Leveraging Teenagers Feedback in the Development of a Domain-Specific Language

The Case of Programming Low-Cost Robots

Ankica Barišić, João Cambeiro
NOVA LINCS, DI, FCT
Lisboa, Portugal
a.barisic@campus.fct.unl.pt
jmc12976@campus.fct.unl.pt

Vasco Amaral, Miguel Goulão
NOVA LINCS, DI, FCT
Lisboa, Portugal
vma@fct.unl.pt
mgoul@fct.unl.pt

Tarquínio Mota
Artica Creative Computing
Caparica, Portugal
tarquinio@gmail.com

ABSTRACT

Domain Specific Languages (DSLs) empower end-users to express software tasks that were traditionally developed by software engineers. DSLs allow users to express themselves in terms closer to the way they think about their problems, rather than in computational terms. However, conceiving a DSL with an adequate user experience for its end-users is not a trivial task, and the process of engineering that adequacy tends to be performed *ad-hoc*. The Gyro Creator Language (GCL) is an open-source DSL for controlling low-cost rover-like Arduino robots, designed for being used by teenagers with no previous computing skills, so they can be introduced to programming in a fun way. In this paper, we discuss an iterative process building on teenagers' early feedback, collected in a series of empirical evaluations with 128 teenagers, and how this has helped us driving GCL to a competitive level in terms of usability, when compared to well-established alternatives such as Lego, or Scratch.

CCS CONCEPTS

• **Software and its engineering → Software usability**; **Domain specific languages**; **Visual languages**;

KEYWORDS

Programming languages for children, Robotics Programming

ACM Reference format:
Ankica Barišić, João Cambeiro, Vasco Amaral, Miguel Goulão, and Tarquínio Mota. 2018. Leveraging Teenagers Feedback in the Development of a Domain-Specific Language. In *Proceedings of SAC 2018: Symposium on Applied Computing , Pau, France, April 9–13, 2018 (SAC 2018),* 9 pages.
https://doi.org/10.1145/3167132.3167264

1 INTRODUCTION

The increasing software pervasiveness fosters a growing concern for making some of its development accessible to end-users with no formal training in programming. Creating Domain-Specific Languages (DSLs) for empowering end-users is challenging, as we need to bridge the gap between computation concepts and the concepts mastered by the end-users. Two complementary ways of bridging this gap are (i) nurturing 'computational thinking' [37] skills in end-users, and (ii) devising adequate metaphors that hide the unnecessary complexity of computational concepts from those end-users. We were contacted by a company interested in developing an open source web-based DSL, called Gyro Creator Language (*GCL*), formally known as *Visualino*, targeted to teenagers, to empower them to control low-cost rover-like Arduino robots. The challenge was how to assess the DSL in a timely way, so that end-user feedback could lead to a competitive product. In particular, the company was concerned with the user-friendliness of the DSL.

However, the development process of DSLs lacks a systematic and iterative approach to evaluating and detecting usability issues since the early stages of the DSL construction. Therefore, we add to the common iterative life cycle of DSL development an evaluation task involving the end-users, to be performed in each iteration. The teenagers' feedback is used to help to steer the GCL's evolution through the identification of several improvement opportunities in the language. This evaluation stage in each iteration is often not reported in the context of developing DSLs but is key for our development effort. In this assessment, we contrast GCL with two popular DSLs that are used to control rover-like robots: a commercial competitor (Lego)[22] and an open source initiative (Scratch)[32].

We report on the design and results of the empirical studies used in this evaluation that helped us identifying the language's strengths and weaknesses. We discuss how this lead to the improvement of the *GCL* language. To help to achieve this higher level goal, we answer two more detailed research questions:

- ***RQ1:*** *How does the current GCL (GCL2) compare to baselines (a previous version of GCL (GCL1), Lego and Scratch) regarding the* ***Effectiveness*** *of the teenagers when programming a robot?*
- ***RQ2:*** *How does the current GCL (GCL2) compare to baselines (a previous version of GCL (GCL1), Lego and Scratch) regarding the* ***Satisfaction*** *of the teenagers when programming a robot?*

The chosen baselines are aimed at providing a comparison basis with (1) a previous version of GCL (GCL1), so that we can assess the extent to which the feedback collected with that previous version has helped in its evolution and (2) two popular competitor languages, so that we can assess how competitive the GCL can be, when contrasted with those languages.

SAC 2018, April 9–13, 2018, Pau, France

ACM ISBN 978-1-4503-5191-1/18/04…$15.00
https://doi.org/10.1145/3167132.3167264

2 RELATED WORK

2.1 Language Usability

Language usability is the degree to which a language can be used by specific users to meet their needs to achieve specific goals with effectiveness, efficiency and satisfaction in a specific context of use (adapted for the particular case of languages from [13]).

User-centered design (UCD) [28, 36] can contribute to more usable DSLs. For example, [1] presented an innovative visualisation environment, which eases and makes more effective the experimental evaluation process, implemented with the help of UCD. A visual query system was also designed and implemented following the UCD approach [6]. Although there is a lack of general guidelines and best practices to conduct language usability evaluations, they are slowly being recognised as an important step in the Language Engineering life-cycle [20]. An iterative approach allows us to trace usability requirements and the impact of usability recommendations througout the DSL development process [3].

2.2 User characterization and evaluation

Teenagers who are familiar with computers and technologies tend to be more successful in new computer-related tasks [12]. Also, they are likely to work and play in groups, (e.g. sharing a single computer) [8]. Involving teenagers as subjects in empirical studies is a valid option to evaluate usability on software products targeted to teenagers [34] but also challenging, as teenagers have a high variability in cognitive and development abilities at a given age [7]. Nevertheless, it has been shown that teenagers can identify and report usability problems using methods like 'think aloud', interviews or questionnaires [25].

Previous studies (e.g. [15, 18, 31]) concentrate on issues related to K12 courses curricula (i.e. covering from kindergarten to 12 years of basic education), the motivation of students to engineering, and education of computational teaching. Teenagers have been used in the past as subjects for language-related studies. For example, a hybrid approach combining text and visual notations offers the best compromise to increase efficiency and effectiveness of teenagers while using that language [19]. However, we are not aware of other formal studies concerned about improving software language usability, involving teenagers as subjects.

