“Free Libre OpenSource Software” as a catalyst for a shift to constructivism in Computer Science teaching

Andrea Trentini
Dipartimento di Informatica e Comunicazione, Università di Milano
Via Comelico 39, 20135 MILANO
andrea.trentini@unimi.it

Constructivism (by Piaget) describes the process of acquiring human knowledge as a continuous and iterative process of knowledge refining by interaction between an internal representation of the world and the direct experience of the world itself, helped by a “facilitator”. The facilitator can be a human being but also an environment, a context.
“Free, Libre and OpenSource Software” (FLOSS) is a software philosophy that strongly encourages interaction with the internals of any (free) system giving users formal rights to use the program, study it and propagate knowledge about it.
In teaching there are two main antithetic patterns: 1) “top-down” - traditional authoritative teaching, front lessons, noninteractive, students in passive mode; 2) “bottom-up” - laboratory mode, students very active, they are presented with the things they have to learn and they try and understand by themselves, with minimal contribution from the teacher. Many teachers are still using pattern #1.
Studies show that a mix of the two should be used to effectively bring students up & running fast and with enthusiasm.
This paper argues that, in Computer Science teaching, a FLOSS approach (both in terms of content taught and in method) is best suited to ease the shift from #1 to #2.
After a short chronology of “freedom” in the world of software and documentation this article will compare the two main teaching approaches (traditional and constructivist), analyze how FLOSS may help teaching and describe an object model explicitation to help students better understand their learning process.

1. Introduction

Teaching is not always easy. A teacher should combine many qualities... and knowledge is just one of them. The author works as assistant professor at the University of Milano (Italy), Department of Computer Science, and has taught topics such as Object Oriented Analysis & Design, Operating Systems, Programming for a few years (since 1996). During these years his teaching style has changed, a lot. If such a change can be considered positive will be left to the judgement of his students and colleagues of course, but some considerations and reflections can be gathered here to be evaluated by the reader.

Any teacher has probably started his career as a "traditional teacher": someone who sees himself as special, someone chosen to represent knowledge, to be a master among would-be adepts, an authority deserving respect... and so on. A "top-down" teacher, referring to the prevalent direction of interaction, who leaks knowledge to his students at his pace. But, soon, the very same teacher will face the difficulties of his role: preparation of course material, confronting many students (in Italy we may have classes with hundreds of them), exams, self motivation and students motivation, keeping up with the subject, many questions and, last but not least, trying not to loose authority over students.

Any student entering university has to face a completely new environment, a big switch from high school. He must learn a new way to study, he has more independence, he is less directed, he can self-allocate study resources, timeframes, exams order, etc. It's a suitable time to help him in his learning career, attitude and method. A traditional teacher would ignore this aspect and would probably try to keep the "distance" between himself and the students as big as possible, seeing them just as receptacles, knowledge receivers. Maybe there is a different approach.

The rationale of this article is that since some form of Constructivism-based teaching/learning (see Sections 2 and 5) is the most effective way to transfer knowledge while the Free Software philosophy (see Sections 3 and 4) implies constructivism, the conclusion is that FLOSS (and a FLOSS attitude) should be widely used in teaching. Roles (teacher, learner) are changing, authority on any freed topic is more evenly spread, no one can declare himself "ultimate expert" since information is public and can be used openly. The educational approach should change towards a more open and abstract system/object model, towards the reification of a meta model in the process. This article argues that the teaching process should constructively take into account this difference in object models: capitalize it by reifing the model during teaching and giving students the means to "learn to learn". It's a change in teaching style: in classroom the model should be explicitly explained and the teacher role should drift from "teacher of contents" to "teacher of methods" (with an explicit and dynamic object model). Of course, this meta-method is already exercised by any good teacher... this article wants to assess that FLOSS environments naturally ease this transition.

