УДК 004=111

SOFTWARE COMPLEXITY

Obedin N.V., Chebareva Y.B.
Scientific supervisor – Associate professor Chebareva Y.B.

Siberian Federal University

"The price of reliability is the pursuit of the utmost simplicity. It is a price which the very rich find most hard to pay", - Sir Antony Hoare, 1980

The twentieth century is known as a century of achievements. The most important one is a computer. It has taken the main place in our life and work with a startlingly rapid progress. But the chip itself which we call "a computing machine" would be a useless heap of metal without one thing that makes it execute our orders – the program. At the dawn of computer era programs were quite easy but it was hard to compose them because there wasn't such computer language that would be easy to understand by both machines and humans. Later while the process of technical evolving such languages appeared, but programs became more difficult too. All these lead us to the modern and quite complicated problem of computer science in general and programming particularly – the complexity of developing software.

In 1970 a theory was stated by American engineer and one of the co-founders of the "Intel" company, Gordon Moore. Later it became to be known under the name of "Moore's Law" and it says that the number of transistors that can be placed on an integrated circuit doubles approximately every two years. It's true: the amount of transistors increased from 2300 in Intel 4004 released in 1971 to 731 million in Intel Core i7 released in 2008. In fact it's not the linear but the exponential growth. This supposition is fair not for hardware only but also for software. For example, for eight years from 1993 to 2001 the number of lines of code in popular operating system Microsoft Windows increased from 4 million to 45. Now there are about 2500 people working on it and they're divided into twenty groups. Division into groups has been necessary because it's hard to work on such difficult program and almost no one knows the whole internal architecture of this project. It also involves growth of errors and complication of amending them. A famous engineer and operation system researcher Andrew Tanenbaum said: «The most important problem is making software secure and reliable. Software should be as good as a TV set: the average user will not have ANY failures of any kind in a 10-year ownership period». But it'll be uneasy to satisfy this condition for many programs all around the computer world because of their complexity.

In the article "The end of computer science?" a Dutch researcher and computer scientist Edsger Dijkstra wrote: "In academia, in industry, and in the commercial world, there is a widespread belief that computing science as such has been all but completed and that, consequently, computing has "matured" from a theoretical topic for the scientists to a practical issue for the engineers, the managers and the entrepreneur <…> I would therefore like to posit that computing's central challenge, viz. "How not to make a mess of it", has not been met. On the contrary, most of our systems are much more complicated than can be considered healthy, and are too messy and chaotic to be used in comfort and confidence." This quotation has something in common with his other article called "Why American computing science seems incurable?" Both articles tell us that the problem is in attitude to computer science. It is believed to be in final shape like mathematics or physics and that's why the main direction of scientists' work is to develop and improve methods and solutions that have already been invented, but not to create new ones. Especially it applies to programming languages and algorithms because in spite of their plenty and development the basic ones are still the same. It requires rather great amount of research work to develop new ways of software creation and to improve machine logic altogether. But if we have a look at the most technically developed country in the world – the United States of America - we'll see that even there government and industry don't assign enough funds on such researches. It is proved by reports of the United States National Science Foundation: from 638 millions of dollars inquired in 2010 on research work in the field of computer science only 20 million were assigned to work on the problem of computations beyond reaching the physical and conceptual limitations of current technology. In other words less than 1 percent of all funds was assigned on creating new algorithms and methods of building hardware and software that could help to prevent a possible stagnation in the nearest future.

The reason hides in the relations between computer science and industry. For industry computer science has become a quite appropriate tool in exactly the same state which it is in now. Unwillingness to assign many funds on developing new ideas in computer science makes sense because it's a long-term and rather risky investment. That's why big companies prefer to spend money developing old good technology which has established a good reputation and preparing new developers who are really great professionals, but most of them cannot bring something new in a computer science. It is not because they're not as talented as Donald Knuth or Brian Kernighan, but because of their education. Universities are exerted a strong pressure from the industry not to indulge in such activity as scientific education, but to restrict itself in teaching only professional skills. Industry needs new computer developers, but computer science cannot continue developing without scientists and creators.

Separately I want to say a word about delusions in computer science. The greatest one says that "software developing has nothing in common with mathematics". It presents this hard process of creating like a kind of handicraft that can be done by everyone. This delusion has gained a huge popularity recently and this is not good. People cannot see the whole problem of complexity because of mistaken opinions like this one and think that this is normal to have a huge amount of tangled procedures and functions in their programs. They don't want to study basic algorithms and don't want to use mathematic methods to optimize their programs. It's okay while we do not reach the Moore's Law top limit, but sooner or later we have to deal with it and inconvenience of such programs will be the number one problem.

There is a plan of actions to make the situation better following from the reasons below:

1. We need to get rid from the prejudices about computer science. The common ones are "Computer science is in final shape and is suitable for everyday usage" and "It's easy to make programs now". It's not right and can be proved by complexity of huge program systems even for a team of developers. Such systems also contain a lot of errors which are uneasy to reveal and correct. The reason to this software complexity is imperfection of algorithms and developer tools.
2. The process of computer science education also needs to be corrected. Main direction of this process now is creating new professionals, not scientists and creators, but we need them to develop science.
3. Industry and government should increase the amount of funds to assign on research works in new fields of computer science nevertheless their long-term perspectives look worse than perspective of proved ones. Great profit is gained by risky things so we shouldn't be afraid of it.

If we start to overcome the situation in the right time and treat a problem of software complexity and other problems of computer science more seriously we can not only prevent further complication of programs' creation and usage, but improve their quality and simplify support. It will make computer science a truly stable and intelligible science of future.