2.3 Programming languages for teenagers

Current robot platforms offer Application Programming Interfaces (APIs) for programming with languages such as *Java* (e.g. [9]), [30]) or *.NET Framework* languages (e.g.[14]). These languages may exclude many teenagers who are not (yet) proficient in programming. Without proper training, it is hard for them to program and master a textual programming language with a complex syntax full of technical concepts. This is true even with the help of powerful Integrated Development Environments (IDEs). We argue it is far from trivial to design a DSL for teenagers (and even more so for younger children).

There are several programming languages designed specifically for children (including teenagers). Examples include *Alice* [10], *Blockly* [11], *Lego* [22] and *Scratch* [32]. We used for comparison purposes with GCL *Lego*, one of the most widely used languages for programming toy robots, and *Scratch*, a popular visual programming language for teaching programming concepts to children which can also be used for controlling rover-like robots.

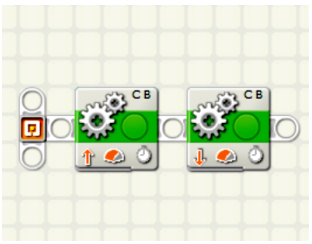

Figure 1: LegoMS

Figure 2: Scratch

2.3.1 Lego. Inspired by Papert's seminal work [29], *Lego Mindstorms (LegoMS)* is one of the most well-known technologies, in educational robotics for children. Several case studies have been done for this technology in order to understand which are the main individual needs of the children, when working with robots. *LegoMS*'s technology combines hardware and software, so that users can develop and deploy programs specifying behavior to be executed by the *Lego* robot.

Lego Mindstorms NXT 2.0, used in our usability trials, presents building blocks (like bricks) as elements to build a program. Each block represents a programming concept, such as an execution control element, *e.g.* loops, conditions, arithmetic, or an actuate block that interacts with the robot components, *e.g.* motors. The sequence of blocks is constructed by a behavior flowchart, structuring the program's blocks.

LegoMS has an appealing notation and is used only with relatively expensive *Lego* robots. However, the development of complex behaviors may become difficult [26]. The increasing number of blocks and the size of the diagram make it difficult to analyze and read the program solution. Non-trivial behaviors are difficult to implement in a visual programming language like this. Fig. 1 exemplifies the move back and forth example. The diagram contains two composition elements describing how the robot should move.

2.3.2 Scratch. Scratch [32] relies on actions (Blocks) to operate specific objects (Sprites). They can be seen as a visual abstraction to the Object Oriented programing paradigm with some restrictions (no support for custom objects and the dynamic generation of sprites). Open source tools like *mBlock* [33] and *Enchanting* [21], are built upon *Scratch*, and meant to be simple and used for robot programming. *mBlock* is a solution which compiles to the *Arduino* open source hardware platform, which makes it suitable for our assessment study, as it uses the same hardware as GCL.

Fig. 2 illustrates how to program the robot to move forward during one second and to go backwards during one second. However, as this language is general purpose, it does not have abstractions of operations like move forward or backwards. Therefore, the user needs to detail the movement operator. This includes to describe the pins of each motor on the Arduino board and deal with concepts like motor rotation directions, rotation speed, angular velocity.

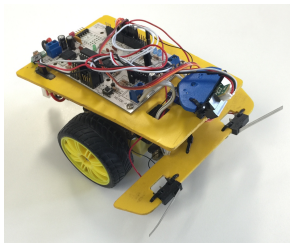

Figure 3: Arduino Robot

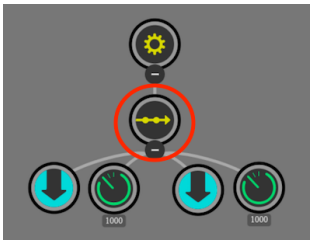

Figure 4: GCL1 - Back and Foward

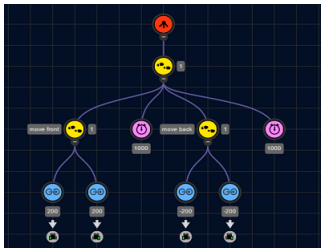

Figure 5: GCL2 - Back and Foward with bumpers

3 CONTEXT

GCL is a visual language that allows the user to implement programs for robot behavior, through the manipulation of visual elements, or objects. This manipulation is expected to help the user to understand quickly the programming mechanisms. Visual languages are thought useful for introducing programming concepts to children [16], while robots perform the developed code in the real world. The observation of the program running on a physical robot provides an engaging feedback on the implications of changes in the program.

The robot is an *Arduino* board with a set of sensors and actuators. *Arduino* is an open-source prototyping platform [2]. The low-cost robot (see Fig. 3) which works with GCL has the following configuration: (i) An infra-red distance sensor; (ii) Two bumpers (e.g. collision detectors sensors); (iii) One motor for each wheel; (iv) A Servo motor that actuates under the distance sensor; (v) A simple LED light.

Programming the *Arduino* textual code requires technical skills not owned by most teenagers. To mitigate this problem, the robot may be programmed using *GCL*, which is then automatically translated into the *Arduino* textual code.

GCL's visual syntax is based on the behavior tree paradigm [24], a mathematical model of plan execution, used for a diversity of areas including robotics, control systems and software games. A complex behavior is mapped into smaller and simpler behaviors through its branches. This descending order of complexity provides a structured way of defining complex behaviors (which are used to define objectives) through simple tasks defined hierarchically. Each node may have a specification that determines how the actions of its children will be executed (in parallel, or sequentially). The child node returns its status to the parent node, and this successively happens until the root of the tree.

Fig. 4 illustrates how to program the robot to move back and forth. The root node is decomposed into a single sequential node (highlighted with a red circle). The sequential node runs all its children (in this case, the leaf nodes) in depth first traversal order. Leaf nodes represent the most primitive actions that could be taken by an agent. In Fig. 4 the nodes represented by arrows pointing downwards are outputs actions. In this case they are left and right motors commands. The outputs can also be visual using LEDs or audio using a buzzer. Both nodes displayed by 2 round timer shapes are wait commands in which the execution of the next node is delayed by the duration determined by the end user for each node.