2. Learning and constructivism

One of the definitions of Constructivism is the following: "a theory of knowledge (epistemology) that argues that humans generate knowledge and meaning from an interaction between their experiences and their ideas." Jean Piaget [Piaget, 1999] formalized it at the end of the fifties, according to this theory every person grows knowledge through the processes of accommodation and assimilation. Accommodation is the mechanism by which failure leads to learning: when we expect some behavior from a system we are examinating and the expected behavior does not happen we say that we have failed, our mind failed, so that we need to accomodate (reframe) our model of the system. i.e. we refine our model on failures. Assimilation occurs when our mind model does not fail on the real system, in this case we are fixating our model more and more.

Constructivism is usually associated with teaching approaches promoting active learning, or learning by doing, by trial and error. Studies such as [Kim, 2005] found that in academic environments some form of constructivism influences positively the learning process, above all for the motivation injected into students. Another study [Dogru and Kalender, 2007] found that constructivism brings "better retention". Pure constructivism (i.e. without external intervention) is seen as not effective [Kirschner et al., 2006] since some form of help from a teacher or a "facilitator" is always needed. "According to the social constructivist approach, instructors have to adapt to the role of facilitators and not teachers" [Bauersfeld, 1995]. Some also argue that the responsibility of learning should reside increasingly with the learner [Von Glasersfeld, 1989]. And "Social constructivism" [Liu and Matthews, 2005] emphasizes the importance of the learner being actively involved in the learning process, unlike previous educational viewpoints where the responsibility rested with the instructor to teach and where the learner played a passive, receptive role. Although some criticism [Holloway, 1999] arose in the pedagogical community all the researchers agree on the fact that a good shift away from the traditional/top-down/from-pedestal teacher is needed towards a more modern and efficient new/bottom-up/social facilitator.

"A teacher tells, a facilitator asks; a teacher lectures from the front, a facilitator supports from the back; a teacher gives answers according to a set curriculum, a facilitator provides guidelines and creates the environment for the learner to arrive at his or her own conclusions; a teacher mostly gives a monologue, a facilitator is in continuous dialogue with the learners" [Rhodes and Bellamy, 1999].

Summarizing: traditional teaching (top-down) lacks in students motivation and participation while pure constructivism (bottom-up) can be dangerous if not guided because it can generate wrong models.

3. Freedom of software

Author's hope is that this section will no longer be required in the near future, as more and more people will already know and understand what Free Software is. Feel free to skip if you already know...

Free (as in "Freedom") Software (http://www.gnu.org/philosophy/free-sw.html) is a way of giving rights to users. Users of Free Software have the following "freedoms to...":

#0: run the program, for any purpose.

#1: study how the program works, and change it to suit the user. Access to the source code is a precondition for this.

#2: redistribute copies.

#3: distribute copies of modified versions to others. By doing this the whole community has a chance to benefit from changes. Access to the source code is a precondition for this.

The two interesting freedoms for the purpose of this article are #0 and #1.

#0 lets the user run the program any times he wants, in any condition (purpose), so that the program can be tested at will, thus fully reverse engineered (if there is no interest for the source code) without any legal limitation. By contrast, there are many proprietary EULAs (End User Licence Agreement) that explicitly prohibit reverse engineering practices (http://wikipedia.org/wiki/Software_license_agreement#Reverse_engineering) or even study the performance of a program.

#1 lets the user study the internals of the program without even running it, i.e. by reading the source code, thus giving the user the ability to fully analyze it and to "construct" an internal representation in his mind (more in Section 5).

The fourth (#3) freedom is also important because of motivation. The possibility to modify the program and then redistribute it is a good incentive, albeit the reward is just a slice of fame and not money.

4. Object model

In computer programming the term "object model" [Fowler, 1997] has a well defined meaning, here we will use it in a lighter way, i.e.: "a collection of objects representing a system to be described in an abstract way".

From the educational approach point of view, there's an interesting distinction in "object model" between open and closed products: 1) different licensing implies a different set of objects; 2) matching names in the two sets may have different meanings or importance, e.g., the "manual" in open products may be more outdated than in the case of closed software. For the sake of this article it is enough to define a list of objects related to the context of computer science education with a minimal description, because the interesting aspects are the differences between the world of proprietary software and the free one, differences in: meaning and source of authority (battled between the producer and the users).

