Towards J.A.R.V.I.S. for Software Engineering: Lessons Learned in Implementing a Natural Language Chat Interface

Rahul Pandita, Steven Bucuvalas, Hugolin Bergier, Aleksandar Chakarov, and **Elizabeth Richards**
Phase Change Software LLC
Golden, Colorado, USA
{rpandita, sbucuvalas, hbergier, achakarov, and erichards}@phasechange.ai

Abstract

Virtual assistants have demonstrated the potential to significantly improve the digital experiences of information technology workers. We, at Phase Change Software, are working on developing a virtual assistant *MIA* that helps software developers with program comprehension. This work summarizes the key lessons learned and identifies open questions during the initial implementation of the *MIA* chat interface.

Introduction

According to the U.S. Bureau of Labor Statistics[1], software developer jobs are projected to increase by 17 percent from 2014 to 2024; a rate that is significantly faster than the average across all occupations. This growth is fueled by an increasing demand for computer software.

One of the problems faced by the software development industry today is the loss of critical knowledge attributed to the high turn-over rate for developers. (Rigby et al. 2016) noted that a tight relationship between a developer and authored code makes organizations susceptible to information loss. One major factor is that in today's job market, developers change jobs readily and often. Moreover, for some technologies like COBOL, the developer population is simply retiring (with an average age of 50+) and creating a void to be filled by a diminishing supply of COBOL developers. All this contributes to an increasing need for training new developers in the mastery of antiquated programming languages and management of existing large applications.

Brook's law (Brooks 1975) states that adding a developer to a mature project impacts productivity negatively. Specifically, (Foucault et al. 2015) show that newbies who join the team often introduce defects in the code as they move towards proficiency, thereby, bringing down overall team productivity in the process. Recently, (Minelli, Mocci, and Lanza 2015) estimated that on average developers spend 70% of their time comprehending source code. Moreover, (Maalej et al. 2014) find that developers are reluctant to maintain and reference documentation, opting for direct communication instead. With this growing body of evidence showing that the developers spend most of their time understanding what the code does, finding effective means to tackle code comprehension has the potential to significantly impact developer productivity.

With continuous advancements in artificial intelligence, natural language processing, speech recognition, and text-to-speech technologies, virtual assistants are transforming the way commercial organizations interact with customers. For instance, product websites are increasingly implementing chat-bot assistants to answer frequently asked questions and common customer complaints. We have seen a gradual maturity of virtual assistants from Microsoft's iconic but failed Clippy to the sophisticated television quiz show winner IBM Watson®. When implemented correctly, virtual assistants provide economic alternatives to hiring human operators who cater to the information needs of customers.

To address the problem of improving developer productivity, we at Phase Change are working on developing a virtual assistant technology to assist a new programmer to quickly become proficient in a new system. As we develop this technology, we took inspiration from the success and value proposition of generic consumer facing products like Amazon's Alexa®, Apple's Siri®, Microsoft's Cortana®, and the Google Now® assistant. We refer to our assistant as *MIA* (short for "My Intelligent Agent") and as a first step towards realizing *MIA*, we focus on program comprehension, and gradually expanding *MIA*'s capabilities to program composition.

This paper outlines some of the lessons learned in implementing the first iterations of *MIA*'s chat interface.

The *J.A.R.V.I.S.* Metaphor

The primary objective of the *MIA* interface is to facilitate software comprehension. Since learning is a crucial part of comprehension and people learn in different ways, *MIA* has to be able to accommodate different styles and approaches.

We envision *MIA*'s interface akin to Marvel's fictional AI assistant J.A.R.V.I.S.[2] A user interacts with *MIA* using multiple modalities such as traditional keyboard and mouse inputs, gesture and touch-based manipulation of objects displayed on the screen, and user voice commands to interactively perform complex tasks. Specifically, what we aspire

[1] See https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm

[2] J.A.R.V.I.S. ("Just A Rather Very Intelligent System") is a highly advanced AI agent developed by Tony Stark.

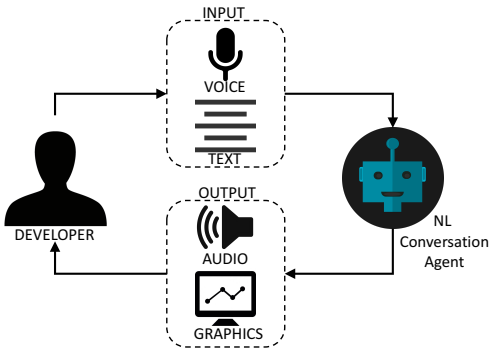

Figure 1: A generic NL Conversation Interface.

for with the J.A.R.V.I.S. metaphor is the "liveness" and natural human-like conversation ability to actively augment human capabilities as if it were an active collaborator rather than a reactive question-answer system.

