CAN END-USERS PROGRAM?

Ido Perez
Ben-Gurion University of the Negev, Beer-Sheva, NA, Israel, idoprz@hotmail.com

Arnon Sturm
Ben-Gurion University of the Negev, Beer Sheva, NA, Israel, sturm@bgu.ac.il

Ido Perez and Arnon Sturm, 2014, "CAN END-USERS PROGRAM?", Proceedings of the European Conference on Information Systems (ECIS) 2014, Tel Aviv, Israel, June 9-11, 2014, ISBN 978-0-9915567-0-0
http://aisel.aisnet.org/ecis2014/proceedings/track12/6

DEV4ME: CAN END-USERS PROGRAM?

Prototype

Perez, Ido, Ben-Gurion University of the Negev, Israel, idoprz@hotmail.com

Sturm, Arnon, Ben-Gurion University of the Negev, Israel, sturm@bgu.ac.il

Abstract

In recent years, personal computing has changed direction and is now more inclined towards the world of mobile computing. This means that end-users expect a simpler and more personalized experience. To achieve the highest level of customization, end-users must develop their own applications. However, end-users usually lack in having the right skills for that task. To address this problem, many end-users programming languages and frameworks have been devised. These are particularly aimed at reducing syntax and cognitive gaps. However, some of the existing solutions reduced the expressiveness of the language and thus reduced the generality of the program, while others remains too complex for end-users. In this work we devise a new framework, named Dev4Me, designed from the ground up to provide end-users a solution for developing mobile and personal apps. The framework is made up of a new form-based language, an Integrated Development Environment (IDE) and an execution environment. The new framework enables the users to develop, test, debug and use their own apps in a few simple steps, using familiar form filling experience.

Keywords: End-User Programming, End-User IDE, Mobile Applications.

1. Introduction

In today's world, personal computing has taken a new form. Smartphones and tablets have replaced the PC, and provide a truly personal computing. Personalization is enhanced, as these devices allow users a high level of customization. Following this trend, users have developed a higher level of expectation from their devices. They expect the devices to allow them expressing their personal, business, and entertainment needs. Indeed, mobile computing does so to a limited extent using mobile apps, which are custom tailored to a very specific task. For each task, the users can look for specific apps which would meet their needs. However, this trend does not always provide the high level of personalization that users have come to expect. To facilitate that expectation, users need to develop their own apps. Nevertheless, as developing and programming apps require certain skills – which only a small portion of the potential users acquire – the adoption of end-user programming for the purpose of personalization is limited. End-users encounter difficulties at three levels: syntax, semantics, and pragmatics (Guibert et al., 2004).

- Syntax refers to the language structure. Dealing with syntax problems is a relatively easy task, as compilers can help in detecting and fixing such problems. Yet, it is desired that the occurrences of such problems would be prevented.
- Semantic errors are misconceptions of the machine model. According to Pea (1986) and Guibert and Girard (2003) they can be categorized into three classes: (1) Temporal –misconception of the control structures; (2) Anthropomorphic – misinterpretation of the code in a way that does not follows what is actually written; (3) Cognitive gaps – a mismatch between the task domain and the computer domain.

- Pragmatics problems refer to incompleteness of the model. As there are many edge cases, the model representing the problem tend to be incorrect.

Lieberman et al. (2006) identified two types of end-users activities relates to programming: (1) Parameterization or customization: these activities enable the end-user to choose from alternative behaviours that are pre-defined; and (2) Program creation and modification: these activities relate to creating or modifying software artifacts. In this work, we focus on the latter.

Several end-user programming environments do exist. However, they only address the aforementioned problems to a limited extent. In this work, we aim to extend the support in addressing the aforementioned problems and thus allow app programming for a variety of end-users. The presented prototype is intended to explore the way end-users act upon the introduction of an abstraction over a programming language and tools.

The paper is organized as follows: We commence with a brief overview of the state-of-the-art in end-user programming paradigms, referencing some of the leading environments. Next, we introduce the proposed framework, followed by an analysis of its contribution. We then describe the technological infrastructure of the proposed solution. Next, we describe a qualitative evaluation of the system and finally, we conclude with future research directions.

