Where is Software Headed?

W. Pree, G. Pomberger
C. Doppler Laboratory for Software Engineering
University of Linz
Altenbergerstr. 69
A-4040 Linz, Austria
Voice: +43.70.2468.9431
Fax: +43.70.2468.9430
e-mail: {pree, pomberger}@swe.uni-linz.ac.at

Before we outline future trends regarding software, let's briefly look back. Due to the enormous difficulties encountered with the development of software, many expected a single panancea to overcome the problems of the state of the art in software development. Computer aided software engineering, prototyping, automated programming, object-orientation and visual programming are just a few examples of technologies that staked the claim to cope with the known deficiencies of software development.

The past taught us the following: No single technology or concept comprises a breakthrough. Furthermore, promising technologies are not applied immediately in industrial software development environments. On the contrary, it often takes decades until new technolgies exercise an impact outside of research laboratories. There are manyfold reasons for this dilemma, especially that many companies are stuck with legacy software and often believe that they cannot afford to overcome this hurdle. The computer industry "helps" them by providing products that are compatible with the older ones.

Another phenomenon characterizes software. Though it's true that the problems software should solve are complex and that this implies many difficulties in the development process, unnecessary complexity is added in most software systems. Programmers are often proud of producing complicated solutions, probably in order to become indipensible and to justify the high costs. Furthermore, meticulous engineering is not rewarded.

What can we expect from the future? A pessimistic scenario results from extrapolating the past and current situation as sketched above. This means that the struggle (summarized as software crisis) continues and even gets worse due to additional domains where software is required and due to the fact that rediscovered or new technologies and concepts won't migrate into the mainstream.

Such a pessimistic view is corroborated by taking a look at current and soon-to-be-established defacto standards. Take object-oriented technology as an example. Though object-orientation could contribute to overcoming essential problems in software development, the most wide-spread languages applied to realize object-oriented software are antiquated and complicated and thus form no adequate tool from the viewpoint of state of the art in software engineering. Unfortunatley, higher level standards, e.g., for object/component distribution and operating systems, are being built on top of these languages.

Such premature standards significantly add complexity to software products. Programmers are forced to produce unnecessarily complicated and unprofessional solutions for problems which could otherwise be solved much more efficiently.

Though standards are becoming en vogue in the computer industry, they cement the software crisis. Despite the negative experience with defacto standards, the industry continues going this way.

Thus we dare to predict that adopters of such standards will not be able to exploit the potential of the underlying concepts and probably will get stuck in a dead end. It is simply too early to establish standards. If we do so, we will continue to get the impression that primarily marketing people and economic forces, not scientific advances, drive the software technology.

The trend of forming increasingly larger project teams to develop software comprises another obstacle to overcoming the software problems. Wirth (1995) states that "the belief that complex systems require armies of designers and programmers is wrong. A system that is not understood in its entirety, or at least to a significant degree of detail by a single individual, should probably not be built."

Software production still differs from other engineering disciplines in that software is developed almost from scratch. The percentage of reused components is very low. This implies that software suffers from teething troubles and quality problems that are common to newly built products. This would not be necessary. Especially object-oriented concepts would allow overcoming the reusability problem when they are used to build generic software architectures (= frameworks) for a particular domain where components can easily be replaced and/or added. Again, existing and emerging defacto standards as well as an industry which hesitates to apply this technology delay the long awaited breaktrough.

Though we draw a realistic picture of the future of software by extrapolating the current situation and trends, there is also hope. The future looks bright for those who depart from the current trail. An increasing number of companies that have applied computer technology almost since its emergence, such as banks, recognize that maintaining legacy software does not allow meeting current and future requirements any longer. They have the chance to show courage and replace the old systems by really new ones without compromises.

Those will be disappointed who remain in their lethargy and wait for a silver bullet to overcome the software crisis. No single concept, method or tool will result in a breakthrough. The key to successful software development lies in overcoming the obstacles sketched above and in applying a combination of already well-known concepts, methods and tools in the development process that is adequate for the problem at hand. Nevertheless, the development of real-world software systems will remain a difficult task that requires creativity and expert knowledge.

To sum up, we hope that the few who will set for new shores are so successful that the rest are forced to follow.

References

Wirth N (1995) *A Plea for Lean Software*, IEEE Computer, 28(2)