Three years of teaching using collaborative tools: patterns and lessons learned

Andrea Trentini
Department of Computer Science, Università di Milano, via Comelico 39, Milano, ITALY
andrea.trentini@unimi.it

Keywords: Programming Learning, Collaborative Editing, Wiki, Flip Teaching

Abstract: The author has taught computer science (Programming 101 and Operating Systems 101) for about fifteen years. He introduced the use of a student-collaborated wiki website for his courses about ten years ago. Then, three years ago, he also began extensively using a collaborative editor (Gobby) in classroom, to let students actively participate during lessons. This paper describes the author's course "workflow", summarizes tools (wiki and collaborative editor) functionalities, collects some context pattern and tries to draw a few conclusions (lessons learned) about the course methodology.

1 INTRODUCTION

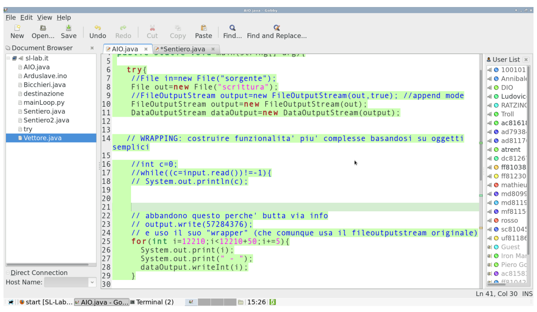
The author of this paper is currently teaching Computer Programming 101 at an italian University. He has taught alternatively Computer Programming and Operating Systems for more than fifteen years. He is also the founder of a Laboratory on Free Software.

The Programming 101 course is organized using a fairly standard structure: 1) traditional "frontal" lesson time, with blackboard, slides and projector and 2) traditional (before introducing the system described here) lab time, with a PC for every student, a programming environment with online documentation and... blackboard, slides, and projector again. We use GNU/Linux as our main operating system so that our students, mainly coming from previous (if any) experiences with Windows or Mac, are immersed in a different and unfamiliar environment, in the (often vain, alas) hope to wake them and spark some curiosity about the internals of an operating system and about programming besides windows, mouses and the damages of autocompletion. We adopted Java for the Programming 101 course even if there has always been some debate over using an Object Oriented language as the first language [(Clark et al., 1998), (Hadjerrouit, 1998), (Hosch, 1996) and (Pears et al., 2007)], we are trying to introduce advanced language concepts as soon as possible. Our course is a bit crowded (just less than 200 students) so we usually split labs into three turns. Classrooms are standard: about 50 PCs, GNU/Linux, (almost full) network access, overhead projectors and a PA audio system. Since the beginning of his work in teaching, the author (inspired by the F/OSS (Free/OpenSourceSoftware) communities where collaboration is the main "tool" to achieve a goal) started using collaborative tools to let students participate in the learning process. The first tool was a **wiki** (description in 2) website. Then, about three years ago (academic years: 10-11, 11-12, 12-13), the author introduced and began extensively using a **collaborative editor** (Gobby, more in 2) in classroom, during lessons.

2 TOOLS

A **wiki**(-website)[1] is a website that can be modified by its readers/users. Wikipedia is the most famous one of course, but there are probably millions of almost unknown wikis scattered through the Internet. A wiki is a convenient way of sharing knowledge in a community: any member of the community can add/modify/delete/upload/etc. the set of available pages, thus augmenting, refining, ameliorating the knowledge base of the community itself. A wiki website can be set up in a matter of minutes, there are versions written in almost every language ever invented on earth[2]. Editing is done using a very simple and easy to learn tagged language, sometimes resembling HTML. Software policies allow the wiki maintainer to tune edit permissions, i.e. giving read/write/etc.