Our vision for the *MIA* interface is to cater to the needs of various user classes. For instance, if *MIA* caters to developers, a typical Integrated Development Environment (IDE) is a likely choice to model the *MIA* interface. Likewise, if the intended audience is a business analyst or a program manager, the ideal *MIA* interface reflects what they use day-to-day to carry out their tasks. However, in this paper we only focus on the voice and conversational aspects of *MIA*. We next describe a generic NL conversation agent.

NL Conversation Agent

An AI assistant is an *agent* that helps humans understand and react to a system. In our context, *MIA* allows a user to comprehend and manipulate the software system, by accepting as inputs user intension in the form of natural language.

In *Artificial Intelligence: A Modern Approach* (Russell and Peter 1995), Russel and Norvig define an agent as a program that maps the input (perceived by the sensors) in an environment using a *mapping function* to an output (realized by actuators) that is manifested in the environment. A generic Natural Language (NL) conversation agent is shown in Figure 1. A simplified work-flow can be thought of as:

1. the user speaks/types into the interface;
2. the interface captures audio/text through its sensors;
3. the captured audio/text is processed (interpreted) to perform a task to obtain results;
4. the results are then relayed to the user in the form of a user interface change or in the form of audio.

(Hirschberg and Manning 2015) outline the major components of an NL conversation agent (specifically, a *spoken dialog system*) as Speech-to-Text (STT) system, a Dialog Management (DM) system, and a Text-to-Speech (TTS) system. *MIA* leverages various commercial off-the-shelf components to achieve this interaction. For instance, *MIA* uses external STT and TTS libraries to do audio manipulations. However, a key component of the interaction is the DM system, i.e., translating natural language (in text format) into executable actions in the *MIA* interface.

Specifics of implementing a DM component are beyond the scope of this paper. For the general audience, we discuss a naive strategy to realize a DM: at its simplest, a DM can be thought of as a giant rule-based system, where all possible supported interactions are encoded as if-then-else rules.

Although simple to implement, such a system may not scale well: a developer may have to encode all possible variations of natural language utterances for a given command. Expecting users of NL conversation system to use uniform and restricted vocabulary may be unreasonable. It may also be unreasonable to expect users of the *MIA* system to structure their language in a specific style. Fortunately, existing Natural Language Processing (NLP) techniques such as WordNet (Miller 1995) based synonym analysis and word embedding models such as word2vec (Mikolov et al. 2014) help deal with variations in the vocabulary. Additionally, Parts of Speech Tagging (Klein and Manning 2003), Phrase and Clause Parsing (Klein and Manning 2003), and Typed Dependencies (de Marneffe and Manning 2008) effectively deal with the structural and stylistic variations in a sentence (Hirschberg and Manning 2015). With recent advances in neural network technology, it is possible to skip a rule-based approach altogether and learn a domain-specific conversation model (Xu et al. 2017) given a large data-set.

We leave the specifics of implementing conversation agents to focus on generalizing our experience with *MIA*.

Usability Considerations

In this section, we briefly discuss following Nielsen's Usability Heuristics (Nielsen 2005) in the context of the *MIA* interface. Although, we used these heuristics as guidelines to design *MIA*, we believe these are generic guidelines for designing an effective NL conversation agent interface.

Visibility of system status: Ideally, agent conversations should strive for real-time performance. However, there exist cases when a request takes time to process. Agents should keep their users informed about processing through appropriate feedback.

Match between system and the real world: Agents should respond in the users' vocabulary (user words and familiar phrases), rather than system-oriented terms.

User control and freedom: Agents should allow a user to quickly undo an action that a user may have performed in error. Likewise, an agent should support a quick redo of an action that a user may have undone in error.

Consistency and standards: Agents should communicate with their users in a consistent vocabulary. This allows a user to navigate unfamiliar situation with the established conventions of the terminology.

Error prevention: Agents should actively support error prevention. In case of ambiguity, users should be consulted to confirm before an agent commits to an action.

Flexibility and efficiency of use: Agents should cater to the needs of novice users as well as expert users. The interface should allow for expert users to input complex utterances by means of shortcuts (keywords).

Help users recognize, diagnose, and recover from errors: Error messages should be crafted such that they help users to effectively understand the problem and should also constructively suggest a solution.

Lessons Learned

In this section, we outline some of the lessons learned during the course of implementing the first iteration of *MIA*. This list is by no means exhaustive as we are actively developing the *MIA* system. We, however, hope that this list provides a starting point for other teams that are in the initial phase of implementing an NL virtual assistant.

Reuse components to quickly prototype. Although this may be generic advice, we did learn that reusing existing components such as TTS, STT, NLP annotations, and rule engines allowed us to quickly prototype the *MIA* system. Furthermore, it bought us time to focus on the interaction itself rather than on the technologies that normally involve significant investments in time and resources.

