RED. Revista de Educación a Distancia
E-ISSN: 1578-7680
mzapata@um.es
Universidad de Murcia
España

Bender, Walter
The Sugar Learning Platform: Affordances for Computational Thinking
RED. Revista de Educación a Distancia, núm. 54, junio, 2017, pp. 1-19
Universidad de Murcia
Murcia, España

RED. Revista de Educación a Distancia. Núm. 54. Artíc. 1. 30-06-2017
DOI: http://dx.doi.org/10.6018/red/54/1
http://www.um.es/ead/red/54/bender.pdf

# The Sugar Learning Platform: Affordances for Computational Thinking

## La plataforma de aprendizaje Sugar: *Affordances* educativas1[1] para el pensamiento computacional

Walter Bender
Sugar Labs. Massachusetts Institute of Technology. Cambridge, USA
walter@mit.edu

**Abstract**
After ten years of deployment of the Sugar Learning Platform, we reflect on the specific tools and affordances deployed to engage learners in computational thinking with the overarching goal of fluency. These tools include multiple media-rich programming environments and also mechanism for debugging, collaboration, expression, and reflection. We motivate our selection of tools by reviewing the pioneering work of Seymour Papert, Marvin Minsky, and Cynthia Solomon, who first brought multimedia computing to elementary schools in the late 1960s with the goal of engaging children in the mastery of many of the heuristics and algorithms we associate with computational thinking. Multiple examples of how these tools have been used by teachers and students are discussed. We further describe the role that Free/Libre Software plays in providing scaffolding for deep and personal expression through programming and for surfacing personal responsibility, a sense of community, and unbounded expectations of Sugar users turned Sugar developers.
**Keywords**: Constructionism, Programming, Logo, Free/Libre Software, Computational Thinking

**Resumen**
Diez años después del lanzamiento de *Sugar Learning Platform*, reflexionamos sobre las herramientas específicas y las *affordances* educativas promovidas por esta plataforma para

[1] Nota del editor: El término "*affordance* educativa" ha adquirido un significado que se relaciona con “la respuesta a la búsqueda de expresar las propiedades de un entorno que, al interactuar con un usuario, mejora el potencial de aprendizaje”. En palabras de Kirschner (2002, p.14) :

"Educational affordances are those characteristics of an artifact (e.g. how a chosen educational paradigm is implemented) that determine if and how a particular learning behavior could possibly be enacted within a given context (e.g., project team, distributed learning community). Educational affordances can be defined [...] as the relationships between the properties of an educational intervention and the characteristics of the learner (for CSCL: learner and learning group) that enable particular kinds of learning by him/her (for CSCL: members of the group too)."

"*Affordances* educativas son las características de un artefacto (por ejemplo, cómo se implementa un paradigma educativo determinado) que determinan si una modalidad particular de aprendizaje podría ser asumida en un contexto determinado (por ejemplo, trabajar un proyecto en equipo, establecer una comunidad de aprendizaje distribuido) y cómo se produce. Una *affordance* educativa se puede definir [...] como las relaciones entre las propiedades de una intervención educativa y las características de los alumnos que permiten que se produzcan determinados tipos de aprendizaje en ellos. "

implicar a los estudiantes en el pensamiento computacional con el objetivo general de lograr su dominio. Estas herramientas incluyen múltiples entornos de programación multimedia, así como diversos mecanismos para la depuración, colaboración, expresión y reflexión. Nuestra selección de herramientas viene determinada por la revisión exhaustiva de la obra pionera de Seymour Papert, Marvin Minsky y Cynthia Solomon, que fueron los primeros en integrar la informática multimedia en los centros de Educación Primaria a finales de la década de los 60, con el objetivo de implicar a los estudiantes en el dominio de muchos de los conceptos heurísticos y algoritmos que se asocian con el pensamiento computacional. En el presente artículo, se analizan muchos ejemplos de cómo han utilizado estas herramientas tanto los docentes como los discentes. Además, describimos el papel que desempeña el Software de Código Abierto a la hora de vertebrar el andamiaje para una expresión personal y profunda a través de la programación y para poner de relieve la responsabilidad personal, el sentido de comunidad y las expectativas sin límite de los usuarios de Sugar que se convierten en desarrolladores.

**Palabras clave:** Constructionismo, Programación, Logo, Software Free/Libre, Pensamiento Computacional

# 1.0 Introduction

The Sugar Learning Platform (Sugar) was first developed as the software platform for the One Laptop per Child (OLPC) program (Bender et al., 2012), a spin-off project from the MIT Media Lab in 2006. The founders of OLPC, Nicholas Negroponte, Seymour Papert, and Walter Bender promoted the idea that learning is not a luxury and lack of an educated society is not just an inconvenience. Learning is fundamental to a free society that aspires to be both prosperous and sustainable. All societies value the role that education plays in economic development. And education is essential to the survival of democratic societies. How should we go about leveraging the latent capacity to learn? How can we transform a consumer-oriented culture into a learning-oriented culture? And is it possible to design a learning platform that respects the diversity of educational context found across the developed and developing world? These questions motivated the creation of the Sugar Learning Platform and the Sugar Labs community that develops and maintains the platform.

## Learning with Logo

In their 1970 paper, "Conceptual advances derived from the Muzzey experiment" (Papert and Solomon, 1970), Papert and Solomon, co-inventors of the Logo programming language (along with Wally Feurzeig), described their pioneering efforts to use the Logo language as a vehicle for introducing children to computational thinking, which they define in terms of heuristics: things to think with and to reflect upon. The following year, Papert and Solomon published “20 Things to do with a Computer” (Papert and Solomon 1971), a harbinger of what today is often referred to as the “Maker Movement”. Minsky contributed to many of the 20 project ideas, which ranged from robotics to music to the visual arts. Children in the Muzzey experiment were using Logo as a tool for the autonomous construction of meaningful artifacts and pursuing mastery in support

of solving problems that were personally meaningful. (Arguably one difference between the Maker Movement and the work of Papert, Minsky, and Solomon is that “makers” tend to focus on the artifact being created, whereas Papert et al. focused on the learning associated with the construction of the artifact.)