[1]http://en.wikipedia.org/wiki/Wiki
[2]http://en.wikipedia.org/wiki/List_of_wiki_software



```java

  try{
   //File in=new File("sorgente");
   File out=new File("scrittura");
   //FileOutputStream output=new FileOutputStream(out,true); //append mode
   FileOutputStream output=new FileOutputStream(out);
   DataOutputStream dataOutput=new DataOutputStream(output);


  // WRAPPING: costruire funzionalita' piu' complesse basandosi su oggetti
semplici

   //int c=0;
   //while((c=input.read())!=-1){
   // System.out.println(c);



   // abbandono questo perche' butta via info
   // output.write(57284376);
   // e uso il suo "wrapper" (che comunque usa il fileoutputstream originale)
   for(int i=12210;i<12210+50;i+=5){
     System.out.print(i);
     System.out.print(" - ");
     dataOutput.writeInt(i);
   }

```

Figure 1: The Gobby editor.

rights to users. In the author's case, his course wiki is configured to let any student contribute upon registration. For the sake of completeness the author's choice for wiki software goes to DokuWiki[3].

The **Gobby**[4] editor (see Figure 1) is a "collaborative" editor. The word "collaborative" means that this editor can be simultaneously used by a group of people, potentially located far away from one another, to edit files together, everyone seeing and interacting on each other's work. For a group to work togehter someone must install a server that must be reachable by everybody through the Internet. Then every member of the group must connect to that server and, if the connection does not fail, a new user (a self-assigned unique nickname is required) appears in the "user list" of every Gobby client connected to that server. Users are assigned a unique colour because every edit action is colour marked. Selection actions of other users are shown, so that any user can have a look at "what others are looking at or doing". When switching to a particular file, on the righthandside of the Gobby user interface there is a list of the users currently editing that file, the file-users association is many-to-many. The main editing area is in the centre while on the left there is a list of available servers and open documents. On the bottom of the window a chat area (not shown in Figure 1) can be activated, it is mostly used during long distance session to coordinate editing ("out of band" communication). Any user can locally save any file at any moment, this is a perfect fit for a collaborative programming environment since anybody can save and compile on its own to verify errors or the correct execution of the source code of the program under examination. There is also a recently introduced *undo* function. Gobby supports language syntax colour coding. The current version does not support any kind of formal coordination, i.e. different roles in a group of users, for example to manage some kind of "ask permission to edit", "edit approval", etc. In classroom, the author's workflow goes like this: 1) launch the server; 2) tell everybody the *url* to connect to and wait for them to complete; 3) start editing a source file; 4) speak while showing various programming examples; 5) let the students add/modify the code **guiding** them by voice (see 3).

[3]https://www.dokuwiki.org

[4]http://gobby.0x539.de

Alternatives: the previous (pre 0.5) versions of Gobby were far form perfect so at one point we had some brief experience with "google docs/google drive" which is a cloud application server enabling users to create and share various types of documents, e.g. formatted text documents, spreadsheets, presentations, etc. The overall functionality is almost the same (usernames, colours, multiple simultaneous editing, etc.) but the main disadvantages are that: 1) it is oriented towards **formatted** documents[5]; 2) every piece of information is managed by Google while the infinoted server is installed locally. Also, some colleagues use *git*[6] as a collaborative platform, but in that case the collaboration is not interactive.

3 PATTERNS

First, a brief note about our course **wiki** usage during these ten years, then the rest of this section will delve into some analysis about collaborative editing patterns. Alas, the wiki is in fact almost only used as a conventional website. I.e. in terms of contributions there are very few students that "dare" to edit what the teacher puts online. And this trend is worsening: while a few years ago there were at least a couple of students for each course who edited and contributed, this year the number of editors is still zero (they haven't even registered), and we are at 75% course completion. There are too many factors involved to draw some conclusion about the cause(s) of this decrement, one of them could of course be the diminished ability of the teacher to spread enthusiasm among the students... some quantitative study should be undertaken before worrying. On the other hand, the usage of **Gobby** in the classroom has got a foothold among the students, there are always at least five of them (on an average attendance of 25[7]) actively interacting with the author. This wide appreciation has led the author to a fair grade of usage

[5]They can be exported to a text only file (loosing comments and any metatext) and there is no language syntax color coding

[6]http://git-scm.com/ a distributed version control system created by Linus Torvalds

[7]We do not oblige students to attend lessons and it is common to all courses to observe a natural decay in attendance: from 100% for the first couple of lessons to the average 50% for the rest of the course

standardization (“routine”) and to the identification of a few **patterns** described in the following sections.

