# Chapter 1

# The Role of Science, Technology, and the Individual on the Way of Software Systems Since 1968

Zeynep Altan
https://orcid.org/0000-0002-0383-9261

Beykent University, Turkey

## ABSTRACT

The NATO conference held in Garmisch in 1968 was on the future of the computer and software world, and it presented the process of realization of what has been talked about in those dates to the present day. This chapter also examines the development of software systems since 1968, depending on the technological developments. The contribution of mathematics and physics to the development of information systems was explained in chronological order by comparing the possibilities of yesterday and today. Complementary contributions of science and technology have been evaluated in the evolutionary and revolutionary developments ranging from the definition of information theory in 1948 to teleportation. It can clearly be seen that discrete mathematics directly affects the improvements in computer science. This review study clearly shows that it would not be possible to talk about digital transformation and quantum computation if the discoveries of Shannon, Turing and Neumann, and the studies of other scientists before them did not exist.

DOI: 10.4018/978-1-7998-2142-7.ch001

## INTRODUCTION

At the NATO conference in Garmisch in 1968, the future of the computer and software world was dealt with. The arguments at the sessions were in preparation for today's digital world. The conference was held for two consecutive years and software engineering was recognized as an independent discipline. One of the editors of the meeting booklets of today's software infrastructure, Randel (2018) evaluated fifty years of software engineering. The paper summarizes the development of software engineering as a new discipline. The transfer of conference reports to an electronic platform in 2001 (McClure, 2001) informed the IT industry about the conversations held at the conferences and confirmed that all progress in information technologies and the software sector aligned with previous experiences.

The roles of science, technology, and human beings are all entwined when examined in the context of software engineering. Although all engineering disciplines utilize these three parameters, software engineering differs from others. The resulting software product in this discipline is abstract and is the product or a portion of the product that must entirely be completed. However, in civil engineering, for example, it is possible to open a bridge before the side roads are completed. This example should not be confused with the delivery of the software product to the user in parts. No matter how small a software product is, it is a stand-alone product and must be delivered to the user in a fully operational state. Examples can be increased for all other engineering fields that offer tangible, concrete products. Therefore, software engineering is different from other engineering disciplines because customer satisfaction is most prominent; the software needs to meet the needs of the user. This puts the human factor in the first three criteria. Today's rapidly changing software products require developers to work closely with the customer. In this context, the evaluations of satisfaction and performance for software engineers and software developers are indispensable criteria for this engineering discipline. In fact, the quality of the work performed in most labor sectors has been measured psychologically by scientific studies for a long time. There are many models investigating the relationship between the pleasure of the working environment and quitting the working, and intention not to work. While the intention of a software engineer or a product developer to quit their job is an important risk factor for the company, doing the job willingly is a positive appraisal for the business. In fact, the degree to which an organization's employees are satisfied with the working conditions and the working environment is an indication of how much that organization attaches importance to its employees.

In the first fifty years of software engineering, and by changing over from hardware engineering to software development processes, new techniques were developed and tools were used to deal with the complexity problem. During the evolution of



software engineering, advanced engineering principles and complicated mathematical foundations have been applied to solve more complex software problems. While the effectiveness of the developed product was important in the early applications of software engineering, the accuracy and usability of the result (i.e., software) became more substantial as the problems became more complex. Therefore, the contribution of individuals on the product had to increase. Complex, real-world problems led

of individuals on the product had to increase. Complex, real-world problems led to the development of reusable products, and new development tools and methods have emerged as a result. However, since these tools solve increasingly complex problems and are designed according to the scope of human capabilities, the success in producing solutions with today's new technologies will guide the solution of complex systems in the future. Therefore, the integration of software engineering with system engineering and the cognitive psychology highlighted by human factors have important roles in overcoming the problems of today's engineering world. On the other hand, it is also necessary to consider the extent to which it is appropriate to use technological developments and important revolutions in solving current and future problems. Many of the negative conditions that can be caused by the modern world, especially security, are important issues that engineers must consider.

The technological advances of the 60s and 70s made significant contributions to the development of the software sector and it continued to develop in the 80s in cooperation with the software industry. Technology-driven software products that were effective in the 90s have transformed into a business world that uses technology as a focus today. The improvement of software engineering projects through the success stories of the past have always moved this new discipline forward. It is reported in the Standish Reports that similar, previously completed software projects have been utilized in many software projects. Furthermore, if there had been no major changes in the past in software technologies, it would have been impossible to imagine future of software applications.

Becoming an independent discipline in 1968 stemmed from prior scientific studies. Later, these studies were applied in the software field. In the 1950s, the people who focused on solving a software problem were either mathematicians or engineers who specialized in hardware. In this context, the problems were focused on the military, scientific calculations, and space research. The software development processes of the problems were being developed in the computing capacity of the computers of that time and under the guidance of hardware engineering. This is clear from the fact that the hourly cost of room-sized computers was 300 times higher than the cost of the engineer working on the system. However, it should be remembered that the system that solved the problems were also the first application of sequential waterfall models in the 70s. The solution from the 60s focused on developing the software product as the antithesis of the 50s. Thus, many software applications have changed from hardware intensive to human-intensive. Since human-computer interaction





issues have become more significant than the strict engineering rules applied in software projects, the projects have been evaluated from a psychological perspective.