3.0.1 From GCL0 to GCL1. The first development iteration of *GCL* GCL0 started with a domain analysis, followed by the design, implementation, and evaluation, in order to quickly deploy an early prototype. The last phase involved 22 (10+12) children, with the age range from 8 to 12, as subjects included in an exploratory study [23]. Children under 10 had a hard time learning *GCL*. Further development resulted in the next release GCL1 (Fig. 4), which improved the interaction model and provided a web based solution.

3.0.2 From GCL1 to GCL2. GCL1 was evaluated experimentally and compared to the one of the most popular commercial competitors in the market, *LegoMS*. Based on the results of this empirical study, some improvements were suggested and applied to GCL. The focus was on improving the user interface providing better readability of the programs being developed and improving error prevention. The suggestions were:

- Improve error feedback and suggestions to solve problems.
- Add auto-alignment of new nodes added to a sequence node, preventing situations where the visual order of nodes from left to right did not match the order of their execution.
- Highlight when a user selects a new node to add to the diagram the nodes that are available to form a connection. This provides better user assistance and error prevention.
- Allow the users to create blocks of tree structures. This feature promotes reuse of previously developed structures and allows to share and slowly introduce more complex behaviours to novice users.
- Show the Icon's label with mouse-over events. This helps the user to recognize the available options.
- Introduce a different method of collapsing and expanding tree structures. The use of the keys "+" and "-" for these operations was not clear to the user.
- Introduce new zooming options to allow an easier navigation of the tree structure by introducing a fit to screen operation of the entire tree structure or only the selected nodes.

These suggestions lead to the release of a new improved version of the language *GCL2* (Fig. 5). Finally, GCL2 was compared with Scratch.

4 EXPERIMENT PREPARATION

In this section, we describe the experiment protocol. Additional materials can be found in this paper's companion site [4]. We had two runs of the same experiment, with the second prototype release (GCL1) which we compared to Lego; and second GCL version

(GCL2) which was compared to Scratch. The experiment was announced as a robot programming challenge to engage teenagers to participate.

During the first run, GCL1 was compared to the best product in the area but appropriate only for costful Lego robots. Therefore, the robots used in this run were different. However, during the second experiment run, both GCL2 and Scratch programs were running on the same robot (see Figure 3). The two different experiment runs were designed as two parts of the same experiment. Both participant's profile and the tasks were similar, so that, in practice, we can think of them as a single experiment with four different languages (two of which are different versions of the same language).

4.1 Experiment objectives

The high-level objective of our study is to evaluate the usability of *GCL*, *"Usability of Programming the robot"*, as it is expected to be used by a broad group of people that are not expected to have previous programming skills.

As we already have different generations of a functional prototype of GCL, we measure *GCL*'s *effectiveness* and *satisfaction*. The metrics for these usability requirements are defined in Table 1. We are not particularly concerned with other requirements such as efficiency or learnability at this point because we are at a stage of the language development where we want to know if the teenagers can program the robot, which is a prerequisite for measuring other characteristics. The experiment objectives follow the GQM template [5] and are defined as follows:

G1 - **Analyse** the effect of *GCL2*, **for the purpose** of evaluation, **with respect to** its impact on the *effectiveness* in programming a robot **when compared to** three *baselines*, namely *GCL1*, *Lego* and *Scratch*, **from the point of view** of researchers, **in the context of** an experiment conducted with secondary school subjects.

G2 - **Analyse** the effect of *GCL2*, **for the purpose** of evaluation, **with respect to** its impact on the *satisfaction* in programming a robot **when compared to** three *baselines*, namely *GCL1*, *Lego* and *Scratch*, **from the point of view** of researchers, **in the context of** an experiment conducted with secondary school subjects.

Table 1: Usability requirements

Requirement	*Metric*
Effectiveness: Is the <User> able to correctly implement a given <Use Case>?	PCorrUCInst - percentage of correctly implemented concepts (i.e. garanteeing that an expected outcome is reached) in a given <Use Case>
Satisfaction How much is the user satisfied with *GCL*?	ConfLevel - self rated confidence score in a Likert scale LikeLevel - self rated likeability score in a Likert scale LearnLevel - self rated learnability score in a Likert scale

In particular we test the following (null) hypotheses:

- $H1_0$: Using *GCL2* has no influence in the *effectiveness* of programming the robot when compared to programming the robot with the *baselines GCL1*, *Lego* and *Scratch*.
- $H2_0$: Using *GCL2* has no influence in the *satisfaction* of programming the robot when compared to programming the robot with the*baselines GCL1*, *Lego* and *Scratch*.

4.2 Experiment design

We used a *between groups* design where participants were randomly assigned to the task of either programming a robot using *GCL1* or *Lego*, in the first run, or *GCL2* or *Scratch*, in the second run. Each participant only participated in one of the four alternatives to avoid learning effects. The experimental process was similar for all runs and subject groups, starting with a learning session that lasted 30 minutes, during which the participants filled in a background questionnaire and learned about programming robots using their assigned language. Then, the participants had 15 minutes to solve a programming challenge. The contents of the computer screen during the session were recorded. Finally, there was a feedback session, taking up to 5 minutes, to collect the teenager's subjective opinions about using the language they were assigned to.

4.2.1 Participants, teams and groups. The participants were high school students recruited through convenience sampling, among the visitors of two different open doors days in the same university. In each day, participants were randomly assigned to one of the two groups: in the first day, there was a group with GCL1 and another one with Lego; in the second day, there was a group with GCL2 and another one with Scratch. Students and their teachers were aware of this study and volunteered to participate. Two lecturers guided each language's groups. The teenagers were requested to participate, in teams of up to three elements, using their assigned language and corresponding robots. Each group had a maximum of 5 teams participating in the same session. Undergraduates helped in the experiment, while a researcher monitored the data collection process.

4.2.2 Technical, social and physical environment. Each team worked on a desktop computer with OS Windows 7 Professional (Athlon 64x2 Dual Core 5000 2.6 GHZ processor, 4 GB RAM and a 17-inch monitor with a screen resolution configured at 1280x720) to implement the applications. The interaction between user and computer was achieved by the use of keyboard, mouse, and screen, and was captured by *Debut* video recorder [27].