## Constructing Cognitive Towers

“A six-year-old child can become an expert.”—Marvin Minsky

In a 2008 memo (Minsky, 2008) questioning “general” education, Minsky proposed that we “re-aim our schools towards encouraging children to pursue more focused hobbies and specialties—to provide them with more time for (and earlier experience with) developing more powerful sets of mental skills, which they later can extend to more academic activities.”

Minsky encourages a child to construct multilevel “cognitive towers”, built upon instinctive reactions, learned reactions, deliberate thinking, reflective thinking, self-reflective thinking, and self-conscious reflection. These levels are reflected in the theory of human self-critical thinking that he details in *The Emotion Machine* (Minsky, 2006), one in a series of books in which he develops a model of the mind. The levels span agencies, each of which specializes in areas such as gaining knowledge from experience, planning and causal attribution, the construction of models, and identifying values and ideals. A focus on achieving meaningful goals, not just the accumulation of simple knowledge objects, exercises all of the levels in a cognitive tower, helping a child “develop proficiencies that can be used in other domains.”

In his review of *The Emotion Machine*, Todd Stark articulates the central implication of Minsky's model as “an epistemological stance that resourcefulness in human thinking is a matter of switching between different kinds of representations, each used in a different way of thinking, each of which captures something essential about specific things in our world while necessarily leaving out other details” (Stark 2006). As a model for learning, the levels within Minsky’s cognitive towers represent skills that can be applied broadly.

Minsky’s towers are inherently complex and require a learner to be motivated and persistent in pursuit of their construction, a process that he once described as "hard fun." This is consistent with Daniel Pink, who reviews four decades of studies (Pink, 2009) that demonstrate that motivation for learning comes from autonomy to explore and express ideas; the confidence and space to master knowledge; and opportunity to engage in authentic problem solving, which leads to a sense of purpose. The children in the Muzzey experiment were motivated learners, pursuing meaningful goals that led to the develop proficiencies that could be used in other domains in the creation of their “towers”. A key insight of Minsky, Papert, and Solomon was to give children tools that they can use to explore, that they can master and that they can apply to problems that they are passionate about. In the Muzzey experiment, learning was something a child did, not something that was done to or for a child.

## Computational thinking

In their 2016 book, *Algorithms to Live By: The Computer Science of Human Decisions*, Brian Christian and Tom Griffiths describe many of the core problems of computer science in layman's terms and apply insights from decades of algorithm development to human decision-making (Christian and Griffiths, 2016). In the process, they make a compelling case for computational thinking beyond the realm of learning to code. While the specific algorithms they discuss—searching, sorting, optimal stopping, predicting, etc.—are useful in and of themselves, the real power of computational thinking lies it its systematic approach to debugging and problem-solving. Learning that problems can be addressed systemically is the underlying "powerful idea" of programming. The process of writing and then repairing or "debugging" a program provides a basis for active learning through trial and error, regardless of what the problem that is actually being solved.

The Logo language was designed to introduce children to programming and to computational thinking. The "curriculum" developed by Solomon and Papert immersed children in problem-solving and debugging. Children were given agency to work on problems they were passionate about in a context where there was an expectation that there were no predetermined solutions or prescribed paths to a solution. It was during the Muzzey experiment that Solomon first observed that "debugging is the greatest learning opportunity of the 21st century." While engaged in problem-solving, children were developing and refining the algorithms employed by the agents in the various levels of their cognitive towers.

## Sugar and Sugar Labs

The Sugar Learning Platform was designed with Minsky's cognitive towers in mind. Sugar provides the user with affordances (things we designed for the possibility of taking action by the user (Gaver, 1991)) and pathways to engage in developing skills mirroring the various agencies in the tower levels. Sugar has been criticized as being a collection of tools rather than an instructional curriculum, but this was by design. This is not just because curricula tend to reflect very local norms and needs, and as a consequence, are resistant to global solutions. It also reflects an explicit rejection of instruction as a pedagogical framework. With Sugar, we try to give agency to the learner, knowing that in all likelihood, the institution of school would likely be trying to take away agency. At each design decision point, we asked ourselves: "how will this impact the learning?" and "how will this impact the autonomy and agency of the learner and the teacher?"

Agency is made possible in part because of the choice of a software license, the General Public License (GPL) (Smith, 2014), which ensures that the learner has permission to both use Sugar and to modify it. We go a step further by giving the learner affordances to engage in software development and debugging, i.e., to exploit the license. We provide a context for encouraging the learner to take initiative, both by deliberately leaving the platform "incomplete" and by providing a social norm in which it is expected to take initiative. Even if it were possible to make Sugar "complete", we would have chosen not to. We wanted there always to be "itches" needing to be scratched. In a related design decision, Sugar was never intended to be an endpoint in

and of itself. Rather, we envisioned it as a waypoint along a lifelong journey of learning. We encourage our users to outgrow Sugar and even provide them with a means of advancing from the Sugar desktop, with its tools for exploration to the GNOME desktop, with its more powerful tools for production. (Indeed, many of our youth developers moved on to other software projects. And some moved on to different fields of expression altogether: Daniel Francis, a prolific programmer and member of the Sugar Labs oversight board when he was a teenager, is now an aspiring classical pianist. He still contributes the occasional patch to Sugar.)

The rest of this article describes the Sugar Learning Platform and Sugar Labs, the organization that develops and maintains the platform. Specifically, we delve into design decisions that were made in support of an approach to computation that focuses on agency and problem solving, and the ways in which the community has organized itself around end-user participation and learning.

## 2. The Sugar Learning Platform

Development of Sugar began in the spring of 2006. Rather than starting from scratch, we built Sugar upon decades of research into how to engineer software to promote learning. We coupled our efforts with key partnerships within the Free/Libre Software movement. Our principal partners in Sugar development were a small engineering team from Red Hat, a major distributor of the open-source computer operating system GNU/Linux, and Pentagram, an international design partnership that does work in graphic design, identity, and product development. (The name Sugar was coined by Marco Presenti Gritti, a Red Hat engineer, who said learning should be sweet. All of our attempts to turn SUGAR into an acronym were unsuccessful, but the name stuck.) In six months, this core group was able to produce a basic framework for Sugar upon which a community of pedagogists and software engineers could build learning activities.