Since the software does not wear out like hardware, the maintainability concept of software was measured differently from the maintenance evaluation of the previous period. During this period, the human factor had been an effective parameter for planning and determining the cost of untouched software products. In the 70s, this factor made the solution difficult even though the problem was not very complex. Because the quality of software would become the focus of the product, the determination of this abstract concept would become more difficult. The prominence of the product to be developed led to the rapid development of programming languages. However, the languages used in software development in those years

were not developed according to any software standard. NATO Conferences gave software engineering a formal structure that started in the 70s. Software products were being developed based on a software development methodology so that it was possible to work on large software projects. These large software projects were the first examples of today's digital world. Almost all design and programming principles of today's applications were created in this period. The software products of this era were compute-intensive. Further technological improvements were required to implement more data-intensive applications. These would constitute the electronic world of the 21st century.

In the 80s, the scalability of the products began to rise. The most important principle of the 80s was productivity. The specification of the software development standards was one of the main reasons for the increase in product productivity. Software development standards were examples of software products developed in the following periods.

Other important criteria for productivity are the usage of software tools, the realization of the environment configurations, and the integration of them; together, they developed software products on a global scale. Utilizing the abstract mathematics theories, formal software development principles have contributed significant effort to the improved productivity of software products. With the support of knowledge-based approaches, advanced software development environments have resulted in the first expert system applications. As the scope of the problems widened, the complexity of the software that would challenge the productivity solutions have appeared. Since the early 00s, new software development approximations have continuously been designing. They resolve the disadvantages seen in the previous projects and continue updating the methods to solve real-world problems.

Conceptualization and visualization in the solutions to problems are guided by the reuse of software as an important solution approach in the later periods of the software discipline and became one of the basic parameters in increasing productivity. With the spread of the Internet, distributed software engineering applications have



increased worldwide. Further, with the combination of software departments of many companies on an international scale, some problems such as communication difficulties, timetable issues, infrastructure deficiencies, and documentation mismatches have occurred. As an outcome of technological developments and improvements in software applications, it has been seen that today's global projects could easily overcome these obstacles. Concurrent software applications of distributed software development methodology have strongly contributed to the transformation of open-source software development.

Returning back to the 90s, human-computer interaction has found its place in the computer world and usability has become an important criteria for product development. Thus, the focus was on the group developing the product rather than the performance of the individuals. This approach constitutes the agile software development methods of the 00s. It was also the beginning of rapid changes in information technologies due to globalization.

The remainder of the chapter is organized chronologically. The next section describes the birth of the first computers and the advances from the calculations performed with the existing computers to the future of the calculations, namely

performed with the existing computers to the future of the calculations, namely quantum computing. Then, the indispensable contribution of logic to the evolutionary development of technology and applications is discussed, and the study continues by discussing the evolutionary developments and revolutionary innovations. The ongoing study concludes with a general assessment of the human factor in successful software products.

## COMPUTERS AND COMPUTING FROM THE PAST TO THE FUTURE

### First Computers and Information Theory

George Stibitz developed the electrical digital computer that could compute complex numbers in 1937 and is considered one of the fathers of the first modern computers (Saxson, 1995). With Stibitz, Boolean logic was used for the first time in electronic world applications (Ceruzzi, 2012). At Bell Labs, where he was also a researcher, he designed computer circuits with adding, subtracting, and storing operations. Stibitz did not know that Konrad Zuse, in Berlin, was working on the same subject; but he was aware that Claude Shannon had designed binary relay circuits using symbolic logic in his doctorate study at MIT (Collins, 2002). These studies were carried out independently of each other in the same years and are the first examples of modern-day computers. Shannon also studied coding theorems, which is one of the concepts of information.





The general definition of information theory is the quantification, storage, and communication of information. In 1948, Shannon developed a general system that would form the basis of communication theory by enabling the transmission of information by signals, just as in today's data compression. A transmitter described by the system provided the appropriate signal and processed the message from the source throughout the channel. The entropy value of the source calculated by the noiseless channel coding theorem measured the optimal compression of incoming messages (Shannon, 1948). The capacity of a channel measured the speed of maximum transmission of the information at a certain noise level with the noisy channel coding theorem (Shannon & Weaver, 1949). Error-correcting codes, a fundamental principle of Shannon's information theory, were obtained after these theoretical studies and are still valid in boosting the transmissions of today's modems.

In the 1980s, modems carried out data transmission via telephone lines with a maximum speed of 9.6 kilobits per second. In those years, the need to increase the speed of data transmission meant that there were a large number of errors in the data. If Shannon had not developed information theory, there would have been no rapid data transmission, not enough space on disks, and no Internet world would emerge. The definition of Shannon entropy is the measurement of the information in the message in bits, while the Shannon limit is the information that can be sent with zero error at maximum speed (Lombardi, Holik & Vanni, 2016a). This can be explained using the following example: if a 3-bit message is transmitted 3 times, the receiver will receive 9 bits. But, the correct 3 bits will be sent with the error

correction code. When the noisy status of the channel increases, more information will be needed to meet the errors. Shannon's studies have shown that the longer the code, the more accurate the message will be. Today, every device used has an error-correction function.

