Polski Uniwersytet na Obczyźnie
w Londynie

Zeszyty Naukowe

Seria trzecia: nr 2, 2014

Stefan Stańczyk

PUNO w Londynie

ON COMPUTER-ASSISTED SOFTWARE ENGINEERING – A PERSONAL VIEW[1]

Computing has permeated virtually all aspects in our lives and has done so with much greater dynamics than any other human invention. Billions of lines of computer program code govern us. We are all affected: engineers and artists, doctors and lawyers, scientists and businessmen, farmers and journalists, celebrities and common people, politicians and their electors, leaders and followers. In banking, shopping, driving, making conversations, writing letters, designing, learning… with – or often without – so realising, we are.

But computing does not mean only computers, computer programming or computer engineering. It is more – and it fundamentally alters our thinking in most human activities.

Computers do not process anything, they do not calculate, they do not compute, they do not do any searching or sorting, they do not store anything. All they do is transform one series of electric impulses into another. It is us, humans, who interpret these impulses. Thus, we do not communicate with machines; the communication is between one user (however experienced) and another (such as a programmer) who – albeit not present – left his thoughts and interpretations embedded into the machine.

[1] The thoughts and views expressed here come from my own observations, reflections, readings and discussions and are not necessarily those of PUNO.

Computing requires handling very complex forms of (mental) structures. Its final outcomes – software systems – aim to encapsulate and to represent in a discrete way (the only way contemporary computers can function) parts of the real world. Furthermore, the transformations from the thoughts, words and actions of the real world to a form with restricted representation may result in loss of accuracy and completeness – yet those are the essential attributes of the outcomes of computing.

Computing roots are in mathematics, sciences and humanities. Mathematics gives it a notation, formalism and rigour, sciences (physics, chemistry, biology) make it possible to transfer the mathematical abstractions into physical existence while humanities offer support for communication, perception of symbolism, ergonomic design, and certain intuitive and comfortable manners in which we use the systems' functionalities.

The outcomes of computing are software systems, developed to support operations of a road network, a railway system, a bank, a production company or a service provider. The common name for the process of constructing those software systems is software engineering.

In what sense does this variant of engineering differ from the more established ones? The traditional manufacturing model aims at producing thousands of copies of a prototype – the prototype that was designed and tested in a (typically long) creative and essentially hierarchical process. First, requirement specifications and functional properties loosely define the to-be final product. Then, the product is broken down into correlated and interfaced subproducts. The subproducts are designed in exactly the same way (but perhaps with greater accuracy), i.e. by decomposing them into smaller pieces, and these into smaller still. Finally, the fundamental elements of the product are designed with the utmost accuracy and in every conceivable detail.

Then, mass production essentially reverses this process for multiple copies of elements that are put together to form multiple copies of subproducts, which, in turn, form multiple copies of the final product.

This aspect of engineering – single decomposition/multiple integration – does not appear to be particularly attractive as an analogy to software production for multiple copies of an element (say, a routine) are not frequently needed and, in any case, producing copies is a trivial task.

Software projects can more convincingly be equalled to large architectural works. There, the final product is bound (or at least expected) to be rather unique, still its elements must precisely fit the overall structure. They also must work well when combined – similarly as in spacecraft engineering. Shuttles are not produced in thousands (at least not yet). Nonetheless, every element of a spaceship represents a technological achievement and is a result of its designer's ingenuity, craft and creativity.

Programming – to use an old-fashioned word – is an engineering discipline. As in any engineering, there are theoretical foundations, recorded practices, rules and standards to be observed. However, the fundamentally different nature of the raw material the software has to deal with makes the difference.

In civil, mechanical or electronic engineering the raw material is tangible matter. It is processed, shaped, restructured to a new form. But it is very unlikely it does any own processing at all. In software engineering, we deal with an entirely new kind of material – thoughts, abstract concepts and interpretations. They eventually get represented by tangible matter (such as electric impulses), but even though – they never lose their intangibility.

Dealing with abstracts and interpretations, programming is usually incomparably more complex than classical engineering. Quite logically, meta-programming (to use a generic word for some elements of software engineering) is more complex still. Typically, proving the correctness of a program (other than an illustrative example) is far from being trivial and often more complex than writing the program itself. This is specifically relevant to systems of a more applicative nature, where a significant number of requirements are non-functional and difficult to quantify.

