Henri Lesourd

Sketch of a research & teaching project

Application to fill a position as

Professor in Open Source Software

at the Computer Science Department in the Faculty of Technical Engineering
University of Erlangen-Nuremberg, Germany

1 Manifesto

I decided to do research in computer science because I deem computers should be easier to *understand*, and easier to *drive*, too. In my masters and PhD studies, I have thus been involved in devising tools to explain the behaviour of the software running inside a UNIX system [LES02], along with applying AI techniques to the domain of scripting the UNIX system [ETZ93][LES95].

During this endeavour, I discovered that computer systems could be much more versatile, especially, one can design environments where the actions of investigating and modifying the running programs are easier to perform, thus allowing more users to practice and to *learn* these tasks [SMA76][ROS99].

This led me to gradually pay more attention to the fact that writing software is an inherently *social* activity, and to notice that such phenomenons have been studied relatively early in the history of computing [MCK90].

Thus, it changed my perspective, for in such a context, the question of how a given community becomes able to perform a given activity, to collectively ponder and understand realities, and how finally, as a group, such a community becomes able to increase its own *collective intelligence*, this is a question of paramount importance.

Afterwards, I carried on a postdoc as researcher in the Ωmega project, which is an ambitious project[1] aimed at prototyping the desktop & internetworked tools of the mathematicians and math students of the future.

During this postdoc, I developed the first prototypes of a middleware-based, user-friendly user interface to the intelligent mathematical tools my colleagues and I were working on. This work led to peer-reviewed publications[2], and later, it was further developed in a PhD project[3], which I helped to appropriately kickstart.

I now have a better view of how the internet is creating a kind of revolution by means of enabling sharing information at an unprecedented scale. I also perceive some important obstacles to this revolution: basically, the Internet is a patchwork of badly integrated technologies, which, although extraordinary useful, remains difficult to use.

There is a continuum between pure technology on the one hand, and on the other hand, its many uses, along with how it can be perceived and understood by its users. Overall, because its principles allow information to flow freely, the Open Source model appears as an excellent approach to create a technology which fulfills very well the needs of its users: but in practice, the complexity of the engineering often remains a challenge for the developers, and then the resulting excessive complexity of the software becomes a problem for the users. Thus, the success of the Open Source model appears currently incomplete.

---

1. `http://www.ags.uni-sb.de/~omega`. The underlying SFB 378 research network was established in 1996.

2. Serge Autexier, Christoph Benzmüller, Armin Fiedler, Henri Lesourd. Integrating Proof Assistants as Reasoning and Verification Tools into a Scientific WYSIWIG Editor, In *David Aspinall and Christoph Lüth (Ed) Proceedings of UITP'05, ENTCS*, January, 2006.

3. The *Plato* proof assistant: `http://www.ags.uni-sb.de/~omega/omega/index.php?target=plato`

Finally, I would like to raise, and look up answers to the following questions:

i. from a *technological* point of view: how can one promote the use of better tools and practices in the Open Source community ?

ii. from a *cultural* point of view: how can one enable a successful communication between groups of people having different cultures, attitudes and skills ?

Of course, the insights obtained in such inquiries could be used to create new Open Source projects and/or to contribute to, or to advise the members of existing Open Source projects.

2 Experience in Open Source Software

Along the course of my postdoc, I also had to develop and to contribute the appropriate extensions[4] to the GNU TEXMACS scientific editor, which was our chosen content editor. I thus gained a valuable experience in the inner workings of complex, object-oriented software and user interfaces (document rendering and wysiwyg edition, GUI), in a C++/X11 context. I also enjoyed very much being involved in an OSS project and with people in the Free Software community.

Finally, my experience in developing the TEXMACS software led me to become more and more aware of the problems related to the use of an *extension language*. Namely, extension languages tend to be a late add-on to an already existing software, in order to be able to *customize*, or to *script* the software. Once such an extension language exists, it tends to be used to write more and more important parts of the software (extension languages are usually dynamic and very convenient, e.g. one can directly modify the system while it is running, etc.). But this tends to raise problems, because of the limitation of current extension languages. To address this extension language problem in software engineering, I started the ABEL[5] project in 2007. My current activities revolve around the design of the related environment, and of applications built on top of this environment (i.e. a programmable browser for developing semantic web applications). Once the design will be complete, and the implementation sufficiently advanced, I plan to recruit programmers and to organize a collective work to develop the project further.

