Natural Language User Interface For Software Engineering Tasks

Alexander Wachtel, Jonas Klamroth, Walter F. Tichy

Karlsruhe Institute of Technology
Chair for Programming Systems Prof. Walter F. Tichy
Am Fasanengarten 5, 76131 Karlsruhe, Germany
Email: alexander.wachtel@kit.edu, jonas.klamroth@student.kit.edu, walter.tichy@kit.edu

***Abstract*—In this paper, we present the idea to use natural language as the user interface for programming tasks. Programming languages assist with repetitive tasks that involve the use of conditionals, loops and statements. This is what is often challenging users. However, users can easily describe tasks in their natural language. We aim to develop a *Natural Language User Interface* that enables users to describe algorithms, including statements, loops, and conditionals. For this, we extend our current spreadsheet system to support control flows. An evaluation shows that users solved more than 60% of tasks. Although far from perfect, this research might lead to fundamental changes in computer use. With natural language, programming would become available to everyone. We believe that it is a reasonable approach for end user software engineering and will therefore overcome the present bottleneck of IT proficients.**

Keywords–Natural Language Processing; End User Development; Natural Language Interfaces; Human Computer Interaction; Programming In Natural Language; Dialog Systems.

I. INTRODUCTION

Since their invention, digital computers have been programmed using specialized, artificial notations, called programming languages. Programming requires years of training. However, only a tiny fraction of human computer users can actually work with those notations. With natural language and end user development methods, programming would become available to everyone and enable end users to program their systems or extend their functionality without any knowledge of programming languages. Myers [1] and Scaffidi [2] compared the number of end users and professional programmers in the United States. Nearly 90 million people use computers at work and 50 million of them use spreadsheets. In a self-assessment, 12 million considered themselves as programmers, but only 3 million people are professional programmers. According to Liberman [3], the main question in the End User Development (EUD) area of research is, how to allow non-programming users who have no access to source code, to program a computer system or extend the functionality of an existing system. In general, spreadsheets have been used for at least 7000 years [4]. The created spreadsheets are not only the traditional tabular representation of relational data that convey information space efficiently, but also allow a continuous revision and formula-based data manipulation. It is estimated that each year hundreds of millions of spreadsheets are created [5].

Our Vision

Programming languages assist with repetitive tasks that involve use of loops and conditionals. This is what is often challenging for spreadsheet users. We work on *Natural Language User Interface (NLUI)* that enables users to describe algorithms in their natural language and provides a valid output by the dialog system for given user description:

- *Find the maximum element of a set:*
  Use an auxiliary variable. Initialize the variable with an arbitrary element of the set. Then visit all the remaining elements. Whenever an element is larger than the auxiliary variable, store it in the auxiliary variable. In the end, the maximum is in the auxiliary variable.
- *Selection sort of a set:*
  The result is a vector. Initially it is empty. Find the minimal element of the set and append it to the vector. Remove the element form the set. Then, repeatedly find the minimum of the remaining elements and move them to the result in order, until there are no more elements in the set.
- *Switching sort of an array:*
  If there are two elements out of order, switch them. Continue doing this until there are no more elements out of order. Out of order means that an element is larger than its right neighbor. The right neighbor of an element x[i] in a vector x is x[i+1].

Ordinary, natural language would enable almost anyone to program and would thus cause a fundamental shift in the way computers are used. Rather than being a mere consumer of programs written by others, each user could write his or her own programs [6]. However, programming in natural language remains an open challenge [7].

Our paper is structured as following: section II describes our previous work on NLUI. Followed by the Section II-D that presents our current work on control flows, discussing conditional and loop statements. Section III evaluates prototype in an user study. Section IV presents related work in the research areas of programming in natural language, End User Programming and natural language dialog systems. Finally, section V presents a conclusion of our topic and future work.

Copyright (c) IARIA, 2017. ISBN: 978-1-61208-538-8

II. Natural Language User Interface

In 1979, Ballard et al. [8][9][10] introduced the Natural Language Computer (NLC) that enables end users to program simple arithmetic calculations using natural language.

