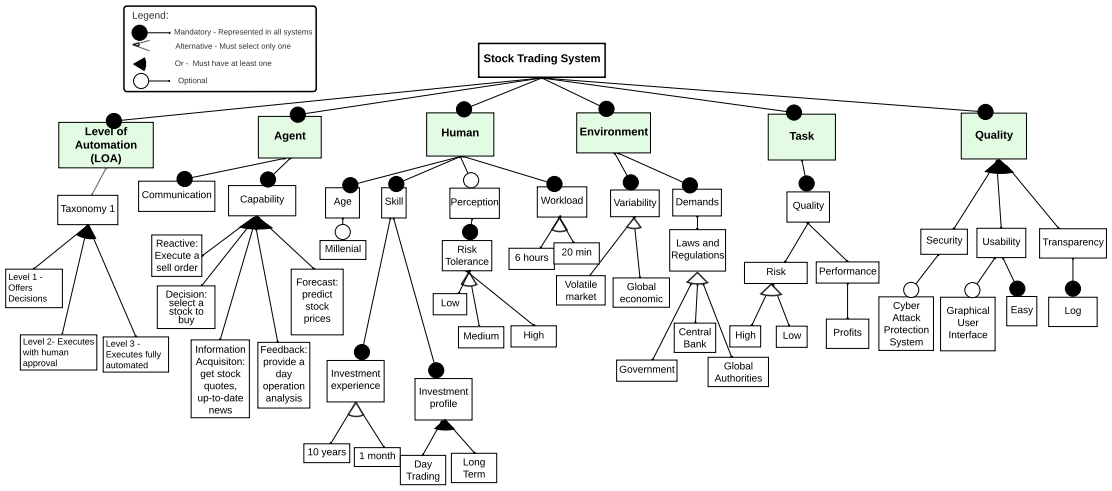

Figure 6.6: Scenario C: LOA Variability Model in Chatbot for Stock Trading.

To develop this stock trading chatbot, we first need to define the system's requirements, examine the contextual factors in this domain application that can impact the LOA, and understand the interrelationships among these factors as follows:

- **Application Requirements:** In extending this step for our financial chatbot, we've taken into account the insights from the works of Jang et al. [72], which offers an

array of chatbot solutions for financial services, and Zhang et al. [180], providing a more sophisticated approach. These studies give us crucial considerations for designing the LOA feature diagram. In addition to LOA, the application requirements also influence the development of the User Interface and Service Integration components. For instance, the need for a user-friendly, intuitive interface would guide the UI design, while service requirements such as integrating forecast algorithms and real-time news services would shape the Service Integration component.

- **LOA Requirements:** To establish the required automation level for stock trading, we consider factors like various trading scenarios, user profiles, and interaction objectives (e.g., stock advice or portfolio management). User investment knowledge, market analysis complexity, and platform quality also contribute to shaping the desired LOA.

- **LOA Contextual factors:** In determining the LOA for stock trading chatbots, we examine various contextual factors. These span dimensions such as quality (trustworthiness, usability), task (data processing load), agent (communication abilities), human (user's financial expertise), and environment (market volatility). Converting these factors into features allows us to assess their impact on the LOA methodically.

- **LOA Feature Diagram:** To adapt the feature-oriented design approach (Figure 6.3) for systems with LOA variability, we created a feature diagram specifically for the stock trading application, as shown in Figure 6.6. For instance, the chatbot's capabilities could vary from simply executing a user-requested sell order to predicting future stock prices. The users can also have variable characteristics, differing in their age, trading experience, and dedication (like professional traders). Furthermore, we need to take into account environmental factors, such as market volatility, global economics, and various laws and regulations. Once these factors are identified, we can delve into their interdependencies. This involves understanding how the inclusion or exclusion of certain features can affect the LOA. For example, a novice user may prefer less automation, when interacting with a system that requires user approval for order execution. Conversely, a more experienced user with limited time might benefit from a chatbot capable of autonomous decision-making and task execution, which provides a summary of activities at the day's end.

After these three steps, we can proceed with the design of the application, expanding

the three components of the feature-oriented design:

- **User Interface (UI):** A refined UI enables the chatbot to understand financial jargon and manage complex dialogues. Depending on the chatbot's automation level, the interface adjusts accordingly, simpler for a more autonomous chatbot and more intuitive for a chatbot requiring human approval.
- **Service Integration (SI):** This component connects with financial databases, trading platforms, and news feeds, enabling the chatbot to retrieve and process real-time data. It also manages internal services like trade execution within the user's portfolio.
- **Level of Automation Autonomic Controller (LOAAC):** This component adjusts the chatbot's automation level considering user factors and task complexity, from basic tasks for beginners to advanced portfolio management for experienced users.

In creating the LOAAC, we consider the five key factors described in [106]:

- **Quality:** The bot needs to be reliable, accurate, and trustworthy, capable of providing clear advice. It must also be adaptable to different user profiles and legal requirements related to stock trading.
- **Task:** Tasks in stock trading vary from simple tasks like showing current stock prices, to complex ones such as suggesting portfolio adjustments based on market changes. The complexity, frequency, and risk associated with these tasks would influence the level of autonomy the chatbot should have. For instance, more complex and riskier tasks might require a lower LOA to allow user validation.
- **Agent (System):** The chatbot needs strong communication skills. Its ability to react to different market situations, provide relevant feedback, and make informed decisions based on current data influences the LOA.
- **Human:** The user's age, cognitive ability, and perception of the bot will determine its automation level. For instance, younger users well-versed in technology might prefer a high LOA, while older users might require more control. Moreover, the user's trust in the system and their risk tolerance also influence the LOA.
- **Environment:** The chatbot must be adaptable to sudden market changes and be able to handle competing tasks simultaneously, such as monitoring multiple stocks and providing updates to different users.

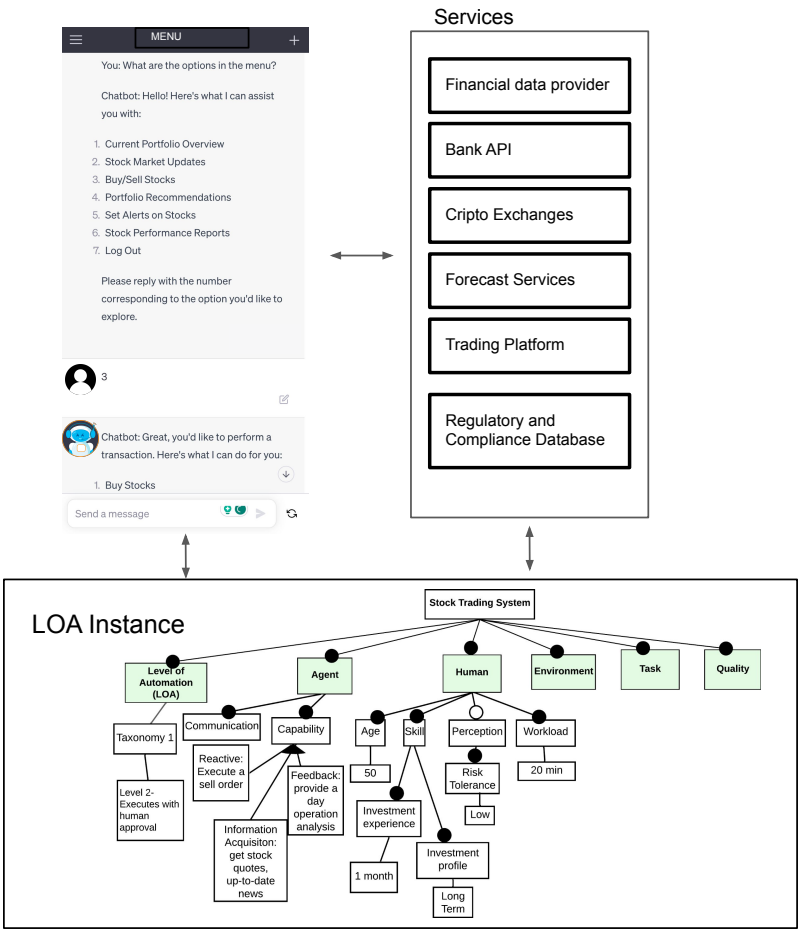

Figure 6.7: An illustration of deploying the application for novice users.

Figure 6.7 illustrates a Level 2 Autonomy system operating with human approval, catering to less experienced users with low-risk tolerance. In this configuration, the chatbot is not authorized to trade stocks independently but is limited to executing users' orders. The user interface is thoughtfully adapted to align with this autonomy level, offering options that correspond to the chatbot's capabilities. In the depicted scenario (Figure 6.7), the service's availability can be programmable in the chatbot's backend, and the instances will be generated according to the combination of factors and LOA. In other words, the chatbot instance generated will depend on the agent, human, environment, tasks and system quality factors. For instance, users with less experience in trading may have a more automated platform that simplifies the stock purchasing

process, while more seasoned traders would be granted access to an instance of the application that allows them to issue stock purchases manually. This adaptive approach aims to optimize the user experience and aligns with the objectives of the autonomy level variability system, striving to strike a balance between human and autonomous functionalities.

6.7 Discussion

To date, studies that outline factors that affect automation-level decisions are scarce. However, scholars have attempted to identify various levels of automated taxonomies, each having a particular function. These definitions can extract the factors that authors use to determine the LOA at which the system and its specific tasks should operate. These taxonomies often cover a spectrum spanning from fully manual to completely autonomous.

Regarding the identified factors, quality factors are mentioned, as well as system and task factors. For the system, quality is usually acquired with testing and process verification. Task quality factors are different in that they are related to the performance of the execution of a task or the task's complexity. Situation awareness is also related either to a task or a system. While a person can be "situationally aware" through knowledge, experience, and human sensory and decision-making abilities, a system can be "situationally aware" through sensors and contextual information. Likewise, according to Villani et al. [173], demographics including age, are essential factors since they allow, for instance, customizations specifically targeting elderly or inexperienced system operators, supporting them to achieve tasks they would otherwise be unable to perform.

These findings support understanding the factors determining whether tasks should be more or less automated and to what extent. Therefore, it is conceivable to assume that systems that anticipate various LOAs might also be built to recognize the variables that impact the amount of automation and adapt as necessary. For example, an autonomous car has categories for how aggressive the driving might be. This determines, for instance, how distant another car must be for the autonomous car to merge into a lane. The more aggressive the driving mode is, the shorter the distance between the autonomous car and the other car when merging lanes. Although the action to merge lanes is autonomous, the user (driver) must manually set the aggressive mode category. Suppose we were to analyze the factors that could influence the driving mode, such as weather, total driving distance, drivers' agendas (how fast they need to get somewhere),

who is driving the car or even the landscape of the road. In that case, the driving mode could be set automatically without manual input for most of these factors.

The interaction between humans and autonomous systems is a multifaceted domain with complex considerations. One consideration is the importance of metrics, such as precision and recall in Natural Language processing tools. In the context of autonomous systems, precision equates to how well a system can accurately perform a task, while recall relates to when the system was efficient, doing the right thing when it should have done it right. According to the perspectives of Berry et al. [17], sometimes, it is more efficient to have a less intelligent tool handle a specific part of a task, rather than a highly intelligent tool failing to perform the entire task. This perspective suggests that the quality of an autonomous system plays a pivotal role in determining the LOA required. Still considering precision and recall, however this time mentioning a human factor as attentiveness, the work of DiMatteo et al. [45] introduces the critical element of human attentiveness in partially autonomous vehicles (AVs). The authors claim that as the LOA increases, humans tend to become less attentive. This implies that the human factor, including factors like fatigue, can substantially impact the appropriate LOA. Balancing false positives and false negatives in detecting inattention becomes crucial. Here, the system quality and human factors interplay as the system needs to adapt to the human's attention state. Achieving the right balance is essential to optimize system performance while ensuring human safety and efficiency.

The development of a software engineering discipline that addresses autonomous system design and identifies factors that influence the variance of LOAs in autonomous systems could potentially significantly impact the field of autonomous systems and artificial intelligence research. This could lead to more efficient and effective design and development of autonomous systems' software and better understanding and management of the relationship between humans and machines in such systems. The findings suggest that a software engineering approach that considers the factors determining the appropriate LOA for a task can be beneficial. Such an approach would allow for the development of systems anticipating various LOAs and recognizing the variables that impact the amount of automation required. This approach can lead to the development of autonomous systems that can adapt to changing conditions and user preferences, thus increasing their effectiveness and efficiency. The work of Berry et al. [18] delves into the domain of AI and how requirements for AI can be established. This work emphasizes the importance of defining measures, acceptable values, and the AI's context to create a comprehensive requirements specification (RS). In the context of human-autonomous system interaction, understanding the context and defining appropriate measures are essential in deciding the LOA that aligns with human capabilities and expectations.

In AI, our proposed approach can lead to the development of AI systems that can learn from experience and adapt to changing conditions. By considering the factors that influence the appropriate LOA for a task, AI systems can be designed to be more flexible and adaptable. Additionally, this approach can lead to the development of AI systems that can be designed to operate at different LOAs depending on the context, thus increasing their usability and applicability.

6.8 Threats to Validity

In this section, we address potential threats to the validity of the research methodology employed in the systematic literature review (SLR) to identify factors influencing LOAs in systems that support human-machine interactions.