The team worked with the Free/Libre Software community to leverage many preexisting software projects, including Fedora and the GNOME toolkit. This community is made up of a large group of programmers who are committed to an ethic of user freedom to run, copy, distribute, study, change, and improve software. The community openly shares programs they have written with one another. Any member of the community has the right to use a program as is, or to adapt it as desired, without needing permission of any sort (Smith, 2014).

With the engagement of the core development team and a growing number of volunteers from the Free/Libre Software community, it took only six months to move from idea to completion of the first version of Sugar. The team used an iterative-design process: rapid prototyping of ideas, followed by critiques, followed by coding. We went through two to three cycles per week until we reached consensus on a basic framework. It was at this point that we were able to set higher-level goals enabling participation by a broader community of developers (and eventually the Sugar users themselves).

"The broader development community, which was dispersed across five continents, was engaged in addressing the same problems and met 24/7 in multilingual online chat forums. This was a global movement: the lead developer lived outside of Milan, Italy, the lead community contributor lived in Siberia, and the principal testing team operated out of a coffee shop in Wellington, New Zealand. Significant contributions were made by a high-school student from Wunstorf, Germany, an energy-management consultant living in Melbourne, Australia, and a student at the University of San Carlos in Brazil. The use of modern software-development tools, such as distributed source-code management and wikis enabled members of the development community to collaborate anywhere and at any time. We were also able to pilot Sugar in a wide range of contexts, getting hands-on experience and feedback in schools in Nigeria, Thailand, Cambodia, and Brazil." (Bender et al., 2012) This distributed process of development directly influenced the selection of affordances and tools surfaced in the Sugar interface itself, as we aspired to give the Sugar users a similar experience to that of the Sugar developers.

## A platform for collaboration, reflection, and discovery

The Sugar Learning Platform was designed to promote collaborative learning through tools and activities that encourage critical thinking. Sugar puts an emphasis on divergent thinking. A related goal is to make that thinking visible to the learner. Sugar equally promote cultures of expression and reflection. With Sugar, we provide teachers and learners with a collection of open-ended tools and activities, applicable to problems of their own choosing.

The choice and organization of the tools is analogous to how one might organize a kitchen. Everyday tools, e.g., a coffeemaker, are on the low shelf, for easy access. Less commonly used tools, e.g., a special baking pan for a holiday cake, are on a higher shelf, since they are not needed as often. The selection of "low shelf" tools in Sugar is left up to the teacher and student, although Sugar has a default selection that emphasizes tools for expression and reflection: write, record, program, chat, journal, portfolio, etc.

Sugar offers an alternative to traditional "office-desktop" software based on three affordances: (1) Sharing: Collaboration is a first-order experience. The interface always shows the presence of other learners who are available for collaboration. Sugar allows users to dialog, support, critique, and share ideas with each other. (2) Reflecting: A ''journal'' records each learner's activity. It is a built-in space for reflection and assessment of progress. (3) Discovering: Sugar tries to accommodate a wide variety of users with different levels of skill in terms of reading and language and different levels of experience with computing by providing activities with a ''low floor'' and, where possible, ''no ceiling.''

## The only time collaboration is called cheating is when you are in school.

Sugar drew inspiration for its activities and the fluid interface between activities from observing how the free software community collaborates. Software developers chat, socialize, play games, share media, and collaborate on media creation and programming in both formal and informal settings. The Sugar parallels to the Free/Libre Software movement are tools of expression,

children creating content as well as consuming it, and a strong emphasis on collaboration, co-creation, and helping one another. As with Free/Libre Software, Sugar encourages every child to be a creative force within their community.

Sugar users are given access to a variety of commonly used tools for collaboration, e.g., Chat, and IRC. By default, the IRC application opens into the `#sugar` channel on `irc.freenode.net`, where Sugar developers discuss their work. Sugar users are given real-time access to the developers of the very tools that they are using and the opportunity to watch (and participate) in the deliberations of professional software developers working on problems that directly impact them. Many want-to-be developers come into the community this way at an age as young as nine or ten years. (It is a testament to the generous spirit of the Sugar community that they have welcomed these children, who often behave as children, disrupting and straying off topic. Providing an opportunity for learning trumps all else.)

David Cavallo, in his work on emergent design (Cavallo, 2000), describes the “motorcycle culture” of rural Thailand, where children would gather around mechanics, observing and helping as they repaired and modified their vehicles. Bender has observed a similar scene in rural India, halfway between Delhi and Pilani, where tractor repairs are done by mechanics while surrounded by swarms of attentive children. The open problem-solving by software developers in IRC is another example of children learning from experts who do their work in the open.

In addition, Sugar provides a framework, the Neighborhood, based on Collabora’s Telepathy package (Sugar Toolkit, 2017), for making any application capable of sharing data in real time, one-to-one, one-to-many, or many-to-many. When given an opportunity, applications were enhanced by collaboration: the word processor, the camera application, various programming environments, etc. Sugar enhancements to Abiword (Sugar Write, 2008) enabled real-time peer writing and editing, similar to the now commonplace Google Docs, but without the necessity to utilize the Cloud or expose personal data to third parties. The Sugar journal enabled file sharing, similar to Google Drive, but again without the necessity to utilize the Cloud or expose personal data to third parties. In “server mode”, shared data need not ever leave the classroom. In “ad hoc” or “mesh” mode, shared data remain strictly between those participating in a collaboration. Sugar provides powerful tools for collaboration and productivity while it respects the privacy of children and their teachers.