The atmosphere was set up to be challenging and educative, but also playful and entertaining, to keep the teenagers still interested to participate and to reduce the sensation of failure. Team members were allowed to talk to each other. They could seek help if they had technical issues with the robots.

4.2.3 Exercises and Challenge. During the learning sessions, participants solved three basic robot programming exercises. They tested each of those exercises on the corresponding robot. Furthermore, they were encouraged to ask questions and to ask for help if necessary.

The first exercise was to program the robot to *move forward* and then *move back* (see Fig. 6). It gave the participants a notion of how to program the robot to move in a basic way. The numbering in Figures 6 through 10 represents the order in which the sequential commands would be executed. The second exercise was to program the robot to move along a path similar to a '5' (see Fig. 8).

In each turn, the robot needed to execute the move operation using the same amount of time and then to make a 90^o angle turn. The third exercise was to program a robot to *move forward* until it *bumped* into some object, then it would *move back* and *stop* (see

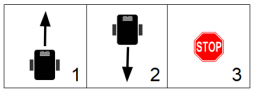

Figure 6: Exercise 1

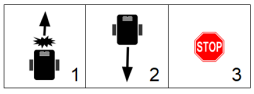

Figure 7: Exercise 3

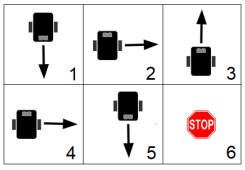

Figure 8: Exercise 2

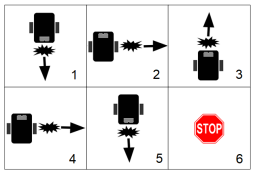

Figure 9: Challenge 1

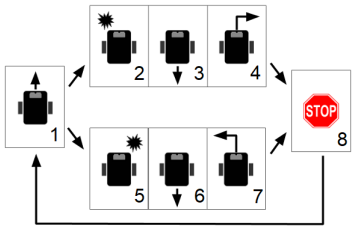

Figure 10: Challenge 2

Fig. 7). During this exercise, participants learned how to use the detector sensors and to compose their behaviour in a sequence with other modelling elements.

The challenge in the first experiment was to program the robot to make a shape of '5' but the robot would only turn when it would hit the bumper (see Fig. 9). It was composed by basic user story actions of the exercises. The challenge in the second experiment was slightly different (see Fig. 10). The robot was expected to detect if it hits the obstacles with the left or the right bumper, and it was supposed to turn accordingly to the opposite direction. It either executed the instructions associated with hitting the right bumper, or those associated with hitting the left bumper.

When the team thought they had a solution, they were invited to physically test it in the arena with a given robot. If they were not happy with the result of the test, they could go back and try to fix whatever was wrong with their solution.

4.3 Experimental instruments and measurements

During the experiment, we used survey forms, video recordings and a competition arena to collect data for further analysis. The survey forms were composed of *"Smileyometers"* which are found to be appropriate for teenagers questionnaires [35]. While answering to the forms, the teenagers were assisted by an adult (one of the experiment assistants), to ensure that there were no misinterpretations of questions and answers and to confirm that the participants did not experience reading problems. As we grouped the participants into teams, the participants' individual answers to questionnaires were merged. We computed the mean response within each group, for each answer.

Table 2: Instruments and Scales

	Instrument	*Value*
Profile	Background Questionnaire	0 / 0.5 / 1
Effectiveness	Video recording	0 / 1
Satisfaction	Satisfaction Questionnaire	-1 / 0 / 1

Profile is a measure influenced by *Experience* factors on Computer Games, Programming or Programming a robot, and *Tendency* factors reflecting the teenager's tendency to Mathematics, Physics or to Learn programming. The data for calculating the *Profile* was collected through the background questionnaire, that consisted of questions designed to assess those pre-defined factors. Each answer was encoded in three possible answers: 'Yes' (with a score of 1), 'Intermediate' (0.5), and 'No' (0).

We used the recorded videos to evaluate the **Effectiveness**. The challenge could be solved by composing elements of the training exercises, which are marked as Success (S) or Failure (F). Effectiveness measures the percentage of modelling elements correctly built and composed to achieve the solution.

Satisfaction is characterized by: a *Confidence* factor, reflecting ConfLevel metric that tells how confident teenagers were about their solution; *Likeability* factor, reflecting LikeLevel metric that tells how interesting and enjoyable they found the challenge itself; and, *Learnability* factor, reflecting LearnLevel metric that tells how useful they found what was taught during the learning session which helped them to face the final challenge. The data was collected trough a satisfaction questionnaire, that consisted of questions designed to assess the defined factors. A 'Yes' scored 1 point, 'Intermediate' scored 0, and 'No' scored -1.

5 RESULTS

5.1 Descriptive statistics

Table 3 presents the descriptive statistics for the metrics collected in our data analysis. In the *Characteristic* column we present the property under scrutiny. For each of these characteristics, we show four rows, one for each *Language* (in the second column). We further detail the *mean*, *standard deviation*, *skewness*, *kurtosis*, and the *p-value* for the Shapiro-Wilk normality test. The number of participant teams is not always the same for all languages. The shape of the distributions concerning most of the variables suggests that, in general, normality is not a reasonable assumption (p-value < 0.05). The exceptions are metrics concerning *profile*, *experience* and *tendency*, where, for most languages, normality is a reasonable assumption (*p-value* >= 0.05). The visual inspection of boxplot diagrams, Q-Q plots and kernel density plots (omitted here for the sake of brevity) further reinforced our assessment concerning data normality. As several of the variables have a non-normal distribution, we assume this for the remainder of our analysis. Note also that all teams expressed the maximum *confidence* level on their solution for GCL2, as well as the maximum *likeability* level for GCL2 and Scratch.

The variance of the distributions is not similar, when comparing the characteristics metrics distributions for the different languages, as shown in table 4.