- Construct Validity:
  - Scope of Inclusion Criteria: The inclusion criteria focused on studies published in English after the year 2000. This scope might exclude relevant studies or factors from earlier research or non-English literature.
  - Categorization Bias: The categorization of factors into abstract concepts could introduce bias or subjectivity. The lack of a clear justification for this categorization may lead to potential misunderstandings or misinterpretations.
- Internal Validity:
  - Literature Sampling: The LR primarily relies on backward and forward snowballing and online databases for literature sampling. This approach may not capture all relevant studies, leading to potential omissions of important factors.
  - Inter-Rater Reliability: While the study reports high agreement between the two researchers categorizing the identified factors, potential subjectivity in categorization could introduce biases or inconsistencies.
- External Validity:
  - Generalizability: The focus on papers published after 2000 and in English may limit the generalizability of findings to a broader range of factors and studies that existed before this time frame or in other languages.

6.9 Conclusion

In this study, we aimed to answer the following research question: Which factors affect the variance of LOAs in autonomous systems? We performed a systematic literature review to identify the factors related to varying LOAs. We describe the methodology of the approach and list all the identified factors in a table, linked with their corresponding source article(s). We then provide a categorization of the identified factors that affect LOAs in systems. We categorize these factors into five main categories: Quality, Task, Agent (System), Human and Environment factors. To continue the work, we refine these factors by demonstrating how systems can capture and embed them in their operation. We also introduce a representation of these factors and their variability, demonstrating the relationship of factors with specific LOAs. Lastly, we demonstrate how these factors can be applied to three different scenarios with illustrative examples. Prior studies have recognized the value of research into the definition of the taxonomies of LOAs. This research complements taxonomy research, by investigating the factors that affect the choice of one LOA over another, contributing to the development of adaptive autonomous systems independently of the LOA taxonomy being used.

The present results are significant in at least two major respects. First, it highlights the existence of these factors, categorizes them and presents how a combination of different factors can influence how intelligent autonomous systems work. We also demonstrate how systems can capture factors and systemically implement such factors. Finally, we achieve system adaptability and characterization by identifying and representing these factors as feature models.

Our work also raises questions for future research. One of these questions concerns the challenges of implementing these factors in autonomous systems. How can these factors be implemented in autonomous systems while still ensuring that the systems remain safe and reliable? Another important area of research is validating the proposed approach by implementing real-world autonomous systems and testing them under various conditions. Future work can explore using the proposed model to assess the automation of problems in specific fields, such as software engineering and other domains. We also aim to implement this model, investigate its contribution to different application scenarios in conversational agent architectures, and investigate which taxonomies are ideal for conversational agent solutions. Keeping a history of the factors that impact specific tasks can also be used to measure the performance of the task and evaluate the selection of the LOA. Additionally, such models could potentially contribute to proving the decisions made.

In conclusion, this study contributes to the development of adaptive autonomous systems by identifying and categorizing the factors that affect the choice of one LOA over another. By using the identified factors and their variability, software engineers can design autonomous systems that can operate more effectively in various conditions. The results of this study have practical implications for the development of autonomous systems and provide a foundation for future research in this field.

Chapter 7

Conclusion

The combination of heightened demand for software developers, the growing complexity of software development projects, the dynamic nature of the field, and the collaborative aspect of team-based development are all key factors shaping the way developers work. Additionally, the introduction of AI-based conversational systems, in the form of chatbots and virtual assistants, represents a promising avenue for developers to navigate this evolving landscape more effectively and efficiently.

In this thesis, we embarked on a research journey to investigate the design of context-based adaptive interactions between software developers and chatbots, with a particular focus on understanding developers' expectations and the desired levels of automation. This exploration was driven by the increasing complexity of software development and the emergence of advanced AI-based conversational systems, which hold the potential to transform how developers work. We aimed to bridge the gap between developers' needs and the capabilities of chatbots, ultimately seeking to provide optimal support for software development processes.

We made contributions to both the theoretical and practical aspects of context-based chatbot interactions. We conduct literature reviews to identify contextual factors in software engineering (RQ1) and factors influencing LOAs in autonomous systems (RQ3). We categorized these factors and analyzed their relationships, shedding light on how they impact LOAs.

Our empirical studies provided insights into developers' perspectives and requirements when interacting with chatbots (RQ2). Through a set of user studies, we extracted essential information on how chatbots can best support developers in their tasks. We discovered that developers are interested in chatbot support for tasks such as task and

repository management, but their preferences for guidance or automation vary. Additionally, we found that context plays a crucial role in enhancing chatbot interactions, as developers desire a deeper contextual understanding of these virtual assistants. Furthermore, we explored the relationships between developers' automation preferences and their experience levels, highlighting the importance of tailoring chatbot responses to individual skill levels.

In our thesis statement, we claim that the integration of chatbots to support developers in their work has become prominent and that there is a pressing need to address the challenges in human-chatbot interactions, which has been addressed throughout this research. We have explored the complexities of context-based chatbot interactions, gaining a deeper understanding of the contextual factors that influence software development and the impact of automation on human-system interactions. By conducting systematic literature reviews, and user studies, we have contributed to the development of effective solutions and knowledge that optimally support software developers.

Looking ahead, there are several avenues for future research in this domain. Further investigations can explore the integration of context into developer-chatbot interactions in more depth, considering specific support mechanisms that chatbots should offer to developers. Additionally, the quality of chatbot recommendations and their impact on task performance can be assessed through qualitative and quantitative studies. As advanced language models like ChatGPT become increasingly prevalent, the possibilities for enriching human-machine collaboration in software development hold promise and should remain a focus of continued investigation. An interesting study can also evaluate the quality of software with the same requirements being developed with the support of a chatbot, and without. Of course, many avenues to implement chatbots for software developers with the requirements presented in this thesis are also open.

In conclusion, our research has paved the way for more context-aware and personalized support for software development, harnessing the power of chatbots and AI advancements to address complexities in the software development process. As the software development landscape continues to evolve, our findings and contributions can be considered, guiding the development of adaptive developer-chatbot interactions in the years to come.

References

[1] Ahmad Abdellatif, Khaled Badran, Diego Elias Costa, and Emad Shihab. A comparison of natural language understanding platforms for chatbots in software engineering. *IEEE Transactions on Software Engineering*, 48(8):3087–3102, 2022.

[2] Ahmad Abdellatif, Diego Costa, Khaled Badran, Rabe Abdalkareem, and Emad Shihab. Challenges in chatbot development: A study of stack overflow posts. In *Proceedings of the 17th International Conference on Mining Software Repositories*, MSR '20, page 174–185, New York, NY, USA, 2020. Association for Computing Machinery.

[3] Raja Ben Abdessalem, Annibale Panichella, Shiva Nejati, Lionel C Briand, and Thomas Stifter. Testing autonomous cars for feature interaction failures using many-objective search. In *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 143–154. IEEE, 2018.

[4] S. Ahmadi-Karvigh, A. Ghahramani, B. Becerik-Gerber, and L. Soibelman. One size does not fit all: Understanding user preferences for building automation systems. *Energy and Buildings*, 145:163–173, 2017. cited By 30.

[5] H. Ait Malek, A. Etienne, A. Siadat, and T. Allavena. A literature review on the level of automation and new approach proposal. *IFIP Advances in Information and Communication Technology*, 591 IFIP:408–417, 2020. cited By 0.

[6] Bowen Alpern, Alan Carle, Barry Rosen, Peter Sweeney, and Kenneth Zadeck. Graph attribution as a specification paradigm. *ACM SIGPlan Notices*, 24(2):121–129, 1988.

[7] Vander Alves, Nan Niu, Carina Alves, and George Valença. Requirements engineering for software product lines: A systematic literature review. *Information and Software Technology*, 52(8):806–820, 2010.

[8] Bruno Antunes, Francisco Correia, and Paulo Gomes. Context capture in software development. *arXiv prmisc arXiv:1101.4101*, 2011.

[9] Johan Aronsson, Philip Lu, Daniel Strüber, and Thorsten Berger. A maturity assessment framework for conversational ai development platforms. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, pages 1736–1745, 2021.

[10] B. Ashok, Joseph Joy, Hongkang Liang, Sriram K. Rajamani, Gopal Srinivasa, and Vipindeep Vangala. Debugadvisor: A recommender system for debugging. In *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC/FSE '09, pages 373–382, New York, NY, USA, 2009. ACM.

[11] Noppadol Assavakamhaenghan, Raula Gaikovina Kula, and Kenichi Matsumoto. Interactive chatbots for software engineering: A case study of code reviewer recommendation. In *2021 IEEE/ACIS 22nd International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 262–266, 2021.

[12] Samira Badrloo, Masood Varshosaz, Saied Pirasteh, and Jonathan Li. Image-based obstacle detection methods for the safe navigation of unmanned vehicles: A review. *Remote Sensing*, 14(15):3824, 2022.

[13] Paul L Baker. Ada as a preprocessor language. *ACM SIGAda Ada Letters*, 10(1):83–91, 1990.

[14] José L. Barros-Justo, Fabiane B. V. Benitti, and Ania L. Cravero-Leal. Software patterns and requirements engineering activities in real-world settings: A systematic mapping study, 2018. ID: 271914.

[15] Jenay M Beer, Arthur D Fisk, and Wendy A Rogers. Toward a framework for levels of robot autonomy in human-robot interaction. *Journal of human-robot interaction*, 3(2):74, 2014.

[16] Cinzia Bernardeschi, Pierpaolo Dini, Andrea Domenici, Ayoub Mouhagir, Maurizio Palmieri, Sergio Saponara, Tanguy Sassolas, and Lilia Zaourar. Co-simulation of a model predictive control system for automotive applications. In *Software Engineering and Formal Methods. SEFM 2021 Collocated Workshops: CIFMA, CoSim-CPS, OpenCERT, ASYDE, Virtual Event, December 6–10, 2021*, pages 204–220. Springer, 2022.

[17] Daniel Berry, Ricardo Gacitua, Pete Sawyer, and Sri Fatimah Tjong. The case for dumb requirements engineering tools. In *Requirements Engineering: Foundation for Software Quality: 18th International Working Conference, REFSQ 2012, Essen, Germany, March 19-22, 2012. Proceedings 18*, pages 211–217. Springer, 2012.

[18] Daniel M Berry. Requirements engineering for artificial intelligence: What is a requirements specification for an artificial intelligence? In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 19–25. Springer, 2022.

[19] U.S. BLS. Bureau of labor statistics, u.s. department of labor, occupational outlook handbook: Software developers, quality assurance analysts, and testers. Accessed: 2022-11-06.

[20] Nick Bradley, Thomas Fritz, and Reid Holmes. Context-aware conversational developer assistants. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 993–1003, 2018.

[21] Petter Bae Brandtzaeg and Asbjørn Følstad. Why people use chatbots. In Ioannis Kompatsiaris, Jonathan Cave, Anna Satsiou, Georg Carle, Antonella Passani, Efstratios Kontopoulos, Sotiris Diplaris, and Donald McMillan, editors, *Internet Science*, page 377–392. Springer International Publishing, 2017.

[22] Chris Brown and Chris Parnin. Sorry to bother you again: Developer recommendation choice architectures for designing effective bots. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, pages 56–60, 2020.

[23] Liang Cai, Haoye Wang, Bowen Xu, Qiao Huang, Xin Xia, David Lo, and Zhenchang Xing. Answerbot: an answer summary generation tool based on stack overflow. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1134–1138, 2019.

[24] G. Calhoun. Adaptable (not adaptive) automation: Forefront of human–automation teaming. *Human Factors*, 64(2):269–277, 2022.

[25] Antonio Carzaniga, Alfonso Fuggetta, Richard S Hall, Dennis Heimbigner, André Van Der Hoek, and Alexander L Wolf. A characterization framework for software deployment technologies. Technical report, Colorado State Univ Fort Collins Dept of Computer Science, 1998.

[26] Cristiano Castelfranchi and Rino Falcone. Founding autonomy: The dialectics between (social) environment and agent's architecture and powers. In *International Workshop on Computational Autonomy*, pages 40–54. Springer, 2003.

[27] Marcelo Cataldo and James D Herbsleb. End-to-end features as meta-entities for enabling coordination in geographically distributed software development. In *2009 ICSE Workshop on Software Development Governance*, pages 21–26. IEEE, 2009.

[28] Jhonny Cerezo, Juraj Kubelka, Romain Robbes, and Alexandre Bergel. Building an expert recommender chatbot. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, pages 59–63. IEEE, 2019.

[29] 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*, 37(8):729–758, 2021.

[30] Lianping Chen, Muhammad Ali Babar, and Nour Ali. Variability management in software product lines: a systematic review. In *Proceedings of the 13th International Software Product Line Conference*, pages 81–90. Citeseer, 2009.

[31] Lianping Chen and Muhammad Ali Babar. A systematic review of evaluation of variability management approaches in software product lines. *Information and Software Technology*, 53(4):344–362, 2011.

[32] Leon Ciechanowski, Aleksandra Przegalinska, Mikolaj Magnuski, and Peter Gloor. In the shades of the uncanny valley: An experimental study of human–chatbot interaction. *Future Generation Computer Systems*, 92:539–548, 2019.

[33] Bruce T Clough. Metrics, schmetrics! how the heck do you determine a uav's autonomy anyway. Technical report, Air Force Research Lab Wright-Patterson AFB OH, 2002.