A. Dialog System

In 2015, first prototype of an assistant has been presented that uses natural language understanding and a dialog management system to allow inexperienced users to manipulate spreadsheets with natural language [11]. Motivated by a pilot study based on the selected problems from Frey's book *Microsoft Excel 2013* [12] the system requests missing information and is able to resolve ambiguities by providing alternatives to choose from. Furthermore, the dialog system must resolve references to previous results, allowing the construction of complex expressions step-by-step. The system architecture consists of a user interface responsible for human interaction, as well as a natural language understanding and a dialog management unit (See Figure 1).

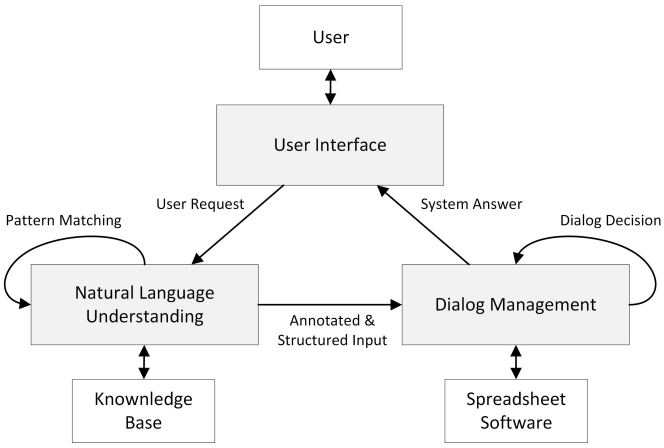

Figure 1. Architecture Overview

In a first step, the natural language understanding unit (NLU) performs essential language analysis relying on a basic vocabulary specifically built to cover the system's domain. Synonyms are substituted using a handcrafted synonym database. Mathematical terms and numerical values as well as references to regions within the spreadsheet are tagged. In the following step, the system groups elements representing a sentence or clause to enable subsequent analysis.

The purpose of the dialog management unit (DMU) is to deal with the tree structure that has been created by the NLU unit, resolve references, create a valid spreadsheet formula and generate a human-like response to the user input.

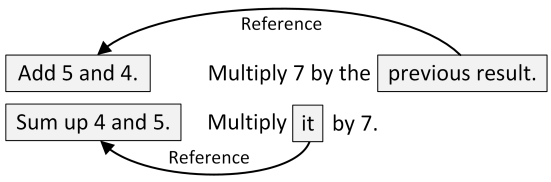

Figure 2. Resolution of references

The evaluation of the prototype exceeded expectations. 80% of 170 tasks have been solved successfully. The system helped users to solve tasks and received positive feedback from nearly two thirds of the users. Inspired by the Turing Test, the authors asked 17 independent spreadsheet users to formulate requests for particular calculation tasks. Each task was answered by both, the prototype and a human, independently. Afterwards the participants were encouraged to identify the computer generated response. This however turned out to be surprisingly hard to decide. With 34 decisions made in total, 47.1% falsely identified the dialog system answer as human. This result indicates that the prototype is capable of generating suitable responses for sufficiently specific requests within the language domain.

B. Active Ontologies

In early 2016, the natural language dialog system has been extended with a natural language dialog system based on active ontologies which enables the user to create and manipulate excel sheets without having to know the complex formula language of excel [13]. Our system is able to resolve references, detect and help resolve ambiguous statements and ask for missing information if necessary. While already quite powerful this system was not able to handle conditions properly or understand statements involving loops or instructions affecting multiple cells. In this paper, we will present an approach on how to attack these weaknesses.

By adding additional information to an ontology, such as a rule evaluation system and a fact store, it becomes an execution environment instead of just being a representation of knowledge. Sensor nodes register certain events and store them in the fact store. An evaluation mechanism tests the new facts against the existing rules and performs the associated action if one or more rules apply to the stored facts. The old system consists mainly of two active ontologies. One in charge of interpreting the user input and one generating answers according to the interpretation.