=> **Pattern: “think ahead”**
They in fact interact so much that the author must tell them to wait before writing some final snippet of code because he wants to show them every step, even wrong ones. They are sometimes thinking ahead of the teacher, at least in terms of the final solution to a problem, but since this is a programming course they must be exposed also to common mistakes.

=> **Pattern: “leave traces for future reference”**
To keep every snippet of code, even wrong (or just worse) ones we decided to leave every significant code statement inside the current example just by commenting out the ones we do not want to be compiled. The following is a very small example of the way we organize examples by leaving common mistakes in the code itself:

```
// this is done right
Random r=new Random();
for(..;..;..) {
 ...
 r.nextInt();
 ...}
// this one stresses the GC
/* for(..;..;..) {
 Random r=new Random();
 ...
 r.nextInt();
 ...} */
```

The second half is commented out using standard Java syntax: “//” and “/* .. */”. This way they can study every possible solution in a single compilable example code. It is a kind of *poor man’s version control*: by commenting and decommenting lines students can experiment with different “versions”. It may also be useful to have some kind of visual branching tool to show different solutions to the same problem with different code snippets, but it is the author opinion that it would also complicate too much the interface of Gobby itself.

=> **Pattern: “out-of-band... inline”**
Instead of using the (out-of-band[8]) chat to coordinate and to ask questions, we ended up inlining questions and comments using... code comments (again). While the author is writing - and explaining/commenting by voice - some example code, students prepare questions and comment simultaneously by using standard Java comments. So that any answer, be it in form of a code snippet (thus in the code and compilable) or text (thus in a following comment) remains strongly attached to the code that stimulated the question itself. By the way, at the end of every session all the code created during the lesson is usually uploaded to the wiki.

=> **Pattern: “emergent syntax”**
Sometimes there is a need to communicate “graphically” and, since Gobby does not support anything other than text we have to resort to the good old ASCII art[9], albeit very simple and naive, like:

```
// <<<===  see other doc in the list!
```

Or:

```
Stringhe di comando
|-D(igital)
|       \
|        |-I-pinNumber [ritorna valore]
|        |-O-pinNumber-H/L
|-A(nalog)
|       \
|        |-I-pinNumber [ritorna valore]
|        |-O-pinNumber-value
|-V(ersion) [ritorna valore]
|-NP(umber of pins)  [ritorna valore]
```

Or:

```
SCHEMA
(Client)                 (Server)
Socket               other Socket
|-InputStream  <= OutputStream-|
|-OutputStream =>  InputStream-|
```

Everything is always left in the code (thus in the wiki) for future reference. In one single compilable file students can find examples, comments, questions, answers, common mistakes, different solutions to the same problem, and they participated in building that content.

=> **Pattern: “pillory demythified”**
Sometimes the author gives an exercise in the classroom to be solved by the students themselves. When a resonable (to complete the exercise) amount of time has passed the author *jokingly* asks if any of them wants to be “put in the pillory” by copying & pasting their code in public, reassuring them that if there are mistakes it will be a better chance for everybody to learn. And it works! There is usually no need to force anybody to “volunteer” their code, there is always one or two of them willing to show their work. Humour is a good way to lower their fear. And using their code instead of the teacher’s is a better chance to discuss about different solutions and about common mistakes, and the ones who learn more are the ones offering their code since they are more involved of course.