2. End-User Programming – The State-of-the-Art

The domain of end-user computing has evolved during the last two decades to provide solutions for end-user programming. These solutions are of various types, as we elaborate in the following:

- *Scripting languages and macros* are languages that are mainly used to automate tasks. They are often popular among system administrators. These tools can perform tasks in specific domain like sorting emails, or perform a wide variety of tasks like any other programming language (Blackwell, 2006). These techniques are not easy to learn (Ko et al., 2004), and are often used only by professionals. The scripts are actual code, thus they may suffer from all the three of the programming difficulties. Nevertheless, exiting tools help in reducing syntactic errors, and help in maintaining and testing the code.
- *Programming by Example (PbE)* is a paradigm in which programmers create a set of examples, and the computer deduces the program from these examples (Lieberman, 2001). It can be based on either graphical or textual tools. Examples of such systems include Pygmalion (Myers, 1986), ToonTalk (Morgado and Kahn, 2007) and StageCast creator (Smith, et al., 2000). Machine learning frameworks also use this paradigm to provide the machine with a model (Menon, et al., 2013). The disadvantage of this paradigm is that the specification of the program is either too general or too specific and does not address all cases. Using PbE, users can create "rule-based" apps, but they still lack proper data model, connectivity to web services and data sources. Because of the dynamic nature of the language, temporal errors can be avoided; however, it is prone to anthropomorphic errors because user manipulations and examples are sometimes unclear. In addition, the user cannot see direct feedback of his manipulations, and thus cannot understand his actions immediately (Blackwell, 2006).
- *Block Programming* (Tempel, 2013) is a paradigm in which each block is a basic software component, which can be part of another software component. LogoBlocks (Begel, 1996) is one of the origins of this paradigm. It is derived from Logo, and was the inspiration for many block programming environments. Programmers assemble their program using these blocks (McCaffrey, 2006). A program which is built using a lot of blocks is not easy to read and understand, and also is hard to maintain. It also requires the programmer to be familiar with many blocks and with the system of using control flows to assemble the blocks. This solution only uses graphical tool to hide the syntax complexity of programming, but yet does not offer a

new methodology. Thus, syntax problems can be reduced or avoided, and the black-boxing of the business logic can reduce cognitive gaps. However, temporal and anthropomorphic errors are not eliminated, because it is just a graphical representation of the code. The paradigm also suffers from a pragmatic problem, because it does not provide the means to capture the entire program.

- *Mashups* uses modern web technologies to collect information from the web and assemble it in a new single location (Grammel and Storey, 2008). Stolee et al. (2011) observe that mashups tend to suffer from common software programming deficiencies like complexity, using outdated modules, the use of non-standard patterns duplicate modules, etc. Cappiello et al. (2011) identify four fundamental parts for a mashup composition tool: (1) Domain-specific focus; (2) Abstraction from technical details; (3) Continuous feedback; and (4) Composition support. DashMash is a development environment designed to respond to these parts. In common with blocks programming, it only hides the code using graphical tools, but does not reduce the gap between the end-user mental model and the program model (Zang, 2009). Thus, it does not solve any of the semantic or pragmatic problems.
- *Domain Specific Language (DSL)* Systems are written for a specific domain and thus are easy to use by domain experts (Fowler, 2010), (van Deursen, et al., 2000). However, they are expensive to develop and are limited by their functionality. Even though it is easier to develop graphical tools for these languages, some of these systems still require training regarding the syntax. Other systems expose control structures in graphical tools. These systems deal mainly with syntax issues, and do not address semantics problems. Ortiz-Chamorro et al. (2009) suggest "Hypertextual programming" to solve the DSL syntax problems. They define hypertextual programming as "*a form of programming that uses navigation as the primary tool to inspect and edit the application code, and is supported by a computer system that: i) represents the entire program source code as hypertext; and ii) allows all the possible finite language instances to be generated as navigation paths through it.*"
- *Form-Based Systems* are another name for spreadsheets. They are the most used form of end-users development (Blackwell, 2006). Spreadsheets enable end-users to instantly review the results in the corresponding cells. Nardi (1993) claims that the use of forms reduce memory load, help to avoid typing errors by providing selections and menus and providing a prototype solution. The end-user can select formulas from menus, thus it reduces syntactic errors. Also, the end-user does not need to be familiar with complex programming concepts like control flow (Wilde, 1993). Burnett et al. (2001) enhance the spreadsheet paradigm by generalizing it and adding external events and assertions. The program is mixed with the data in the cells, and thus lack an abstraction layer between the program and the data. It also provides low visibility on the relationship between formulas, rows, columns and formatting. Following this approach, syntax problems can be reduced or avoided, and there are no control structures (besides "if"). There is a lack of abstractions and low visibility of the program, so end-users may not see the entire picture and be moved into pragmatics errors. Fogli and Parasiliti Provenza (2011) are using this form-based metaphor to construct e-government services by end-user developers. They claim that this metaphor proved to be adequate for end users, due to the low cognitive burden and because of its similarity to paper based forms. Burnett et al. (2006) investigate testing aspects of end-users programming using spreadsheets.