In our system each rule is represented by a separate node in the active ontology. By connecting nodes the developer decides which type of facts are relevant to which node. In [13], we presented four different types of nodes:

1) Gather-Nodes: These nodes gather the information of all children nodes and only create a new fact if all necessary children facts exist.
2) Selection-Nodes: These nodes gather all information of their children and pass on the most fitting according to some score.
3) Pass-Nodes: These nodes bundle all obtained information of their children into 1 new fact.
4) Sensor-Nodes: These nodes are the "leaves" of the ontology and react directly to the user input.

Each node-type can be seen as one possible evaluation mechanism. While with these types a developer is able to cover most parts of standard domains of dialog systems one can think of far more complex ones. This is where our new system comes into play. By allowing the developer to use his own evaluation mechanisms we created an infinite amount of new possibilities what our system is capable of.

Copyright (c) IARIA, 2017. ISBN: 978-1-61208-538-8

C. Interactive Spreadsheet Processing Module

Interactive Spreadsheet Processing Module (ISPM) [14] is an active dialog management system that uses machine learning techniques for context interpretation within spreadsheets and connects natural language to the data in the spreadsheets. First, the rows of a spreadsheet are divided into different classes and the table's schema is made searchable for the dialog system (See Figure 3).

	A	B	C	
1	**Table 1: persons**			CAPTION
2	**name**			SUPER HEADER
3	**first name**	**last name**	**age**	HEADER
4	**group A**			GROUP HEADER
5	Sloane	Morgan	37	DATA
6	Dustin	Brewer	33	DATA
7	Valentine	Yates	38	DATA
8	Michael	Gregori	50	DATA
9	**group B**			GROUP HEADER
10	Ina	Hoffman	40	DATA
11	Oliver	Hopkins	27	DATA
12	Damon	Vasquez	22	DATA
13	Mark	Richards	25	DATA

Figure 3. A spreadsheet table annotated with row labels

In the case of a user input, it searches for headers, data values from the table and key phrases for operations. Implicit cell references like "people of age 18" are then resolved to explicit references using the schema. Using the ISPM, end users are able to search for values in the schema of the table and to address the data in spreadsheets implicitly, e.g., *What is the average age of people in group A?* (See Figure 4).

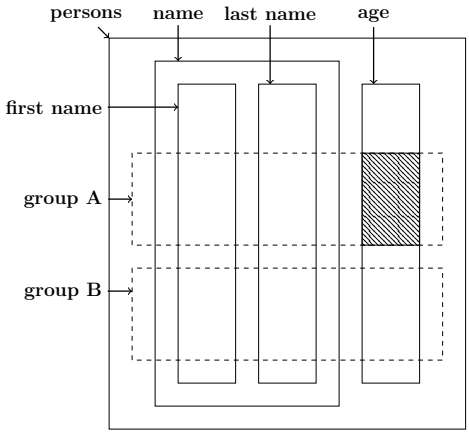

Figure 4. Context detection of user input

Furthermore, the system enables users to select (88% successfully solved), sort (88%), group (75%) and aggregate (63%) the spreadsheet data by using natural language for end user software engineering.

D. Control flows

Two new modules have been developed to extend the current system for support of control flows. The first module is capable of handling conditional instructions and the second is able to understand statements that contain loops.

1) Conditional Statements: Conditional instructions are often hard to understand due to their complex grammatical and contextual structure. Also references are complicated to resolve in this kind of sentences. The advantage of conditions is that they have a small set of key-words (such as if, in case of, etc.) that indicate that the user uses a conditional statement. In domain of spreadsheet manipulation to be able to understand, the condition has to result in a boolean operation. These two facts enable us to develop a specialized service dealing with conditional statements. We react to the keywords and try to find a boolean value in the user input. If we can not find any boolean operation, dialog system asks the user directly for it. After user answers, it is just recognizing which action the user wants to perform. Unconditional statements were already supported in our older system, so we can rely on it to find the proper action.