The following table is not complete of course, it is just a "start of discussion", it is also not related to a particular product, the author tried to think as generally as he could. Also, for the sake of brevity, only one aspect has been emphasized in form of a keyword for every item, but many objects deserve a more thorough discussion.

object	*proprietary*	*free/libre*
author	not known	known
distributor	profit	"nonprofit"
version	marketing	functionality
source code	unavailable	available
license	**restrictions**	**rights**
course	certification	knowledge
manual	elegant	outdated...
knowledge base	customer support	user based solutions
(publish a) bug	**criminal!(1)**	**thank you!(2)**
bug tracking system	unavailable	public
price	>0	=0
distributor	profit	nonprofit

1) http://lwn.net/Articles/368861
2) "given enough eyeballs, all bugs are shallow" [Raymond, 2001]

There is one last object in the model that is only somewhat part of the model itself, it is an aspect of many items listed above: **authority**. The questions are: "Who has authority over a particular object?", "Where this authority comes from?", "Can this authority be trusted?". We may define authority such as an entity accepted as a source of reliable information on a subject. In libre software can be anyone, anyone can study a product and prove to be expert by demonstrating it in the battlefield, his/her knowledge is always verifiable (remember you can study the source code!). There is a good example of this attitude in the Hacker ethics [Levy, 1994]: "Hackers should be judged by their hacking, not bogus criteria such as degrees, age, race or position.". In proprietary products can be almost anyone, BUT his/her authority derives directly from some kind of certification by the owner/producer/distributor and his knowledge can be verified up to a certain border: the industrial secret border - anything that resides beyond this border cannot be verified (maybe partially through reverse engineering) and must be taken from granted, you must believe the product owner - a very famous example is the case of Windows suspected of sending private data back to MicroSoft servers, users unaware... if a MicroSoft expert/teacher is asked about this "feature" he will tell you the "official truth" MicroSoft has told him, so you will have to trust the complete chain of information.

As you can see, in the case of FLOSS authority is questionable, verifiable, dynamic while in the case of proprietary software authority can only be trusted, the only choice you have is whether to use the product or not. Of course this is an exaggeration, an extremization, but it summarizes the opposite directions, trends, tendencies of FLOSS and proprietary: FLOSS is based on verification, proprietary on "secretization".

5. Constructivism and free software

Following the definition of constructivism given in Section 2 we may depict a schematic diagram of the learning process, in Figure 1 and in Figure 2 the reader will find the author's interpretation. The goal of knowledge is to create a mind map of a "thing" (a physical object, a software product, anything) we want to know so that we may reason about it, expect some kind of behavior and in general usefully interact with it.

On the left side of the diagrams there is an "internal representation": a set of abstract objects that we can play with in our brain, this set of objects should mimic the real thing as closely as possible if we want to know the real system. The curved arrows on both figures represent the possible interactions/connections between the mind map and the real system.