Our analysis demonstrates that although end-user programing solutions do exist, they address the syntax, semantics, and pragmatics issues to a limited extent, and neglect the holistic view of these.

3. Dev4Me: Allowing Novice to Program

Addressing the various problems mentioned before, we developed a "Form-based" programming framework, Dev4Me. It uses "Forms" (not spreadsheets) as a way to create apps. The Integrated Development Environment (IDE) incorporates four steps (forms), which end-users will fill out in order to create apps. Figure 1 describes these steps. In the first step, the end-user creates the app and the data-model; in the second step the functions of the app are specified; in the third step more sophisticated time-dependent events are handled; and in the last step, the generated UI can be customized. By following these steps, we anticipate the reduction of the gaps end-users encounter when they develop applications. Each form, representing a step within the development procedure, collects a different kind of relevant information for the application. The forms structure helps end-users to better understand the machine model, and lowers the gaps between the machine model and their mental model. The steps are ordered to help the end-users establish their applications from the data model first. Yet, they can navigate between the steps and re-iterate. An additional tool (and an important one) in the Dev4Me framework is the side-by-side app emulator view. The emulator actually executes the developed application and shows its view immediately when the end-user makes changes to the app. For example, if an end-user creates a new item, instances of this item can be created and updated instantly in the side-by-side emulator. This enables the understanding the status of the application, and bridging the gaps between the mental model and the application model. The end-user can execute the actions and see the results instantly. This enables the "debugging" and "testing" of the application logic while developing it.