Gradually migrate from rule-based to statistical approaches. Our implementation of the *MIA* interface started as a simple "if-this-then-that" system. We hand-crafted the initial set of rules of the NLP component based on *preliminary requirements*. We also chose to collect interaction data as a means to for later improvements to the system.

Query reformulation (Hearst 2009) is a query strategy in which users first execute a general query and then iteratively refine subsequent queries to fine-tune the search results. Search engine users frequently employ this strategy (Jansen, Spink, and Pedersen 2005) and we empirically corroborated that *MIA* users employ similar ones. We used this opportunity to collect variations of multiple queries as formulated by the users. This data was used to further refine the rules in the NLP component to improve performance.

We realized rule-based approaches do not scale as new interaction functionality is added to *MIA*; however, we did not initially have data to use statistical approaches. Most teams starting chat-bot implementations rarely do.

One alternative is to train a model on external yet similar data and then use transfer-learning to customize the model for the specific domain. However, we observed that often the requirements and capabilities of the interface are not well-formed initially but evolve over time before settling. Given the instability of requirements and the large upfront cost of statistical approaches, we recommend starting with simple rule-based approaches and gradually migrating to statistical ones.

Adopt Recommendation Systems. While developing *MIA*, we painstakingly encoded every possible scenario that our team could imagine so that *MIA* could handle almost anything thrown at it. However, even after multiple iterations, we still missed some combination of tokens in the query. Initially, *MIA* handled such scenarios by responding generically: "I don't quite understand, let's try again." While it is a fair response, users get frustrated when *MIA* does not provide ways to remedy the situation. We put in a basic recommendation system (Resnick and Varian 1997) that allows *MIA* to propose alternate queries as a function of keyword matching whenever *MIA* does not understand a user query. In practice, we found that responding with a recommended query *almost* always remedies the situation.

On the flip side, we learned that users are also irked if there are too many recommended queries or they are too far from the user's intention. Thus, we filter our recommendations using a threshold and only show recommendations if: a suggestion scores above the predetermined aggregate threshold (we employ multiple scoring mechanisms) and the total number of recommendations does not exceed three (derived empirically).

Over time users stop using fully formed sentences. In practice, we noted that although bot interaction started with well-formed sentences, it quickly moved to keyword utterances and sentence fragments. Thus, an interpretation component should be designed to handle such multi-modal interactions. This is especially pertinent to rule-based interpretation components. We still emphasize handling fully-formed sentences as we observed that whenever *MIA* users encountered unexpected results, they reverted to using fully-formed sentences. Additionally, we noticed that users seldom use more than two clauses while interacting with *MIA*. This allows us to optimize the design by targeting up to two clauses as opposed to an arbitrary number of clauses.

Subliminal Priming. In psychology and cognitive science, subliminal priming (Kiesel, Kunde, and Hoffmann 2007) is the phenomenon of eliciting a specific motor or cognitive response from a subject without explicitly asking for it. We stumbled upon this effect while developing *MIA*.

Our initial beta tests achieved a fair performance. However, *MIA* often completely missed what a user requested. As testing progressed, we observed that *MIA*'s performance improved significantly, without changing the implementation. We investigated and observed that beta users slightly altered their approach to interacting with the system. They spoke with a slightly affected accent on the words that caused problems for the STT component which resulted in a better translation. Likewise, users also tried variations and then adhered to a specific vocabulary which they perceived worked.

Encouraged by observations, we experimented with a feature in the *MIA* interface, where *MIA* spoke back a normalized version of the user's query. We experimented with text mining techniques such as collapsing synonyms and using operational forms of verbs. We observed that speaking back a normalized version of the query caused the users to quickly converge on a specific vocabulary and sentence construction that in turn significantly improve the performance of *MIA*.

Data driven prioritization: Due to resource constraints, we often face the question of prioritizing action items for the next iteration of implementation. While, in an ideal world we would have an infinite number of resources, that is not necessarily the case in real world. In such situations, we found data to be extremely valuable in arriving at decisions.

For instance, during one of the iterations, we had a choice of upgrading our STT component or further improving the performance of our NLP engine. The general sentiment in the team was to improve the NLP engine, primarily because the component was developed in-house and that was where

most of the team had focused. However, the error analysis indicated that if we improved the STT component, the overall accuracy of the system would increase significantly (by a few percentage points) more than by improving the NLP engine. The error analysis included manually analyzing a random (yet significant) sample of the errors observed with each component to assess the perceived improvements. Although, we were using STT as an external service, switching serves would require significant work to accommodate the requirements of the different provider.

Open Questions

Going forward with *MIA*'s development there are two key open questions that we are investigating.