Table 3: Descriptive statistics

Characteristic	Language	N	Mean	Std. Dev.	S-W
Age	Scratch	8	16.7062	1.23782	0.023
	GCL2	9	16.3689	0.91408	0.180
	GCL1	17	16.6059	1.31694	0.018
	Lego	14	16.4429	1.28465	0.039
Profile	Scratch	8	0.6875	0.17107	0.542
	GCL2	9	0.6019	0.13029	0.213
	GCL1	17	0.6225	0.16435	0.455
	Lego	14	0.6250	0.22349	0.201
Experience	Scratch	8	0.5625	0.23465	0.241
	GCL2	9	0.4259	0.14699	0.338
	GCL1	17	0.5294	0.24463	0.214
	Lego	14	0.4643	0.31473	0.042
Tendency	Scratch	8	0.8125	0.18767	0.197
	GCL2	9	0.7778	0.20412	0.122
	GCL1	17	0.7157	0.20211	0.059
	Lego	14	0.7857	0.2210	0.004
Effectiveness	Scratch	8	0.8000	0.23905	0.041
	GCL2	9	0.9556	0.08819	0.000
	GCL1	17	0.3765	0.40548	0.001
	Lego	14	0.7857	0.32783	0.000
Satisfaction	Scratch	8	0.9583	0.05893	0.002
	GCL2	9	0.9630	0.07349	0.000
	GCL1	17	0.6275	0.23221	0.085
	Lego	14	0.8571	0.22746	0.000
Confidence	Scratch	8	0.9688	0.05786	0.000
	GCL2	9	1.0000	0.00000	-
	GCL1	17	0.5000	0.30619	0.234
	Lego	14	0.8571	0.25409	0.000
Learnability	Scratch	8	0.9063	0.12939	0.000
	GCL2	9	0.8889	0.22048	0.000
	GCL1	17	0.4412	0.42875	0.023
	Lego	14	0.7500	0.37978	0.000
Likeability	Scratch	8	1.0000	0.00000	-
	GCL2	9	1.0000	0.00000	-
	GCL1	17	0.9412	0.24254	0.000
	Lego	14	0.9643	0.13363	0.000

Table 4: Test of Homogeneity of Variances

Characteristic	Levene Statistic	df1	df2	Sig.
Age	0.353	3	44	0.787
Profile	1.758	3	44	0.169
Experience	3.588	3	44	**0.021**
Tendency	0.537	3	44	0.660
Effectiveness	6.975	3	44	**0.001**
Satisfaction	2.892	3	44	**0.046**
Confidence	8.079	3	44	**0.000**
Learnability	2.828	3	44	**0.049**
Likeability	1.711	3	44	0.178

5.2 Dataset preparation

The video recordings were analysed following a protocol previously established by the research team. Apart from (i) assessing the success, we checked if the team (ii) reused the concepts from previous exercises, (iii) experienced technical problems or functional errors, (iv) had interaction difficulties (e.g. using copy/paste for visual objects or connecting the same objects in sequence), (v) reused previously constructed sequences (within the same exercise), or (vi) used any other additional language features (e.g. zooming). The remaining data was extracted from the questionnaires [4].

5.3 Hypotheses testing

For testing our hypotheses, we used the Welch t test, as it is robust to deviations from the normal distribution, different sample sizes and different variance in the samples, thus following the recent recommendations on data analysis for Software Engineering empirical evaluations (which summarises best practices in statistical analysis on other domains) [17]. Table 5 summarizes the results of these tests. Note that, because *Confidence* and Likeability had a constant value (the top possible score for each of them, the corresponding lines are not filled in table 5.

Table 5: Welch t test scores

	Statistic	df1	df2	Sig.
Age	.170	3	21.109	.915
Profile	.423	3	20.891	.739
Experience	.921	3	21.381	.448
Tendency	.527	3	20.597	.669
Satisfaction	10.723	3	24.108	**.000**
Confidence	-	-	-	-
Learnability	5.678	3	23.334	**.005**
Likeability	-	-	-	-
Effectiveness	10.886	3	20.483	**.000**

We used a Games-Howell post-hoc test to determine which languages were significantly different from which languages, according to our set of characteristics under scrutiny. Table 6 summarises this test's result, for the comparisons involving either GCL1 or GCL2, or both. We observe that GCL1 lead to a significantly lower *Satisfaction* and *Effectiveness* when compared to Scratch, GCL2 and Lego. GCL1 was also significantly harder to learn than Scratch and GCL2, but not significantly different when compared to Lego. In contrast, GCL2 was consistently as good as Lego and Scratch, while superior to GCL1 in terms of Satisfaction, Learnability and Effectiveness.

Table 6: Games-Howell test

Characteristic	(I)Tool	(J)Tool	MD(I-J)	Std. Err.	Sig.
Satisfaction	GCL2	Lego	.10582	.06554	.397
	GCL2	Scratch	.00463	.03216	.999
	GCL2	GCL1	.33551	.06142	**.000**
	GCL1	Lego	-.22969	.08287	**.046**
	GCL1	Scratch	-.33088	.06005	**.000**
Learnability	GCL2	Lego	.13889	.12531	.689
	GCL2	Scratch	-.01736	.08657	.997
	GCL2	GCL1	.44771	.12734	**.009**
	GCL1	Lego	-.30882	.14531	.169
	GCL1	Scratch	-.46507	.11360	**.003**
Effectiveness	GCL2	Lego	.16984	.09242	.293
	GCL2	Scratch	.15556	.08948	.362
	GCL2	GCL1	.57908	.10264	**.000**
	GCL1	Scratch	-.42353	.12967	**.018**
	GCL1	Lego	-.40924	.13171	**.021**

*RQ1: How does the current version of GCL (GCL2) compare to the used baselines (GCL1, Lego and Scratch) regarding the **Effectiveness** of the children when programming a robot?*

As seen in table 5, there was a statistically significant difference among languages, with respect to the overall *Effectiveness*, with GCL2 (M=.9556; SD=.08819), Scratch (M=.8; SD=.23905) and Lego

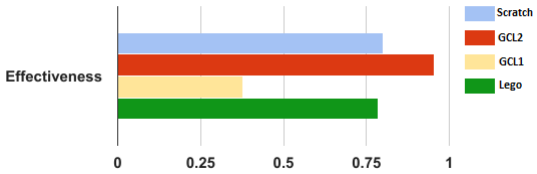

Figure 11: Effectiveness results

(M=.7857; SD=.32783) clearly allowing the teenagers using it to outperform those using GCL1 (M=.3765; SD=.40548), as detailed in table 6. This suggests that **GCL2 has achieved a similar level of *Efficiency* compared to the two commercial baseline languages, and has significantly improved when compared to its previous iteration, GCL1, in this aspect**. Figure 11 illustrates this. As such, we can reject our null hypothesis $H_0 1$.

