CASE Support for the Software Process: Advances and Problems

Bernard Lang

INRIA
B.P. 105, 78153 Le Chesnay CEDEX, France
bernard.lang@inria.fr

Je maintiendrai
Motto of the House of Nassau

There are many facets to the software process, and much controversy over their identification or importance. However it seems obvious to me that we can at least distinguish between the technical and the managerial aspects. Being an academic, I am of course biased towards the technical issues, but even management has its techniques.

1. Achievements or Advances.

There have been many major advances in the technical aspects of the software process. To begin with, lets us recall that the problems predate their identification as a Software Engineering discipline, and complex software, presenting to a large extent similar problems, was already developed in the sixties. These problems were addressed as they emerged and were understood, with a varying degree of success. Though now a popular acronym, CASE is not a new concept, and we have been building tools since the inception of Computer Science.

Bearing this in mind, I would assert that one of the greatest achievement of CASE technology has been in the realm of **reuse**. Indeed, early advances in software technology have been based on the development of operating systems and of languages and compilers. Operating systems actually started as a systematization of various routines commonly used by programs, and which were standardized both to regulate the use of the machine and to avoid repetitious programming tasks. Similarly, programming languages were developed to embody in a systematic way programming techniques that had been identified as useful. There is only a difference of degree between a simple reusable component, a parameterized component, a component generator, and a compiler (for a language). This trend is far from abating: many languages have been and are being developed embodying new programming knowledge either w.r.t. coding technology or w.r.t. problem domain technology.

One of the best example is the current development of synchronous real-time languages that simplify the programming of large classes of reactive systems by embodying in a compiler advanced knowledge of the problem domain. An older example concerns parameterized components, and component generators such as Yacc and Lex to cite the most well known. A last example is the development of new extensions to operating systems that provide collections of ready made components for various purposes, most notably the creation of man-machine interfaces.

Finally, I would note that the so-called "*object oriented*" techniques that have gained so much recognition lately — and are essentially expressed as new structures in programming languages or data-bases — have and will considerably improve reuse technology.

I believe the second major achievement is the development of new operating system layers and tools, most notably in *networking, data-bases and man-machine interfaces.* To be effective, computerized support has to properly integrate all the services offered by the machine, and also to integrate the contributions of all the people involved in the software process. Indeed such integration mechanisms are one of the foremost and most difficult research topics at this time. However, none of the essential work would be possible without the fantastic evolution of the available **platforms**.

Similarly, the progress in graphical interfaces has considerably improved the responsiveness and usability of our tools. Indeed, one of the major difficulties for the software engineer or the programmer is the very mass of facts, commands, rules (etc...) that have to be learned and memorized in order to cope with current computer environments. Self explicative modern tools, as pioneered by the Macintosh, considerably relieve this load, and graphics considerably increase the communication bandwidth. Also the quality of graphical interface generators makes it easier to create and customize new tools.

The third achievement I would like to mention concerns the evolution of the **architecture of CASE tools and environments**, which should not, as is sometimes the case, be confused with the architecture of the supporting platform (e.g. distributed implementation). Monolithic systems are now being replaced by open architectures aiming at the integration of independently produced CASE tools. Simultaneously, many tools are now replaced by parameterized tools, or tool generators. Though far from over, this evolution already brings many benefits to users (if also some losses to manufacturers):

- creation of a competitive market of integratable tools, that can be produced by the most competent companies in each concerned area.
- customization of tools, and of environments built from tools to adapt to the need of companies and software projects.
- ability of tools and environments to evolve with the needs of users, and with the progress of technology.
- disappearance of scale factor preventing the contribution of smaller companies or countries as producers or consumers of CASE environments.
- better integration of CASE tools, w.r.t. functionality, and particularly w.r.t. to look and feel (see also previous point).

Another benefit of this architectural evolution is that it positively influences the design and architecture of the software systems produced.

The development of numerous methods that assist the design and development of complex software project might also be seen as an essential achievement, considering their crucial role in most industrial projects. It is not clear to me whether these tools are fundamental, or whether there are only (*very necessary*) crutches intended to compensate for the current inadequacy of other tools (that could well be changed, e.g. programming languages), or worse to compensate for the inadequacy or limited ability of the people involved (whom we have to live with).

2. Open issues.

Since one of the roles of a panel is to be controversial, I shall state that what I consider a major success, namely **reuse**, is also a major open issue, if only because of its remaining potential for increasing productivity. The problem is still open in many respects:

- we are more demanding and expect to reuse all types of documents/data that occur in the software process: code of course, but also design, development history, etc.
- the interaction between reuse and software certification is not too well understood.
- we are now aiming for very large data bases of reusable components, opening many hard research problems w.r.t. the organization of these data bases and the component retrieval algorithms.
- the problem is further compounded by the availability of component generators, and by mechanized combination of components, leading to higher order problems.

Possibly, an analysis of how reuse has been dealt with in the past, through its implicit contribution to the evolution of systems and languages could give us some hints for new paths to follow.

Another glaring open problem is **modularity** and **standards**. Considerable progress has been made from procedures, to modules and Abstract Data Types (ADT), to "object oriented" techniques. But many issues are still not well understood and raise complex theoretical problems, many of which are being addressed by the research on type systems. In this respect, software engineering could benefit from a more mature attitude towards programming techniques. The community focuses too much on fashion, religion and buzzwords like logic programming, ADT, object orientation, with very little respect, attention and connection to the actual mechanisms and constructions they offer and permit. I found this lack of clear identification of technical concepts to be a major problem in large team work.

More specifically related to the architecture of CASE environments is the issue of properly identifying the useful CASE functionalities, and even more importantly of characterizing their interfaces and the (logical or physical) types of data/structures and control they should exchange. A typical example, still seeking consensus after 15 years, is the representation of programs. Identifying, and standardizing these interfaces is crucial for the success of open and customizable CASE environments or software factories.

This leads me to what I believe to be one of the most controversial and open problem, though it is not strictly speaking a technical one: **patents** and **interface copyright** laws.

It is not my intention here to take sides on this issue. However I am surprised how little serious discussion there has been in the community to determine what kind of laws would be the most appropriate to protect the interest of creators, and of users, without preventing progress. I have seen little response to the very cogent (if controversial) positions of the founders of the Free Software Foundation.

Considering the importance of standards, and the fact that they often correspond to marketed products, this issue will certainly bear on the progress of CASE environments.

Concerning the managerial aspects of the software process, it seems to me that much work has been done in other areas for automating production processes, but this work seems seldom taken into account in the SE literature. Are there specific reasons that justify this?

I would also like to point out that software production is to a large extent the production and management of complex inter-related documents. Hence the development of office automation should have a notable impact on software production technology. Though I cannot assess myself whether that is the case, I believe this issue should be raised.

3. Conclusion

My own work being more with design and implementation techniques for CASE tools for advanced users, I am moderately qualified to actually assess their effectiveness in general industrial environments, and this has certainly biased my choices.

I have tried to strictly adhere to the panel topic as stated, namely CASE support. However, it is my strong belief that much of the progress in software engineering stems for other, often unexpected, sources. CASE is only a way to mechanize technical knowledge that has been previously acquired. Experiments and theoretical research are still much needed to increase this knowledge.