## Today's Computers and Neumann Architecture

Developments in the field of electronics, along with the Von Neumann architecture (O'Connor & Robertson, 2003), were led by Moore's law (Moore, 1965). Moore expressed the need to integrate the circuit design with a greater number of components. Therefore, Moore, the father of semiconductor components, should be mentioned together with Neumann, the father of computers. In 1975, computers were generated by squeezing 65,000 components in a single silicon chip where the unit cost was reduced by taking advantage of Moore's law (Thackray et al. 2015). This was the beginning of many technological developments striving to meet the need for intensive data processing from integrated circuits to the present day. Computations made in the technological conditions of those days were performed at a much lower cost due to the lower power consumption than in previous years and with a higher





performance where power densities increased dramatically. With the introduction of silicon chips, the era of low-efficiency hardware was over. By 1989, with the 486 processor, the transistor number rose to 1.4 million.

Increases in today's computer capacities in the last quarter of the 20th century enabled the development of product configurations; therefore, certain large computer manufacturers focused on platform-oriented products. The transformation into an all-encompassing architecture with a general-purpose processor has spawned the concept of reconfigurable computing. To meet the calculation need, operating systems that perform multi-task management, resource management, and time-sharing were developed. In these systems, global data transmission is realized according to tasks defined by making graph partitioning. Scientists think that the correction in the error-correction functions of devices will lead to the improvement in much more effective techniques with quantum information. In other words, if information carriage in problem-solving and data processing is a quantum system instead of a binary system, the problem of error correction will be eliminated. It is claimed that data transmission will be much more secure than today's technology (Lombardi, Holik & Vanni, 2016b). The binary structure of the computer architecture still in use also performs protection with entanglement systems, providing security by processing only slightly more knowledge. In today's computer systems, error corrections perform automatic communication with self-integration instead of the combination of different devices that are required in older applications. In other words, computer scientists are constantly designing more efficient algorithms to prevent traditional computers from lagging behind in the race.

Recently, requirements for new processor configurations for companies using data-intensive, powerful machine learning techniques such as deep learning have been rapidly increasing. Even today's supercomputers are forced to update their calculations on a machine with the same capacity. Still, 3D chips continue to improve processor performance. Another solution to the computing power of machines has been the

performance. Another solution to the computing power of machines has been the migration of data centers to a single computation entity. Cloud services offer fairly cheap and easy alternatives, called in-house software. Despite these developments, the question of how to produce more powerful hardware that is compatible with complex data science, analytics applications, and self-learning systems is already one of the major problems of the processor manufacturers.

## Quantum Information Theory and Quantum Computers

Quantum technology is rapidly improving, though it is controversial whether quantum computers will close the gap between classic and quantum computers. In companies using information technology, hardware capacities are constantly being increased, but this is not enough. Although large-scale quantum processors are very expensive





today, quantum theory and algorithms will be the hardware infrastructures of future generations of computers, and the huge information of the future will be easily processed with these processors. Nations equipped with high technology are in a race to achieve supremacy in quantum computing, and huge systems have quantum computer requirements. The IBM Q System is the first quantum computer created in laboratories with quantum researchers and is preparing to serve the scientific and business world (Lardinois, 2019; Vincent, 2019). Despite being designed for commercial purposes, it is reported that this computer is not yet ready for widespread use. Google, on the other hand, claims that they have quantum supremacy and have a quantum computer with one million processors. Google's studies on artificial intelligence (AI) continue with powerful chips growing at an exponential speed relative to those in use (Hartnett, 2019).

Returning to how this radical quantum technology transformation began in information technology, it was claimed in 1999 that Shannon's knowledge was incompatible with the quantum context (Brukner & Zeilinger, 1999). When information storage, processing, and transmission were realized with quantum information theory according to the laws of quantum physics, no powerful computer operating with classical information technology would have access to this computational power that evolved at an exponential speed. Quantum bits (qubits) are the two states of a classical data bit, in which the transistor on the chip has two different voltage forms (Neilsen & Chuang, 2010). In other words, qubit symbolizes the state of a bit in a 2-dimensional vector space as 0 or 1. Qubit has the same characteristics as bits if it is single-way. Any qubit state space has a permanently expanding schedule. It is possible to summarize the existent application areas of quantum computing with the following titles:

1. Sophisticated modeling of financial services,
2. Research in medicine and pharmacy, simulations on biomedical,
3. Supply chain logistics applications in which billions of trillions of operations are Performed per person, where existing optimization problems will be insufficient for the Solution,
4. Applications that contain huge information in which exponentially faster data analysis is performed, in other words, machine learning.

One of the biggest changes in the Internet is that cloud computing made data computing power much more important as a revolution of the 21st century. In sum, quantum information theory studies continue to increase in three subjects depending on the field of application: super dense coding, quantum teleportation, and quantum computation (Wilde, 2019).