[34] Francesco Cosimi, Pierpaolo Dini, Sandro Giannetti, Matteo Petrelli, and Sergio Saponara. Analysis and design of a non-linear mpc algorithm for vehicle trajectory tracking and obstacle avoidance. In *Applications in Electronics Pervading Industry, Environment and Society: ApplePies, 2020*, pages 229–234. Springer, 2021.

[35] Krzysztof Czarnecki. Overview of generative software development. In *International workshop on unconventional programming paradigms*, pages 326–341. Springer, 2004.

[36] Krzysztof Czarnecki. Variability in software: State of the art and future directions. In *International Conference on Fundamental Approaches to Software Engineering*, pages 1–5. Springer, 2013.

[37] Krzysztof Czarnecki and Simon Helsen. Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3):621–645, 2006.

[38] Leandro Ferreira D'Avila, Jorge Luis Victória Barbosa, and Kleinner Silva Farias de Oliveira. Sw-context: a model to improve developers' situational awareness. *IET Software*, 14(5):535–543, 2020.

[39] Kleber Rocha de Oliveira and Mauro de Mesquita Spínola. Porei: patterns-oriented requirements elicitation integrated–proposal of a metamodel patterns-oriented for integration of the requirement elicitation process. In *Proceedings of the 2007 Euro American conference on Telematics and Information Systems*, pages 1–8, 2007.

[40] Mustafa Demir, Nathan J McNeese, and Nancy J Cooke. Team situation awareness within the context of human-autonomy teaming. *Cognitive Systems Research*, 46:3–12, 2017.

[41] Statista Research Department. Current chatbot ability in customer service in the united states, canada and u.k. in 2019. https://www.statista.com/statistics/1015841/customer-service-chatbot-ability-us-canada-uk/. Published: 2019-07-06.

[42] Nicolas Devos, Christophe Ponsard, Jean-Christophe Deprez, Renaud Bauvin, Benedicte Moriau, and Guy Anckaerts. Efficient reuse of domain-specific test knowledge: An industrial case in the smart card domain. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 1123–1132. IEEE, 2012.

[43] C. Di Ciccio, A. Marrella, and A. Russo. Knowledge-intensive processes: Characteristics, requirements and analysis of contemporary approaches. *Journal on Data Semantics*, 4(1):29–57, 2015.

[44] Cem Dilmegani. Top 9 customer service chatbot use cases and examples. https://research.aimultiple.com/customer-service-chatbot/. Accessed: 2022-12-19.

[45] Johnathan DiMatteo, Daniel M Berry, and Krzysztof Czarnecki. Requirements for monitoring inattention of the responsible human in an autonomous vehicle: The recall and precision tradeoff. In *REFSQ Workshops*, 2020.

[46] Pierpaolo Dini and Sergio Saponara. Processor-in-the-loop validation of a gradient descent-based model predictive control for assisted driving and obstacles avoidance applications. *IEEE Access*, 10:67958–67975, 2022.

[47] John V Draper. Teleoperators for advanced manufacturing: Applications and human factors challenges. *International Journal of Human Factors in Manufacturing*, 5(1):53–85, 1995.

[48] Tore Dybå, Dag I.K. Sjøberg, and Daniela S. Cruzes. What works for whom, where, when, and why? on the role of context in empirical software engineering. In *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '12, page 19–28, New York, NY, USA, 2012. Association for Computing Machinery.

[49] Mica R Endsley. The application of human factors to the development of expert systems for advanced cockpits. In *Proceedings of the Human Factors Society Annual Meeting*, volume 31, pages 1388–1392. SAGE Publications Sage CA: Los Angeles, CA, 1987.

[50] Mica R Endsley and David B Kaber. Level of automation effects on performance, situation awareness and workload in a dynamic control task. *Ergonomics*, 42(3):462–492, 1999.

[51] Mica R. Endsley and Esin O. Kiris. The out-of-the-loop performance problem and level of control in automation. *Human Factors*, 37(2):381–394, 1995.

[52] M.R. Engsley, E. Onal, and D.B. Kaber. The impact of intermediate levels of automation on situation awareness and performance in dynamic control systems. In *Proceedings of the 1997 IEEE Sixth Conference on Human Factors and Power Plants, 1997. 'Global Perspectives of Human Factors in Power Generation'*, pages 7/7–712, 1997.

[53] Juan Carlos Farah, Basile Spaenlehauer, Vandit Sharma, María Jesús Rodríguez-Triana, Sandy Ingram, and Denis Gillet. Impersonating chatbots in a code review exercise to teach software engineering best practices. In *2022 IEEE Global Engineering Education Conference (EDUCON)*, pages 1634–1642, 2022.

[54] Åsa Fasth. *Quantifying Levels of Automation to Enable Competitive Assembly Systems*. Chalmers Tekniska Hogskola (Sweden), 2012.

[55] Afreza Fereidunian, Caro Lucas, Hamid Lesani, Matti Lehtonen, and Mikael Nordman. Challenges in implementation of human-automation interaction models. In *2007 Mediterranean Conference on Control and Automation*, pages 1–6. IEEE, 2007.

[56] Alireza Fereidunian, Matti Lehtonen, Hamid Lesani, Caro Lucas, and Mikael Nordman. Adaptive autonomy: Smart cooperative cybernetic systems for more humane automation solutions. In *2007 IEEE International Conference on Systems, Man and Cybernetics*, pages 202–207, 2007.

[57] Frank Flemisch, Matthias Heesen, Tobias Hesse, Johann Kelsch, Anna Schieben, and Johannes Beller. Towards a dynamic balance between humans and automation: authority, ability, responsibility and control in shared and cooperative control situations. *Cognition, Technology and Work*, 14(1):3–18, 2012.

[58] Asbjørn Følstad and Petter Bae Brandtzæg. Chatbots and the new world of hci. *interactions*, 24(4):38–42, 2017.

[59] Martin Ford. *Rise of the Robots: Technology and the Threat of a Jobless Future*. Basic Books, 2015.

[60] Matthias Galster, Paris Avgeriou, Danny Weyns, and Tomi Männistö. Variability in software architecture: current practice and challenges. *ACM SIGSOFT Software Engineering Notes*, 36(5):30–32, 2011.

[61] Matthias Galster, Danny Weyns, Dan Tofan, Bartosz Michalik, and Paris Avgeriou. Variability in software systems—a systematic literature review. *IEEE Transactions on Software Engineering*, 40(3):282–306, 2013.

[62] Matthias Galster, Uwe Zdun, Danny Weyns, Rick Rabiser, Bo Zhang, Michael Goedicke, and Gilles Perrouin. Variability and complexity in software design: Towards a research agenda. *ACM SIGSOFT Software Engineering Notes*, 41(6):27–30, 2017.

[63] Marko Gasparic, Gail C. Murphy, and Francesco Ricci. A context model for ide-based recommendation systems. *Journal of Systems and Software*, 128:200–219, Jun 2017.

[64] Allen Goldberg. Reusing software developments. *ACM SIGSOFT Software Engineering Notes*, 15(6):107–119, 1990.

[65] John D. Gould, John Conti, and Todd Hovanyecz. Composing letters with a simulated listening typewriter. *Commun. ACM*, 26(4):295–308, April 1983.

[66] Deep Learning Bible H. Traditional NLP hangeul. Dialogue and conversational agents. https://wikidocs.net/191160, Unknown. Accessed 03. Jul 2023.

[67] Reid Holmes and Gail C. Murphy. Using structural context to recommend source code examples. In *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, page 117–125. ACM, 2005. event-place: St. Louis, MO, USA.

[68] Samuel Idowu, Daniel Strüber, and Thorsten Berger. Asset management in machine learning: a survey. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 51–60. IEEE, 2021.

[69] Nasif Imtiaz, Justin Middleton, Joymallya Chakraborty, Neill Robson, Gina Bai, and Emerson Murphy-Hill. Investigating the effects of gender bias on github. In *Proceedings of the 41st International Conference on Software Engineering*, pages 700–711. IEEE Press, 2019.

[70] Toshiyuki Inagaki. Situation-adaptive degree of automation for system safety. In *Proceedings of 1993 2nd IEEE International Workshop on Robot and Human Communication*, pages 231–236. IEEE, 1993.

[71] Samireh Jalali and Claes Wohlin. Systematic literature studies: database searches vs. backward snowballing. In *Proceedings of the 2012 ACM-IEEE international symposium on empirical software engineering and measurement*, pages 29–38. IEEE, 2012.

[72] Moonkyoung Jang, Yoonhyuk Jung, and Seongcheol Kim. Investigating managers' understanding of chatbots in the korean financial industry. *Computers in Human Behavior*, 120:106747, 2021.

[73] David B Kaber. Issues in human–automation interaction modeling: Presumptive aspects of frameworks of types and levels of automation. *Journal of Cognitive Engineering and Decision Making*, 12(1):7–24, 2018.

[74] Raman Kazhamiakin, Salima Benbernou, Luciano Baresi, Pierluigi Plebani, Maike Uhlig, and Olivier Barais. Adaptation of service-based systems. *Service Research*

Challenges and Solutions for the Future Internet: S-Cube–Towards Engineering, Managing and Adapting Service-Based Systems, pages 117–156, 2010.

[75] J. F. Kelley. An iterative design methodology for user-friendly natural language office information applications. *ACM Trans. Inf. Syst.*, 2(1):26–41, jan 1984.

[76] Mik Kersten and Gail C. Murphy. Mylar: a degree-of-interest model for ides. In *Proceedings of the 4th international conference on Aspect-oriented software development*, pages 159–168. ACM, 2005.

[77] Mik Kersten and Gail C. Murphy. Using task context to improve programmer productivity. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, SIGSOFT '06/FSE-14, page 1–11. ACM, 2006.

[78] Thanh Tung Khuat, David Jacob Kedziora, and Bogdan Gabrys. The roles and modes of human interactions with automated machine learning systems. *arXiv prmisc arXiv:2205.04139*, 2022.

[79] Alison Kidd. The marks are on the knowledge worker. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 186–191, 1994.

[80] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):4909–4926, 2021.

[81] Barbara Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering. *Technical Report EBSE 2007-001*, 2007.

[82] A Kontogogos and P Avgeriou. Towards modelling variability-intensive soa systems. *University of Groningen, The Netherlands, Technical Report*, 2009.

[83] Stefan Kugele, Ana Petrovska, and Ilias Gerostathopoulos. *Towards a Taxonomy of Autonomous Systems*. Springer International Publishing, Cham, 2021.

[84] JR Landis, GG Koch biometrics, and undefined 1977. The measurement of observer agreement for categorical data. *JSTOR*, 1977.

[85] Thomas D. LaToza, Gina Venolia, and Robert DeLine. Maintaining mental models: A study of developer work habits. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, page 492–501, New York, NY, USA, 2006. Association for Computing Machinery.

[86] Paul J Layzell and Pericles Loucopoulos. A rule-based approach to the construction and evolution of business information systems. In *1988 Conference on Software Maintenance*, pages 258–264. IEEE Computer Society, 1988.

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

[88] Amanda Lee and Jeffrey C Carver. Floss participants' perceptions about gender and inclusiveness: a survey. In *Proceedings of the 41st International Conference on Software Engineering*, pages 677–687. IEEE Press, 2019.

[89] Hongwei Li, Zhenchang Xing, Xin Peng, and Wenyun Zhao. What help do developers seek, when and how? In *2013 20th Working Conference on Reverse Engineering (WCRE)*, pages 142–151, 2013.

[90] A. M. Lima, R. Q. Reis, and C. A. L. Reis. Empirical evidence of factors influencing project context in distributed software projects. In *2015 IEEE/ACM 2nd International Workshop on Context for Software Development*, pages 6–7, 2015.

[91] Chun-Ting Lin, Shang-Pin Ma, and Yu-Wen Huang. Msabot: A chatbot framework for assisting in the development and operation of microservice-based systems. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, ICSEW'20, page 36–40, 2020.

[92] Jun Lin. Context-aware task allocation for distributed agile team. In *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*, ASE'13, pages 758–761, Piscataway, NJ, USA, 2013. IEEE Press.

[93] Luis Fernando Lins, Glaucia Melo, Toacy Oliveira, Paulo Alencar, and Donald Cowan. Pacas: process-aware conversational agents. In *International Conference on Business Process Management*, pages 312–318. Springer, 2021.

[94] Dongyu Liu, Micah J. Smith, and Kalyan Veeramachaneni. Understanding user-bot interactions for small-scale automation in open-source development. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, page 1–8, New York, NY, USA, 2020.

[95] Martin Lochner and Daniel Smilek. The uncertain advisor: trust, accuracy, and self-correction in an automated decision support system. *Cognitive Processing*, 24(1):95–106, 2023.

[96] Bernd Lorenz, Francesco Di Nocera, Stefan Röttger, and Raja Parasuraman. The effects of level of automation on the out-of-the-loop unfamiliarity in a complex dynamic fault-management task during simulated spaceflight operations. In *Proceedings of the human factors and ergonomics society annual meeting*, volume 45, pages 44–48. SAGE Publications Sage CA: Los Angeles, CA, 2001.

[97] T. Machado, A. Ahonen, and R. Ghabcheloo. Towards a standard taxonomy for levels of automation in heavyduty mobile machinery. In *Proceedings of ASME/BATH 2021 Symposium on Fluid Power and Motion Control, FPMC 2021*, 2021.