As already annotated *if* gets recognized as a condition keyword. The system already knows that it is dealing with a conditional statement. *A1 is greater than 3* is a boolean operation and may be used as condition. The trivial statement *save 5 in B1* can be easily recognized as unconditional action and handled by our system (See Figure 5).

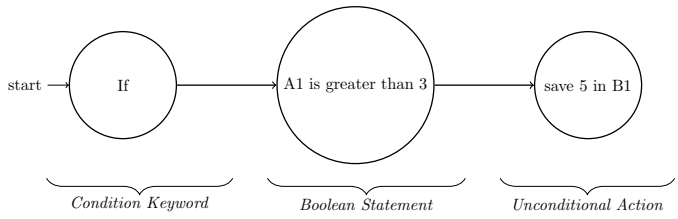

Figure 5. Example for a conditional statement

2) Loop Statements: Dealing with statements that affect more than one cell can be seen as a looped instruction. In that case, the target of the instruction is the loop variable. Knowing this, we can handle it in a similar way we used for conditional statements. In contrast to conditions, loop do not necessarily have to contain clear keywords. Often times these keywords are hidden within the sentence like *for all*, *for each*, *as long as*. However, there may also be explicit instruction like *do something three times*.

Once any of these keywords are recognized, the system has to find the corresponding action and execute it for the given range of cells. In order for our active ontology to be able for recognition the proper action, we introduced a Looptarget. This is an operator that acts like a normal cell and is able to be recognized by normal actions (such as save, multiply), but at the same time indicates that it can be executed on a range of cells. Once activated by a loop keyword, the loop service reacts to actions containing the regarding targets. In this case, next step is to split input into several instructions for each cell of the given range. This may also be combined with conditional instructions. User input *add 2 to B2 for each cell of column A*

Copyright (c) IARIA, 2017. ISBN: 978-1-61208-538-8

which is greater than B2 is correctly handled by our system (See Figure 6).

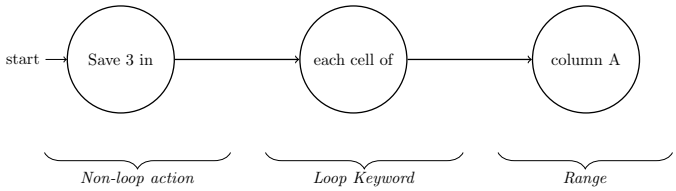

Figure 6. Example for a loop statement

3) Sample Dialog: To demonstrate the extended features, we present a short dialog between the NLUI and user. For each iteration of the dialog we listed the user input (User), the operation the system executed as consequence of this input (NLU) and the response generated by the system (DMU):

User: Write 30 in all cells of column A

NLU: save(30, A:A)

DMU: Already done. Is there anything else i can do?

User: Subtract 10 from each cell between A1 and A10

NLU: save(A1:A10 - 10, A1:A10)

DMU: Ok, I did that for you.

Can I do something else for you?

User: No

DMU: Alright

...After a break, user turns back to the NLUI

User: For each cell of column A which is greater than 25

add 2 to B2

NLU: conditionalAction(A:A>25, save(B2 + 2, B2))

User: If A3 is greater than A2 write 2 in A3 else in B3

NLU: conditionalAction(A3>A2, save(2, A3), save(2, B3))

DMU: Ok, I did that for you.

User: If A5 is greater than B5

NLU: NoAction

DMU: I see you used a condition.

What do you want to do if the condition is true?

User: Save 4 in B3

NLU: conditionalAction(A5 > B5, save(4, B3))

DMU: Everything done as you wished.

III. EVALUATION

To evaluate the quality of our system we conducted an user study. We asked 12 participants to solve given tasks by using our system and recorded their experience with it. In detail we asked for each task

- whether they were able to solve the task,
- if the system was helping them to solve the task,
- if the system output was natural,
- and if the system was able to understand their input.

The participants were Non-native English speakers and the majority of them have never used our system before. Most of them stated that they knew and already used excel before, but not on a regular basis.