4. Especially, a vector graphics renderer and integrated editor inside TEXMACS, the correct implementation of a full-screen mode for doing presentations, along with discussing issues and refining APIs, hunting various bugs, supporting the TEXMACS user community (several tutorials, answering questions on the mailing list), and presenting TEXMACS at various Free Software conferences. GNU TEXMACS is written in C++ directly on top of X11.

5. A Better Extension Language (ABEL): `http://www.nongnu.org/abel`

3 Research interests

I dream of a day when egoism will not rule science anymore, of a day when we will join forces to study. And instead of sending closed letters to academies, we will hurry to publish our least observations, as soon as they are new, and we'll say: "I don't know the rest" – Evariste Galois

In any given society, there would be no art (and neither technology, nor science as well) without the patient, tireless and collective improvement of a widely shared corpus of knowledge.

In this respect, widely available, network-mediated communications hold the promise of an unprecedented rise in the availability of knowledge and in its speed of evolution.

There are in fact many communities sharing different kinds of informations thru the medium of (possibly distributed) software tools: communities of hackers, but also scientific communities, communities of artists.

How we use our tools has an influence on the surrounding organization and on its culture; on the other hand, this organization has an influence on how we use and evolve the tools. Moreover, the use of computers in a group has been characterized as co-adaptative phenomenon [MCK90], as a *situated activity* [SUC87][LES02], and thus, continuously co-adapting the software seems to be an inherent part of the human use of a computer[6].

I could focus my energies on the following main areas:

i. *New tools for developing technological content* (e.g. mathematics, software, music): these developments are often extremely expensive, because it takes an unusual lot of time (several years) to achieve the required quality. Developing such tools is often considered as not being first-class research, thus there is few funding available. As a consequence of that, either the tools are developed following proprietary business models (at a huge cost for universities and public institutions later on), or even worse, they are never developed.

   A sensible approach to this problem would be trying to identify a common, cross-domain architecture, and then develop (from scratch or from an already existing development) the more important parts of this architecture (e.g. reconfigurable authoring tool, versioning, etc.) one after the other ;

ii. Programmability, fostering data integration and reuse in science: programmability and data integration remains an important concern in many areas of sciences, e.g. in the TEXMACS user's community, which I know quite well, many researchers (e.g. in physics) use TEXMACS as an *integrator*, i.e. as a front end to various heterogeneous tools (e.g., Computer Algebra Systems), with the ability to

6. In this respect, it should be noted that this need for the programmability of computers by their users is precisely one of the roots of the Free Software movement.

integrate the output of all these tools in one document. At Institut Pasteur, research has been carried out along related lines, to foster *end-user programming* for bioinformaticians[7].

As such, the use of programming is recognized as a genuine topic in arts as well (in the graphic arts community, e.g. the Processing[8] software is of interest, along with software like Pure Data[9], developed in the avant-garde and/or electronic music communities).

In all these instances we are in front of domains made of a « patchwork of standards », such standards having often been created independently. There is currently no easy solution for data integration, nor for gluing together different components, and bridging the gap between heterogeneous tools: in any case, a script has to be written to translate from one data format to another, and agreement on common formats is often slow (in Computer Algebra Systems (CAS), there are standards like OpenMath, but even today, several major CAS do not implement it; in theorem provers, it's only recently that e.g., an important proportion of TPTP's provers effectively implemented a common proof syntax).

Identifying and implementing good architectures for heterogeneous, distributed component architectures is an interesting issue. Devising efficient ways to implement bridges between tools without depending on the implementors of these tools would be an important progress ;

iii. *Advanced tools for programming*: proving software, test & debugging, integrating documentation and code annotation are also topics of interest, all the more because such tools are not always used much in OSS projects ;

iv. *Techniques for summarizing technological content*: often in technological content, one only need to consult *subsets* of the data (also called *views*) to perform a given task. There is a need for more principled ways to extract views of a data repository, especially we would like to be able to appreciate *how much information* contains the view, if some coherence properties, or dependencies are preserved or not, etc. To our knowledge, to a certain extent, such research has been conducted in particular domains e.g. in ontologies[10], but not in other domains, especially not in non formal domains ;

7. C. Letondal (2005). *Participatory Programming: Developing programmable bioinformatics tools for end-users*. In H. Lieberman, F. Paterno, & V. Wulf (Eds.), End-User Development. Springer/Kluwer Academic Publishers.