=> **Pattern: “correct as the teacher speaks”**
The author frequently introduces common programming mistakes and asks the students to find them. Then he leaves the keyboard and starts speaking and explaining (maybe pointing at something on the projected screen), trying to lead the classroom towards the correct solution. Usually, while the teacher is

[8]http://en.wikipedia.org/wiki/Out-of-band

[9]http://en.wikipedia.org/wiki/ASCII_art

speaking they correct the errors (leaving, i.e. commenting out, wrong pieces of code for future reference) on the fly so that the visual effect (just looking at the projector) is similar to a vocal interaction between the teacher and a programming environment. The advantage in this case is that the teacher has good feedback on the students understanding level. Why? Because the speed and quality of their corrections is an immediate and direct measure of their ability to discuss about programming Moreover the author believes they grasp better since they can interact with the ongoing reasoning.
=> **Pattern: "humorous relief"** When in need of a coffee break they comment in the code:

```
// COFFEE!!!
```

And the teacher usually may comment like this:

```
// in five minutes, let me finish this example
```

I.e. there is a lowered authorithy gap while writing, people usually dare a little more in written form (such as in chats or in this collaborative editing case) than in person.

4 CONCLUSIONS

The first lesson learned by the author is that "fully opening" the teaching loop to students intervention completely changes the way teaching must be addressed: since there cannot be strictly fixed and prepared learning material (just a general lesson goal that must be achieved) the teacher must be much more flexible and prepared (e.g. the number of questions raised is far higher than during a standard lab lesson) to any path the students are willing to follow. With limitations of course: the teacher must be very good at leading them where he wants to bring them. The teacher must earn his authority in the field - "the hackers way" (Levy, 2001) - his ability and knowledge is put more to the test in this context. The teacher must also be more open to suggestions, needs, curiosities and of course criticisms raised by the students. And this approach stimulates more curiosity: students can become very knowledge greedy. The most important and useful habit the author has acquired during these years is the "declaration of methodology", the complete disclosure of the approach used, even at the meta level: i.e. the author explains in classroom almost every pattern described in this paper and discusses with the students any form of possible amelioration of the teacher-student interaction.

The approach described (and also proposed) here is a very light kind of "flip teaching" since our students are still exposed first to traditional theory and next they have to apply the theory during a very interactive lab. This way of teaching is at the opposite end from the traditional (blackboard and slides) lesson and it also represents, according to the author of course, a progress with respect to the traditional lab lesson (in which the teacher "gives work" and then circulates among students to help them singularly) because it does stimulate interaction (as in a standard lab) but it also **shares/spread** this interaction among all the students in the classroom. Other interesting learning patterns can be found in (Flood and Lockhart, 2005) and (Hickey et al., 2005), some of them based on student debate, they also could be applied through gobby, the author will try to bring the most useful ones in his classroom. This approach is lightly coupled to any particular programming language, for patterns more related to programming concepts the interested reader may start from (Gomez-Albarran, 2005), a survey of tools for learning (visualization, animation, etc.).

REFERENCES

Clark, D., MacNish, C., and Royle, G. F. (1998). Java as a teaching language - opportunities, pitfalls and solutions. In *Proceedings of the 3rd Australasian conference on Computer science education*, ACSE '98, pages 173–179, New York, NY, USA. ACM.

Flood, R. and Lockhart, B. (2005). Teaching programming collaboratively. *SIGCSE Bull.*, 37(3):321–324.

Gomez-Albarran, M. (2005). The teaching and learning of programming: A survey of supporting software tools. *The Computer Journal*, 48(2):130–144.

Hadjerrouit, S. (1998). Java as first programming language: a critical evaluation. *SIGCSE Bull.*, 30(2):43–47.

Hickey, T. J., Langton, J., and Alterman, R. (2005). Enhancing cs programming lab courses using collaborative editors. *J. Comput. Sci. Coll.*, 20(3):157–167.

Hosch, F. (1996). Java as a first language: an evaluation. *SIGCSE Bull.*, 28(3):45–50.

Levy, S. (2001). *Hackers: Heroes of the computer revolution*, volume 4. Penguin Books New York.

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M., and Paterson, J. (2007). A survey of literature on the teaching of introductory programming. In *Working group reports on ITiCSE on Innovation and technology in computer science education*, ITiCSE-WGR '07, pages 204–223, New York, NY, USA. ACM.