Since we already evaluated our system for standard arithmetic tasks, response time as well as scalability in our last paper [13], we specifically designed the tasks to test the discussed control flow features. e.g.:

- Insert the specified value 10 into all cells of a column.
- Multiply all cells in a range (between A1 and A10) that are greater than 2 by 3.
- If the value in cell A3 is greater than A2, they should add 2 to B1, else to B2.

The results show that the users where able to solve more than 60% of the tasks at least partially and found our system as useful in over 60% of the cases (see Figure 7). Additionally nearly 70% of the system outputs were considered as natural by the participants. The participants stated in over 50% of the cases that the system didn't understand their input. The improvement of the systems output has to be worked on. Overall, the system's quality was rated at 3.33 out of 5 stars, and except for one participant all participants said that they would use our system.

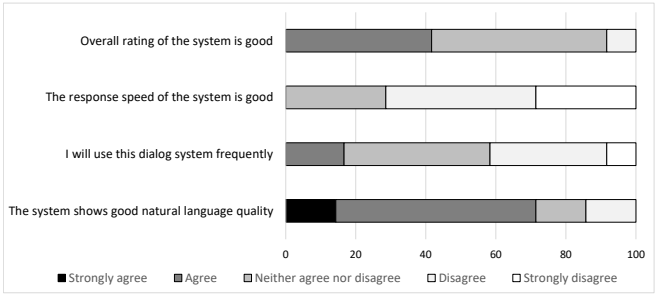

Figure 7. Overall results in %

While this result demonstrates that our system is far from perfect it also shows that there is added value when using the system especially for inexperienced users. Knowing that nearly half of the unsolved tasks stemmed from the same question and the most common problem were synonym problems which are easy to fix the results we achieved are auspicious. Since our system will most likely only improve in coming versions due to the growing number of services and the size of our word databases we consider this a promising approach.

Copyright (c) IARIA, 2017. ISBN: 978-1-61208-538-8

IV. Related Work

The idea of programming in natural language was first proposed by Sammet in 1966 [15], but enormous difficulties have resulted in disappointingly slow progress. One of the difficulties is that natural language programming requires a domain-aware counterpart that asks for clarification, thereby overcoming the chief disadvantages of natural language, namely ambiguity and imprecision. In recent years, significant advances in natural language techniques have been made, leading, for instance, to IBM's Watson [16] computer winning against the two Jeopardy! world champions, Apple's Siri routinely answering wide-ranging, spoken queries, and automated translation services such as Google's becoming usable [17][7]. In 1979, Ballard et al. [8][9][10] introduced their Natural Language Computer (NLC) that enables users to program simple arithmetic calculations using natural language. Although NLC resolves references as well, there is no dialog system. Metafor introduced by Liu et al. [18] has a different orientation. Based on user stories the system tries to derive program structures to support software design. A different approach regarding software design via natural language is taken by RECAA [19]. RECAA can automatically derive UML models from the text and also keep model and specification consistent through an automatic feedback component. A limited domain end-to-end programming is introduced by Le. SmartSynth [20] allows synthesizing smartphone automation scripts from natural language description. However, there is no dialog interaction besides the results output and error messages.

Paternò [21] introduces the motivations behind end user programming defined by Liberman [3] and discusses its basic concepts, and reviews the current state of art. Various approaches are discussed and classified in terms of their main features and the technologies and platforms for which they have been developed. In 2006, Myers [1] provides an overview of the research in the area of End-User Programming. As he summarized, many different systems for End User Development have already been realized [22][23][24]. However, there is no system such as our prototype that can be controlled with natural language. During a study in 2006, Ko [22] identifies six learning barriers in End User Programming: design, selection, coordination, use, understanding and information barriers. In 2008, Dorner [25] describes and classifies End User Development approaches taken from the literature, which are suitable approaches for different groups of end users. Implementing the right mixture of these approaches leads to embedded design environments, having a gentle slope of complexity. Such environments enable differently skilled end users to perform system adaptations on their own. Sestoft [26] increases expressiveness and emphasizing execution speed of the functions thus defined by supporting recursive and higher-order functions, and fast execution by a careful choice of data representation and compiler technology. Cunha [27] realizes techniques for model-driven spreadsheet engineering that employs bidirectional transformations to maintain spreadsheet models and synchronized instances. Begel [28] introduces voice recognition to the software development process. His approach uses program analysis to dictate code in natural language, thereby enabling the creation of a program editor that supports voice-based programming.