## THE CONTRIBUTION OF LOGIC IN THE EVOLUTION OF MACHINES AND SOFTWARE TECHNOLOGY

### Decision Problem and Computation Theory

Kurt Gödel's incompleteness theorem has made significant contributions to computer science and the modern-day computer world (Raattkaine, 2015). The Turing machine was a machine that called everything that has been calculated computable. This was Hilbert's solution, who was a famous decision and problem-solver in mathematics. Turing developed the Turing Machine to answer the following question, which Hilbert asked in 1928:

How can a general algorithm be written so that any mathematician can solve any mathematical expression?

This machine, with which Turing proved that a machine that can calculate everything, put an end to the famous decision problems of mathematics, suggested by Hilbert. Alan Turing's paper on computable numbers is proof that he is the principal srcinator of modern computers (Turing, 1936). Hilbert described the axiomatic formalization of systems before the solution of the decision problem with a number of formulas, and Turing also designed his famous abstract machine with axioms in formal terms. As a universal computational device where any algorithm is solved, the Turing machine performs calculations using the sequence of symbols found in an infinite tape. The number of states used to solve the problem is finite and the right end of the tape is limited to the operations that the computer can perform at once.

John von Neumann, who worked on the incompleteness theorem during the same period as Turing, solved this theory and described the infrastructure of computer architecture that has been in use since the 1940s. This solution represents the basis of modern-day architecture as Von Neumann Architecture was designed concretely with an input device, control mechanism, memory and output device, and the processing of the expressed system used in the solution of the incompleteness problem (Istrail & Marcus, 2013).

The English physicist Deutsch pioneered a new computing system by asking the following question aimed at Turing's theory:

Is there a single (universal) computing device that can effectively simulate any other physical system? (Deutsch, 1985)

In addition to physics, the graph theory, which is an important subfield of discrete mathematics, has been used in solving both hardware and software problems of





systems. An example of this can be given from circuit design: a VLSI (Very Large Scale Integration) design is the algorithm of a multilevel graph partition (Karypis & Kumar, 1998). An implementation of this is the realization of a highly qualified hypergraph partitioning in a short time is in the design of telephone networks.

## Ontology and Semantic Technologies

Since the perception and reasoning of knowledge are different in various people, it is natural to encounter communication difficulties. With the emergence of different terminologies, the data that refers to the same concept may not be able to connect different storing devices. When there is a huge amount of data using different terminology, ontology is a good technology that can solve this problem.

Ontology is the formal depiction of a shared conceptualization (Borst, 1997). The most important advantage of using ontologies is the symbolization and sharing of knowledge with the vocabulary used. In short, ontology acts as a format provider for information exchange, enables interoperability, reuses information, and integrates knowledge with automated verification. Thus, the machine's readability and operation can be more easily understood. The concept of lattice (described as Galois lattice) had important applications in data mining; it is a partial ranking system symbolized by the Hasse diagram (Berry & Sigayret, 2004), and it is the indispensable of ontology. Ontology languages such as OWL have also been developed based on description logic. OWL (Web Ontology Language) is widely used in many applications for representing and reasoning processes, planning and recognizing AI (Gil, 2005), modeling business processes (OMG, 2011), in web services, and for recognizing human behavior (Rodriguez, 2014). Different conceptualizations between systems exist, and different terminologies and meanings can be shared using ontology. The integration of ontologies is achieved if the relations between two or more ontology concepts are deduced in the solutions of real-world problems. Thus, the realization of heterogeneous data applications in distributed systems through ontology is a success of green information technology (Binder & Suri, 2009). In sum, the negative impact of information technology operations is minimized in computers and in every product related to computers at every stage from design to product manufacturing.

The main problems in the industry can be summarized as semantic interoperability, ubiquitous computing, enterprise integration, and business convergence. Mobile computing has been an integral part of daily life in the last five years. Mobile application developers are increasingly utilizing the benefits of semantic technologies in sharing, reusing knowledge, and knowledge decoupling. The wisdom of developed applications comes from the measure of new knowledge extracted from the inferred information (i.e. semantic reasoning). But, purpose-specific, standardized, and unambiguous data prepared with semantic technology imitates human logic by using

simpler algorithms. Much simpler algorithms are used in operations using big data, and these algorithms attempt to imitate human memory.

## Solving the Problems with Inference Rules

As mobile platforms became more widespread, ontology-based reasoning became indispensable and using semantic data became prevalent. The conceptual solution of many problems with uncertainty in this symbolization is generalized with probabilistic description logic. As description logic is determined in standardizing decision problems in big problems, the usage of finite automata has also been seen. Mobile devices on different platforms have established their own semantic APIs (Application Program Interface) using description logic with the improvement of semantic web technology. For example, description logics defined for health care and life sciences (e.g., tourism and augmented reality) have been realized with different ontology languages. Further, the solution for many real-world problems can easily be transformed into applications with influence diagrams (Borgida et al., 2019).

A classic example of influence diagrams used in many studies is the problem of deciding whether to go to a weekend picnic by looking at weather forecasts. To automatically transform the inference rules on mobile applications between platforms, rules defined in different languages need to be transformed syntactically (i.e., from Java to Swift) because such definitions play an important role in the performance evaluation between APIs (An et al., 2018). Another example of a transformation between APIs of two different platforms is for Java and C# (Nguyen et al., 2014). The success criterion of the transformation between platforms is determined by defining the unambiguous grammar used by the intermediate representations of each API. The possibility of a duplicate solution from the code with intermediate representations (e.g., the occurrence of condition before and at the end of the loop) is out of the question.