Conversational Models: The current implementation of *MIA* can best be described as a one-shot-interpretation. This means that *MIA* accepts as input natural language and determines the best possible match for the action to be undertaken. While simple to engineer as a system, one-shot-interpretation is far from a natural conversation form. In natural conversation, the current utterance is both influenced and resolved as a function of past utterances. Furthermore, natural conversations are often guided by a goal. The goal in turn influences the directions a conversation takes and serves as a mechanism to *detect* and *recover* from digressions.

We are experimenting with strategies such as maintaining a cache of the past N utterances to explicitly model the latent state. It is still unclear how to efficiently model conversations and make *MIA* interactions more natural.

Bot cognition: We currently evaluate virtual assistants using the traditional quantitative statistical measures of precision, recall, and accuracy. In contrast, we qualitatively evaluate them on human-like conversation, problem-solving ability, and cognition. However, it is unclear what we mean by the analogy of evaluating cognition when applied to bots.

Bloom's Taxonomy (Anderson et al. 2001) has been used in education as a model to classify a learning objective using levels of complexity. In particular, Bloom's Taxonomy proposes the following levels of objectives (orders from lowest to highest): Remember ⇒ Understand ⇒ Apply ⇒ Analyze ⇒ Evaluate ⇒ Create.

Although it is an appropriate and valuable lens to look at human cognition, Bloom's Taxonomy (or some variation of) may not be appropriate or even applicable to a virtual assistant. Nevertheless, it is highly desirable to have a lens to evaluate and better understand such aspects of bot cognition.

Conclusion

In this paper, we presented our motivation for building *MIA*, our J.A.R.V.I.S-like conversational agent, to bring the benefits of virtual assistants to software developers. *MIA* is a work in progress, and we summarized some of the key lessons learned and open questions we encountered while engineering the first iteration on *MIA*. We hope that our experience helps other developers or teams working on virtual assistants. In addition, we hope that the questions of effective conversation model and how to evaluate bot cognition garners further discussions and scientific work in the area.

References

Anderson, L. W.; Krathwohl, D. R.; Airasian, P.; Cruikshank, K.; Mayer, R.; Pintrich, P.; Raths, J.; and Wittrock, M. 2001. A taxonomy for learning, teaching and assessing: A revision of bloom's taxonomy. *New York. Longman Publishing. Artz, AF, & Armour-Thomas, E.(1992). Development of a cognitive-metacognitive framework for protocol analysis of mathematical problem solving in small groups. Cognition and Instruction* 9(2):137–175.

Brooks, F. P. 1975. The mythical man-month.

de Marneffe, M. C., and Manning, C. D. 2008. The stanford typed dependencies representation. In *Workshop COLING*.

Foucault, M.; Palyart, M.; Blanc, X.; Murphy, G. C.; and Falleri, J.-R. 2015. Impact of developer turnover on quality in open-source software. In *Proc. of the 10th FSE*, 829–841. ACM.

Hearst, M. 2009. *Search user interfaces*. Cambridge University Press.

Hirschberg, J., and Manning, C. D. 2015. Advances in natural language processing. *Science* 349(6245):261–266.

Jansen, B. J.; Spink, A.; and Pedersen, J. 2005. A temporal comparison of altavista web searching. *Journal of the Association for Information Science and Technology* 56(6):559–570.

Kiesel, A.; Kunde, W.; and Hoffmann, J. 2007. Mechanisms of subliminal response priming. *Advances in Cognitive Psychology* 3(1-2):307.

Klein, D., and Manning, C. D. 2003. Fast exact inference with a factored model for natural language parsing. In *Proc. 15th NIPS*, 3 – 10.

Maalej, W.; Tiarks, R.; Roehm, T.; and Koschke, R. 2014. On the comprehension of program comprehension. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 23(4):31.

Mikolov, T.; Chen, K.; Corrado, G.; Dean, J.; Sutskever, L.; and Zweig, G. 2014. word2vec.

Miller, G. A. 1995. Wordnet: a lexical database for english. *Communications of the ACM* 38(11):39–41.

Minelli, R.; Mocci, A.; and Lanza, M. 2015. I know what you did last summer: an investigation of how developers spend their time. In *Proc. of the 23rd ICPC*, 25–35. IEEE Press.

Nielsen, J. 2005. Ten usability heuristics.

Resnick, P., and Varian, H. R. 1997. Recommender systems. *Communications of the ACM* 40(3):56–58.

Rigby, P. C.; Zhu, Y. C.; Donadelli, S. M.; and Mockus, A. 2016. Quantifying and mitigating turnover-induced knowledge loss: case studies of chrome and a project at avaya. In *Proc. of the 38th ICSE*, 1006–1016. ACM.

Russell, S., and Peter, P. N. 1995. *Artificial Intelligence. A modern approach*. Prentice-Hall, third edition.

Xu, A.; Liu, Z.; Guo, Y.; Sinha, V.; and Akkiraju, R. 2017. A new chatbot for customer service on social media. In *Proc. of the CHI*, 3506–3510. ACM.