[98] Keith Marzullo and Douglas Wiebe. Jasmine: A software system modelling facility. *ACM SIGPLAN Notices*, 22(1):121–130, 1987.

[99] Christoph Matthies, Franziska Dobrigkeit, and Guenter Hesse. An additional set of (automated) eyes: Chatbots for agile retrospectives. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, pages 34–37, 2019.

[100] Michael F McTear. The rise of the conversational interface: A new kid on the block? In *International Workshop on Future and Emerging Trends in Language Technology*, pages 38–49. Springer, 2016.

[101] Glaucia Melo. Designing adaptive developer-chatbot interactions: Context integration, experimental studies, and levels of automation. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 235–239, 2023.

[102] Glaucia Melo, Paulo Alencar, and Don Cowan. Context-augmented software development in traditional and big data projects: Literature review and preliminary framework. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 3449–3457. IEEE, 2019.

[103] Glaucia Melo, Edith Law, Paulo Alencar, and Donald Cowan. Understanding user understanding: What do developers expect from a cognitive assistant? In *2020 IEEE International Conference on Big Data (Big Data)*, pages 3165–3172, 2020.

[104] Glaucia Melo, Luis Fernando Lins, Paulo Alencar, and Donald Cowan. Supporting contextual conversational agent-based software development. In *2023 IEEE/ACM 5th International Workshop on Bots in Software Engineering (BotSE)*, pages 9–13, 2023.

[105] Glaucia Melo, Nathalia Nascimento, Paulo Alencar, and Donald Cowan. Understanding levels of automation in human-machine collaboration. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 3952–3958, 2022.

[106] Glaucia Melo, Nathalia Nascimento, Paulo Alencar, and Donald Cowan. Identifying factors that impact levels of automation in autonomous systems. *IEEE Access*, 11:56437–56452, 2023.

[107] Glaucia Melo, Nathalia Nascimento, Paulo Alencar, and Donald Cowan. Variability-aware architecture for human-chatbot interactions: Taming levels of automation. In *Proceedings of the 1st International Workshop on Artificial Intelligence for Autonomous Computing Systems (AI4AS 2023), co-located with ACSOS 2023*, 2023. To appear.

[108] Glaucia Melo, Toacy Oliveira, Paulo Alencar, and Don Cowan. Retrieving curated stack overflow posts from project task similarities. In *International Conference on Software Engineering Knowledge Engineering*, pages 415–418, 2019.

[109] Alex C. Meng. On evaluating self-adaptive software. In Paul Robertson, Howie Shrobe, and Robert Laddaga, editors, *Self-Adaptive Software*, pages 65–74, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[110] A. N. Meyer, L. E. Barton, G. C. Murphy, T. Zimmermann, and T. Fritz. The work life of developers: Activities, switches and perceived productivity. *IEEE Transactions on Software Engineering*, 43(12):1178–1193, 2017.

[111] André N. Meyer, Earl T. Barr, Christian Bird, and Thomas Zimmermann. Today was a good day: The daily life of software developers. *IEEE Transactions on Software Engineering*, 47(5):863–880, 2021.

[112] Mia Mikic and Joy Malala. The impact of artificial intelligence on the future of work. In *The Home in the Digital Age*, pages 143–159. 2021.

[113] Vesna Mikulovic and Michael Heiss. "how do i know what i have to do?" the role of the inquiry culture in requirements communication for distributed software development projects". In *Proceedings of the 28th international conference on Software engineering*, pages 921–925, 2006.

[114] P. Milgram, A. Rastogi, and J.J. Grodski. Telerobotic control using augmented reality. In *Proceedings 4th IEEE International Workshop on Robot and Human Communication*, pages 21–29, 1995.

[115] Neville Moray, Toshiyuki Inagaki, and Makoto Itoh. Adaptive automation, trust, and self-confidence in fault management of time-critical tasks. *Journal of experimental psychology: Applied*, 6(1):44, 2000.

[116] Gail Murphy. Beyond integrated development environments: adding context to software development. In *Proceedings of the 41st International Conference on Software Engineering*, pages 73–76. IEEE Press, 2019.

[117] Gail C. Murphy. Getting to flow in software development. In *Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2014, pages 269–281, New York, NY, USA, 2014. ACM.

[118] Gail C Murphy. The need for context in software engineering (ieee cs harlan mills award keynote). In *33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 5–5, 2018.

[119] Gail C. Murphy. The need for context in software engineering (ieee cs harlan mills award keynote). In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE 2018, page 5, New York, NY, USA, 2018. Association for Computing Machinery.

[120] M. Müller, T. Müller, B. Ashtari Talkhestani, P. Marks, N. Jazdi, and M. Weyrich. Industrielle autonome systeme: ein Überblick über definitionen, merkmale und fähigkeiten. *At-Automatisierungstechnik*, 69(1):3–13, 2021. cited By 4.

[121] Nathalia Nascimento, Paulo Alencar, Carlos Lucena, and Donald Cowan. An iot analytics embodied agent model based on context-aware machine learning. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 5170–5175. IEEE, 2018.

[122] Damir Nesic, Jacob Krueger, Stefan Stanciulescu, and Thorsten Berger. Principles of feature modeling. In *ESEC/SIGSOFT FSE*, pages 62–73, 2019.

[123] Quynh N Nguyen and Anna Sidorova. Understanding user interactions with a chatbot: A self-determination theory approach. *24th Americas Conference on Information Systems – Emergent Research Forum*, 2018.

[124] F. Novakazi, M. Johansson, H. Strömberg, and M. Karlsson. Levels of what? investigating drivers' understanding of different levels of automation in vehicles.

Journal of Cognitive Engineering and Decision Making, 15(2-3):116–132, 2021. cited By 1.

[125] M. Nylin, J. Johansson Westberg, and J. Lundberg. Reduced autonomy workspace (raw)—an interaction design approach for human-automation cooperation. *Cognition, Technology and Work*, 24(2):261–273, 2022.

[126] Dušan Okanović, Samuel Beck, Lasse Merz, Christoph Zorn, Leonel Merino, André van Hoorn, and Fabian Beck. Can a chatbot support software engineers with load testing? approach and experiences. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, ICPE '20, page 120–129, New York, NY, USA, 2020. Association for Computing Machinery.

[127] M. Pai, M. McCulloch, J. D. Gorman, N. Pai, W. Enanoria, G. Kennedy, P. Tharyan, and Jr J. Colford. Systematic reviews and meta-analyses: an illustrated, step-by-step guide. *The National medical journal of India*, 17(2):86–95, 2004. pmid:15141602.

[128] Elahe Paikari, JaeEun Choi, SeonKyu Kim, Sooyoung Baek, MyeongSoo Kim, SeungEon Lee, ChaeYeon Han, YoungJae Kim, KaHye Ahn, Chan Cheong, et al. A chatbot for conflict detection and resolution. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, pages 29–33. IEEE, 2019.

[129] Elahe Paikari and André Van Der Hoek. A framework for understanding chatbots and their future. In *Proceedings of the 11th international workshop on cooperative and human aspects of software engineering*, pages 13–16, 2018.

[130] Raja Parasuraman, Michael Barnes, Keryl Cosenzo, and Sandeep Mulgund. Adaptive automation for human-robot teaming in future command and control systems. *The International C2 Journal*, 1(2):43–68, 2007.

[131] Raja Parasuraman, Thomas B Sheridan, and Christopher D Wickens. A model for types and levels of human interaction with automation. *IEEE Transactions on systems, man, and cybernetics-Part A: Systems and Humans*, 30(3):286–297, 2000.

[132] Zhenhui Peng and Xiaojuan Ma. A survey on construction and enhancement methods in service chatbots design. *CCF Transactions on Pervasive Computing and Interaction*, 1:204–223, 2019.

[133] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1–18, 2015.

[134] Lesley M. Pickard, Barbara A. Kitchenham, and Peter W. Jones. Combining empirical results in software engineering. *Information and Software Technology*, 40(14):811–821, 1998.

[135] Klaus Pohl, Günter Böckle, and Frank Van Der Linden. *Software product line engineering*, volume 10. Springer, 2005.

[136] L. Ponzanelli, G. Bavota, M. D. Penta, R. Oliveto, and M. Lanza. Prompter: A self-confident recommender system. In *2014 IEEE International Conference on Software Maintenance and Evolution*, page 577–580, Sep 2014.

[137] Luca Ponzanelli, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Michele Lanza. Mining stackoverflow to turn the ide into a self-confident programming prompter. In *Proceedings of the 11th working conference on mining software repositories*, pages 102–111, 2014.

[138] Abhisek Omkar Prasad, Pradumn Mishra, Urja Jain, Anish Pandey, Anushka Sinha, Anil Singh Yadav, Rajan Kumar, Abhishek Sharma, Gaurav Kumar, Karrar Hazim Salem, et al. Design and development of software stack of an autonomous vehicle using robot operating system. *Robotics and Autonomous Systems*, 161:104340, 2023.

[139] Ryan W Proud, Jeremy J Hart, and Richard B Mrozinski. Methods for determining the level of autonomy to design into a human spaceflight vehicle: a function specific approach. Technical report, National Aeronautics and Space Administration Houston TX Lyndon B Johnson . . . , 2003.

[140] PTOLEMUS. What are the six levels of vehicle automation? https://www.ptolemus.com/what-are-the-six-levels-of-vehicle-automation/. Accessed: [Insert Access Date].

[141] Chetan Surana Rajender Kumar Surana, Shriya, Dipesh B. Gupta, and Sahana P. Shankar. Intelligent chatbot for requirements elicitation and classification. In *2019 4th International Conference on Recent Trends on Electronics, Information, Communication and Technology (RTEICT)*, pages 866–870, 2019.

[142] M.A. Ramos, C.A. Thieme, and X. Yang. Human-system concurrent task analysis: An application to autonomous remotely operated vehicle operations. *30th European Safety and Reliability Conference, ESREL 2020 and 15th Probabilistic Safety Assessment and Management Conference, PSAM 2020*, pages 629–636, 2020.

[143] Quelita ADS Ribeiro, Moniky Ribeiro, and Jaelson Castro. Requirements engineering for autonomous vehicles: a systematic literature review. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, pages 1299–1308, 2022.

[144] Victor Riley. A general model of mixed-initiative human-machine systems. In *Proceedings of the Human Factors Society Annual Meeting*, volume 33, pages 124–128. Sage Publications Sage CA: Los Angeles, CA, 1989.

[145] Vincent Rosener and Denis Avrilionis. Elements for the definition of a model of software engineering. In *Proceedings of the 2006 International workshop on Global Integrated Model Management*, pages 29–34, 2006.

[146] Denise M Rousseau and Yitzhak Fried. Location, location, location: Contextualizing organizational research. *Journal of organizational behavior*, pages 1–13, 2001.

[147] Daniel Salber, Anind K Dey, and Gregory D Abowd. The context toolkit: Aiding the development of context-enabled applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 434–441, 1999.

[148] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, 4(2), may 2009.

[149] Juliana Saraiva. A roadmap for software maintainability measurement. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 1453–1455. IEEE, 2013.

[150] Farhana Sarker, Bogdan Vasilescu, Kelly Blincoe, and Vladimir Filkov. Socio-technical work-rate increase associates with changes in work patterns in online projects. In *Proceedings of the 41st International Conference on Software Engineering*, pages 936–947. IEEE Press, 2019.

[151] Nicholas Sawadsky and Gail C. Murphy. Fishtail: From task context to source code examples. In *Proceedings of the 1st Workshop on Developing Tools as Plug-Ins*, TOPI '11, page 48–51, New York, NY, USA, 2011. Association for Computing Machinery.

[152] Mark W Scerbo. Theoretical perspectives on adaptive automation. In *Automation and human performance*, pages 37–63. CRC Press, 2018.

[153] Moch Akbar Selamat and Nila Armelia Windasari. Chatbot for smes: Integrating customer and business owner perspectives. *Technology in Society*, 66:101685, 2021.

[154] A. Sharma and M.A.M. Capretz. Application maintenance using software agents. In *Proceedings First IEEE International Workshop on Source Code Analysis and Manipulation*, pages 55–64, 2001.

[155] Thomas B Sheridan and William L Verplank. Human and computer control of undersea teleoperators. Technical report, Massachusetts Inst of Tech Cambridge Man-Machine Systems Lab, 1978.

[156] Joseph Sifakis and David Harel. Trustworthy autonomous system development. *ACM Transactions on Embedded Computing Systems*, 22(3):1–24, 2023.

[157] M. Simmler and R. Frischknecht. A taxonomy of human–machine collaboration: capturing automation and technical autonomy. *AI and Society*, 36(1):239–250, 2021. cited By 4.

[158] Janice Singer, Timothy Lethbridge, Norman Vinson, and Nicolas Anquetil. An examination of software engineering work practices. In *CASCON First Decade High Impact Papers*, CASCON '10, page 174–188, USA, 2010. IBM Corp.

[159] Kamal Souali, Othmane Rahmaoui, Mohammed Ouzzif, and Ismail El Haddioui. Recommending moodle resources using chatbots. In *2019 15th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)*, pages 677–680. IEEE, 2019.

[160] Natalie B Steinhauser, Davin Pavlas, and Peter A Hancock. Design principles for adaptive automation and aiding. *Ergonomics in Design*, 17(2):6–10, 2009.