NLyze [29], an Add-In for Microsoft Excel that has been developed by Gulwani, Microsoft Research, at the same time as our system. It enables end users to manipulate spreadsheet data by using natural language. It uses a separate domain-specific language for logical interpretation of the user input. Instead of recognizing the tables automatically, it uses canonical tables which should be marked by the end user. Another Gulwani's tool QuickCode [30] deals with the production of the program code in spreadsheets through input-output examples provided by the end user [24]. It automates string processing in spreadsheets using input-output examples and splits the manipulations in spreadsheet by entering examples. The focus of his work is on the synthesizing of programs that consist of text operations. Furthermore, many dialog systems have already been developed. Commercially successful systems, such as Apple's Siri, actually based on active ontology [31], and Google's Voice Search [32][33] cover many domains. Reference resolution makes the systems act natural. However, there is no dialog interaction. The Mercury system [34] designed by the MIT research group is a telephone hotline for automated booking of airline tickets. Mercury guides the user through a mixed initiative dialog towards the selection of a suitable flight based on date, time and preferred airline. Furthermore, Allen [35] describes a system called PLOW developed at Stanford University. As a collaborative task agent PLOW can learn to perform certain tasks, such as extracting specific information from the internet, by demonstration, explanation, and dialog.

V. Conclusion and Future Work

In this paper, we presented the new idea to use of natural language as the user interface. Nowadays, users can easily describe a tasks including conditionals, loops and statements (See Section I). To enable the system for end user development, these parts should be recognized correctly by the system. In the current version of our prototype, the system supports control flows, such as conditionals and loops, but the challenge is to understand the user input at run time and put the different statements in the right order. There is a lot of work on our system still needs to be done. The goal is to implement valid scripts from natural language input that describes some sorting algorithm. We are also exploring ways to extend the system functionality with the help of the dialog. The system needs to be extended for handling graphs, and charts. Furthermore, there are some properties of tables, which are not considered in the current system and can potentially lead to problems.

References

[1] B. A. Myers, A. J. Ko, and M. M. Burnett, "Invited Research: Overview End-User Programming," CHI, 2006.

[2] C. Scaffidi, M. Shaw, and B. Myers, "Estimating the numbers of end users and end user programmers," in Visual Languages and Human-Centric Computing, 2005.

[3] H. Liberman, F. Paternò, M. Klann, and V. Wulf, "End-user development: An emerging paradigm, human-computer interaction series, volume 9," 2006.

[4] M. Hurst, "The interpretation of tables in texts," University of Ediburgh, Ph.D., 2000.

[5] R. Abraham, "Header and Unit Inference for Spreadsheets Through Spatial Analyses," in IEEE Symposium on Visual Languages - Human Centric Computing, 2004.

Copyright (c) IARIA, 2017. ISBN: 978-1-61208-538-8

[6] W. F. Tichy, M. Landhäußer, and S. J. Körner, “Universal Programmability - How AI Can Help. Artificial Intelligence Synergies in Software Engineering,” May 2013.

[7] C. L. Ortiz, “The Road to Natural Conversational Speech Interfaces,” IEEE Internet Computing, March 2014.

[8] B. W. Ballard and A. W. Biermann, “Programming in natural language: NLC as a prototype,” Association for Computing Machinery (ACM), Volume 10, 1979.

[9] A. W. Biermann and B. W. Ballard, “Toward Natural Language Computation,” American Journal of Computational Linguistics, Volume 6, Number 2, 1980.

[10] A. W. Biermann, B. W. Ballard, and A. H. Sigmon, “An experimental study of natural language programming,” International Journal of Man-Machine Studies, 1983.