### Reflection and assessment in the context of Sugar

In the early days of the development of the Sugar user interface, one of the biggest points of contention with the OLPC advisory board was when we told them that we were not going to use file browsing as the primary mode of navigation. We were asked, “how will the children learn to use Windows?” Our response was, “why do they need to learn to use Windows?” They can engage with the powerful ideas of computation without acquiescing to the idiosyncrasies of a specific vendor of proprietary software. In retrospect, we probably made the correct decision in that very few contemporary interfaces use file browsing; it is not in the critical path to computation.

Instead of browsing a filesystem, Sugar gives the user a journal or notebook into which one's work is "kept" rather than "saved". The interface tries to keep things that offer value automatically in the Sugar journal. (The journal was the brainchild of Ivan Krstić, who also designed the OLPC security model.) The primary function of the journal is as a time-based view of the activities of a learner. As with physical media, such as pen on paper, no explicit "saving" step is needed. The individual journal entries are treated much like pages in a laboratory notebook. There is a title, room for taking notes, and adding tags. The learner is encouraged to adopt a routine where by time is taken to write about what they are doing either while they are doing it or immediately afterward. This process of note taking becomes the basis upon which they can subsequently engage in reflection. This mechanism is similar to the "commit message" used in source-code management systems, e.g., git, which would be familiar to software engineers. It is also parallels practice common in the arts and sciences (Hetland et al., 2007).

Sugar acknowledges the need for measurement and evaluation. While Sugar does not take a position in the debate on high-stake testing, our goal is to have learning have some positive socio-economic impact on children, we do advocate for an evaluation of our interventions that look more broadly than those data that are captured by standardized tests. We developed a series of recommendations for innovation in evaluation at different levels (Urrea and Bender, 2012) (Bender and Urrea, 2015): at the level of individual students, teachers, and parents; at the level of a classroom or school; and national and global indicators.

The primary tool for assessment within Sugar is the digital portfolio. Sugar journal entries are directly incorporated into digital portfolios to support reflection that can help students (as well as teachers and parents) be aware of their own learning, and do so by documenting their work and thinking over time. Digital portfolios are part of a "comprehensive system that combines formal, informal, and classroom assessment, including portfolios, to inform the state, the district, the school, and the teacher" (Stefanakis, 2002). Without a way to make visible what students do and what teachers teach, it is difficult to make changes to improve those dynamics.

The Sugar journal also has a fixed set of metadata entries that are displayed in the journal detail view for all entries, e.g., "description", "tags", "preview", et al., as well as activity-specific metadata. For example, when assessing student work, it is of interest to teachers to know what tools a student may have used and, perhaps how many iterations a student made in creating an artifact. These data may vary from activity to activity, hence an enhancement to the journal "expanded view" enables Activities to specify which metadata fields would be useful to display. Utilizing these data are some Sugar tools that deploy rubrics to help teachers understand the impact and evolution of the program in a larger context—at the level of the classroom or the school. We have designed tools that navigate and visualize data automatically derived from the learning activities in which the learners are engaged. These data help teachers, administrators and stakeholders understand the impact of a program and make adjustments to it.

Some Sugar applications incorporate strategies for understanding the use of computation in learning at a much larger scale. These strategies involve the design and implementation of a

repository of objects or artifacts designed by children from different programs. There are a number of similar repositories with artifacts from an individual already in existence, e.g., the Scratch website (Scratch, n.d.) and Turtle Blocks (Turtle Blocks, n.d.). Such collections make possible the analysis and understanding of impact at a large scale, and the learning that emerges, not only at the individual, but also at the collective level. And these collections present another opportunity for learners to learn from each other.

### Discovery: Low floor and no ceiling

Sugar comes with a variety of programming environments that allow children to use a computer as a tool for creativity. These include many programs such as Turtle Blocks, which is based on Logo and allows children to make images while learning concepts of geometry and programming; Scratch, a programming language that enables the creation of interactive stories, games, music, and art; and Squeak Etoys, a multimedia authoring environment and visual programming system that is meant as an educational tool to teach children powerful ideas (primarily science and math) in new ways.

These tools are by-in-large designed to have a low floor, i.e., easy to learn, and a high ceiling, i.e., few if any constraints on how the tools can be applied. A core example is Turtle Blocks (Sugar Turtle Blocks, 2010), an activity with a Logo-inspired graphical “turtle” that draws colorful art based on snap-together visual programming elements. Its low floor provides an easy entry point for beginners. It also has high-ceiling programming features which will challenge the more adventurous student. These features include blocks to access sensors and various inline and external mechanisms for adding extensions to the block language. Turtle Blocks has been used to create, for example, presentation software, paint, multimedia chat, an oscilloscope, etc. Instead of just using Power Point™, the children can write their own presentation tools. Turtle Blocks also allows the user to export their block-based program as Python code, which in turn can be converted to a stand-alone Sugar activity, setting the learner down the path of text-based programming and embarking on the journey from user to developer.

## 3. Sugar in the field

One of the first formal studies of Sugar took place in Uruguay in 2009–10. Uruguay was the first country to provide every child a free internet-connected laptop computer. They began distributing OLPC XO laptops running Sugar in 2007. Even though Uruguay is a relatively small country, with less than 500,000 children, it took several years before they could achieve full coverage. The last region to receive laptops was Montevideo. Montevideo was last because there was less need there than in the more rural regions, since many urban children already had access to computers. The delay in deploying in Montevideo presented an opportunity to study the impact of Sugar (DSPE-ANEP, 2010). Children were asked in 2009—before they has Sugar—what they did with their computers. It should come as no surprise that they used their computers to play games (See Figure 1). The same children were asked in 2010—after almost one year of using Sugar—what they did with their computers. Again they responded that they used their computers to play games. They were still children after all. But they also used their computers to write, chat, paint, make and watch videos, search for information, etc. In other

words, with Sugar, they used the computer as a tool. Children play games. But given the opportunity and the correct affordances, they can leverage computation to do much much more.