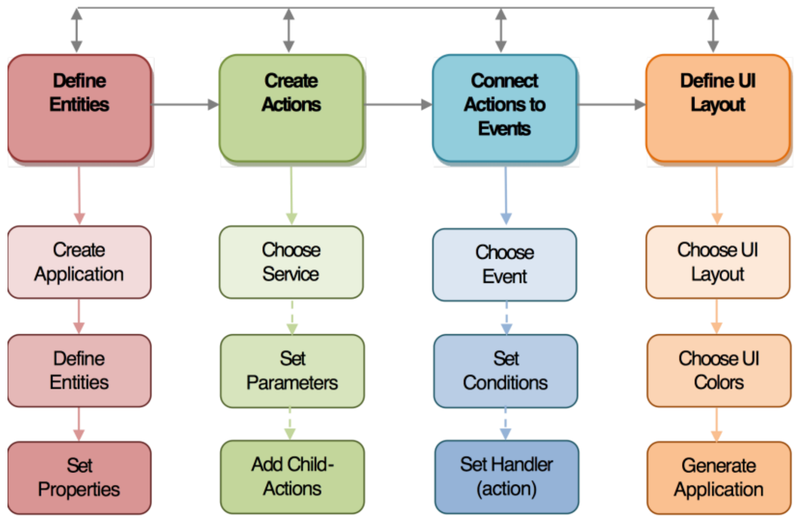
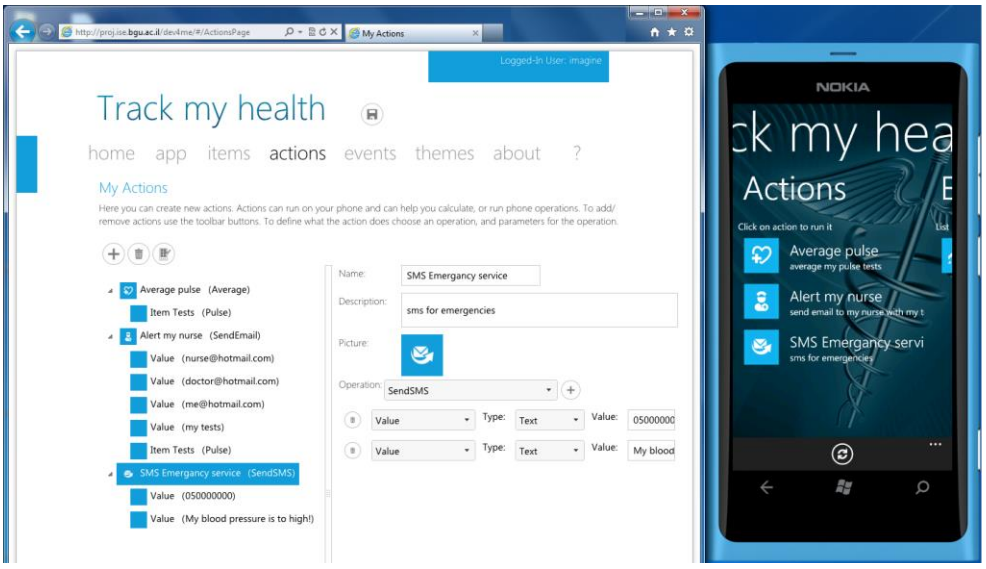


Figure 1. Steps of the new form based approach

In the following, we elaborate on the development steps elaborated above. However, before developing an application there is a need to fill out the relevant application information such as the name and an icon.

The first step is actually the development of the data model. For the sake of clarity for end-users, we term it "items". Each item has a name, a description, a picture and properties that hold more information and are typed. The type can be either an existing type (such as Text, Number, Date, Picture, Contact person) or other items. By allowing using other item as a type, we enable the specification of relationships between the items.

The second step is intended for developing the functionality of the app. The end-user creates new actions and chooses what the action does from a list of pre-defined operations. The operations may be mathematical operations, string operations, or phone operations (like "Send SMS"). Each operation has parameters. The parameters can be constants, values from "items", values that are received in run-time, or other actions. By using other actions, the end-user can create a call hierarchy and perform complex tasks. Figure 2 shows the IDE focused on the "actions" form, and the emulator on the run actions part of the application.



Track my health

My Actions

Here you can create new actions. Actions can run on your phone and can help you calculate, or run phone operations. To add/remove actions use the toolbar buttons. To define what the action does choose an operation, and parameters for the operation.

- Average pulse (Average)
  - Item Tests (Pulse)
- Alert my nurse (SendEmail)
  - Value (nurse@hotmail.com)
  - Value (doctor@hotmail.com)
  - Value (me@hotmail.com)
  - Value (my tests)
  - Item Tests (Pulse)
- SMS Emergancy service (SendSMS)
  - Value (050000000)
  - Value (My blood pressure is to high!)

Name: SMS Emergancy service

Description: sms for emergencies

Picture:

Operation: SendSMS

Value	Type: Text	Value:	05000000
Value	Type: Text	Value:	My blood

ck my hea

Actions

Average pulse
average my pulse tests

Alert my nurse
send email to my nurse with my t

SMS Emergancy servi
sms for emergencies

Figure 2. The side-by-side view focused on the actions tab

The third step consists of subscribing to the system events. The events can invoke actions whenever data changes or other rules are satisfied; for example, whenever the average value of an item is higher than a certain threshold. The events happen automatically on the application and invoke the desired action automatically.