Such ambiguity problems are eliminated by defining the formal representations of grammar rules with multiple statements. In addition, the syntactic tree may be designed in such a way that intermediate representations do not allow expressions to be nested and the source code is normalized. Even though these features seem to be commonplace, they are important criteria that allow the solution of the problem to be transformed into code as clean code.

## Toward Quantum Computing

Since computer theory was developed so rapidly, computer scientists were looking for a new universal computer; they frequently expressed the possible contribution of physics in universal computers and the changes to the laws of physics (Feynmann,

1982). With the exact simulations of quantum physics, computers were intended to operate in complete harmony with nature. According to the researchers, this was only possible in a finite volume of space and time and therefore, a finite number of logical operations could be fully analyzed. These studies reveal that quantum computing has been built on Turing's legacy.

The main goal of simulation studies, which have been going on since the 80s, is that when a large physical system is fully simulated, the boundary value of exponentially growing computers in power will drop. In this context, incomplete physical knowledge and its conversion into practice are still being studied with great effort. Additionally, the speed of continually renewed computer processors in the last fifty years according to Moore's law (e.g., the logarithmic increase in the number of transistors on chips in every two years) has begun to slow down in the second half of the 2010s. It is expected that this slowdown will be fixed in the 2020s and computers that operate quantum computing with qubit chips whose prototype studies continue, will commercially begin to be implemented (Health, 2018). But, despite the theoretical quantum teleportation studies that began in 1993 and developed rapidly (Bennett et al., 1993), the transmission of an unknown qubit state to another location in the early 2000s still had not succeeded. The studies of those years drew attention to the concept of entanglement and teleportation and scientifically laid the foundations for the computer world by stating that information will not be copied, just as it is today.

Teleportation, which took place in 2017 between entangled photons in a 1200 km distance between China and Tibet, was an application of the new technology studies (Messier, 2019). This Chinese experiment has been the longest quantum teleportation phenomenon ever made. The explanation by the scientists confirms that the experiments depending on Einstein's statement "spooky action at a distance" told in 1948 (Musser, 2015) continue to increase the speed.

## Numbers Theory and Cybersecurity

As quantum computers have enormous power in computing, it is envisaged that it will solve the problem of cybersecurity as well. The security studies, which began entirely intuitively, evolved from ad-hoc designs from the 1980s to a methodology whose principals were determined. These methodologies were determined as (1) a mathematical problem that had not yet been solved for any security problem was selected and (2) whether the solution to this problem provides the targeted confidence was investigated. Numbers theory has been the biggest contributor to solving these problems. The basis of today's encryption, cryptography and cybersecurity theorems are also based on numbers theory. When the first studies on numbers theory were investigated, Pythagoras who lived in Ancient Greek in 600 BC was found. In 300



The Role of Science, Technology, and the Individual on the Way of Software Systems Since 1968

BC, Euclid's work on this subject was seen in converting the mathematics of that period into deductive science. Further, the Sumerians (2500 BC) had studies on number systems and the Babylonians (2000 BC) had mathematical tables written

on tablets before these two Greek scientists (Ore, 1948). The theory of numbers, the fundamental theorem of arithmetic, has found its application in many important and critical issues today, thanks to the mathematicians and scientists who have followed the important inventions of prehistory.

## TRACES OF THE NATO CONFERENCES IN CONTEMPORARY SOFTWARE PRODUCTS

### Software Engineering in 1968 and Agile Manifesto

The concept of a software crisis, defined by Bauer in 1968 at the NATO Conference, revealed that programs were not sufficient in solving problems using computers and in the control of solutions. Therefore, it was suggested that software engineering should be used as a term (Bauer, 1987). This meant that the software product that was developed was not successful for different reasons. The developments in the software world from the 50s to the beginning of the 70s generally followed the technological evolutions in the first, second, and third-generation computers. Looking at NATO conferences (Buxton & Randell, 1969; Naur & Randell, 1968), Ercoli (1968) addressed the cost of very large operating systems per instruction in his talk. Harr, on the other hand, emphasized the importance of developing the operating system according to a scheduling algorithm for quality as user requirements change. D'Agapeyeff asked questions about whether a single operating system would be sufficient for a computer, and whether a computer made up of a single framework could meet the requirements of different users. Lampson made additions to Harr's opinion at the conference in 1969 and said that systems should be designed as open-ended so that they could be expanded to meet changing requirements. All of these talks were the predictions of the development stages of hardware systems described in the previous chapter.

Another speaker at the conferences, Dijkstra, said that as the problems become complicated, the need for the distribution of knowledge would be inevitable and this would lead to incorrect coding of the problem. In addition, as it is impossible to determine the quality of the product being developed, he argued that the design process and implementation must be interpenetrated. The views advocated in those years still remain valid today. It is no coincidence that the importance (and to some extent necessity) of formal representations of solving software problems was mentioned before the 70s. It is possible to see the role of conceptualization in the



progress of computing and the computer world as much as possible in the previous sections. Dijkstra advocated for:

Complexity controlled by the hierarchical ordering of function and variability explained the solutions to the software crisis in his book (Dijkstra, 1976).