Then, what is software engineering, after all? Traditionally perceived, software engineering is essentially a linear process (albeit with repetitions) composed of a number of subsequent phases, each producing a more and more detailed plan to carry out the appropriate works in the next phase. Effectively, we have a series of translations – from a vague, informal, imprecise and, therefore, erroneous description to the systematic and rigorous form required by a machine – an intolerant object, which was produced in a similar way.

During subsequent production phases, certain tools are available – such as, for instance, modularization, stepwise refinement, data and program correspondences and so on. Nearly all of them stem from some sort of a structural approach imposing *ipso facto* the hierarchical structure on the final product.

However, software engineering is not really a single discipline, as no engineering is. It is rather a combination of various, mutually permeating, disciplines that are continuously being developed – not necessarily within one particular domain. For instance, a successful technique in one kind of engineering may prove to be merely effective in another.

Three domains form the skeleton of software engineering: the product itself settled in a space (physical, social, business, abstract), the matter the product is made of (so are the tools to make it), and finally the environment of comprehension, i.e. human capabilities that make us able to understand, to analyse and to appreciate technological phenomena. The boundaries among these domains are not sharply defined. On the contrary, the developments occurring in one domain affect and influence the two others.

Compared to the other branches, software engineering is much more fragmentary. There are numerous grey areas – theories not discovered, techniques not developed and, alas, experiences not recorded.

Theory aims to clarify the phenomena concerned, proving facts and disproving fallacies. In this work, abstraction and generalisation are used, for implementational details would rather obscure the analysed issues. On the other hand, practical methods concentrate on constructing objects and take logical and analytical formulae for granted. The objects constructed are to exist in real space and, therefore, great attention to detail is inevitable.

Just as theories must be communicative to method constructors, the methods themselves must communicate with the users. No method in engineering is appreciated unless supported by a suitable graphical notation. Graphics carries, quite naturally, more information and this information is a lot easier and faster to absorb. This explains relative popularity of various techniques – flowcharts, system charts, data flow diagrams, entity life histories, different forms of nets etc. – despite the fact that at least some of them are not fashionable.

No matter how complex a theory is, the corresponding design method may add a few complexities on its own. Then a particular design may assume a form that will not be comprehended easily. If a problem description is difficult to grasp, so are the results to obtain. It is a well-known fact that some theoretically sound methods with great practical potential are just useless if not supported by a computer system. The final element method is a primary example. It was a real breakthrough in several branches of engineering – very well grounded on discrete mathematics and theory of elasticity, clear and easy to understand, applicable to a variety of cases and provably accurate. However, the sheer time of manual processing would have made it entirely inapplicable.

The use of a computerised system for designing other computerised systems appears to be inevitable and indeed computer-assisted programming is becoming more and more widespread. Such systems maintain the entire designers' knowledge of the project, detect inconsistencies, errors and omissions and may produce alternative solutions whenever fed with new facts, constraints or criteria.

Stefan Stańczyk

INŻYNIERIA OPROGRAMOWANIA WSPOMAGANA KOMPUTEROWO – SPOJRZENIE OSOBISTE

STRESZCZENIE

Artykuł prezentuje informatykę jako dziedzinę nauki powstałą z połączenia elementów matematyki, nauk ścisłych i humanistycznych. Końcowy produkt informatyki – informatyczne systemy użytkowe – przewyższają stopniem komplikacji wiele zadań inżynierskich ze względu na odmienny typ przetwarzanych materiałów, wysoki poziom abstrakcji i konieczność dyskretnego prezentowania obiektów i procesów zachodzących w ciągłej rzeczywistości. Artykuł wskazuje również na podobieństwa i różnice pomiędzy inżynierią oprogramowania a tradycyjnymi dziedzinami inżynierskimi, takimi jak budownictwo, mechanika czy elektronika, i na tej podstawie stawia tezę o wysokiej przydatności systemów komputerowych do budowy aplikacji informatycznych.

Słowa klucze: komunikacja, komputer, inżynieria komputerowa, informatyka, programowanie, technika programowania, oprogramowanie