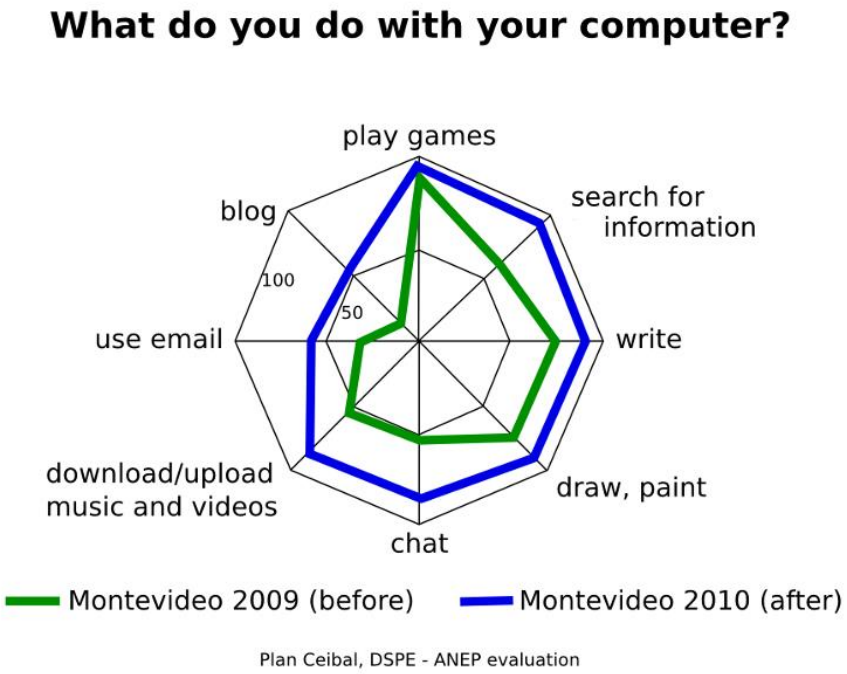


Figure 1: Data from a DSPE-ANEP survey of students in Montevideo before and after the deployment of Sugar

## Free/Libre Software as an affordance

Sugar was designed so that new uses emerging from the community could easily be incorporated, thus Sugar could be augmented and amplified by its community and the end users. We encouraged our end users to make contributions to the software itself. This was in part out of necessity: we were a small team with limited resources and we had no direct access to local curricula, needs, or problems. But our ulterior motive was to engage our users in development as a vehicle for their own learning.

One of the first examples of end-user contributions took place in Abuja, Nigeria, site of the first OLPC pilot project. While teachers and students took to Sugar quickly, they did confront some problems. The most notable of these was that the word-processor application, Write, did not have a spelling dictionary in Igbo, the dialect used in this particular classroom (and one of the more than three-hundred languages currently spoken in Nigeria). From a conventional software-development standpoint, solving this problem (300 times) would be prohibitively expensive. But for children in Abuja, equipped with Sugar, the solution was simple: confronted with the problem of lacking a dictionary, they made their own Igbo dictionary. The did not look for others to do the work for them. The took on the responsibility themselves. The Free/Libre Software ethic built into Sugar enabled local control and innovation.

John Gilmore heard the about our aspiration to reach out to our end users—children—at the 2008 Libreplanet conference. He asked, “how many patches have you received from kids?” At the time, the answer to his question was zero. But over the past nine years, the situation has changed dramatically. By 2015, 50% of the patches in our new releases were coming from children (See Table 1); our release manager in 2015–16 (Sugar v0.108) was Sam Parkinson, a fifteen-year-old from Australia; our current release manager (Sugar v0.110) is Ignacio Rodríguez, an eighteen-year-old from Uruguay who began hanging out on our IRC channel at age ten and contributing code at age twelve.

| Release number (date) | Total commits | Youth commits | Release notes URL |
|---|---|---|---|
| 0.102 (July 2014) | 424 | 108 | https://wiki.sugarlabs.org/go/0.102/Notes |
| 0.104 (February 2015) | 249 | 127 | https://wiki.sugarlabs.org/go/0.104/Notes |

Table 1: Sugar commits by youth contributors

When the now former president of Uruguay, José Mujica, learned that a twelve-year-old from a small town east of Montevideo had programmed six entirely new Sugar activities for the XO, he smiled and said triumphantly: “Now we have hackers.” In his eyes, this one child’s ability to contribute to the global Sugar development community was a leading indicator of change and development in his country.

None of this happened on its own. End-user contributions are not simply an artifact of Sugar been Free/Libre Software. Open-source Software gives you access to the code and that Free/Libre Software gives you a license to make changes. But without some level of support, very few people will have the means to exercise the rights granted to them under the license. For this reason, we built scaffolding into Sugar to directly support making changes and extensions to Sugar applications and Sugar itself.

## Opening up the “black box” through “view source”

In addition, Sugar has no black boxes: the learner sees what the software does and how it does it. Sugar is written in Python and comes with all of the tools necessary to modify Sugar applications and Sugar itself. We chose Python as our development language because of its transparency and clarity. It is a very approachable language for inexperienced programmers.

With just one keystroke or mouse click, the Sugar “view source” feature allows the user to look at any program they are running. A second mouse click results in a copy of the application being saved to the Sugar Applications directory, where it is immediately available for modification. (We use a “copy on write” scheme in order to reduce the risk of breaking critical tools. If there is no penalty for breaking code, there is better risk-reward ratio for exploring and modifying code.) The premise is that taking something apart and reassembling it in different ways is a key to understanding it.

Constructionist-learning advocate Gary Stager once remarked (in a private conversation) that teachers should not have to use git. (The context of the conversation was the process for installing extensions to Turtle Blocks.) But git is a tool that provides access to the powerful ideas inherent in content management. Why should teachers (or learners) be discouraged from using it? In Sugar, we provide a link to the git repository of every application so that the full development history and commit messages are available to would-be developers. While it is not necessary to use git in order to use, view, or modify Sugar, we strive to introduce our users to the powerful tools and best practices of the software development industry.