[161] Margaret-Anne Storey, Alexander Serebrenik, Carolyn Penstein Rosé, Thomas Zimmermann, and James D. Herbsleb. BOTse: Bots in Software Engineering (Dagstuhl Seminar 19471). *Dagstuhl Reports*, 9(11):84–96, 2020.

[162] 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.

[163] Hossein Tajalli and Nenad Medvidović. A reference architecture for integrated development and run-time environments. In *2012 Second International Workshop on Developing Tools as Plug-Ins (TOPI)*, pages 19–24. IEEE, 2012.

[164] Gabriel Tamura, Norha M. Villegas, Hausi A. Muller, Laurence Duchien, and Lionel Seinturier. Improving context-awareness in self-adaptation using the dynamico reference model. In *2013 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 153–162, 2013.

[165] Marialena Vagia, Aksel A. Transeth, and Sigurd A. Fjerdingen. A literature review on the levels of automation during the years. what are the different taxonomies that have been proposed? *Applied Ergonomics*, 53:190–202, 2016.

[166] Jilles Van Gurp, Jan Bosch, and Mikael Svahnberg. On the notion of variability in software product lines. In *Proceedings Working IEEE/IFIP Conference on Software Architecture*, pages 45–54. IEEE, 2001.

[167] R. Van Solingen, V. Basili, G. Caldiera, and H. D. Rombach. Goal question metric (gqm) approach. *Encyclopedia of software engineering*, 2002.

[168] R. van Tonder and C. Le Goues. Towards s/engineer/bot: Principles for program repair bots. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, pages 43–47, 2019.

[169] Jay van Zyl. Class of solution dilemma: selecting business relevant software solutions. In *IEMC'03 Proceedings. Managing Technologically Driven Organizations: The Human Side of Innovation and Change*, pages 41–45. IEEE, 2003.

[170] S. Vasanthapriyan, J. Tian, and J. Xiang. A survey on knowledge management in software engineering. In *Software Quality, Reliability and Security-Companion (QRS-C), 2015 IEEE International Conference on*, pages 237–244. IEEE, 2015.

[171] Davor ČubraniĆ, Gail C. Murphy, Janice Singer, and Kellogg S. Booth. Learning from project history: A case study for software development. In *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*, CSCW '04, pages 82–91, New York, NY, USA, 2004. ACM.

[172] Jéssyka Vilela, Jaelson Castro, and João Pimentel. A systematic process for obtaining the behavior of context-sensitive systems. *Journal of Software Engineering Research and Development*, 4:1–57, 2016.

[173] V. Villani, L. Sabattini, P. Baranska, E. Callegati, J.N. Czerniak, A. Debbache, M. Fahimipirehgalin, A. Gallasch, F. Loch, R. Maida, A. Mertens, Z. Mockallo, F. Monica, V. Nitsch, E. Talas, E. Toschi, B. Vogel-Heuser, J. Willems,

D. Zolnierczyk-Zreda, and C. Fantuzzi. The inclusive system: A general framework for adaptive industrial automation. *IEEE Transactions on Automation Science and Engineering*, 18(4):1969–1982, 2021. cited By 3.

[174] Norha M. Villegas, Hausi A. Müller, Gabriel Tamura, Laurence Duchien, and Rubby Casallas. A framework for evaluating quality-driven self-adaptive software systems. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '11, page 80–89, New York, NY, USA, 2011. Association for Computing Machinery.

[175] Patrick Wagstrom and Subhajit Datta. Does latitude hurt while longitude kills? geographical and temporal separation in a large scale software development project. In *Proceedings of the 36th International Conference on Software Engineering*, pages 199–210, 2014.

[176] Eko K Budiardjo Wahyudianto and Elviawaty M Zamzami. Feature modeling and variability modeling syntactic notation comparison and mapping. *Journal of Computer and Communications*, 2:101–108, 2014.

[177] Lihui Wang, Sichao Liu, Hongyi Liu, and Xi Vincent Wang. Overview of human-robot collaboration in manufacturing. In *Proceedings of 5th International Conference on the Industry 4.0 Model for Advanced Manufacturing: AMP 2020*, pages 15–58. Springer, 2020.

[178] Frederic Weigand Warr and Martin P. Robillard. Suade: Topology-based searches for software investigation. In *Proceedings of the 29th International Conference on Software Engineering*, ICSE '07, page 780–783. IEEE Computer Society, 2007.

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

[180] James Y Zhang, Zhi Li, Hao Fang, Jun Wu, Zhongnan Shen, Jing Zheng, Wei Chu, Weiping Duan, and Peng Xu. China's first natural language-based ai chatbot trader. In *Companion Proceedings of the ACM Web Conference 2023*, pages 1253–1256, 2023.

[181] Davor Čubranić and Gail C. Murphy. Hipikat: Recommending pertinent software development artifacts. In *Proceedings of the 25th International Conference on Software*

Engineering, ICSE '03, page 408–418. IEEE Computer Society, 2003. event-place: Portland, Oregon.

APPENDICES

Appendix A

Systematic Literature Review in Contexts

A.1 Software Engineering Contexts Table

Table A.1: RQs Table Summary - RQ1.1 to RQ1.4

ID	RQ1.1	RQ1.2	RQ1.3	RQ1.4
A1	Static software structure		IDEs provide static source code artifacts as context to tools hosted in the environment	
	Dynamic system execution		Context in the form of dynamic execution information about a system under development.	
	Historical artifact changes		Tools that access historical informatio nabout a system's static artifacts.	
	Developer activity		Context about how humans work to produce the system, and not necessarily what was generated during the system's production. Ex.: Mylyn's degree of interest.	
	Team and organization activity		Treating the activities across a value streamas context.	
A2	Task Context: pieces and relationships in an information space (e.g., artifacts) that are relevant to a software developer as they work on a particular task		Approximate task context by either capturing developers' interactions or using data from repositories to determine if the information captured or used is relevant to new tasks that will be performed.	
A3	Approach uses current task related information generated during sprint assessment phase (or in sprint planning meeting), task completion information generated during previous sprint review phase/meeting, and characteristics of team members.		Produce task allocations recommendations.	Planning (task allocation)
A4	Natural language text, textual rendering of core dumps, debugger output etc.		Have fat query, a query which could be kilobytes of structured and unstructured data containing all contextual information for the issue being debugged. Allows users to search through all available software repositories (version control, bug database, logs of debugger sessions, etc).	When bug occurs
A5	Person, Message, Document, Change Task and File version.		Create a project memory from the artifacts and communications created during a software development project's history to facilitate knowledge transfer from experienced developers to novice.	Coding or when bug occurs.
SBA1	13 IDE contexts. The authors list each and characterize into the following categories: who, what, when and where.	Yes. Contextual factors of interactions of developers with IDEs	Support development in an IDE and support context-aware RSSE systems development.	Mostly coding and testing, which are done in an IDE such as Eclipse.
SBA2	12 elements are modeled. They are related to the current file, local repository, remote repository and other services such as test and assign a reviewer to the code.	Yes. The contexts are modeled so the conversational assistant can use the model to retrieve the responses.	Reducing low-level commands that developers need to perform, freeing them to focus on their high-level tasks through voice commands.	Coding and assigning test.
SBA3	Already mapped during first step of the LR in paper Learning from project history.			
SBA4	Code		Retrieve Stack Overflow Posts according to code typed in IDE	Coding and assigning test.
SBA5	Methods or fields during programming that users specify as relevant (often result of a text search)		Help developers quickly find relevant elements and understand their relationships with the other elements that implement the feature of interest	Coding
SBA6	Code, Methods and Field Declaration		Support developers to locate relevant code to the current code being written	Coding
SBA7	Iteraction events history of a task	Task context model proposed.	The proposed model reduces information overload and focuses on a programmer's work by filtering and ranking the information presented by the development environment	Coding

Table A.2: RQs Table Summary - RQ1.5 to RQ1.9

ID	RQ1.5	RQ1.6	RQ1.7	RQ1.8	RQ1.9
A1					
		Although many research tools have been proposed that use historical information, few tools are available to practicing developers.	Adv: Task contexts enable developers to be more productive by making it easy to recall the source code associated with a given task and by enabling other tools, such as content assist, to order information based on work performed as part of the task.		
			Adv: Enables correlation of downstream effects with upstream choices and would open new opportunities for feedback to be provided to developers. Disad.: Still unexplored.		
A2			Adv: Increased flow reduces information shown to developers and enables different information to be related automatically.		
A3	Tool presented better results than baseline.		The tool also balances a common problem of the increased number of tasks allocated to experienced users, while others were idle.	Recommendation of who should resolve a task through a tool.	
A4	One of the performed evaluations returned useful results (bugs resolutions) for 75% of the cases tried.	Duplicate bug reports because of code clones, which can hamper evaluation results.		Recommendations of information from bug databases considering query of contextual information of issue.	
A5	Qualitative evaluation regarding effective use of history information by newcomers within developed tool that implements the contexts.			Recommendation of artifacts that should be edited according to the project history captured.	
SBA1	Executing IDE commands and verification if contextual factors of the proposed model correlate with commands in the IDE.	Concerned with the privacy of developers, the work is limited to IDE commands.	Advantages: The context modeled provide meaningful information regarding the interactions with the IDE while developing. Disad: there are no evaluations considering if the performance of the developers improved, for example.	During evaluation, the model was populated with information, so statistical analysis was possible.	There are contextual factors abstracted to represent an artifact.
SBA2	Inteview and experiment mixed.		Adv: Allows reduced context switches through natural language.		The speech is abstracted into the set of contexts expected by the tool.
SBA3					
SBA4	Evaluated ranking of Stack Overflow posts and the usefulness of the tool proposed.		Adv.: Supports provision of flow to software development as developers do not need to leave the IDE to search for support when coding.		
SBA5		Suggestions can take long in case of modified code			Users are able to set tags (concerns) which the proposed algorithm also uses when trying to find the related contexts
SBA6	Qualitative evaluation on the usefulness of the recommendations. Results show recommendations were helpful.				
SBA7	Quantitative and qualitative field study with participants. Both produced evidence that the use of task context can make programmers more productive.				Programmers tasks are abstracted into high-level tasks.

Appendix B

Adaptive Context-Augmented Framework

B.1 Adaptive Contextual Framework for Software Development

Because of the variability of context observed in the literature review (environment, people, domain), we propose a framework to capture the software development context, monitor the possible variabilities and recommend specific knowledge and potential next steps to developers.

Context can be defined as something that is part of an environment and can be sensed. A more specific definition applied to software engineering proposed by Murphy [116] is that it "is the information about the system under development and the environment and process in which the system is being developed." A system that can respond to these possible mutable scenarios such as domain, process, technologies involved and people appears to be better than methods that are not prepared for these contextual changes [121].

We propose a framework based on the observed context variability. The following modules are proposed: (i) the software development project where the context model will be applied; (ii) a baseline of a reconfigurable context model; and (iii) an engine that adapts machine learning models to the context model and provides recommendations to software developers. A high-level framework representation is shown in Figure B.1.

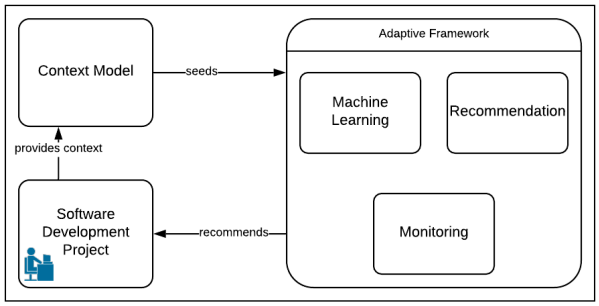

Figure B.1: Proposed high-level adaptive context-augmented framework for software development projects.

An example of a possible context model, according to the information retrieved in the literature review is presented next.

B.1.1 Context Model Example

This Context Model proposed as part of the Adaptive Context-Augmented Framework in Figure B.1 is based on the context types (RQ1) identified in the Literature Review (Section 4.2). We do not claim this model is final, rather we use the model as a basis to understand some of the possible contextual variabilities and as a guide to produce practical examples. The Context Model is presented in Figure B.2.

An illustrative example of how this model can be used and integrated with the proposed framework is presented next.

B.1.2 Illustrative Example

Gabi is a software developer who has been programming in Java for nine years. She has been recently working on Project X, a new project for the company. When there is a new project, Gabi needs to create a minimal viable product (MVP) to show her clients. She deploys the software locally, using a container tool such as Docker and manually uploads the project to a web server. She also reboots the server manually after each deployment, so changes are effective. This way is faster, and she does not have to configure a job or a server to generate a deployment automatically, which would cause the clients to wait much longer for the MVP.

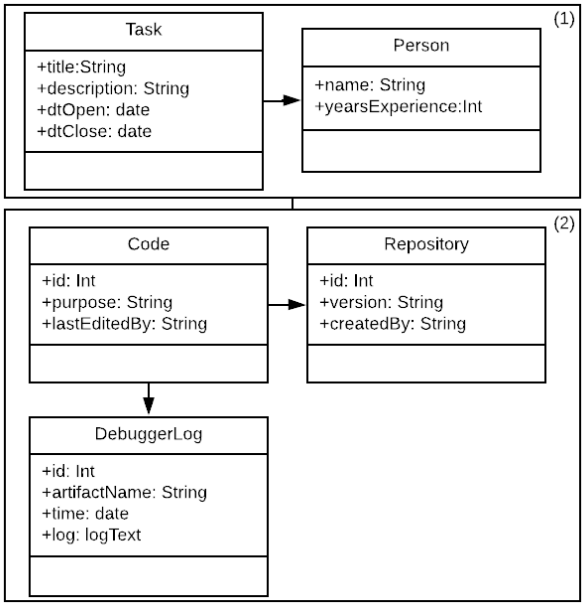

Figure B.2: Context Model.

