Research Statement

I am interested in inventing technologies by studying and modeling both *human factors* and *software engineering factors,* in the context of programming tasks. The primary goal of my research is to empower problem-solving professionals (experts, novice programmers, end-user programmers, and underrepresented communities) by integrating software engineering activities into their existing workflow without changing the nature of their work or priorities by using or inventing human-computer interaction methods. Towards my goal, I follow an iterative approach of (i) performing empirical studies to investigate current needs and problems; (ii) defining theoretical propositions that address these problems; (iii) designing tools that embody the theoretical propositions; and (iv) evaluating the tools and the underlying propositions via confirmatory studies. I have developed new strategies, theories, visualizations, and prototypes for problem-solving professionals; the following sections detail current, past, and future research.

1. Fostering Collaborations

Currently, along with my students, I am working on three projects that aim to foster collaborations with agents or humans for programming tasks. These projects utilize artificial intelligence techniques to enhance the programmer's experience. The following details the current progress on our active projects, as well as future work aimed for the next 5-10 years.

1.1. Design and Develop Inclusive-Conversational Agents [1-6]

- ***Honorable Mention* & NSF CAREER Award***

Advances in Human-Computer Interaction (HCI) and Artificial Intelligence (AI) are heralding the era of intelligent anthropomorphic personal agents (Alexa, Siri, and Google Assistant) that permeate society. This research draws and expands on the underlying ideas from these domains to address the unexplored but compelling problem of designing conversational agents (CA) that support pair programming. The pair programming CA paradigm, which increases code quality, efficiency, knowledge transfer, and self-efficacy, will impact software development, teaching and learning programming, and help retain underrepresented groups in computer science. The overall aim of the project is to enhance infrastructure by providing research datasets, end-to-end conversational agents, and educational materials to the academic community. The following are the components on which progress has already been made.

Design Space of Anthropomorphic Agents [1, 2, 5]: Our in-depth qualitative analysis revealed 12 design guidelines for an interactive pair programming partner. We created an inclusive agent following our 12 design guidelines as well as research from CA, software engineering, education, human-robot interactions, psychology, AI and gender studies. Further, we iteratively enhanced it's design using a series of Wizard-of-Oz studies. The Agents active application of soft skills - adaptability, motivation, and social presence - as a navigator increased participants' confidence and trust, while its technical skills - code contributions, just-in-time feedback, and creativity support - as a driver helped participants realize their own solutions. Part of this work was done in collaboration with researchers from IBM T.J. Watson, New York.

***Feasibility as a Pair Programming Partner [4]**: We investigated the tradeoffs of substituting a human with an agent in a pair programming context with gender-balanced studies of 9 human-human pairs in a remote lab setting and 14 human-agent pairs using Wizard-of-Oz methodology. The results were triangulated with interviews and survey data. Upon comparing human-human and human-agent studies, we found: (i) no significant differences in code correctness, task progress and self-efficacy before and after performing the programming task; (ii) agent partners were unable to explain how the agent-provided code worked, explain its logic and elaborate on their partners' ideas whereas human partners could analyze the logic behind agent-provided code to extract the ideas; (iii) human partners trusted agent instructions often without any

questions, in one case responding, *"I'm going to blindly believe you;"* (iv) human partners interrupted agents when they did not know what to do next, were stuck, were unsure about their problems or wanted clarification whereas human-human pairs were more hesitant and asked follow-up questions to understand, clarify or verify their partners' decisions; (v) human partners showed the same humility towards agents and humans by attributing success to the group using words like "we" and taking personal responsibility for mistakes using "I" or "me;" and (vi) human partners acted on agent ideas or dismissed agent ideas, whereas humans tended to modify/refine their human partners' ideas.