*RQ2: How does the current version of GCL (GCL2) compare to the used baselines (GCL1, Lego and Scratch) regarding the **Satisfaction** of the teenagers when programming a robot?*

As seen in table 5, there was a statistically significant difference among languages, on the overall satisfaction, with Scratch (M=.9583; SD=.05893), GCL2 (M=.963; SD=.07349) and Lego (M=.85710; SD=.22746) providing a higher satisfaction than GCL1 (M=.62750; SD=.23221). This suggests that **GCL2 has achieved a similar level of *Satisfaction* compared to the two commercial baseline languages, and has significantly improved when compared to its previous iteration, GCL1, in this aspect**. Figure 12 illustrates this. As such, we can reject our null hypothesis $H_0 2$.

We can further break down this observation with a closer look to the components of *Satisfaction*: *Confidence*, *Learnability* and *Likeability*. *Confidence* obtained a perfect score (M=1.0; SD=.0) for GCL2, closely followed by Scratch (M=.96880; SD=.05786) and Lego (M=.8571; SD=.25409), but contrasting to GCL1 (M=.5; SD=.30619). Concerning *Learnability*, Scratch (M=.90630; SD=.12939) and GCL2 (M=.8889; SD=.22048) had a statistically significantly higher score than GCL1 (M=.4412; SD=.42875), but not significantly higher than Lego (M=.75; SD=.37978). Finally, the perfect scores of Scratch and GCL2 conderning *Likeability* (M=1.0; SD=.0) were not statistically significantly different from those of Lego (M=.9643; SD=.13363) and GCL1 (M=.9412; SD=.24254).

Participants Background:

We also need to stress that the observed differences are **not** attributable to different participant backgrounds. Indeed, the results from the background questionnaires indicate a comparable profile of participants for all languages. This is visible from table 5, where none of the properties *Age*, *Profile*, *Experience* and *Tendency* are statistically significantly different for any of the languages, i.e., they are essentially comparable.

Most participants had some experience in playing computer games, but very few of them had previously programmed a robot (see Figure 13). Some of the participants also had some knowledge of programming. These three factors gave an average Experience score for teams. The motivation to participate in the challenge was high. Most subjects expressed their tendency to learn to program,

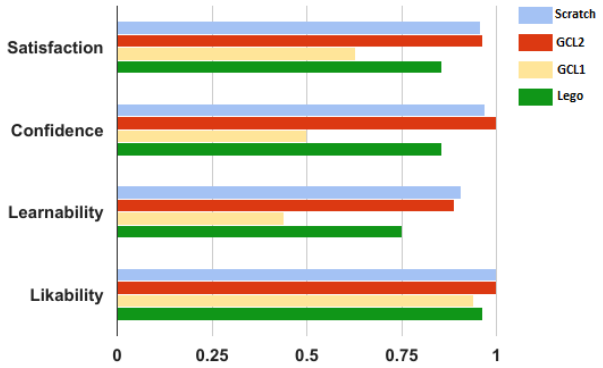

Figure 12: Satisfaction feedback

mathematics and physics, leading to a high Tendency score for all groups. The Profile score, calculated as an average of Experience and Tendency, indicates balanced teams regarding their background.

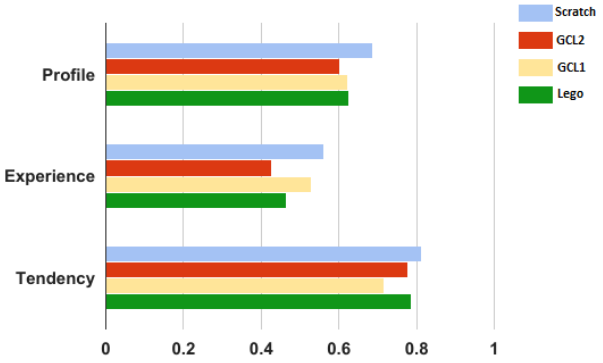

Figure 13: Profile analysis

6 DISCUSSION

We presented a systematic usability evaluation of the *GCL* language. In this section, we discuss the contributions to the software language engineering process, evaluation results and finally the implications for the further development.

6.1 Contributions to the development cycle

By designing the systematic experimental study, it was necessary to describe the context of use of *GCL* during the experiment and its goals explicitly. We have identified "Who will use the language?" and characterised the intended group of end users by the Profile, where each of the relevant characteristics was measured with a particular set of questions and can be further reused and analysed for selection of experimental subjects and specification of *GCL*'s audience. We explicitly tackle the question "Where will the language be used?" by defining its technical, social and physical environment. Also, we systematically analysed the working environment of the *GCL*'s competitors' tools to further identified both limitations and

advantages of *GCL*. Further, we explicitly expressed "How will the language be used?" through user stories. (see [4]). Developers performed the functional testing of these user stories, and the teaching artefacts (i.e. presentation documents and videos) were produced, which can be now be reused as a part of language documentation. The readability and understandability of these presentation materials were validated during pilot experiment sessions and materials can be reused in further evaluations.

Additionally, we address the question "Why will the language be used?" by defining the usability requirements and their GQM definition. Our high-level goal to evaluate usability is context-dependent, and therefore we see that its success is linearly related to the validated requirements (e.g. effectiveness and satisfaction) and a given context (e.g. evaluating a Challenge scenario in a similar experiment with LegoMS, in the first run of the comparative evaluation, using GCL1, and Scratch in the second run of the experiment, this time competing with GCL2). The trial design was successful and can be reused and save time in further evaluation sessions. The experiment results helped in making the further design decisions and identifying the pros and cons of the previous implementation.