## Teachers as mentors

Not every creative use of Sugar involves programming. Rosamel Norma Ramírez Méndez is a teacher from a school in Durazno, a farming region about two-hours drive north from Montevideo, Uruguay. Ramírez had her lessons for the week prepared when, first thing Monday morning, one of her students walked into her classroom holding a loofa. The child asked Ramírez, “teacher, what is this?” Rather than answering the question, Ramírez seized the opportunity to engage her class in an authentic learning experience. She discarded her lesson plans for the week. Instead, on Monday the children figured out what they had found; on Tuesday they determined that they could grow it in their community; on Wednesday they investigated whether or not they should grow it in their community; on Thursday they prepared a presentation to give to their farmer parents on Friday about why they should grow this potential cash crop. Not every teacher has the insight into learning demonstrated by Ramírez. And not every teacher has the courage to discard their lesson plans in order to capture a learning opportunity, But given an extraordinary teacher, she was able to mentor her students as they used Sugar as a tool for problem-solving. Ramírez encouraged her students to become active in their learning, which means that they engaged in doing, making, problem-solving, and reflection. (It is worth noting that Ramírez was subsequently recruited to work with teachers throughout the country, helping them learn to leverage Sugar in their classrooms.)

## Teachers can learn (and contribute) too.

Sometimes teachers have been directly involved in Sugar software development. Sugar has an abacus application (Sugar Abacus, 2010) to help children explore whole-number arithmetic and the concept base (the activity allows the user to switch between various base representations of whole numbers). It also lets children design their own abacus. Teachers in Caacupé, Paraguay, were searching for a way to help their students with the concept of fractions. After playing with the Sugar abacus activity, they conceived and created—with the help of the Sugar developer community—a new abacus that lets children add and subtract fractions (See Figure 2). Sugar didn't just enable the teachers to invent, it encouraged them to invent.

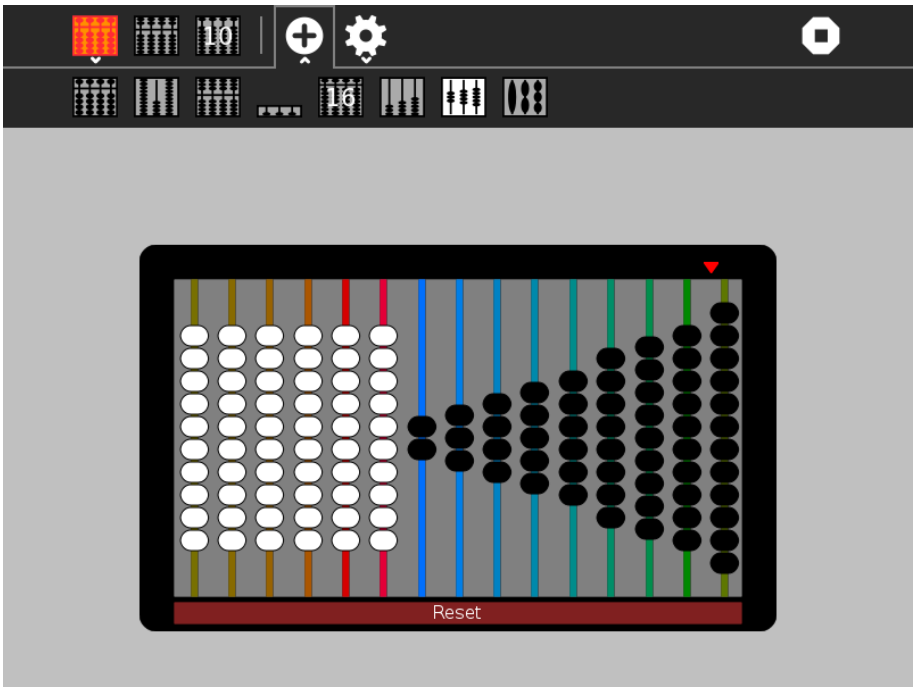


Figure 2: The Caacupé Abacus. The white beads represent whole numbers. The black beads represent fractions.

Guzmán Trinidad, a high-school physics teacher from Montevideo, Uruguay and Tony Forster, a retired mechanical engineer from Melbourne, Australia collaborated on a collection of physics experiments that could be conducted with a pair of alligator clips and a small collection of Sugar applications (Trinidad, 2013). In the process of developing their experiments, they maintained regular communication with the developers, submitting bug reports, documentation, feature requests, and the occasional patch. Other examples of teacher and community-based contributions include Proyecto Butiá, a robotics and sensor platform build on top of Sugar (and GNOME) at Facultad de Ingeniería, Universidad de la República, Uruguay (Butiá, 2009). Butiá inspired numerous other robotics platforms, e.g., RoDI (Robot Didáctico Inalámbrico) (RoDI, 2015) developed in Paraguay, as well as a wealth of projects aligned with the pedagogy of Constructionism. In the spirit of Sugar, these hardware projects were all designed to be “open”: schematics and firmware were made available under Free/Libre licenses.

## Community resources

Professional developers don’t work in isolation and neither should learners. There are a number of existing crowd-sourced resources available on line, e.g., Slashdot, where learners (and professionals) can go for help. There are also a number of resources specific to Sugar that have developed in parallel with the project. In Uruguay, Leticia Diana Romero Cabrera, Pablo Flores, and other local developers organized Asociación Civil Ceibal Jam (Ceibal Jam, 2009) as a means of interfacing between teachers and Sugar developers. Flavio Danesse organized Python Joven (Danesse, 2012), an extra-curricular coding club for youths in the Montevideo

area, which fed a steady stream of developers to the project. Both organizations drew much of their inspiration from the fact that the work that they did was not just an academic exercise, but had immediate impact of all of the children of Uruguay and literally millions of other children around the world.