Royce (1970) proposed the waterfall method to solve large-scale problems and laid the foundations for the new methodologies in software/system development.

The waterfall method requires documentation at each stage before moving from one development stage to another in the product because the output of the previous stage is the input of the next stage. Dijkstra (1968) explained the necessity of documentation for successful software in his speech:

> I have a point with respect to the fact that people are willing to write programs and fail to make the documentation afterward… I would suggest that the reason why many programmers experience the making of predocumentation as an additional burden, instead of a tool, is that whatever predocumentation he produces can never be used mechanically.

The Agile Manifesto (2001) was published by 17 computer scientists in Utah (Beck, 2001). Even though contemporary methods are based on the manifesto have been used in many software products to emphasize that documentation is not necessary and focus on the importance of the software operating in small parts instead, this has not been the case. Soon after the manifesto's applications began to appear, the importance of documentation in a software project was understood, and it was concluded that documentation should be added to new product development approaches.

Dijkstra had another speech at the conference that formed the second of the four key principles of the Agile Manifesto where decisions on the future of software development were made. In this session, he stated that communication between groups will make the flow of negative information easier and this will also positively reflect the development of the product. He also described software developers as manufacturers and stated that it was not right to blame them for each problem. The message jointly given by other speakers in this session was that the software system should be a product that is acceptable to all users. This view is fundamental for today's service-based architecture in the software industry. When associated with the manifesto, the third principle is encountered as customer collaboration.





## Software Engineering in 1968 and Service Contracts

Software developers, called manufacturers at that time, would lose money and time by collecting false data on the product to be developed and should plan accordingly. Randell foresaw further and asked users to demand safeguards with contracts due to the rapid increase in the size of the problems. Until the manifesto, contract negotiations were made between the user and the developer that contained the entire product to be developed. These later were transformed into the contract of the software piece to be developed. Randell's speech at the 1968 conference is a reminder of today's service contracts as the agreement between a remote service and a service consumer. The inbound and outbound data organized in the contract format illustrates the foresight at the NATO conference because it said the operation of the data would be complicated in the future. In today's application, the data used in the contract is decided in the planning stage and could be XML (Extensible Markup

Language), JSON (JavaScript Object Notation), Java Object, etc., and the solutions are compatible with the work produced. Service contracts have an important function in problem-solving and are prepared either as service-based or consumer-driven contracts. The only difference between these contracts is collaboration. While you get to be the sole owner of the contract in the first, the latter has a close relationship between the service and the service consumer. Remote access protocols in Internet applications such as REST or SOAP operate according to the service-based contract principle (Flander, 2009; Richards, 2015). While service-based contracts can be modified by the owner regardless of requirements, this type of contract requires that the consumer complies with the rules. Thus, while healthy solutions are offered to complicated problems, challenging conditions also arise. This is one of the hidden parameters of the software crisis that was first discussed in 1968 and continues to exist even as complex, real-world problems are solved.

Even though the importance of the compiler was emphasized by many speakers of the first NATO conference due to complex problems, could d'Agapeyeff's following question be a prediction of the reduction of today's health system's electronic solution to the mobile platforms?

An example of the kind of software system I am talking about is putting all the applications in a hospital on a computer, whereby you get a whole set of people to use the machine. This kind of system is very sensitive to weaknesses in the software, particular as regards the inability to maintain the system and to extend it freely.

At the time, d'Agapeyeff was designing the bridge between the assembler and compiler by putting middleware between the application program and the control





program. d'Agapeyeff's speech describes the real-world problems of today, in which the inference rules are explained with description logic.

## Software Crisis and Chaos Reports

Chaos reports have been published regularly since 1994 by the Standish Group and have the same levels of complex problems. The lean software sector of the earlier years has developed over time and has spread across industries such as banking, financial, government, healthcare, manufacturing, retail, services, telecom, and others; time overrun in the projects has increased from 50% to 100% (Jensen, 2014; Mulder, 1994). The changes in project size, along with time-out in chaos demographics have also resulted in cost overrun and the limited rate of the performance of the target/scope. Thus, the evaluation criteria in the software projects beginning in 1994 underwent changes according to the conditions of the period, including the region where the products were developed. Despite this, the rates of successful, challenged, failed projects have not changed much since the 2000s. This is due to the systematic development of methods realized with technological advances since the adoption of software engineering as a separate discipline. Successful projects that were 16% in 1994 rose to 29% in 2004. This value is also the rate of successful projects in the Standish Group statistics from 2015. The success rates of software projects between

these years (2004–2015) have increased to 39%. An important reason for this increase is that the executive IT managers at the Standish Group asked project participants about three different statistical results used to solve the problems. Also, the concept of software project management gained great importance. The Software Capability Maturity Model (CMM) was developed by the Software Engineering Institute (SEI) as a standard and started to be implemented in the government and industry sector in 1991. The importance given to standards and project management can be a reason for the increased success of software projects. Continuous improvements by SEI were renamed as CMMI Developers and changed the standard that provided a combination of software and system engineering disciplines. Other process-improvement methods such as ISO 9000, Six Sigma, and Agile also contribute to institutes' timely delivery of service and software products with lower cost and higher quality.