6.2 Evaluation results

RQ1: How does GCL2 compare to the baselines (*GCL1, Lego* and *Scratch*) in terms of the *Effectiveness* of the teenagers when programming a robot? Based on the results obtained in the first run we conclude that programming the robots with *GCL1* resulted in lower Effectiveness scores when compared to programming the robot with *Lego*. However, thanks to the evaluation with the *GCL1* prototype, we found that *GCL* was already usable to some extent by teenagers. The feedback and observations of that initial run gave us insights into particular features that should be implemented to improve *GCL*'s usability. This helped to steer the development of *GCL2*, that was then tested in the second run of the experiment, this time using *Scratch* as a competitor (although, in practice, both *GCL1* and *Lego* can be seen as competitors). The feedback collected in the previous iteration proved extremely valuable, leading to a significant improvement of the *effectiveness* from *GCL1* to *GCL2*, which is now on par to *Lego* and *Scratch*.

RQ2: How does *GCL2* compare to baselines (*GCL1, Lego* and *Scratch*) in terms of the *Satisfaction* of the teenagers when programming a robot? Results showed that both *GCL1* and *Lego* were rated as satisfactory, with *Lego* providing a more intense satisfaction level. This shows that, although not being as successful as with *Lego*, the teams solving the challenge using *GCL1* still had an entertaining and motivating experience. Concerning *Satisfaction*, *GCL2* is significantly more competitive than *GCL1* and on par to *Scratch* and *Lego*.

6.3 Implications for the development of *GCL*

As a consequence of the analysis of all the data collected and the observation of the interaction of the experiment's participants with the *GCL* language, the *GCL* development team identified that new features are needed, and they will be included in future releases.

The focus is on improving the following areas:

Error Prevention and program readability needs: 1) Improve errors' feedback (including suggestions on how to solve the problems found); 2) When a user selects a new node to add to the diagram, the nodes that are possible to connect to the new node should become highlighted; 3) The users should be able to create blocks of tree concepts (this will promote reuse and facilitate the sharing and introduction of complex behaviours to novice users); 4) To prevent misinterpretations of the execution of a sequence node, an auto-alignment of new nodes added to a sequence node feature has to be introduced; and, 5) Icon Labels should be presented on mouse-over events (this should improve the user recognition of the available options).

Diagram navigation: A *fit to screen* of the entire program or only the selected nodes should be available; an improved tree collapse and expand feature (the former approach, using keys "+" and "-" was not clear to the user) should be added.

7 THREATS TO VALIDITY

Usually, it is not safe to rely on teenagers self-rating questions. To maximise the reliability of the responses, we had adult helpers interviewing them. This helped to ensure the validity and integrity of results and gave strength to design recommendations or decisions.

We did not have participants using both languages. Although this design prevents learning effects, it does create the risk of, by accident, having more "competent" teams using one language, or the other. However, by obtaining similar background scores (Profile and Age) in the four groups, we are confident that this threat was mitigated.

We compared two approaches with different robots, in the first experiment run: one using *Arduino*, and the other using *Lego* hardware. This might have introduced a bias in the results if one of the robots was easier to program than the other for some reason not directly associated with the programming language. To mitigate this threat, the second run of the evaluation contrasted *GCL2* with the Scratch for *Arduino* language. So, in this second run, the robot was the same for both languages (and also the same used with GCL1).

The choice of *LegoMS NXT* as the platform to test against *GCL1* can also be regarded as a potential threat. Lego recently released the *Mindstorms EV3* platform that introduces improvements in their development software and robots. However, in the second run of the experiment, we used a different, but also quite popular language (although not EV3), so *GCL2* is progressively being compared to other alternative languages, rather than just to *LegoMS NXT*. This diversity is expected to mitigate the effects of using a single comparison point.

8 CONCLUSION

We reported how we involved teenagers, the end users, in several iterations of the engineering process of a programming language for low-cost robots and performed usability studies. This language, *GCL*, was contrasted with *LegoMS* and *Scratch*.

The evaluation described in this paper, thanks to the involvement of the end-users (teenagers) since early stages of the development process of the language, was helpful to timely detect, prioritise, and improve crucial usability aspects of *GCL* by identifying its strengths and weaknesses. We observed the convergence of the visual language to the degree of usability (regarding satisfaction

and effectiveness) achieved by existing mature and commercial languages.

As future work, we intend to apply the same described technique in the development of DSLs for other purposes and for different end-user profiles. Also, we foresee the need for developing tools to support software language developers to deal with the significant overhead of the assessments and track the improvements on the different features of the language.

ACKNOWLEDGMENTS

The authors would like to thank the NOVA LINCS Research Laboratory (Grant: FCT/MCTES PEst UID/ CEC/04516/2013) and DSML4MA Project (Grant: FCT/MCTES TUBITAK/0008/2014).

REFERENCES

[1] Marco Angelini, Nicola Ferro, Giuseppe Santucci, and Gianmaria Silvello. 2014. VIRTUE: A visual tool for information retrieval performance evaluation and failure analysis. *Journal of Visual Languages & Computing* 25, 4 (2014), 394–413.

[2] Arduino. 2017. Arduino. (2017). Retrieved November 30, 2017 from http://www.arduino.cc/

[3] Ankica Barišić, Pedro , Vasco Amaral, Miguel Goulão, and Miguel Pessoa Monteiro. 2012. Patterns for Evaluating Usability of Domain-Specific Languages. In *Proc. 19th Conference on Pattern Languages of Programs (PLoP), SPLASH 2012*. ACM, Tucson, Arizona, 14:1–14:34. http://dl.acm.org/citation.cfm?id=2821679.2831284

[4] Ankica Barišić, João Cambeiro, Vasco Amaral, Miguel Goulão, and Tarquínio Mota. 2017. Gyro Creator Language - Companion Site. (2017). Retrieved November 30, 2017 from https://sites.google.com/view/vl-empiricalstudy/home

[5] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. 2001. Goal Question Metric Paradigm. *Encyclopedia of Software Engineering* 1 (2001), 528–532.

[6] Emanuela Bauleo, Serena Carnevale, Tiziana Catarci, Stephen Kimani, Mariano Leva, and Massimo Mecella. 2014. Design, realization and user evaluation of the SmartVortex Visual Query System for accessing data streams in industrial engineering applications. *Journal of Visual Languages & Computing* 25, 5 (2014), 577–601.

