The maintenance is the implementation

OR

Why people misunderstand IT systems

Enrico Nardelli
Dept. of Mathematics, Univ. Roma Tor Vergata
nardelli@mat.uniroma2.it - http://www.mat.uniroma2.it/~nardelli/

This short paper has been written more as a position paper intended to stimulate discussion than as a research or experience description paper.

1. The traditional scenario of mechanization

Since the 18th century humankind has reassigned to machines jobs previously performed by human beings and animals, and this process is well understood.

Assume you have to harvest a huge field or to assemble thousands of pieces or to execute billion of calculations.

In the pre-industrial era you needed a lot of people and possibly the help of several animals. In the industrial era you buy a machine and use just a few specialized persons.

The machine does the physical work, while the people drive it and supply the required cognitive processing. If there is a big stone in the field, a broken piece, a meaningless number, the person sees it, recognizes it, and avoids it.

This applies to almost all industry mechanization. Machines are a substitute for physical effort by humans or animals. They need no capability to adapt to a changing environment, since persons driving them have this capability. They need no intelligence (in the human sense), since persons guiding them can supply theirs.

Once you have mechanized and automated a task, more or less 80% of the job is done. It just remains a 20% maintenance job. This is the current view.

2. Computer-based mechanization and flexible/adaptive mechanization

Then computers arrived, and humankind started to automatically solve a different kind of tasks by means of this new kind of machines.

A computer is a machine automatically executing a "problem solving" task working on a suitable abstract representation of problem data and their relations.

However, this time the problems to be solved, i.e. the tasks to be automated, are the ones where mainly human intelligence is required, and not so much strength or manual ability. But, on the other side, a computer just “shuffles symbols around”: each step in the execution of its problem solving

activities only changes strings of bits or symbols, attaching no meaning whatsoever to these representations and operations. E.g., there is no difference between a string representing a picture and one used to reserve a flight seat. It is only the intelligence of the human beings that reads them differently. It is indeed this blind procedure that enable us to apply this new kind of machines and solve so many different kinds of tasks, including running the machines themselves.

This is a completely different scenario from the ones humankind has experienced in the past centuries: now we are automating not just those tasks where there is a physical component and a cognitive component, using machines to execute the former and people driving them to take care of the latter.

We are now mechanizing almost purely cognitive tasks ("services", in modern parlance). Service automation has completely changed the usual mechanization scenario. We are replacing some intellectual capabilities of human beings with machines.

We have been able to successfully automate some of these cognitive tasks. For example those "control" tasks which mainly require a quick decision to react to many sensor-read values to execute a pre-defined action with a low cognitive content, e.g. controlling the strength and speed of car braking. Or those low complexity clerical tasks that can be fully described by means of a small number of fool-proof rules, e.g. money withdrawal from one's own bank account. Both of them are usually industrialized in IT systems with a high success rate. Most of the industrial productivity increase in the last 30 years is due to the introduction of this type of mechanization.

But when more specialized knowledge is required we are truly substituting the human intelligence with a machine. And we do not witness so much success here[1].

One of the essential features of the human intelligence is the adaptability to a changing environment, the flexibility in coping with modified or new requirements. Human beings have a "built-in" capability of adapting to changes and learning from errors. Machines don't have it.

And we know the world around any organization is constantly changing, if not the day after, certainly the month after.

3. A change of paradigm

Apparently, we witness just a slight shift in the meaning of word "mechanization". Instead, we have to cope with a dramatic change of paradigm. Our society has not yet understood it, and that's why people usually think: "now we have an IT system and we are done".

Wrong!

Once an IT system is built, the real job starts. "Panta rei": the environment changes, and the IT system has to be adapted, and requirements change, and the IT system has to be modified. And this goes on and on.

[1] Also, those "common sense" tasks where there is a low amount of specialized knowledge but a huge amount of common sense knowledge coupled to a sophisticate movement control, e.g. cutting the nails to an elderly person, are difficult to mechanize (may be even more than the previous ones), but they usually do not have a high cognitive complexity.

The IT professional community knows very well that software maintenance absorbs the largest share of the resources spent in an IT system lifetime. But our society at large is not aware of this, since most people - especially decision makers - are used to the old view of mechanization.

Hence they do not understand that building an IT system is only 20% of the job. The real - and expensive - job is managing its adaption to an ever changing environment. As an aside, we conjecture that this is one of the economic motivations for open source software: because it gives you for free just the first 20%, and then service companies are happy to be paid for the remaining 80%.

This misunderstanding is the main reason why IT system development efforts are late and over budget in a proportion embarassingly higher than for any other industrialized field. Whether they are a success or a failure is an orthogonal question not discussed here.

4. We can and should do something

Lawyers try to fix in contracts what an IT system should do. But legal descriptions in most cases do not work, since even if you are able to write a perfect contract specifying everything, the moment after you have finished it, the reality around the system has changed. Instead, contracts for hiring people usually work, because the human intelligence fills the unavoidable gaps existing in any contract.