In Project Y, a mature and huge project in the company, when a version of the software has to go to production, all Gabi does is commit the code from the local to the shared code repository. Then, the scheduled automated job in a Jenkins server will take care of the other steps, which are checking out the code, building the project, uploading the package on the server and rebooting the server.

In theory, the steps are the same, but because the projects are different, Gabi's work is different, which means that in the second case, the context model should be expecting Project information or project phase information (e.g., MVP or production phase), which defines how the deployment will be done. If Gabi, who has been working on Project Y for years, forgets she needs to deploy Project X manually, this can be a problem, as she will not be able to see the changes of the new deployment. In this case, the context model should be adapted to recognize and store the project identification context (including the phase/maturity of the project), and should evolve as shown in Figure B.3. This figure also shows several other contextual project attributes, including team expertise, hardware and software technologies, IDE tools, as well as location and timezone, that could potentially influence Gabi's work.

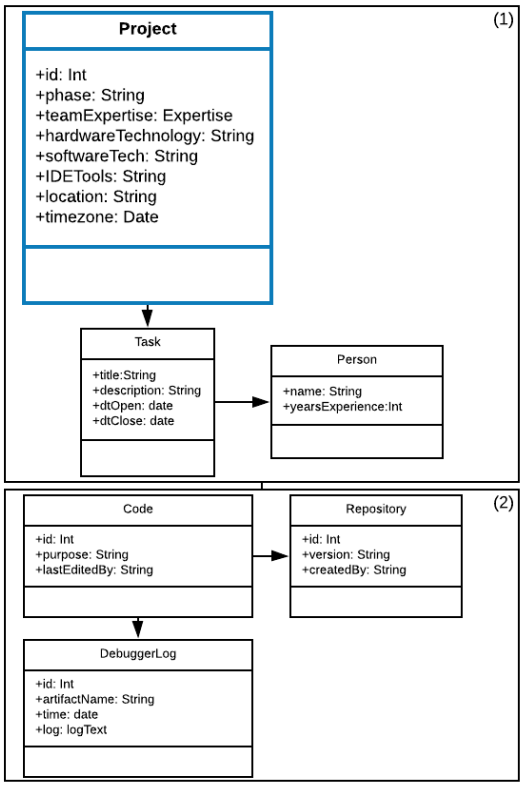

Figure B.3: Extended Context Model.

Appendix C

Rasa Files - DevBot

C.1 General Configuration

There are a few configuration files. Here is a description of such files.

- credentials.yml: This file contains the credentials for the voice & chat platforms which your bot is using. In this file, you will configure the access token that the platform where you are deploying the chatbot will use.
- domain.yml: The domain defines the universe in which your assistant operates. It specifies the intents, entities, slots, responses, forms, and actions your bot should know about. It also defines a configuration for conversation sessions. The domain.yml we used for DevBot is presented in Figures C.1 and C.2.
- config.yml: The configuration file defines the components and policies that your model will use to make predictions based on user input. The config.yml for DevBot is presented in Figure C.3.

Next, are the files related to the language and conversation of the chatbot prototype.

C.2 Data Files Configurations

C.2.1 Greetings.txt

```
Greetings:
goodbye:
  - text: Bye!!
greet:
  - text: Hey! What do you need to know regarding your scenario today?

Scenario responses:
artifact:
  - text: You need to edit class client.java and class diagram OrderMain.
commit:
  - text: OK, I am committing your code to your repo.
priority:
  - text: The task with the highest priority assigned to you is Task 13003.
pullrequest:
  - text: Please, create a pull request to a branch identified with your task ID.
push:
  - text: Please, create a pull request to a branch identified with your task ID.
sync:
  - text: You should synchronize your repo. I bet you know how.
  Click on the sync repo button on your version control page.
tasks:
  - text: Tasks 27405, 67933 and 13003 are open and assigned to you.
tool:
  - text: You have to open Eclipse and synchronize your code. After, assign your a test

did_that_help:
  - text: Did that help you?
  What else can I help you with?

default_fallback:
  - text: Sorry, I can't help you with that. Is there anything else I can help with?
```

C.2.2 nlu.md

The goal of NLU (Natural Language Understanding) is to extract structured information from user messages. NLU training data consists of example user utterances categorized by intent. To make it easier to use your intents, give them names that relate to what the user wants to accomplish with that intent, keep them in lowercase, and avoid spaces and special characters. As conversations happen, new intents will be dynamically added to this file.

intent:Greet

- hey
- hello
- hi
- good morning
- good evening
- hey there
- Hi

intent:Goodbye

- bye
- goodbye
- see you around
- see you later
- That's all for today. bye!

intent:Artifact

- What is the name of the .java class that I'm supposed to be editing?
- What are artifacts

- How do I edit class client.java?
- How many artifacts do I have to change to accomplish the first task?
- Give me the artifacts' names in a list
- Gonna start editing the .java file now, help me bring it up?
- Which artifacts do I need to edit?
- which part of client.java needs to be edited?

intent:Commit

- How do I commit my code
- Devbot, could you commit my code please?
- Just help me commit my tests and changes if the integration tests pass?
- please commit my code
- could you please commit my edits to my repository?

intent:Tasks

- What does my overall day look like today?
- How do I update the status to Done? Is this in the Jira?
- How do I update status to Done.
- What is task 27405
- Where is task 27405 located in the program management tool.
- How does my schedule look like today?
- Okay. Are there any similar tasks like 13003?
- Is there a deadline to the task?
- Is there a person I can go to for this task who can help me of I have doubts ?

- Also, what are my most important tasks now?
- Update my task list
- what tasks do I need to complete?
- ok which task ID should I work on?
- what are my open tasks?
- what do I have to do today?
- what are my tasks?

intent:Priority

- What is the priority of the tasks
- What is the second highest priority task
- The lowest priority task
- What are the priorities of my tasks?
- Which is of my higher priority ?
- which task has higher priority?
- what is the task with highest priority?
- what i have to do first

intent:Pull

- how to do I make a pull request
- How do I make a pull request?
- Ok, create a PR with the usual feature template
- Need that comment into the PR

- Is there a pull request template or procedure that I should follow for the repo I AM: pushing to?
- who should I ping to look at my pull request?
- ok create a pull request for me
- can you make a pull request now?
- create a pull request to a branch identified with my task ID.
- create a pull request
- please create a pull request
- thank you. can you create a PR?

intent:Push

- push code

intent:Sync

- What's the command for syncing my repo?
- and sync it

intent:Tool

- What is the difference between the Eclipse editor and Visual Studio?
- DevBot, what is the difference between Git and Jira?
- What are the technologies needed for that task?

intent:Follow-up

- Did that help you? What else can I help you with?

intent:Bot Service

- DevBot, what can you help me with?
- What can you do?
- How can you help me?

intent:Exception

- When can I merge my code with master?
- Do you know where the coffee machine is?
- What is Git?
- Hey DevBot, can you fetch the newest code?
- Can you help me record what my colleague just said? She said the second artifact is probably not needed for an update
- What tests do you think we need for this task?
- Can you start a new file for the tests?
- Please name it with the usual format
- Actually, can you fill the file in with the test function templates as well?
- Gonna code the tests in now, just help me fix the usual typos and whatnot
- Looks like I got most of them, DevBot is there any test cases I left out?
- Actually, it looks like I haven't logged into Github, wanna help me do that too?
- Insert the comment: "Task 27405 completed referring to bug https://github.com/Repo/issues/13"
- remove 27405
- Actually, assign it to Done, not remove it, my bad
- Nice, that was helpful, put there too that I used about 3 hours for the task

- Awesome, looks like we are good, get some rest and hang out with your chatbot friends, we'll talk again tomorrow
- What testing framework is expected?
- Are there any CLAs that need to be signed to make the pull request?
- Which repository is that task associated with?
- but which repo is it?
- ok. please clone the repo.
- can you please set task 13003 to status Done?
- best way to talk to my boss about getting a raise?

C.2.3 stories.md

Stories serve as a form of training data to educate your assistant's dialogue management model. By employing stories, you can train models capable of extrapolating to unfamiliar conversation trajectories. A story is a representation of a conversation between a user and an AI assistant, converted into a specific format where user inputs are expressed as intents (and entities when necessary), while the assistant's responses and actions are expressed as action names.

Happy Path

- greet
  - utter_greet
- mood_great
  - utter_happy

Sad Path 1

- greet
  - utter_greet
- mood_unhappy
  - utter_cheer_up
  - utter_did_that_help
- affirm
  - utter_happy

Sad Path 2

- greet
  - utter_greet
- mood_unhappy
  - utter_cheer_up
  - utter_did_that_help
- deny
  - utter_goodbye

Say Goodbye

- goodbye
  - utter_goodbye

Bot Challenge

- bot_challenge
  - utter_iamabot

Artifact

- artifact
  - utter_artifact

Tasks

- tasks
  - utter_tasks

Commit

- commit
  - utter_commit

Priority

- priority
  - utter_priority

Pull

- pull
  - utter_pull

Push

- push
  - utter_push

Sync

- sync
  - utter_sync

Tool

- tool
  - utter_tool

Follow-up

- followup
  - utter_followup

Bot Service

- botservice
  - utter_botservice

Exception

- exception
  - utter_exception

Interactive Story 1

- greet
  - utter_greet
- tasks
  - utter_tasks

New Story

- tasks
- tasks
- priority
- exception
- pull

Interactive Story 2

- greet
  - utter_greet
- priority
  - utter_priority
- pull
  - utter_pull
- commit
  - utter_commit
- exception
  - utter_exception
- goodbye
  - utter_goodbye
  - action_restart

devbot / domain.yml

glauciams Add files via upload

Code Blame 80 lines (80 loc) · 2.15 KB

```yaml
session_config:
  session_expiration_time: 60.0
  carry_over_slots_to_new_session: false
intents:
- pull
- priority
- greet
- commit
- goodbye
- exception
- tasks
- affirm
- deny
- mood_great
- mood_unhappy
- bot_challenge
- artifact
- push
- sync
- tool
- followup
- botservice
responses:
  utter_artifact:
  - text: You need to edit class client.java and class diagram OrderMain.
  utter_botservice:
  - text: I can help you discover which tasks are assigned to you,their priorities,
```

Figure C.1: domain.yml file in Rasa, for DevBot prototype (1).

```
28          how to do a pull request and you can ask meto commit your code.
29      utter_cheer_up:
30      - image: https://i.imgur.com/nGF1K8f.jpg
31        text: 'Here is something to cheer you up:'
32      utter_commit:
33      - text: OK, I am committing your code to your repository.
34      utter_did_that_help:
35      - text: Did that help you?
36      utter_exception:
37      - text: Sorry, I can't help you with that. Is there anything else I can help with?
38      utter_followup:
39      - text: Did that help you? What else can I help you with?
40      utter_goodbye:
41      - text: Bye!
42      utter_greet:
43      - text: Hey! What do you need to know regarding your scenario today?
44      utter_happy:
45      - text: Great, carry on!
46      utter_iamabot:
47      - text: I am a bot, powered by Rasa.
48      utter_priority:
49      - text: The task with the highest priority assigned to you is Task 13003.
50      utter_pull:
51      - text: Please, create a pull request to a branch identified with your task ID.
52      utter_push:
53      - text: Please, download or clone your repository to your local machine.Keep it
54          synchronized.
55      utter_sync:
56      - text: You should synchronize your repo. Click in the sync repobutton on your version
57          control page.
58      utter_tasks:
59      - text: Tasks 27405, 67933 and 13003 are open and assigned to you.
60      utter_tool:
61      - text: You have to open Eclipse or VSCode and synchronize your code.After, assign
62          a test task to the test team.
63    actions:
64    - utter_artifact
65    - utter_botservice
66    - utter_cheer_up
67    - utter_commit
68    - utter_did_that_help
69    - utter_exception
70    - utter_followup
71    - utter_goodbye
72    - utter_greet
73    - utter_happy
74    - utter_iamabot
75    - utter_priority
76    - utter_pull
77    - utter_push
78    - utter_sync
79    - utter_tasks
80    - utter_tool
```

Figure C.2: domain.yml file in Rasa, for DevBot prototype (2).

devbot / config.yml

glauciams Add files via upload

30 lines (29 loc) · 739 Bytes

Code Blame Raw

```yaml
# Configuration for Rasa NLU.
# https://rasa.com/docs/rasa/nlu/components/
language: en
pipeline:
  - name: WhitespaceTokenizer
  - name: RegexFeaturizer
  - name: LexicalSyntacticFeaturizer
  - name: CountVectorsFeaturizer
  - name: CountVectorsFeaturizer
    analyzer: "char_wb"
    min_ngram: 1
    max_ngram: 4
  - name: DIETClassifier
    epochs: 100
  - name: EntitySynonymMapper
  - name: ResponseSelector
    epochs: 100

# Configuration for Rasa Core.
# https://rasa.com/docs/rasa/core/policies/
policies:
  - name: MemoizationPolicy
  - name: TEDPolicy
    max_history: 5
    epochs: 100
  - name: MappingPolicy
  - name: "FallbackPolicy"
    nlu_threshold: 0.4
    core_threshold: 0.3
    fallback_action_name: "utter_exception"
```

Figure C.3: config.yml file in Rasa, for DevBot prototype.