[7] Natacha Borgers, Edith De Leeuw, and Joop Hox. 2000. Children as respondents in survey research: Cognitive development and response quality. *Bulletin de methodologie Sociologique* 66, 1 (2000), 60–75.

[8] Amy Bruckman and Alisa Bandlow. 2002. Human-Computer Interaction for Kids. In *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies, and Emerging Applications*, Julie Jacko and Andrew Sears (Eds.). Lawrence Erlbaum and Associates, Boca Raton, FL, USA.

[9] Lejos Community. 2017. Lejos. (2017). Retrieved November 30, 2017 from http://www.lejos.org/

[10] Wanda P Dann, Stephen Cooper, and Randy Pausch. 2011. *Learning to Program with Alice (w/CD ROM)*. Prentice Hall Press, Upper Saddle River, NJ, USA.

[11] Google. 2017. Blockly. (2017). Retrieved November 30, 2017 from https://developers.google.com/blockly/

[12] Maria Hatzigianni and Kay Margetts. 2012. 'I am very good at computers': young children's computer use and their computer self-esteem. *European Early Childhood Education Research Journal* 20, 1 (2012), 3–20.

[13] International Standard Organization. 2011. ISO/IEC FDIS 25010:2011 Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models. (March 2011). Retrieved November 30, 2017 from http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=35733

[14] Jared Jackson. 2007. Microsoft robotics studio: A technical introduction. *Robotics & Automation Magazine, IEEE* 14, 4 (2007), 82–87.

[15] J. Johnson. 2003. Children, robotics, and education. *Artificial Life and Robotics* 7, 1 (2003), 16–21.

[16] Caitlin Kelleher and Randy Pausch. 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys (CSUR)* 37, 2 (2005), 83–137.

[17] Barbara Kitchenham, Lech Madeyski, David Budgen, Jacky Keung, Pearl Brereton, Stuart Charters, Shirley Gibbs, and Amnart Pohthong. 2017. Robust statistical methods for empirical software engineering. *Empirical Software Engineering* 22, 2 (2017), 579–630. https://doi.org/10.1007/s10664-016-9437-5

[18] Frank Klassner and Benjamin Schafer. 2014. Using the New Lego MindStorms EV3 Robotics Platform in CS Courses (Abstract Only). In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE '14)*. ACM, New York, NY, USA, 745–746. https://doi.org/10.1145/2538862.2539024

[19] Roxane Koitz and Wolfgang Slany. 2014. Empirical Comparison of Visual to Hybrid Formula Manipulation in Educational Programming Languages for Teenagers. In *Proceedings of the 5th Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU '14)*. ACM, New York, NY, USA, 21–30. https://doi.org/10.1145/2688204.2688209

[20] Tomaž Kosar, Marjan Mernik, and Jeffrey Carver. 2012. Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments. *Empirical Software Engineering* 17, 3 (2012), 276–304. https://doi.org/10.1109/MS.2003.1231149

[21] Lego. 2017. Enchanting. (2017). Retrieved November 30, 2017 from http://enchanting.robotclub.ab.ca/

[22] Lego. 2017. Lego Mindstorms. (2017). Retrieved November 30, 2017 from http://mindstorms.lego.com/en-us/Software/Default.aspx

[23] Pedro Leonardo. 2013. *Child Programming : An adequate Domain Specific Language for programming specific robots*. Master's thesis. Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa.

[24] Xiao-Wen Terry Liu. 2005. *An intuitive and flexible architecture for intelligent mobile robots*. Ph.D. Dissertation. The University of Manitoba.

[25] Panos Markopoulos and Mathilde Bekker. 2003. On the assessment of usability testing methods for children. *Interacting with computers* 15, 2 (2003), 227–243.

[26] Myles McNally, Michael Goldweber, Barry Fagin, and Frank Klassner. 2006. Do Lego Mindstorms Robots Have a Future in CS Education?. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '06)*. ACM, New York, NY, USA, 61–62. https://doi.org/10.1145/1121341.1121362

[27] NCH Software. 2016. Debut Video Capture Software. (2016). Retrieved November 30, 2017 from http://www.nchsoftware.com/capture/

[28] Donald A Norman and Stephen W Draper. 1986. *User centered system design*. Lawrence Erlbaum Associates, Hillsdale, NJ, USA.

[29] Seymour Papert. 1980. *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc., United Kingdom.

[30] Pololu. 2017. Pololu 3pi. (2017). Retrieved November 30, 2017 from https://www.pololu.com/product/975/

[31] Uvais A Qidwai. 2007. A LAMP-LEGO Experience of Motivating Minority Students to Study Engineering. *SIGCSE Bull.* 39, 4 (Dec. 2007), 41–44. https://doi.org/10.1145/1345375.1345411

[32] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. 2009. Scratch: programming for all. *Commun. ACM* 52, 11 (2009), 60–67.

[33] Eric Rosenbaum, Evelyn Eastmond, and David Mellis. 2010. Empowering Programmability for Tangibles. In *Proceedings of the Fourth International Conference on Tangible, Embedded, and Embodied Interaction (TEI '10)*. ACM, New York, NY, USA, 357–360. https://doi.org/10.1145/1709886.1709974

[34] Gavin Sim, Brendan Cassidy, and Janet C. Read. 2013. Understanding the Fidelity Effect when Evaluating Games with Children. In *Proc. 12th International Conference on Interaction Design and Children (IDC '13)*. ACM, New York, NY, USA, 193–200.

[35] Gavin Sim and Matthew Horton. 2012. Investigating Children's Opinions of Games: Fun Toolkit vs. This or That. In *Proceedings of the 11th International Conference on Interaction Design and Children (IDC '12)*. ACM, New York, NY, USA, 70–77. https://doi.org/10.1145/2307096.2307105

[36] Karel Vredenburg, Ji-Ye Mao, Paul W. Smith, and Tom Carey. 2002. A Survey of User-centered Design Practice. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '02)*. ACM, New York, NY, USA, 471–478. https://doi.org/10.1145/503376.503460

[37] Jeannette M. Wing. 2006. Computational Thinking. *Commun. ACM* 49, 3 (March 2006), 33–35. https://doi.org/10.1145/1118178.1118215