## FROM EVOLUTION TO REVOLUTION

### 21st Century Software Products

In the early 2000s, software development and management processes of sectors such as traffic and tourism, banking, automotive, industry and trade, telecommunication





and media, and insurance on issues such as e-commerce, sales and support, order processing and supply chain were called industrial software engineering products. Large-scale software projects that began to be developed in those years included independent modules/components and were designed with architectures using cross-sectional functions. Tool generating and code fragments were also important in these projects. Effort, measured as person-years on an industrial scale suddenly rose up to hundreds of times. Accordingly, code lines reached into the millions in accordance with the size of the problems. With the start of the 2000s, software problems changed from individual solutions to team solutions. Thus, organizations' communication, management, and quality assurance requirements changed. New technological developments and new perspectives on project solutions are the main reasons for the increased success rates of software products developed between 2005 and 2015.

Evolutionary developments and revolutionary changes in the software world have influenced the solutions to complex, real-world problems. On the improvement of software and systems, the usage of mouse, the World Wide Web (WWW) and MapReduce can be considered successively as a revolution. Software products before the 2000s did not focus on the changing needs of customers, and the necessary changes were taking place very slowly. The efficiency of the product developed focused on the choice of priorities. But, by the 2000s, it was decided that the strategic outlook needed to be changed. This was an evolution that aimed to make the developed product have better value compared to its competitors. Since the scope of the problems were gradually expanding, solutions that focused on the direct satisfaction of users needed to be found. This intelligent evolutionary process that continues slowly is similar to the change of the hereditary characteristics of an organism population from one

generation to the other in biology.

The evolution process that was slow, reliable, and continuous turned into a revolution in terms of time and the direction of the requirements. The concept of Industry 4.0 is an example of the transformation from evolution to revolution; there is never enough time for developing the product at the end of the improvement cycles in the modern business world. Thus, problems are solved by taking too many risks. Instead of waiting patiently for the evolution of product development, change was accelerated by speculating revolutionary ideas. In some cases, the importance of the problem and the scale of changes needed on the old system made the implementation of new ideas much more effective because it left old concepts behind. Producing the expected results in the process of quick transformation to Industry 4.0 must be controlled; otherwise, the harms the system might face could exceed the benefits. An answer to the question, “Could an incompleteness in quantum mechanics lead to the next scientific revolution?” (Siegel, 2019) might be this: by rapidly ensuring





cybersecurity and safety, cranking the cryptography can occur very quickly in many protected networks (Herman, 2019).

## Digital Transformation

The revolutionary opportunities of many Internet of Things (IoT) applications such as driverless cars, smart grids, 5G, and cyber and space satellites pose challenges to the dream of switching to quantum computers. Today’s computers will never reach the targeted intelligence with the current technology because the current architecture is a structure that can perform logical reasoning on only a programmed idea-collection, so it cannot produce revolutionary ideas. Industry 4.0 (Kagermann, 2013) focused on automation, innovation, data, cyber-physical systems, processes, and people. Robotics, cloud computing, and the evolutions in operational technology are examples of the Industrial Internet of Things (IIoT). IIoT is a new computer term that refers to the processing of information technology with operational technology. IDC (International Data Corporation) announced in 2011 that big data/analytics, mobile, cloud, and social technology applications underwent a transformation called third platform technologies. This transformation brought a more modern look to the software world than the second platform technologies (i.e., personal computers, local area networks, and Internet and client-server architectures).

The history of using a large number of different applications by different users can be summarized as follows: tens of thousands of applications have been used by hundreds of millions of users in mainframe terminals since the second half of the 80s, while with client-server architecture, tens of thousands of applications have been popularized by hundreds of millions user in the 2010s. The goal of the 2020s has started to be realized in the form of smart intelligent solutions. Mobile broadband, big data/analytics, social business, cloud services, mobile devices, and apps will be used by billions of users is already an indispensable part of everyday life.

At the end of 2019, the cost of digital transformation is expected to reach $1.7 trillion on a world scale (Avanade, 2018). The world’s major companies use intelligence apps extensively and aim to increase customer experience by doing big

intelligence apps extensively and aim to increase customer experience by doing big data analytics. Banking, health care, insurance, and manufacturing, for example, are the leading sectors where the third platform is used. The banking sector highlights customer experiments in its analysis to support more customers. For example, by collecting data through call centers, they raise the quality criteria of the service.

Although such assessments increase the cost of the sector, the goal is to maximize the service by estimating the degree of nervousness and satisfaction from the customer's tone of the voice and the words they choose through AI applications. Health care services enable a detailed diagnosis of diseases and the monitoring of patients with intelligent solutions. Insurance companies perform intelligent enterprise with machine





learning and predictive analytics methods through a widespread network, repairing the organization's losses. Retail chains no longer consider geographic distribution and demographic diversity in sales and market strategies. Rather, the focus is to make online transactions attractive to the customer by means of intellectual corrections that are revolutionary in nature. In addition, information from the user's in-store experiences is measured to include the customer in the service performed and thus the collaboration factor is brought to the fore.