Some of the other opportunities for contributing to Sugar were more centralized. Sugar Labs has participated in two Google-run programs in support of Free/Libre Software: Google Summer of Code and Google Code-in. Summer of Code is an internship program for university students interested in participating in Free/Libre Software projects. Sugar Labs has largely used the program for “moon shots”, i.e., high-risk explorations. For example, the aforementioned Python export mechanism for Turtle Blocks was written by Marion Zepf, a Summer of Code intern from Germany (Zepf, 2013). Google Code-in is a programming contest for teenagers. Participating projects, such as Sugar Labs, compile small (and ambitious) programming challenges over a seven-week window. Hundreds of teenagers participate and hundreds of patches are submitted. Prior to becoming our release manager, Rodríguez, entered the Code-in contest twice and won handily both times (Acosta, 2014). (While he was participating in the contest, he also helped the other contestants with their projects.) He went on to become a mentor to both Code-in students and Summer of Code and he recently represented Sugar Labs at a Google summit.

There also remains a small core team of professional developers who work on Sugar. OLPC still uses Sugar and been employing a consultant to make builds, squash bugs, and interface with the community. Other community members have done some on-again off-again consulting for specific Sugar deployments. But for the most part, Sugar remains the purview of a loosely organized team of global volunteers.

# 4. Conclusion

> Conjecture: once a child builds a cognitive tower that works well in some particular realm, that child will thereafter be better equipped to develop proficiencies that can be used in other domains.
>
> The idea is that it seems plausible that the first few such developments could have a major effect on the qualities of that child’s future ones—because those will the child’s first experiments with organizing such “vertical” structures. If so, then this would imply that our children’s early education should focus on activities, hobbies, and specialties that has the ‘desirable’ kinds of such qualities. Of course, this also implies that we’ll need good theories of what ‘good’ such qualities ought to be—and which kinds of “curriculums” help to promote them. (Minsky, 2008)

We designed Sugar with the goal of providing learners with powerful tools: "things to think with." Or specific focus is on tools for expression—writing, programming, drawing, composing, etc.—collaboration—sharing and mentoring—and reflection—commentary and critique. We chose tools that would encourage open-ended problem-solving and debugging through engagement in

authentic projects, during which time they would develop and hone their mental skills and over time strengthen their cognitive towers.

“It is said that the best way to learn something is to teach it—and perhaps writing a teaching program is better still in its insistence on forcing one to consider all possible misunderstandings and mistakes.” (Papert, 1970) We found that providing activities such as Turtle Blocks, which engage learners in computational thinking within the context of personal expression is necessary, but not sufficient. Giving them tools for reflection enhance the learning experience. And encouraging them to be part of a larger community of learning was motivating (a supportive community does lead at least some learners take real intellectual risks).

One premise is that by basing the platform on the principles of Free/Libre Software, we will achieve both our goal of large-scale deployment of Constructionist Learning and also create an ecology of support for that learning. We want children not just to learn about the computer, but also to learn with the computer. And we want children and their teachers to understand and take responsibility for the tools that they use, so we built specific scaffolding to support maintenance and enhancement of the tools. In building tools, one is forced to “consider all possible misunderstandings and mistakes.”

Returning to the questions we asked in the introduction: How should we go about leveraging the latent capacity to learn? How can we transform a consumer-oriented culture into a learning-oriented culture? It possible to design a learning platform that respects the diversity of educational context found across the developed and developing world?

The ability to not only learn with the machine and software but also to manipulate and change the software and hardware itself opens the door to learning lessons far more important than those necessary to pass a test. It leads children to the discovery that they are “authentic problem-solvers” in the real world. Success at fixing a program also gives students confidence that they can apply the same skills—defining problems, developing hypotheses, creating tests, and executing solutions—to other problems they may encounter.

By giving people tools (mechanical tools, computational tools, and heuristics for problem solving), giving them support (mentoring, facility, and, most important, time) in the application of these tools, and being explicit in setting an expectation that everyone teach and learn, we can make a positive transformation of both individuals and society—a society of learners and teachers, problem-solvers and innovators, both prosperous and happy.

In our discussion of Sugar as a vehicle for computational thinking and fluency, we have used some of the metrics from software development, e.g., pull requests and commits. There is evidence that given the proper conditions, children will use computation in support of problem-solving. Free/Libre Software undoubtedly has a role to play in that it allows students and teachers to operate with autonomy, achieve mastery, and engage in something purposeful.

Access to development tools and a community of mentors are important. Scale matters: Sugar users/developers are part of a global enterprise. And great teachers also have a role to play. As much as we focused on building a learning-centric interface, a master teacher such as Rosamel Norma Ramírez Méndez still makes a huge difference between using a computer and using a computer for learning.

“Learning is hard fun.”—Marvin Minsky

Where have we fallen short? Much has changed since we began Sugar development in 2006. Sugar predates smartphones and Apps, Chromebooks and Google Docs, MOOCs and resources such as the Kahn Academy. Edtech is become big business: selling Apps and content is more lucrative and facile than the hard work of engaging teacher and learners in authentic problem-solving. There is a strong temptation to make things as simple as possible so as to reach the broadest possible audience. But some things are inherently complex. Apps might be fun, but the hard part of “hard fun” is in reaching towards complexity. Children should not miss out on the learning that takes place when learning to use tools. (To be fair, some powerful open-ended tools, such as Google Docs, are seeing rapid uptake in schools.) At Sugar Labs we are attempting to go where the learners are by providing as much of Sugar as we can as a Web app, on Android, or on iOS (Sugarizer, 2017). At the same time, we are encouraged by the growth of the Maker Movement and are working hard to support Sugar on platforms popular with that movement, e.g., Raspbian (Sugar Raspian 2017).

We conclude with a story from the Amazonas region of Peru. In 2012, we were part of a team running a week-long Sugar workshop for more than 60 teachers who had traveled to Chachapoyas, the regional capital. During the day we spend time reviewing the various Sugar activities and discussing strategies for using Sugar in the classroom. In the evenings, we gave a series of optional workshops on specialized topics. One evening, mid-week, the topic was fixing bugs in Sugar. (We had already given a well-attended workshop on repairing the OLPC XO hardware, which like Sugar, was designed to be maintained by end users.) It was not expected that many teachers would attend—in part because we were competing with an annual festival and in part because their background in programming was minimal. But almost everyone showed up. In the workshop, we walked through the process of fixing a bug in the Sugar Mind Map activity and used git to push a patch upstream. Teachers, especially rural teachers, have a hunger for knowledge about the tools that they use. This is in part due to intellectual curiosity and in part due to necessity: no one is going to make a service call to Amazonas. As purveyors of educational technology we have both a pedagogical and moral obligation to provide the means by which our users can maintain (and modify) our products. Enabling those closest to the learners is in the interest of everyone invested in educational technology as it both ensures viability of the product and it is a valuable source of new ideas and initiatives.