In the last step, the end-user can customize the UI by choosing color and background for the application.

4. The Dev4Me Contribution

This proposed framework incorporates several well-known end-user techniques to create a unified solution to the end-users programming problem. It takes into account the mental model of end-users, software engineering considerations, and the emerging of the mobile application domain.

The use of forms is a well-known practice for end-users, and is very popular with many web pages and other business applications. Users are familiar with the concept of filling out forms, by using selections, trees and text fields. Dev4Me uses "forms" as the very foundation of the framework, and thus provides end-users with a familiar environment. Specifying the app behavior, end-users can choose formulas from a list, and choose the parameters (using the selection paradigm). There is no need for control structures such as loops, so it simplifies the semantic (temporal) aspects of the program. There is no use of control structures, rather only events and handlers are used. The side-by-side view provides a quick view to the application, and is thus very helpful in debugging and testing the application.

In developing the Dev4Me, we followed HCI best practices derived from Nielsen's (1994) usability heuristics and Pane and Myers (1996) as follows:

- Visibility of system status: The side-by-side view provides high visibility of the application at all times.
- Match between system and the real world, consistency and standards: Pictures, simple terms and names are used to create easy metaphors for end-users.
- User control and freedom: The end-user has control of the system at all times, and can navigate between the forms easily, as the development follows an iterative process.
- Recognition rather than recall: Recognition is achieved using selection and forms instead of code.

- Aesthetic and minimalist design: Dev4Me focus on minimalism, with an emphasis on clean design.
- Error handling: Selection avoids errors, and the side-by-side view provides a quick view to the system, so end-users can understand how the errors occurred.
- Help and documentation: The documentation and help is integral part of the Dev4Me experience.

In Table 1 we summarize the processes that help in addressing the programming difficulties stated above.

Programming Difficulty Addressed	**Dev4Me Feature**	**Explanation**
Syntax	Use of forms	Provide selections from lists and trees instead of typing.
Semantic: Temporal	Use of forms	No need for control structures.
Semantic: Anthropomorphic & Cognitive	Navigation system	Provide structure and order in creating and maintaining different aspects of the program.
	Side-by-Side emulator	Helpful in debugging. Enables the end-users to see what is actually happens in the program.
Semantic: Cognitive	Use of multiple forms	Lower the cognitive load by separating different aspects to different forms.
Pragmatics	Side-by-Side emulator	Allows for the verification of the model completeness.

Table 1. Mapping programming difficulty to Dev4Me feature.

5. The Dev4Me Infrastructure

The Dev4Me infrastructure consists of four major parts as described in Figure 3.

1. The **Domain Specific Language (DSL)** defines the language in which applications are described. The DSL is the meta-model of the applications.
2. The **Integrated Development Environment (IDE)** manipulates the meta-model, using the graphical forms. The IDE is implemented as a web application, along with the mobile phone emulator.
3. The **Application Server** stores the model of each application, and also the data of the application itself. It exposes the meta-data and the data using RESTful API. It also provides notifications to the application on event occurring. It enables synchronization between the IDE and the run-time of the application, and thus enables the side-by-side view. A professional developer can add more pre-defined operations using a simple programming API which meets a simple interface.
4. The **Phone Client** is the run-time environment of the system. Each phone OS needs a run-time to download and interpret the program. It downloads the meta-data of the application from the server, parses and interprets it, and then generates the UI using the phone OS native language. We decided to develop an example of the run-time for the Windows-Phone OS. The phone is

subscribed to listen to events of the app that are calculated on the server, in order to save battery life, and minimize the need of getting all the data from the server.