Utilizing Machine Learning [2, 3, 6]: Developing an agent poses challenges associated with collecting, labeling, and analyzing programmer conversational data. We explored the viability of integrating ML in a pair programming agent using support vector machine (SVM) and found an accuracy of 57%. Further, we conducted two controlled lab studies and collected 8,324 utterances of pair programming conversations. By following an open coding process, we developed a hierarchical classification scheme of 26 annotations tailored for pair programming. We compared three state-of-the-art transformer-based language models, BERT, GPT-2, and XLNet. Further, we demonstrated the feasibility of using transfer learning. Our qualitative analysis revealed (i) stylistic differences; (ii) questions asked to an agent; and (iii) men participants showed less uncertainty and trusted agent solutions more, while women participants used more instructions and apologized less to an agent. Hence, machine learning (ML) algorithms need to collect and train on gender-inclusive data for programming.

Future Work:

Implementing a Robust End-to-End Programming Agent: To create an end-to-end pair programming CA paradigm, we will use Reinforcement Learning (RL), Imitation Learning to mimic an expert-provided policy, train with a simulated programmer, Human-in-the-Loop, and studies with real programmers.

Automating Code, Test Case, and Feedback Generation: Pair programming CA must uniquely support all stages of software development (i.e., implementing code, testing). Currently, language models exist to generate code (GPT-3 trained on GitHub, GitHub Copilot) and automated test cases (Randoop and EvoSuite); however, these tools must be adapted for our CA to give feedback at method or line level.

1.2. Collective Foraging Intelligence [7-13]

- ***Best Paper* and AFOSR YIP Awards***

Collaborative software development allows collective intelligence but has the core problem of how to optimize the sorting and analysis of a vast array of "information fragments" in order to produce "optimal information" that is meaningful and applicable to a specific goal. The overall aim of the project is to investigate the use of the past collective information seeking behaviors of individual developers, specifically experts, to reduce the overhead of finding relevant information for new developers working on similar tasks. We are currently collaborating with a researcher from Microsoft Research Lab at Cambridge, UK.

This research is inspired from the collective intelligence patterns among animals. Most creatures in the wild have developed ways to communicate information between individuals/groups when performing optimal foraging tasks. For example, ants use pheromone trails and stridulation, while honeybees use the celebrated waggle dance to recruit others to a food source. Similar to this behavior, we believe that developers also leave traces of their intelligence (e.g., commit logs), and their foraging behavior among artifacts (within Open Source Software and on the web) that can be captured and utilized. We are developing our system grounded in a theory, namely Information Foraging Theory (IFT). IFT posits that people seek information in ways similar to how predators (e.g., developer) forage for their prey (e.g., information) in hunting areas or patches (e.g., source file) by gathering scents from cues (signposts e.g., links) in the environment (e.g., IDE, webpage). In the past, I have introduced IFT to two new domains:

End-user Programmers' Debugging and Foraging in Online Repositories [9-13]: We contributed to the IFT theory by creating a model and refining our understanding of the debugging behavior of end-user programmers by separately focusing on the localization and fix of faults. Our study revealed that EUPs spent substantial time foraging information to locate faults and foraging back and forth across the mashup programming environment and websites to fix faults. Further, we investigated information-seeking behavior of EUPs when using online repositories for MATLAB and AppInventor. We found that participants used goals and strategies to bridge the gaps in knowledge by reassessing their knowledge base. Our studies revealed new cues and strategies to reduce the cost and increase the value of a patch/variant.

***Variation Foraging Theory and Computational Models [7, 8]:** Foraging among too many variants of the same artifact can be challenging when they are very similar as it requires high cognitive effort, especially for novice programmers. We developed a theory of variation foraging through a qualitative empirical study investigating how novice programmers reuse variants. Our work provided the first empirical evidence that IFT has gaps when applied as a theory of variation foraging. Further, we created a predictive computational model of foraging, PFIS-V, that accounts for multiple similar variants. Our results show that PFIS-V is up to 25% more accurate than the previous programming computation model.

Future Work:

Creating Global Data and Cognitive models: To address the information-seeking problem for software teams, (i) we will investigate IFT constructs and propositions to represent heterogeneous data sources such as code repositories and the web; (ii) Based on these propositions, we will create configurations of a global data model accounting for the experience of developers. Also, we will determine an optimal data model that is capable of discovering and exploiting collective (teams of developers) foraging knowledge already encoded in shared information sources; (iii) We will enhance the existing IFT cognitive model to use unsupervised, active ML to capture the "wisdom of crowds" for optimal recommendations to new developers; (iv) Finally, we will create agents to simulate data and determine the optimal global data model among various configurations and evaluate the recommendations of "good sources" by the new IFT model. These evaluations will help in fine-tuning the data model and prediction algorithm.