Appendix D

User Study Forms and Resources

D.1 Recruitment Email

Study Group: Controlled Experiment with Software Developers
Recruitment Method: This will be sent out through Snowball Sampling.

Title: Participants Needed ($15 to talk to chatbot)

Dear All,

We are looking for participants for a study called "Exploring Conversational Agents in Software Development". The purpose of this study is to explore how software developers interact with chatbots.

This study is part of an ongoing research project at the University of Waterloo. If you decide to participate, the study will take 1 hour. The entire study is done online. You will use your own computer or mobile phone to speak to a chatbot on Telegram. You will be given a scenario (text - one paragraph) of a day as a software developer and you will ask the chatbot questions that help you go about your day as a software developer, described in the scenario.

You will need internet access and a browser to access our software. Participation does not require downloading anything on your computer. The interactions with the software on the computer screen for this session will be video recorded. The study will be conducted fully online using secured video conferencing tools (e.g., Zoom, Skype, Teams). Unique meeting links will be generated and provided before the study. You will need your personal laptop/desktop for completing the study. You will not be required

to share your video. However, you will have to share your computer screen while you are using our software. We will not take any personal identifying information from you. Your interactions with the system and feedback will be kept completely anonymous.

The study is open to anyone who has at least 1-year experience developing software.

A remuneration of $15.00 CAD will be provided for participating in this study.

The study will be done online. Once you agree to participate, the researcher will send you the demographics collection form and the chatbot URL. Once you have interacted with the chatbot, participants are supposed to answer a questionnaire. The remuneration will be e-transferred after the completion of the study.

To participate or enquire about this study, please contact: Glaucia Melo gmelo@uwaterloo.ca.

This study has been reviewed and received ethics clearance through a University of Waterloo Research Ethics Committee.

D.2 Recruitment Text on Facebook and other Social Media

Greetings All,

We are looking for participants for a study called "Exploring Conversational Agents in Software Development".

The purpose of this study is to explore how software developers interact with chatbots. The study is open to anyone who has at least 1-year experience developing software.

A remuneration of $15.00 CAD will be provided for participating in this study.

To participate or enquire about this study, please contact: Glaucia Melo gmelo@uwaterloo.ca for further details regarding the study.

This study has been reviewed and received ethics clearance through a University of Waterloo Research Ethics Committee.

D.3 Information Letter and Consent Form

Title of Project: Exploring Conversational Agents in Software Development. This study is conducted by Dr. Edith Law and Glaucia Melo at the Cheriton School of Computer Science at the University of Waterloo, Canada. The objective of this exploratory study is to understand how a chatbot can facilitate software developers' tasks, learn how software developers interact with chatbots and how having a chatbot can impact software development.

Study details: If you decide to participate, you will be interacting with a chatbot called DevBot, which is created to help you during your day as a software developer. At the beginning of the session, you will be asked to complete a pre-study questionnaire about your demographics (e.g., gender, age, educational background) and months of experience with software development. Then, you will receive a scenario, detailing what would be your workday as a developer. After understanding the scenario, your interaction with DevBot beings, where you will be asking DevBot questions that would support the tasks of your day (scenario). This interaction should last at least 10 minutes. At the end of the session, you will complete a post-study survey about their experience with DevBot, and will be interviewed about your perception of the chatbot, and different kinds of factors that motivate you to interact with the robot.

In this interview and survey, you will also be asked to describe your general feeling and level of motivation, your perception of the DevBot, and your experience interacting with the chatbot. The entire study is done online, using Google Forms, Skype or Zoom and the DevBot chatbot on Telegram.

The study lasts approximately one hour and must be done in one sitting, i.e., without stopping, including the final interview. When the experiment is over, you will be paid based on what is explained in the Remuneration section.

Remuneration: By completing the experiment, you will be paid $15.00 CAD sent by e-transfer. If you do not complete the experiment, you will be paid based on the portion of the experiment completed. Specifically, you will be paid $1 for completing the pre-study survey, $12 for 10 minutes of continuous interaction with the chatbot (pro-rated based on time spent, if ending early), and $2 for completing the post-study survey and interview.

Inclusion/Exclusion Criteria: We seek participants who have at least 1 year of experience with software development in the industry (full-time employee, part-time, co-op or internship).

Withdrawal: Participation in this study is voluntary. You may decline to answer any questions in the questionnaire at any time you want, and you can withdraw your participation in the study at any time by emailing the researchers. You cannot withdraw from the study after results have been submitted for publication.

Benefits: There is no direct benefit to participants from this study.

Risks: There are no known or anticipated risks from participation in this study.

Videotaping, Audiotaping and Screen Capture: If you are participating in this study, we will record your interaction with the chatbot and video-record your post-survey interview, your opinions may be transcribed/coded in real-time or at a later time. You may decline to participate in the video, audio and screen capture recording at any time. Your face will not be captured, only the video of your screen sharing and audio.

Confidentiality: It is important for you to know that any information that you provide will be confidential. All of the data will be summarized and no individual could be identified from these summarized results. The data, with no personal identifiers, collected from this study will be maintained on a password-protected computer database in a restricted access area of the university and external servers. Paper forms will be stored in a locked cabinet at Davis Center, University of Waterloo. The data will be electronically archived after completion of the study and maintained for 8 years and then erased. The data collected for this study will be kept at the University of Waterloo. Only researchers and external collaborators associated with the study will have access to the data. The data will not be shared publicly. Your name and contact information (e.g., email, Skype ID) will be stored separately from your survey data, used for remuneration purposes only, and will be deleted immediately after you completed the experiment.

This study will use the Google Drive platform to collect data, which is an externally hosted cloud-based service. A link to their privacy policy is available here (https://support.google.com/drive/answer/2450387?hl=en). Please note that there is a small risk with any platform such as this of data that is collected on external servers falling outside the control of the research team. If you are concerned about this, we would be happy to make alternative arrangements for you to participate, perhaps via telephone. Please talk to the researcher if you have any concerns.

You will be completing the study by an online survey operated by Zoom or Skype. When information is transmitted or stored on the internet privacy cannot be guaranteed. There is always a risk your responses may be intercepted by a third party (e.g., government agencies, hackers). We temporarily collect your user ID to avoid duplicate

responses in the dataset but will not collect information that could identify you personally.

When information is transmitted over internet privacy cannot be guaranteed. There is always a risk that your responses may be intercepted by a third party (e.g., government agencies, hackers). University of Waterloo practices are to turn off functions that collect machine identifiers such as IP addresses. The host of the system collecting the data, such as Google Form, may collect this information without our knowledge and make this accessible to us. We will not use or save this information without your consent.

This study has been reviewed and received ethics clearance through a University of Waterloo Research Ethics Committee (ORE#42126). If you have questions for the Committee, contact the Chief Ethics Officer, Office of Research Ethics, at 1-519-888-4567 ext. 36005 or ore-ceo@ Uwaterloo.ca.

Questions: Should you have any questions about the study, please contact Glaucia Melo (gmelo@uwaterloo.ca) or Edith Law (edith.law@uwaterloo.ca). Further, if you would like to receive a copy of the results of this study, please contact either investigator.

Consent for Participation
By agreeing to participate in the study you are not waiving your legal rights or releasing the investigator(s) or involved institution(s) from their legal and professional responsibilities. With your permission, voice recordings and screenshots of your interaction with the chatbot from your participation may be used for transcription or analysis purposes. In these: Your name will not be used, but your face may be seen and your voice may be heard Your face and other identifying details will be cropped, blurred or removed and your name will not be used

By signing below, I consent to my participation in this study designed to help understand the interaction modes of user experience of software developers with chatbots. I have read the letter of information and understand the risks and benefits of participation. I also understand that:

- My identity will be kept confidential.
- If participating in groups, my interaction will be recorded.
- I am free to withdraw from the experiment at any time before, during or after, without reason or consequence.

- I have been told the purpose of the experiment and am free to ask questions at any time.
- I may take any complaints or concerns I may have to the primary experimenter, Dr. Edith Law.

I have read the above statement and freely consent to participate in this research.

oo I want to participate

oo I do not want to participant

Participant's Name:

Date:

D.4 Appreciation Material

University of Waterloo, David R. Cheriton School of Computer Science September, 2020

Project Title: Exploring Conversational Agents in Software Development

Principal Investigators: Dr. Edith Law and Glaucia Melo

We appreciate your participation in our study and thank you for spending the time helping us with our research!

In this activity, you interacted with a conversational agent by asking questions to the chatbot according to the scenario presented. What we wanted to find out is how you would ask questions to the chatbot, what would be your vocabulary, what part of the development would be most interesting to have the support and how much you liked the interaction and support provided.

We collected a lot of useful information from your interactions with the chatbot. Please know that we will protect all this information, so your personal information and identity will not be revealed to anyone. As a reminder, if you decide to withdraw from the study, you can have your data destroyed by contacting us before November 1, 2020. This deadline exists since we will be submitting papers for publication based on this data and it is not possible to remove participants' data once this has occurred.

If you have any questions, or are simply curious about what we found out from this study, please feel free to contact Edith Law by email (edithl.law@uwaterloo.ca) or by

phone (1-519-888-4567, ext. 35751) or Glaucia Melo by email (gmelo@uwaterloo.ca) or phone (1-519-888-4567, ext. 33991).

We really appreciate your participation and hope that this has been an interesting experience for you.

This study has been reviewed and received ethics clearance through a University of Waterloo Research Ethics Committee [ORE#42146]. If you have questions for the Committee contact the Office of Research Ethics, at 1-519-888-4567 ext. 36005 or oreceo@uwaterloo.ca.

Appendix E

On the Integration of Context into Software Development: Challenges and Opportunities

E.1 Motivation

Software development (SD) requires comprehensive support in terms of information and guidance based on the context during task execution [20, 116, 110]. As a process dependent on knowledge workers [79], SD lacks supportive methods based on conversational-guided agents that account for cognitive tasks such as paying attention, remembering, and maintaining mental maps of the software processes. Current practices of developing software also lack techniques that make use of historical implicit or tacit data to infer new knowledge about the project tasks and navigation aspects of the process. While similar tools and solutions provide comparable assistance [77, 76, 171] based on software information, none have considered software process information, and an artificial intelligence and machine learning component. Therefore, we argue that novel approaches should take advantage of the synergy among emerging methods in context-aware software processes, cognitive assistance such as chatbots and systems based on machine learning (ML).

Given these challenges, we provoke a discussion by asking: How can software development be advanced by introducing a new paradigm to realize human-machine software development cooperation based on context, cognitive assistance and machine learning?

SD has already been supported with automated tools [181, 136], and with automatically generated code, commit, and built chatbots [23, 20]. In the future, developers' knowledge and ubiquitous context will be integrated into the development environment, complementing the current state-of-the-art with very effective, timely and supportive relevant information for developers.

This proposed discussion intends to stimulate thinking about the creation of tools and procedures that can advance software development as it relates to software developers and, on a larger scope, companies. There is a direct connection to work being done by large software companies working at the forefront of research and practice involving novel (semi-)automated methods to support the development of software applications while improving software developers' efficiency and productivity. Shaping software development is critical as software systems have become the backbone of much of today's technology and society's functions. Working remotely has become a new reality. Consequently, approaches and tools that help to facilitate software development have become even more essential. Working with software development and its intrinsic implicit context is essential; therefore, we argue there is still the need to improve the machine-developer interaction, instead of purely automating software processes. We discuss the future of this proposed paradigm more in section E.4.

E.2 Motivating Example

It is known that deployment is a challenging task during software development [25], and so, we have presented an example of how changes in the steps that are executed during deployment vary in different contexts. You can refer to this scenario in section B.1.2.

Now, imagine Gabi - the object of the example - is interacting with a chatbot and does not remember exactly how to deploy a new project, since she is not always working on new projects. Or even, if Gabi has deployed Project X, and the changes cannot be seen in the application, because she forgot to reboot the server. In Figure E.1, we present an example of a conversation between Gabi and the chatbot during deployment. To reach the desired level of support, the chatbot should know her context, which includes the server address and folder of the development environment, and, based on the history of tasks previously executed in that context, the chatbot should be able to learn and suggest the task sequence that Gabi needs to follow and provide the information required to execute the tasks.