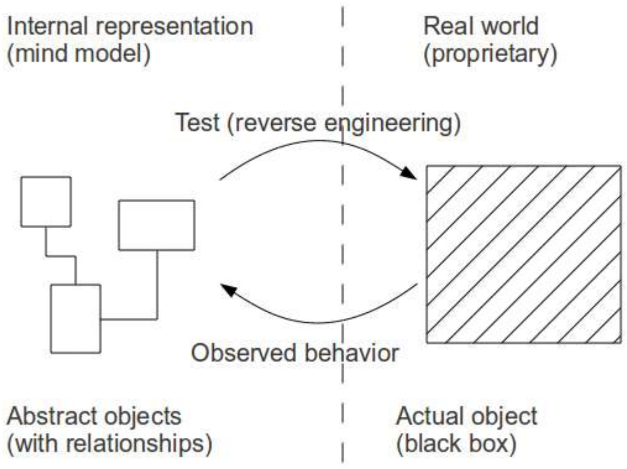

Figure 1: Constructivism: proprietary context

On the right side of Figure 1 is depicted a black-box object, since in the proprietary world no internal details are available. The only way to know a black box-object is by stressing/testing it, i.e. reverse engineer [Chikofsky and Cross II, 1990] it, by trial and error: your mind, based on the current abstract map, hypothesizes an expected behavior so that by actually trying it on the real object you can verify your mind's hypothesis. By iterating this cycle many times your mind should at last create a satisfactory internal representation of the object. How many iterations are necessary to reach a correct model (if a correct model can even be reached...)? Probably not many for simple objects (e.g. a pair of scissors), but what about a complex system such as a word processor or a programming language compiler? The reader may argue that some kind of documentation (user manual, technical manual) should be available in complex cases. Of course, but who can check document correctness in a proprietary context?

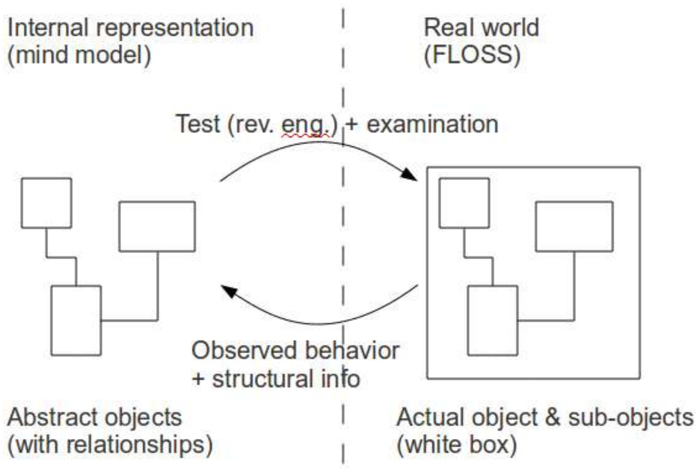

Figure 2: Constructivism: FLOSS context

On the right side of Figure 2 is depicted a white-box object, since in the FLOSS world all internal details are available. You may still use a reverse engineering approach to create your mind map, but at any time you can study the source code to actually see the real object structure. It's up to the learner. Of course any teacher can propose his own knowledge model, but he will have to let students compare it to the model they can create by themselves while freely studying the inner details of the object (software system) under examination.

An example of this attitude, closely tied to the teaching of Java, happened to the author. One day a student asked about the internals of the java.lang.Vector class (sources of all library classes are available), the author had only a partial answer. In a proprietary context a smart teacher could have packaged a FUD (Fear, Uncertainty & Doubt) response, pretty sure about the impossibility to be contradicted. The FLOSS attitude led the author to the explicitation of the whole process: formulate a behavioral hypothesis (the add() method execution speed is not constant, from time to time it takes longer because the inner array has to be cloned), test it black-box and also check it in the source code. Students were impressed and they demonstrated afterwards to have learned the lesson (method).

6. Method proposal

"Do not believe what I say." This is the first phrase the author pronounces at the beginning of the first lesson of every course. It is meant to shock students out of their prejudices about professors and encourage them to verify information. The goal is to lower the "authority gap" to help participate in the learning process. The next thing they hear is: "Please do make questions! The worst thing that can happen is that I don't know the answer... but I'll have the meta-answer!", i.e. the teacher presents himself as a facilitator, one who can help in the constructivist process of learning. Then they are presented with the differences between proprietary and FLOSS worlds and they are explained why they will be offered to study mainly FLOSS artifacts (motivations in Section 5). Then the course flows in what may appear as traditional (front lessons, website for materials, etc.) but it is not.

Front lessons in classroom are supported by the use of a collaborative editor (gobby - http://gobby.0x539.de) where any student can edit the very same source code the teacher is showing, and in fact they do, a lot, even anticipating what the teacher is explaining (sometimes the author has to stop them by voice because he wants to explain the whole sequence of common mistakes in programming before getting to the correct solution).