Connecting Programmers to Relevant Variants: Codoban et al. found that the current version of configuration management tools for developers need better visualizations, organization of history, usability, and change notifications. We will design interaction mechanisms and integrate support that allow programmers to find, select, and integrate code effectively and efficiently. To understand the developers intent when creating and managing variants, I plan to conduct more studies to collect a suite of design principles, prototypes, and interaction techniques for reuse at a very large scale.

1.3. Pairs' Brain Responses and Gender Effects [14, 15]

This project aims to help us understand brain-to-brain interactions between pairs of individuals while working on complex cognitive tasks as well as the effect of gender. Towards this aim, we are using real-time simultaneous neuroimaging ("hyperscanning") of pairs to study their brain-brain interactions as they problem solve. This is in collaboration with the Laureate Institute for Brain Research, supported by IPGP grant.

Gender Differences in Online CS Education [14, 15]: To understand how technology-mediated remote pair programming hinders remote collaboration, we empirically investigated same- and mixed-gender pairs with a lab study and a large-scale survey. Our results revealed that: (i) same-gender pairs tend to be democratic, set aside preferences and divide tasks fairly whereas mixed-gender pairs tend to have an authoritative partner who often take control of the tasks; (ii) women tend to use more non-verbal cues and are more aware of gender differences; (iii) women participants reported that they were interrupted by men partners more; and (iv) men participants believed that their men partners were more rude and gave more negative feedback. Our studies also revealed features for online tools to promote gender-inclusiveness.

Future Work:

Adding Biometric Features for Improved ML Predictions: We will utilize ML algorithms with multiple features identified to classify differences between individuals related to communication style and facial expressions to predict neurobiological processes. Our ML pipeline will use novel feature selection and prediction algorithms with demonstrated ability to integrate heterogeneous data types and identify statistical interactions between brain regions. We will explore approaches such as neural networks, SVM, and decision trees to utilize the large generalization capacity of a combination of data-driven models.

Develop Gender-Inclusivity Inspection Methods: Current gender-inclusivity inspection method GenderMag supports one-user problem-solving activities with technology. We have established that same-and mixed-gender pairs' tasks are affected by the combination of their leadership styles, pair programming roles (navigator/driver), and self-efficacy; the new gender-inclusivity inspection method should reflect these. The new facets for collaborative activities discovered can be integrated in personas and cognitive-walkthrough to develop a gender-inclusive inspection method for collaboration.

Virtual Reality Training to Foster Gender-Inclusivity: Virtual reality social cognition training for collaboratively developing software with different genders can provide dynamic and realistic opportunities for social success in both novice and professional programmers.

2. Supporting Problem Solving Professionals [16-25]

- ***Best Paper Awards****

Our studies have helped to understand the problem-solving professionals' behaviors and strategies; based on the results from these studies, we have iteratively designed and developed tools for supporting the following activities:

Problem Solving [16, 17]: In collaboration with researchers at Oregon State University (OSU), we created a theory-based approach called the "Idea Garden" that considers how EUPs generate ideas when learning programming concepts, based on Minimalist theory. It uses an integrated, just-in-time combination of scaffolding for problem-solving strategies, and for programming patterns and concepts. We empirically investigated our prototype with two formative think-aloud studies at summer camps of 90 teens. Our results showed that participants required significantly less in-person help and the tool supported diverse information processing and problem-solving styles as well as cognitive stages for varying tasks.