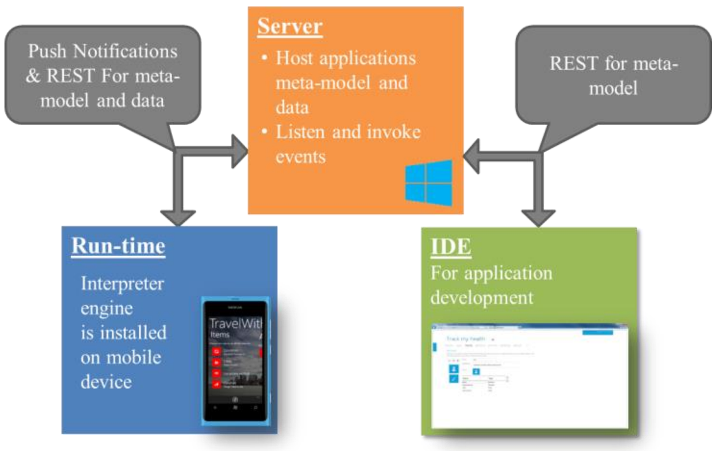

Push Notifications & REST For meta-model and data
Server
• Host applications meta-model and data
• Listen and invoke events
REST for meta-model
Run-time
Interpreter engine is installed on mobile device
TravelWit
IDE
For application development

Figure 3. Communication flow between system components

6. Evaluation

We evaluated the framework following a qualitative evaluation procedure. We chose to use the think aloud technique, in which we provided subjects with tasks, recorded them, and talked to them before, during and after the task. They should have described what they are doing, while we also recorded their actions. We tested the system using eight subjects. The subjects were Humanities majors with only basic knowledge in computer use and no programming experience whatsoever. Every subject received a 30 minutes training session on how to use the system and an example application. They were given two hours to perform two programming tasks: (1) create a home inventory system; (2) create a system to help a lecturer manage a course, students, grades, calculate averages, and send notifications to students. We analyzed the results by viewing the recordings and the transcripts of the observations made. We then categorized the observations according to the programming expertise (Guibert, et al., 2004). We also graded each subject result according to the expected results we set. With regards to their success, the subjects performed well both with respect to the data model (average of 79 percent) and the functionality (average 78 percent). Furthermore, although the second task was more complex, most of the subjects performed better than in the first task, a fact which indicates that the learning pace was quick. There were fewer observations in the second task than in the first one. We attribute this to the fact that there was an effective learning of the framework.

Our analysis of the observations indicates that the forms concept, the navigation and the side-by-side view was very conducive. In the interviews, all participants commended the basic concepts of the system. The main problem with the data model was to understand the difference between the meta-data of the data model and the actual data. This was resolved as they progressed in the task by using the side-by-side view. Another issue was the lack of transparency about hierarchical relations between items in the UI. Creating basic actions was easy enough. The main difficulty in the actions form was with the “if” statements. Users often confused it with the “events” form.

7. Conclusion

In this paper, we demonstrate an integrated approach for end-user application development. In particular, we refer to mobile applications. The proposed framework (and its supporting environment) addresses the issues of syntax, semantics, and pragmatics as follows. Because of the graphical nature of the forms, they eliminate the syntax problems. We enable end-users to choose from list and trees, instead of writing code. The forms structure helps end-users to understand the machine model better, and thus reduces the gaps between machine model and the user mental model. To avoid temporal mistakes, we implemented a form that can help the user to create rules that invoke actions over time and data changes. To reduce the anthropomorphic errors and the pragmatics problems, we show a running application side by side with the development environment. This way, the end-user has a clear view of the program and can test and debug during the app development.

In addition, we also evaluate the prototype using a qualitative approach in which we aimed at understanding what are the benefits and limitations of the proposed approach. We found out that overall, users are able to create apps using the suggested approach. The concepts of the forms, navigation, and the side-by-side view worked well, and the subjects felt comfortable with this solution, specifically indicating the ease of use of the system.

We plan to further evaluate the proposed approach and tools, compare it with alternatives, and to figure out its settings within an organizational context.

References

Begel, A. (1996). LogoBlocks: A Graphical Programming Language for Interacting with the World.

Blackwell, A. F. (2006). Psychological issues in end-user programming. In: H. Lieberman, F. Paternò and V. Wulf, eds. End user development. s.l.:Springer, pp. 9-30.