Presentación del artículo: 6 de abril de 2017
Fecha de aprobación: 25 de junio de 2017
Fecha de publicación: 30 de junio de 2017

Bender, W. (2017). The Sugar Learning Platform: Affordances for Computational Thinking. *RED. Revista de Educación a Distancia*, 54. Consultado el (dd/mm/aaaa) en http://www.um.es/ead/red/

## Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

## Acknowledgments

Sugar is built directly upon the work of Seymour Papert, Marvin Minsky, and Cynthia Solomon, to whom we are extremely grateful. Sugar would not have been possible without the contributions from Red Hat, Pentagram, OLPC, and a global community of volunteers. Google, through both the Summer of Code program and the Code-in contest has provided fora for contributions to the project. The Software Freedom Conservancy has hosted Sugar Labs since 2008. Children and their teachers have been contributing to Sugar since 2007.

## References

Acosta, I. (2014). En código. *La Diaria*. https://ladiaria.com.uy/articulo/2014/2/en-codigo/ Retrieved 2017-06-24.

Bender, W., Kane, C., Cornish, J., Donahue, N., (2012). *Learning to Change the World: The Social Impact of One Laptop per Child*, Palgrave Macmillan.

Bender, W., Urrea, C. (2015). Visualizing Learning in Open-Ended Problem Solving in the Arts. *RED-Revista de Educación a Distancia*, 46(2).

Butiá (2009). *Proyecto Butiá*. https://www.fing.edu.uy/inco/proyectos/butia/ Retrieved 2017-06-24.

Cavallo, D. (2000). Emergent Design and learning environments: Building on indigenous knowledge. *IBM Systems Journal,* 39(3&4), pp. 768–781.

Ceibal Jam (2009). Convenio marco entre la Asociación Civil Ceibal Jam y la Universidad de la República..

Christian, B. and Griffiths, T. (2016). *Algorithms to Live By: The Computer Science of Human Decisions.* Henry Holt and Co.

Danesse, F. (2012). *Programación Python Joven* http://fdanesse.byethost8.com/python_joven/index.html Retrieved 2017-06-24.

DSPE-ANEP (2011). *Informe de evaluación del Plan Ceibal 2010*. Administración Nacional de Educación Pública Dirección Sectorial de Planificación Educativa Área de Evaluación del Plan Ceibal.

Gaver, W.W. (1991). Technology Affordances. In: Robertson, S.P., Olson, G.M., Olson, J.S. (eds.) *Proceedings of the ACM CHI 91 Human Factors in Computing Systems Conference* April 28 – June 5, 1991, New Orleans, Louisiana. pp. 79–84.

Hetland, L., Winner, E., Veenema S., and Sheridan, K.M. (2007). *Studio Thinking: The Real benefits of Visual Arts Education*, Teachers College Press

Minsky, M. (2006). *The Emotion Machine*. Simon & Schuster.

Minsky, M. (2008). http://wiki.laptop.org/go/Questioning_General_Education Retrieved 2017-06-24.

Papert, S. (1970). Teaching Children Thinking, *World Conference on Computer Education, IFIPS,* Amsterdam.

Papert, S. & Solomon, C. (1970). Conceptual advances derived from the Muzzey experiment. (unpublished).

Papert, S. & Solomon, C. (1971). Twenty things to do with a computer. *Artificial Intelligence Memo* No. 248 and *Logo Memo* No. 3. Retrieved 2012-12-21

Pink, D. (2009). *Drive: The Surprising Truth About What Motivates Us*. Riverhead Press.

RoDI (2015). http://rodibot.github.io/ Retrieved 2017-06-24.

Scratch website (n.d.). Retrieved from http://scratch.mit.edu/

Smith, B., (2014). *A Quick Guide to GPLv3*. https://www.gnu.org/licenses/quick-guide-gplv3.html Retrieved 2017-06-24.

Stark, T.I. (2009). Book Review: Minsky's "The Emotion Machine: - Doing Strong AI Right? http://starkreal.blogspot.com/search?q=emotion+machine Retrieved 2017-06-24.

Stefanakis, E. (2002). Multiple Intelligences and Portfolios: A Window into the learner's Mind, Greenwood Press.

Sugar Abacus (2010). https://wiki.sugarlabs.org/go/Activities/Abacus Retrieved 2017-06-24.

Sugar Raspbian (2017). https://wiki.sugarlabs.org/go/Raspbian Retrieved 2017-06-24.

Sugar Toolkit (2017). https://github.com/sugarlabs/sugar-toolkit-gtk3/tree/master/src/sugar3/presence Retrieved 2017-06-24.

Sugar Turtle Blocks (2010). https://wiki.sugarlabs.org/go/Activities/Turtle_Art Referenced 2017-06-24.

Sugar Write (2008). http://wiki.laptop.org/go/Write Retrieved 2017-06-24.

Sugarizer (2017). http://sugarizer.org/ Retrieved 2017-06-24.

Trinidad, G. (2013). *Física con XO*. https://sites.google.com/site/solymar1fisica/fisica-con-xo-investigacion-/fisica-con-xo-el-libro Retrieved 2017-06-24.

Turtle Blocks (n.d.). https://walterbender.github.io/turtleblocksjs Retrieved 2017-06-24.

Urrea, C. and Bender, W. (2012). Making Learning Visible. *Mind, Brain, and Education*, 6: 227–241.

Zepf, M. (2013). Turtle Blocks Python Export Project. https://wiki.sugarlabs.org/go/Summer_of_Code/2013/Turtle_Blocks_Python_export_project Retrieved 2017-06-24.