***Exploratory Programming [7, 8, 18-22]:** This project supported exploratory programming at workspace, file, and online-repositories. This research was part of the Exploratory Programming project, a collaboration involving Carnegie Mellon University (CMU), OSU, University of Nebraska-Lincoln, and University of Washington. Our online survey revealed that EUPs create variants frequently (88%), and rely primarily on memory to manage variants (82%) rather than available tools. To support automatic variation management, we developed AppInventorHelper for the App Inventor (mobile development environment) and PipePlumber for Yahoo! Pipes (web mashup). We discovered design requirements for an EUP variation management system and used novel visualizations to (i) organize their program variants automatically; (ii) visualize all variants at once; and (iii) select appropriate variants based on parameters. Our three empirical evaluations with EUPs showed improvements in reuse and debugging.

***Assessment of Programmers' Socio-Technical Skills [23, 24]:** In collaboration with OSU, University of California, Irvine (UCI), and Masey University, New Zealand, we designed a tool (Visual Resume) to aggregate developer activity traces across GitHub and Stack Overflow to portray their technical and soft skills and help potential employers, as well as recruiters, to locate suitable candidates. Two scenario-based, formative evaluations showed that participants appreciated the ability to compare candidates based on overall summaries, and to drill down to a particular contribution to assess their quality.

Debugging in Web Based Distributed Programming Environments [9, 12, 13, 25]: We classified bugs as Intra-module (bugs that occur within a module) and Inter-module (bugs that involve interactions between modules) for web mashups. Based on these classifications, an anomaly detector automatically detected bugs based on *static analysis, dynamic analysis,* or a *combination of both*. We developed a debugging interface to reduce cognitive load using Nielsen's heuristics and designing error messages using Shneiderman's guidelines. Our lab study showed that debugging support helped EUPs find and fix bugs.

Future Work:

Supporting Big Data Platforms: To support EUPs including data scientists, analysts, researchers, and consumers of the analyses (decision makers, managers), I plan to support relevance of data in big data platforms (e.g., QuantCell, Optique). With the increased relevance of big data's numbers, there is a risk of misunderstanding results and in turn, misallocating important public resources. Further, supplement information along with the big data numbers to provide a better explanation of how the results were derived, based on the inputs, i.e., the provenance of the result data.

Supporting Just-in-time Learning of Third-party APIs: APIs are varied and often complex, making them difficult to use, and programmers at all levels spend significant time learning new APIs. To support better learning of APIs, we are motivated from worked examples in education literature, which have been harnessed to support learning tasks by providing step-by-step procedures and reducing cognitive load.

3. Mining Online Repositories for Predicting User Behavior [26-28]

In collaboration with researchers at the University of Tartu, Estonia and UCI, I have used social network analysis (SNA) and ML techniques to understand the socio-technical behavior of programmers. We investigated the correlation between technical and social skills of developers. Our quantitative analysis of social skills compared with technical skills indicated that better collaboration competency skills are associated with enhanced coding abilities and code quality. We also investigated flocking and migration patterns of developers within and across platforms (GitHub & Stack Overflow) and found mostly flocks of two developers migrated within and across platforms. Further, we investigated the success of apps and found app users' satisfaction and reviews on the App Store did not reflect the developers' preferences and issue reports on GitHub; larger team sizes and the presence of sub teams had a significant impact on ratings.

Future Work:

Behavior Classification for Phishing Attacks: Millions of people experience phishing attacks every year. To help identify malicious phishing attacks, I want to automate the behavior classification by analyzing data logs using hierarchical clustering algorithms and develop personas for company employees and customers based on similar user characteristics such as gender, motivations, and goals. By using ML and SNA techniques and security policies, the incoming traffic can be monitored to identify changes in the behavior for a given persona, the systems can allow/deny access to company resources.

4. Avoiding Code Redundancies [29, 30]

Reuse promotes redundancies. Current code clone detection techniques rely on code structure and their dependencies and are resource intensive, one way to address this is to utilize code comments that carry important domain knowledge and are often used by developers. Our empirical investigation verified that code comments can be used effectively to identify duplicate or near-duplicate code clones.

Future Work:

Avoiding Reanalysis: Utilize program partitioning and regression testing techniques to avoid redundancies caused by re-analyzing and re-testing changed programs in online repositories. These results will be applicable to professionals as well as in EUPs programming environments.