The course website is in fact a wiki (http://en.wikipedia.org/wiki/Wiki) so that any student (even anonymously) can edit, add, remove, pages. They are also encouraged to interact with the teacher (but also among them) by being offered many communication channels, such as email (of course), chat (Msn, Facebook, skype), microblogging (twitter, identi.ca). The choice of many "modern" channels tries to lower the "energy barrier", so that everyone can participate, the hope is that any student can find his suitable way of interaction. And they do: some of them use classical email, some twitter, some chat, some ask during/after lesson, etc.

Please be aware that this is not a (criticized [Kirschner et al., 2006]) "minimal guidance" or "pure discovery-based teaching technique" bottom-up (see Section 2) approach since students are not "left by themselves to try and understand", they are well guided, they are protected from "misconceptions or incomplete or disorganized knowledge"; author's goal is to let them be as active (in the learning process) as they can, inside a coherent knowledge framework.

Is there any problem with this approach? Yes of course. Teachers must jump down from pedestal and let students participate, somewhat loosing their "given" authority while deserving it "in the field" (i.e. proving to be a reliable source). Moreover, in the FLOSS world knowledge runs fast [Trentini, 2009], so the new teacher must keep up with the pace or his students will rapidly overcome his knowledge and he will loose respect, i.e. he must run (metaphorically) to remain in the same place.

Bibliography

[Bauersfeld, 1995] H. Bauersfeld. Constructivism in Education, chapter The Structuring of the Structures: Development and Function of Mathematizing as a Social Practice. Lawrence Erlbaum Associates Publishers, 1995.

[Chikofsky and Cross, 1990] E. Chikofsky and J. Cross II. Reverse engineering and design recovery: A taxonomy. IEEE software, pages 13–17, 1990. ISSN 0740-7459.

[Dogru and Kalender, 2007] M. Dogru and S. Kalender. Applying the Subject "Cell" through Constructivist Approach during Science Lessons and the Teacher. Journal of Environmental & Science Education, page 11, 2007. ISSN 1306-3065.

[Fowler, 1997] M. Fowler. Analysis Patterns. Addison-Wesley, Boston, 1997. ISBN 0201895420.

[Holloway, 1999] J. Holloway. Caution: constructivism ahead. Educational Leadership, 57(3):85–86, 1999.

[Kim, 2005] J. Kim. The effects of a constructivist teaching approach on student academic achievement, self-concept, and learning strategies. Asia Pacific Education Review, 6(1):7–19, 2005. ISSN 1598-1037.

[Kirschner et al., 2006] P. Kirschner, J. Sweller, and R. Clark. Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. Educational psychologist, 41(2):75–86, 2006. ISSN 0046-1520.

[Levy, 1994] S. Levy. Hackers: Heroes of the Computer Revolution. Dell Publishing Co., Inc., New York, NY, USA, 1994. ISBN 0385312105.

[Liu and Matthews, 2005] C. Liu and R. Matthews. Vygotsky's philosophy: Constructivism and its criticisms examined. Published by Shannon Research Press Adelaide, South Australia ISSN 1443-1475 http://iej. cjb. net, 6(3):386–399, 2005. ISSN 1443-1475.

[Piaget, 1999] J. Piaget. The psychology of intelligence. Routledge, 1999. ISBN 0203981529.

[Raymond, 2001] E. S. Raymond. The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. O'Reilly Media, Inc., revised & expanded ed. edition, 2001. ISBN 0596001088.

[Rhodes and Bellamy, 1999] L. Rhodes and G. Bellamy. Choices and Consequences in the Renewal of Teacher Education. Journal of Teacher Education, 50(1):17–18, 1999.

[Trentini, 2009] A. Trentini. The Borg "architecture" as a metaphor for FLOSS. In IADIS International Conference Applied Computing, Rome (IT), Nov 2009. IADIS.

[Glasersfeld, 1989] E. Von Glasersfeld. Cognition, construction of knowledge, and teaching. Synthese, 80(1):121–140, 1989. ISSN 0039-7857.