One possibly effective approach to tackle this scenario is the use of a development "philosophy" inspired to the so called "agile methods". We deliberately avoid the use of the term "methodology" here, since the point is not about following more or less faithfully a sequence of steps but taking a completely different viewpoint on the development of IT systems.

With such a philosophy you build a system a piece at a time, constantly analyzing the solutions found, and adjusting to errors, like a person learns to do her job a bit every day. And you do this having defined a general plan and an overall design, but without detailing too much too early. The focus is on the responsibilities of those developing the system and of those driving, on the client side, this development, rather than on a detailed development plan often inflexible and difficult to adjust.

Another possible approach is to develop application development environments specialized for narrow application classes, where the end-user directly develops what she needs. This is the so-called “*citizen development*” that according to Gartner will have citizens build at least 25% of new business application by 2014 [1]. And the recent release by Google of “*App Inventor*”, the development environment making it possible to users of its Android-based mobile phones to develop their own applications, is a clear step in this direction [2].

Since many years informatics scientists working in the software engineering field are trying to understand better how the software development process can be shaped to better match these needs for adaptability and flexibility of IT systems. And to understand how we can produce software able to evolve in a changing environment by self-adapting their behaviour to new and modified situations ("situational computing" [3] and “self-managing situated computing” [4]), possibly using probabilistic models and reasoning tools to deal with uncertainty, and using the web as source of knowledge. We have an absolute need for this kind of results.

Still, the informatics community should probably act more proactively towards society in educating people that for the development of IT systems:

"***the maintenance IS the implementation***".

5. Conclusions

Twenty five years have passed since the resignation of David Parnas from the panel on "Computing in Support of Battle Management" convened by the Strategic Defense Initiative Organization [5] and his statement "*I believe that I have as sound and broad an understanding of the problems of software as anyone that I know. If you give me the job of building the <SDI> system, and all the resources that I wanted, I could not do it. I do not expect the next 20 years of research to change that fact.*" And I think the basic issues on which this statement is based hold still true, even abstracting away from the very strict requirements that this specific military system had.

In the almost contemporary and equally well-known paper [6] Frederick Brooks cited as the two most important issues to improve software production "*incremental development*" and "*great designers*". The first is a clear anticipation of the viewpoint taken by the Agile Manifesto in 2001, and the second is an anticipation of the conclusions contained in the report produced in 2006 by ACM's Job Migration Task Force led by Moshe Varde and Frank Mayadas, that only standardized IT tasks are offshored and the mission critical ones are kept in-house [7, 8]. Once again, the old paper provided a clear picture of the current situation.

Indeed, at least part of the Informatics community is fully aware of the issue I have addressed in this paper. As it was pointed out by the referee, this year International Conference on Advanced Information Systems Engineering conference (CAISE'10 – http://www.caise2010.rnu.tn/) has as its focus "Evolving Information Systems", stating that "*The evolution of an information system should be a continuous process rather than a single step, and it should be inherently supported by the system itself and the design of the information system should consider evolution as an inherent property of the system*". A perfect match with my views described in this paper.

But if, after more than 20 years, notwithstanding the warnings by two of the most reknown scientists in the software development field, society still considers IT systems just a sophisticate descendant of the plow and not a radically different new species, the problem of the role played by the informatics community-at-large in educating the society remains.

Acknowledgements: The author would like to thank the many colleagues he has shared his reflections with and whose comments greatly helped in focusing them: Silvana Castano, Paolo Ciancarini, Pierpaolo Degano, Paola Inverardi, Letizia Jaccheri, Maurizio Morisio, Daniele Nardi, Giancarlo Succi, Letizia Tanca, Franco Turini. Insightful comments from the reviewer have been greatly appreciated.

References

[1] "*Gartner Says Citizen Developers Will Build at Least 25 Percent of New Business Applications by 2014*", October 22, 2009, http://www.gartner.com/it/page.jsp?id=1212813.

[2] http://appinventor.googlelabs.com/about/.

[3] C. B. Anagnostopoulos, Y. Ntarladimas, S. Hadjiefthymiades: *Situational computing: An innovative architecture with imprecise reasoning*, J Syst. Soft. 80(12):1993-2014, Dec. 2007.

[4] Carlo Ghezzi, *SMSCom*, ERC Advanced Investigator Grant N. 227977 [2008-2013], http://www.erc-smscom.org.

[5] David Lorge Parnas: *Software Aspects of Strategic Defense Initiative*, Comm.ACM 28(12):1326-1335, Dec. 1985.

[6] Frederick P. Brooks: *No Silver Bullets: Essence and Accidents of Software Engineering*, IEEE Computer 20 (4): 10–19, Apr. 1987.

[7] William Aspray, Frank Mayadas, and Moshe Y. Vardi, Eds, *Globalization and Offshoring of Software*, Report of the ACM Job Migration Task Force, 2006.

[8] David Patterson: *Offshoring: Finally Facts vs. Folklore*, Comm.ACM 49(2):41-42, Feb. 2006.