Burnett, M., Atwood, J, Djang, R.W, Gottfried, H., James Reichwein, Sherry Yang (2001). Forms/3: A first-order visual language to explore the boundaries of the spreadsheet paradigm. Journal of Functional Programming, 11(52), pp. 155-206.

Burnett, M., Rothermel, G., and Cook, C. (2006). An integrated software engineering approach for end-user programmers. In: H. Lieberman, F. Paternò and V. Wulf, eds. End User Development, Springer, pp. 87-113.

Cappiello, C., Florian, D. Matera, M., Picozzi, M., and Weiss, M. (2011). Enabling end user development through mashups: requirements, abstractions and innovation toolkits. LNCS 6654, Springer, pp. 9-24.

D. Pea, R. (1986). Language-independent conceptual "bugs" in novice programming. Journal educational computing research, pp. 25-36.

Fogli, D. and Parasiliti Provenza, L. (2011). End-User Development of e-Government Services through Meta-modeling. LNCS 6654, Springer, pp. 107-122.

Fowler, M., 2010. Domain-specific languages. Addison-Wesley.

Grammel, L. and Storey, M. A. (2008). An End User Perspective on Mashup Makers, University of Victoria Technical Report.

Guibert, N. and Girard, P. (2003). Teaching and Learning Programming with a Programming by Example System, International Symposium on End User Development, Bonn, Germany.

Guibert, N., Girard, P., and Guittet, L. (2004). Example-based programming: a pertinent visual approach for learning to program. Gallipoli, Italy, ACM, pp. 358-361.

Ko, A. J., Myers, B. A., and Aung, H. H. (2004). Six Learning Barriers in End-User Programming Systems. Washington, DC, USA, IEEE Computer Society, pp. 199-206.

Lieberman, H. (2001). Your wish is my command: Programming by example, Morgan Kaufmann.

Lieberman, H., Paternò, F., Klann, M., and Wulf, V. (2006). End-user development: An emerging paradigm. In: H. Lieberman, F. Paternò and V. Wulf, eds. End-user development, Springer, pp. 1-8.

McCaffrey, C. (2006). StarLogo TNG : the convergence of graphical programming and text processing. s.l.:Massachusetts Institute of Technology.

Menon, A., Tamuz, O. and Gulwani, S. (2013). A machine learning framework for programming by example. Atlanta, USA, Proceedings of the 30th International Conference on Machine Learning.

Morgado, L. and Kahn, K. (2007). Towards a specification of the ToonTalk language. Journal of Visual Language and Computing, 19 (5), pp. 574-597.

Myers, B. A. (1986). Visual programming, programming by example, and program visualization: a taxonomy. SIGCHI Bull, April, 17(4), pp. 59-66.

Nardi, B. A. (1993). A Small Matter of Programming: Perspectives on End User Computing. Cambridge, Massachusetts: The MIT Press.

Nielsen, J. (1994). Usability inspection methods. New York: John Wiley & Sons.

Ortiz-Chamorro, S., Rossi, G. and Schwabe, D. (2009). Hypertextual programming for domain-specific end-user development. LNCS 5435, Springer, pp. 225-241.

Pane, J. F. and Myers, B. A. (1996). Usability Issues in the Design of Novice Programming Systems. Carnegie Mellon University, School of Computer Science Technical Report CMU-CS-96-132, Pittsburgh, PA.

Smith, D. C., Cypher, A., and Tesler, L. (2000). Novice Programming Comes of Age. Communication of ACM, 43(3), pp. 75-81.

Stolee, K. T. and Elbaum, S. (2011). Refactoring Pipe-like Mashups for End-user Programmers. Honolulu.

Tempel, M. (2013). Blocks Programming.

van Deursen, A., Klint, P., and Visser, J. (2000). Domain-Specific Languages: An Annotated Bibliography. ACM SIGPLAN Notices, 35(6), pp. 26-36.

Wilde, N. P. (1993). A WYSIWYC (what you see is what you compute) spreadsheet. Boulder, CO, IEEE, pp. 72-76.

Zang, N. (2009). Mashups on the Web: End User Programming Opportunities and Challenges. Orlando, Florida, ACM.