УДК 004.41:811.111

Borodach V., Vasilenya M., Beznis Y.

Software. Notion and Development

Belarusian National Technical University
Minsk, Belarus

A software-based system can be neatly compared with a biological entity called a superorganism. Comprising software, hardware, peopleware and their interconnectivity (such as the Internet), and requiring all to survive, the silicon superorganism is itself a part of a larger superorganism [1].

Whether that business is government, academic, or commercial, the software-based system, like its biological counterpart, must grow and adapt to meet rapidly changing requirements. Compared to a biological superorganism, which may take many generations to effect even a minor hereditary modification, software can be modified immediately. This makes it far superior in this respect to the biological entity in terms of its evolutionary adaptability. Software, the brain of the silicon superorganism, controls the action of the entire entity.

Software is the embodiment of logical processes, whether in support of business functions or in control of physical devices. The nature of software as an instantiation of process can apply very broadly, when modeling complex organizations, or very narrowly as when implementing a discrete numerical algorithm. Software has a potentially wide range of application, and that well designed has a potentially long period of utilization [2].

While some would define software as solely the code that a programming language generates from the compilation process, a broader and more precise definition includes requirements, specifications, designs, program listings,

documentation, procedures, rules, measurements, and data as well as the tools used to create, test, optimize, and implement the software [1].

Software at the lowest programming level is termed a source code. This differs from an executable code (i.e., which can be executed by the hardware to perform one or more specified functions) in that software is written in one or more programming languages and cannot, by itself, be executed by the hardware. A programming language is a set of words, letters, numerals, and abbreviated mnemonics, regulated by a specific syntax, used to describe a program to a computer. There are a wide variety of programming languages, many of them tailored for a specific type of application. C, one of today's more popular programming languages, is used in engineering as well as business environments while object-oriented languages such as C ++ and Smalltalk have been gaining acceptance in both of these environments.

The programming language, whether it be C++, Java, Visual BASIC, C, FORTRAN, HAL/s, COBOL, or something else, provides the capability to code such logical constructs as that having to do with: user interface, model calculations, program control, message processing, database, data declaration, simulation, tools and some other.

As a base unit, a line of code can be joined with other lines of code to form many things. In a traditional software environment many lines of code form a program, sometimes referred to as an application program or just plain application. But lines of source code by themselves cannot be executed. First, source code must be run through what is called a compiler to create an object code. Next, the object code is run through a linker which is used to construct an executable code. Compilers are programs themselves. Their function is twofold. The compiler first checks the source code for obvious syntax errors and then, if it finds none, creates object code for a

specific operating system. UNIX, Linux (a spinoff of UNIX), and NT are all examples of operating systems. An operating system can be thought of as a supervising program that controls the application programs that run under its control. Since operating systems (as well as computer architectures) can be different from each other, the object code resulting from the source code compiled for one operating system cannot be executed under a different kind of operating system – without a recompilation [1].

Solving a complex business or engineering problem often requires more than one program. One or more programs that run in tandem to solve a common problem are known collectively as a system. By combining objects it is possible to create more organized systems than those created by traditional means. Software development becomes a speedier and less error-prone process as well. Since objects can be reused, once tested and implemented, they can be placed in a library for other developers to reuse. The more objects in the library, the easier and quicker it is to develop new systems.

The process of writing programs and/or objects is known as software development, or software engineering. It is composed of a series of steps or phases, collectively referred to as a development life cycle. The phases include the following: an analysis or requirements phase, where the business problem is dissected and understood; a specification phase, where decisions are made as to how the requirements will be fulfilled; a design phase; an implementation or programming phase, where one or more tools are used to write and/or generate code; a testing phase, where the code is tested against a business test case and errors in the program are found and corrected; an installation phase, where the systems are placed in production; and a maintenance phase, where modifications are made to the system. But different people develop systems in different ways.

These different paradigms make up the opposing viewpoints of software engineering [3].

A new approach to software engineering is known as development before the fact (DBTF) which includes a technology, a language, and a process (or methodology). With DBTF all aspects of system design and development are integrated with one systems language and its associated automation. Reuse naturally takes place throughout the life cycle. Objects, no matter how complex, can be reused and integrated. Environment configurations for different kinds of architectures can be reused. A newly developed system can be safely reused to increase even further the productivity of the systems developed with it. The paradigm shift occurs once a designer realizes that many of the old tools are no longer needed to design and develop a system. For example, with one formal semantic language to define and integrate all aspects of a system, diverse modeling languages (and methodologies for using them), each of which defines only part of a system, are no longer necessary. There is no longer a need to reconcile multiple techniques with semantics that interfere with each other. DBTF can support a user in addressing many of the challenges presented in today's software development environments [1].

References:

1. Mode of access: http://www.sze.hu/~szenasy/Szenzorok%20%E9s%20aktu%E1torok/Szenzakt%20jegyzetek/Mechatronics_handbook%5B1%5D.pdf. – Date of access: 15.02.2018.
2. Mode of access: https://en.wikipedia.org/wiki/Software_design. – Date of access: 25.02.2018.
3. Mode of access: https://sea.ucar.edu/best-practices/design. – Date of access: 22.02.2018.