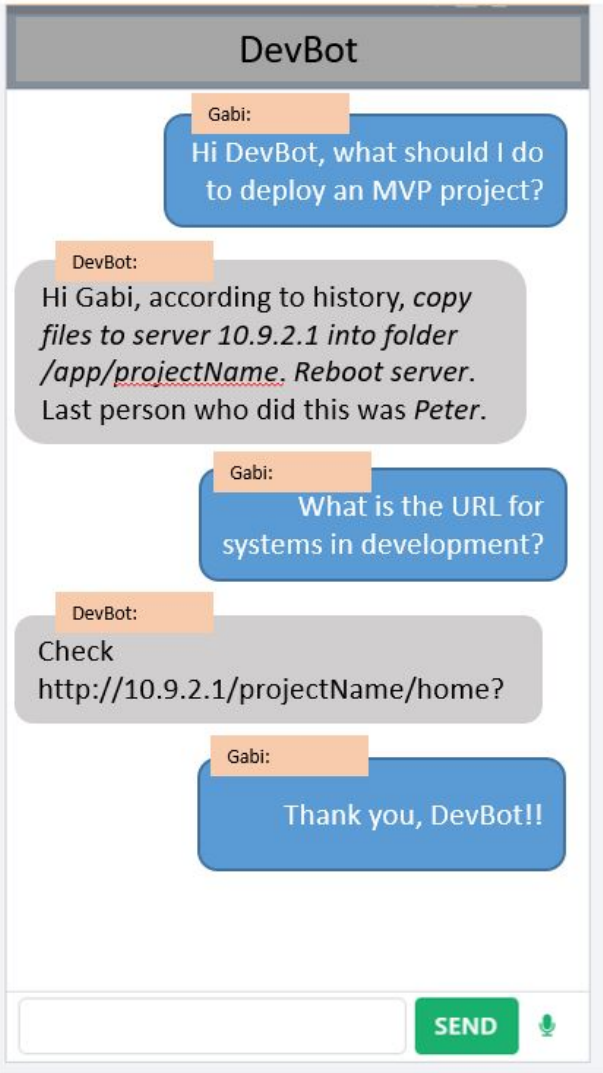

Figure E.1: Illustrative chatbot interaction. [Figure 2.1].

A representation of this scenario and different possible contexts are shown in Figure E.2. The bold information and in italics represent the context that can vary in this example, and therefore, influence the workflow.

This figure does not represent several other contextual attributes, including team expertise, hardware and software technologies, IDE tools, as well as location and time

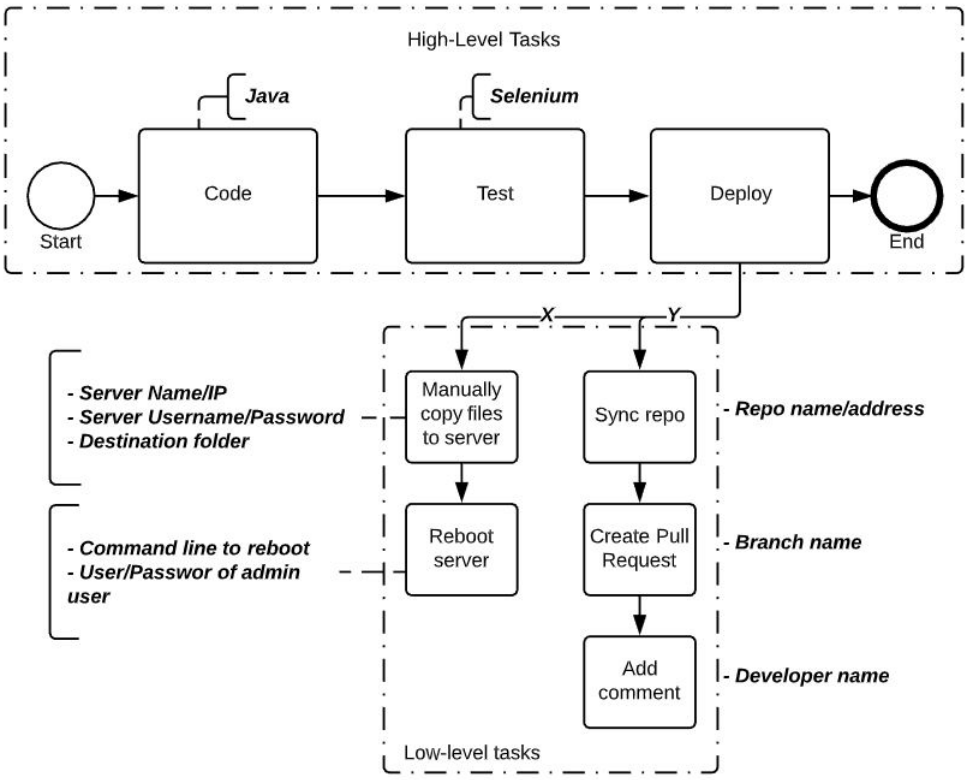

Figure E.2: Example of Workflow with High and Low-Level Tasks.

zone. These other attributes could also influence Gabi's workflow.

The idea is to provide a chatbot that recognizes the developer's current context, having a database containing the history of interactions and workflow, and can suggest the steps the developer should follow. The workflow and context should be embedded in the conversational agent's understanding, and tracked by the chatbot. The following conversation, for example, could occur between Gabi and the chatbot.
Chatbot: Hi!
Gabi: Hi, I need to *deploy* project *x* in *development* environment.
Chatbot: Sure. You should manually copy the files to server 10.1.1.1, into folder `/usr/system/folder1`. Use your username gabidev to access the server.
Gabi: Alright, I copied the files. What is next?
Chatbot: You should reboot the server using this command: pseudo-reboot-n. The password is 1234567890.

Gabi: What's next?
Chatbot: You are all done! Please check the application using the following link localhost:8000/mvp.

Knowing all these different contexts are available and could be communicated to developers using a chatbot, we then executed a study to understand the preferences of software developers when using chatbots to support their work. The next section presents the study, followed by the results.

E.3 The Expected Future

In this discussion, we envision a new paradigm for chatbot use that knows the software development context and relies on machine learning techniques to support developers when executing their tasks. The purpose of the approach is, therefore, to capture the tacit or implicit context and feed it back in a useful way such as by making informed suggestions to developers. Processes based on machine learning and communicated through a chatbot should lower the cognitive load of developers, provide context-aware and real-time support for task execution, and guide developers through the development steps such as deployment. Figure E.3 illustrates the envisioned architecture of the solution.

This novel paradigm can potentially transform the way software development is currently undertaken by allowing developers to receive valuable information and guidance in real-time while they are developing their projects. In contrast with the way developers interact with existing IDEs, the proposed paradigm proactively provides developers with the information and guidance they need, where, when and how they need it. As a consequence, developers will be supported in their interactions, productivity, and decision-making.

We anticipate this approach to be deployed in many different software development scenarios, with tools and procedures that explore the reuse of information on software development, providing support and automation to recurrent tasks. Hence, developers can focus on the creative aspects, use of programming languages, data structures, other product-related concerns, and user-focused solutions. The proposed paradigm is one step further towards bringing more knowledge to developers, both experienced and novice, during specific software development activities. This paradigm builds upon our previous work on reusing relevant information to support developers in executing their tasks [108] and the relevance of context in software development projects [102].

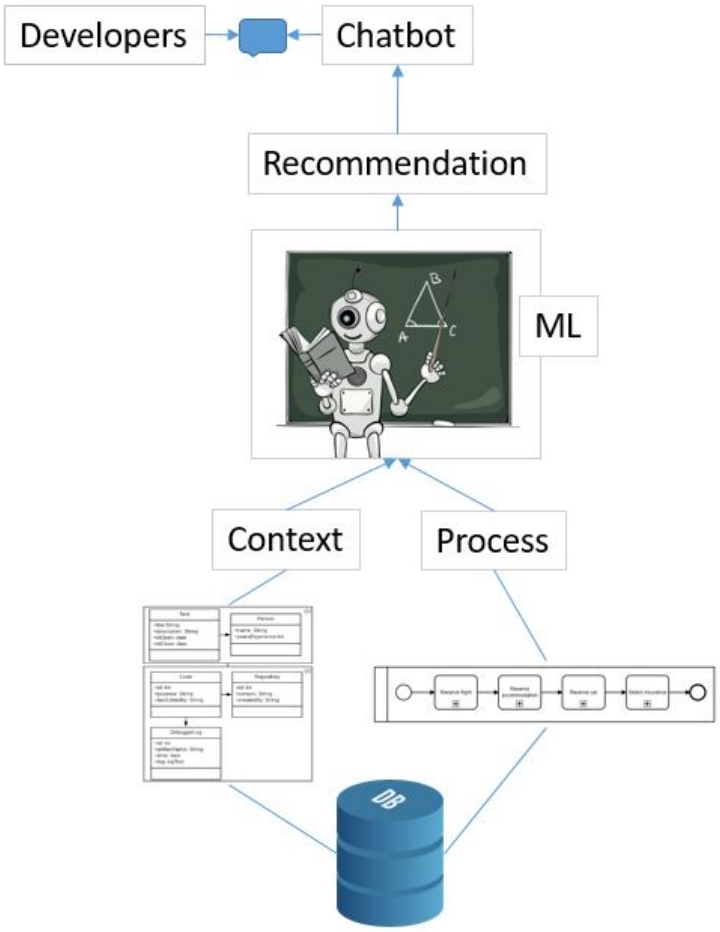

Figure E.3: Prototype of a conversational channel connecting developer, context and process.

E.4 Making this Future Possible

To realize a human-machine software collaboration paradigm based on context, cognitive support, and machine learning, requires performing the following research activities.

Research on Software Development Context. We believe it is essential to understand and capture context for software tasks so that adaptive context relevant to software development can be leveraged. Being able to handle this context would allow

developers to focus on creative tasks rather than on how to execute a specific procedural task or wonder what should be done next to uncover specific information. The volume of information to be handled, as well as the speed at which such information will arrive, is often massive and rapid. Moreover, the variability of context formats is also an aspect to be considered. Examples of relevant contextual information are the next artifact to be edited or read, an API tutorial, a code snippet, or knowledge from another developer. We strongly believe that the nature of capturing the different contexts and presenting the tasks related to that context is already a significant contribution to the field.

Relevant questions include: How can we deepen the use of context to guide developers in real-time? Which context should be leveraged to guide developers in their tasks? How can we better capture and reuse context information in software development? How can massive contextual information be stored so that it can be accessed and used to generate knowledge for software developers?

Environment-Developer Interaction. This approach is within the scope of providing means for developers to interact with a system that is supposed to support them during development. This support is through a conversation where a chatbot supported by a context model should be aware of the developer's set of tasks. This means the chatbot should know both the workflow (process execution and project characteristics) and the contextual information of the project, as well as be prepared to capture information from developers and other resources providing answers. Bradley and colleagues [20] have considered a context model for supporting conversational developer assistants that use the context elements needed to support workflow involving a distributed version control system. In contrast, the focus of our approach is on guiding developers in the steps they must perform throughout the development project, considering different contextual information and how they impact the work in a software project. Automation of tasks could eventually happen as we understand the process and the influence of contextual and cognitive utterances and differences. The goal is to automate contextual communication so that developers do not have to rely on their memories, or mental maps or search huge sets of documentation. The chatbot would act upon a context model and would have embedded machine learning and history information to provide results to the developers. The solution is also intended to be non-invasive, relying on the implementation of techniques such as aggregated data or anonymization. Concerning the feasibility of implementation, our ongoing work on process-aware conversational agents has demonstrated that it is possible to integrate a basic context model into a chatbot tool such as Rasa. We have also implemented a preliminary integration of

the chatbot with a workflow machine called Camunda, allowing the chatbot to receive process execution information [93].

Relevant questions include: How can we establish a communication channel between developers and their environment? ***How can a chatbot be effective in supporting developers? In which ways are developers willing to accept this new technology?***

Knowledge-Intensive Process Guidance. Knowledge workers such as software developers rely on their minds and creativity to implement software solutions. Providing smart solutions when building software is expected, so developers must worry about following patterns, and processes and adjusting to project needs. This information is usually implicit or tacit and in developers' memories. Automated task guidance or task navigation support should improve developers' ability to work more efficiently. This system would count on intelligence from machine learning and project history to recognize the context in which developers are working and suggest the next steps, according to their context. Development tasks are knowledge-guided, and capturing the context is essential. Providing feedback to developers with information should be part of the solution. Once the context is captured and understood, developers have a way to take advantage of this context through the chatbot. A method for process navigation based on machine learning should be provided. The method may include training a machine-learning model by at least processing training data (context and process) with the machine-learning model. The training data may include records of the executed process with the current context at the time of execution. Correlations between context and the executed process shall be done so inferences of the next steps can be provided to developers.

Relevant questions include: How to leverage process information to guide developers in what they must do? What types of suggestions related to the software process can improve SD? How can developers take advantage of process guidance? How can chatbot inferences be integrated with ML inferences to produce rich feedback for developers? How could we maximize the guidance of developers through process and context exploration?

Experimental studies. Qualitative and quantitative studies to demonstrate the feasibility of the proposed approaches as well as the implementation in software companies. Application areas of interest for the studies include deployment, testing, version control and managing issues or tasks.

Relevant questions include: How can we verify the acceptance of a new cognitive assistance-based way of development by software developers? In which types of interactions would developers be most interested? How can we have confidence this idea will do what developers want?

E.5 Conclusion

We discuss a novel paradigm to improve the work of software developers, by providing contextual information about the tasks they are performing through cognitive and intelligent support. It employs capturing the implicit or tacit context and feeding it back in a useful way through suggestions to developers in real-time as they are executing the software project. The novelty of the paradigm arises from the approaches and tools used to capture and inform the developers' context on the fly, considering different contexts. The findings will be of interest since their use should have several advantages including less time to develop software, less effort to share knowledge among team members, enhanced collaboration, application of collective wisdom, knowledge transfer from experts to novices and many other useful contributions. As broader implications of the results, we believe that the impact of the proposed research will contribute to facilitating the development of new avenues in software research as well as support improved ways to develop software, a critical area that is in high demand and has enormous future growth potential. This is the first research program where the combination of three different pillars (context, chatbots and machine learning for process navigation) has been exploited to predict appropriate information during software development.