8. `http://processing.org`

9. `http://puredata.info`

10. Cuenca Grau, Bernardo, Horrocks, Ian, Kazakov, Yevgeny and Sattler, Ulrike: *Extracting Modules from Ontologies: Theory and Practice*, Oxford University Computing Lab., 2007

4 Teaching project

To me, teaching computer science is generally the study of a system, hardware and software, plus possibly a social network & shared culture revolving around these tools. In this respect, it is important to transmit the *philosophy* of the given system, and to show why this particular design is important. From that, it follows that computer science cannot be taught without a minimal perspective on *history*, which is as a matter of fact very useful to show how much subjective the concept of « innovation » in the current industry should be considered: in fact, most of the discoveries and important designs have been devised more than twenty years ago. Once people understand this, it becomes much easier for them to grasp the real degree of innovation of a modern design and/or to position it in the state of the art.

Another important topic is the *social and economic aspects* related to the OSS movement; firstly the Free Software movement [GNU85], which is in fact the original starting point of all these developments. More widely, free markets and open societies [POP44] are an important surrounding element without which OSS can't be properly understood. Finally, a knowledge of other works about the sociological influence of computer systems is in order, e.g. [WEI76].

Following the study of the history, one should also study the different OSS *business models* which have been successful in the past, e.g. Emacs, Cygnus, Ghostview, Qt, Redhat & Ubuntu, etc.

In this respect, it is also interesting to *consider certain important software systems from an OSS perspective*, e.g. Smalltalk, MS-Windows: what is the reason of the limited success of the first one, and of the wide success of the latter one ? What can we say about the software engineering tradition in these two environments ? Of the destiny of the *ideas* stemming from each one of these two experiments ? What are the possible lessons we can derive to devise new successful OSS projects in the future ?

Of course, the students should *master programming & software engineering*, possess a *genuine understanding of hardware* and how a computer operates. They also should have a knowledge of the traditional tools which are used in OSS projects, along with more recent ones (e.g., git, sparse, Eclipse, etc.).

Finally, *launching and/or contributing to an OSS project* should be considered as an important milestone in their curriculum. Such practical works should start by a creative analysis of the current landscape, which leads to identifying some interesting opportunities, some things which are not done, or not well done in the current OSS ecosystem. Once the project has been defined, its realisation should be carried on, which includes devising an initial design & implementation, plus a set of rules and a public relations approach to manage and develop the project.

Why is this teaching project better ? Because, in my opinion:

- when undergraduates reach the bachelor level, they should definitely *know the basics*: how hardware operates, the importance of tests, how to develop a program incrementally. It is my experience that too often nowadays, novice students tend to be exposed immediately to rather abstruse methodologies (e.g. UML, heavily hammered Object Oriented Programming), and then they learn the use of big frameworks (e.g. JavaBeans, .NET), which enable them to develop quick & dirty programs of considerable size, sometimes. These bad practices spread from the universities to the industry (and vice-versa), and the hidden costs are huge ;
- Great discoveries often stem from cross-disciplinary endeavours, and when masters students start a PhD, they should not be experts in only a narrow subset of computer science: as much as possible, they should have a *knowledge of the state of the art* in computer science *at large*, while it is often not the case. This is why insisting on a minimal perspective on history is important. Of course, the point is not to encourage pedantic attitudes: it is to foster a better understanding of *innovation in computer science* ;
- During the masters, developing a real OSS project, studying business models, along with possibly cooperating with students from other horizons (e.g. students from a business school, from media/journalism studies) would definitively be an important *plus* which could attract many students, and lead to *more multidisciplinary studies*. Students with such an experience will have *more options* later in their professional life ;
- Finally, although it is not mentioned explicitely above, students should *have fun*, and it's important to try to foster *creativity* in their minds. No boredom. Do it ;

Bibliography

[ETZ93] Oren Etzioni, Neal Lesh, Richard Segal – Building Softbots for UNIX (1993)
[GNU85] Richard Stallman – The GNU Manifesto (1985)
[LES95] Henri Lesourd – Realisation d'un Softbot sous UNIX, Paris 6 (1995)
[LES02] Henri Lesourd – Le systeme Hammourabi, PhD, University of Paris 6 (2002)
[MCK90] Wendy Mackay – Users and Customizable Software: A Co-Adaptive Phenomenon, PhD, MIT (1990)
[POP44] Karl Popper – The Open Society and Its Enemies (1944)
[ROS99] Guido van Rossum – Computer Programming for Everybody (1999)
[SMA76] Daniel H. H. Ingalls – The Smalltalk-76 Programming System Design and Implementation (1976)
[SUC87] Lucy Suchman – Plans and Situated Actions: The Problem of Human-machine Communication (1987)
[WEI76] Joseph Weizenbaum – Computer Power and Human Reason (1976)