[11] A. Wachtel, S. Weigelt, and W. F. Tichy, “Initial implementation of natural language turn-based dialog system,” International Conference on Intelligent Human Computer Interaction (IHCI), December 2015.

[12] C. D. Frye, “Microsoft Excel 2013, Step by Step,” O’Reilly Media, 2013.

[13] A. Wachtel, J. Klamroth, and W. F. Tichy, “A Natural Language Dialog System Based on Active Ontologies,” The Ninth International Conference on Advances in Computer-Human Interactions, April 2016.

[14] A. Wachtel, M. T. Franzen, and W. F. Tichy, “Context Detection In Spreadsheets Based On Automatically Inferred Table Schema,” 18th International Conference on Human- Computer Interaction, rated with Best Paper Award, October 2016.

[15] J. E. Sammet, “The Use of English as a Programming Language,” Communication of the ACM, March 1966.

[16] D. Ferrucci, “Building Watson: An Overview of the DeepQA Project,” Association for the Advancement of Artificial Intelligence, 2010.

[17] H. Liu and H. Liebermann, “Toward a programmatic semantics of natural language,” Visual Languages and Human Centric Computing, 2004.

[18] H. Liu and H. Lieberman, “Metafor: Visualizing stories as code,” 10th international conference on Intelligent user interfaces, 2005.

[19] S. J. . Körner, M. Landhäußer, and W. F. Tichy, “Transferring Research Into the Real World - How to Improve RE with AI in the Automotive Industry,” 2014.

[20] V. Le, S. Gulwani, and Z. Su, “SmartSynth: Synthesizing Smartphone Automation Scripts from Natural Language,” Proceeding of the 11th annual international conference on Mobile systems, applications, and services (MobiSys), 2013.

[21] F. Paternò, “End user development: Survey of an emerging field for empowering people,” in ISRN Software Engineering, vol. 2013, 2013.

[22] A. J. Ko and B. A. Myers, “Designing the whyline: a debugging interface for asking questions about program behavior,” in Proceedings of the SIGCHI conference on Human factors in computing systems. ACM, 2004.

[23] S. Gulwani, W. R. Harris, and R. Singh, “Spreadsheet data manipulation using examples,” in ACM, 2012.

[24] A. Cypher, “Watch what I do: programming by demonstration,” in MIT Press, 1993.

[25] C. Dorner, M. Spahn, and V. Wulf, “End user development: Approaches towards a flexible software design,” in European Conference on Information Systems, 2008.

[26] P. Sestoft and J. Z. Sorensen, “Sheet-defined functions: Implementation and initial evaluation,” 2013.

[27] J. Cunha, J. P. Fernandes, and J. Mendes, “Bidirectional Transformation of Model-Driven Spreadsheets,” Springer Lecture Notes in Computer Science, 2012.

[28] A. Begel, “Spoken Language Support for Software Development,” Ph.D. Thesis, Berkeley, 2005.

[29] S. Gulwani and M. Marron, “NLyze: Interactive programming by natural language for spreadsheet data analysis and manipulation,” SIGMOD, 2014.

[30] S. Gulwani, “Automating string processing in spreadsheets using input-output examples,” in ACM SIGPLAN, 2011.

[31] D. Guzzoni, “Active: A unified platform for building intelligent web interaction assistants,” in IEEE, Web Intelligence and Intelligent Agent Technology Workshops, 2006.

[32] J. R. Bellegarda, “Spoken Language Understanding for Natural Interaction: The Siri Experience,” Springer New York, 2014.

[33] J. D. Williams, “Spoken dialogue systems: challenges and opportunities for research,” 2009.

[34] S. Seneff, “Response planning and generation in the MERCURY flight reservation system,” 2002.

[35] J. Allen, N. Chambers, and G. Ferguson, “PLOW: A Collaborative Task Learning Agent,” Association for the Advancement of Artificial Intelligence, 2007.

Copyright (c) IARIA, 2017. ISBN: 978-1-61208-538-8