## Third Platform

The third platform is part of the technology system and the evolution of Web 2.0. In an increasing number of industrial products, meaning inference is made using new mobile technologies and their communication with each other using AI. This new structure plays an active role in the purchasing process, people's interactions with each other, shaping work environments, and all kinds of behaviors in daily life. With the emergence of this platform, cloud technology reduces the price of many components and the place of IoT in daily life will become more robust.

The third platform deals with what can be done with it as well as being a technology. In this context, it will take some time for the social, human, and business transformations to take place. Before explaining how business transformations take place, many existing projects that use applications such as Customer relationship Management (CRM) and Enterprise Resource Planning (ERP) have been developed to meet the needs of a single industry, or even a single customer, which increases the complexity of today's problems. However, with the third platform, business processes are simplified. Agility is now provided by receiving a standard application from the cloud and transforming it into compliance with industry-specific requirements. Here, a function of business transformation is to accelerate the transformation and ensure that those who use smart technologies are aware of the innovations and adapt to the system. Digital realization follows new technology's constant advancement such as AI, automation, IoT, and virtual reality/augmented reality.

Manufacturing signifies a revolution in the service area with digital transformation. The customer in the IT sector looks to pay no more than they would use instead of having the whole product. The manufacturer will benefit from the third platform with this business model change and will be able to complete the work at a lower cost by informing the customer in advance how to meet their requirements. By using the same product many times with different customers, the manufacturers will be

more profitable.





## HUMAN CONTRIBUTION

Previous studies that evaluate human satisfaction date back to the 40s. The development of electronic computers in 1941 and computers with stored programs in 1949 were the starting point for AI research. Newell and Simon (1956) took the first steps of modern AI with the computer program Logic Theorist. Being the pioneers of AI and cognitive science studies, Newell and Simon (1972) envisioned emotion directly in their theoretical research. Their first collaborative work on AI aimed to understand human problem solving, and they asked questions in three different categories related to the emotional experience:

1. What happened?
2. What can I do about it?
3. What did I do and what were the results of my actions? (Stein and et al., 1993).

Thus, the expression of the role of emotions in determining goal-oriented behaviors with talks was classified as event-based emotions (Bagozzi et al., 1998). This was also the person's affective reactions to events that are likely to occur.

### Emotional Experiences

The importance of emotional experiences in achieving the goals of the members in a software team has been increasing (Graziotin et al., 2018). The first examples of this were Extreme Programming (XP) applications (Auer, 2002; Beck, 1999). From the agile programming processes used for the first time in 1996, XP highlighted user stories with other agile product development features. As it is known, these are the criteria for the acceptance tests made by the user after the release of the product. This has brought a new perspective to software product development. In XP, the goal is an efficient (positive) release planning where the implementation time of each user story has been optimally determined. Thus, team members working in the pairing structure who concentrate on the target are drawn to a very intensive and stressful environment. In some cases, index cards are used to determine user requirements, while some user stories can be identified with a custom-built documentation tool. No matter which path is chosen to ensure the requirements, the contribution of the goal-directed emotions is important for the success of the developed product (Cao & Park, 2017).

Another principle of XP modeling and the values derived from them is refactoring. Because this process creates an environment with high tension, an individual's emotions are important for strong refactoring (Fowler, 2019). For example, according to a user story, refactoring may not be required, but developers definitely want

to refactor. The opposite is also possible. While developers do not recommend refactoring, refactoring can be included in the user story.

In an article examining three different real-world problems, the problems of the teams during the pairing were evaluated and similar feedback was received (Robinson & Sharp, 2005). This shows that social interactions are inherent in the principle of pairing. These are critical in the continuous integration of code to generate communication in the context of the next release.

The three important criteria of traditional software projects that use classic development models are cost, time, and scope (i.e. quality balance). In agile projects, the goal is to implement the release. This is because the established objective may change based on environmental conditions. It is possible for project managers to make unrealistic decisions when they aim to meet customer requirements. When the teams are forced to work cohesively, developers with different levels of knowledge can make decisions based on their emotional experience. Different developer behaviors and decision-makers cause the software product to succeed/fail, just as it did in the 70s. Thus, the goals of developing an on-time, cost-effective, and quality software product are transformed into successful product parts so the team can quickly respond to changes in user/customer requirements.

## Goal-Directed Emotions

In today's software engineering studies, agile software development teams play an important role in deciding goal directed emotions. Teamwork is the first criterion of the agile approach because it increases the pressure between team members and creates difficulties in managing a large number of stakeholders (Mc Hugh et al., 2012). In modern agile approaches, the customer is also part of the system developed as a stakeholder. Therefore, it is argued that users are perfect in themselves, while the task of ensuring this is in the development team. Team members carry different emotions with different appraisals. When the objective is reached during the operation, the member is happy. If it is not possible to correct any event that does not meet the determined purpose, this makes the whole team unhappy. Depending on the flow of the event, they may decide to change or maintain the current situation with the experienced emotions by the developer. Enacting objective behaviors by monitoring them is not only used in developing the product but also in planning the product and solving the problem (Stein et al., 1993). Another view is that emotional experience can be determined by inaccurate reports because representations of the events enable the understanding of the cognitive effects.

Affective agile design (Pieroni et al., 2017) has been proposed as a reflection of the principle of affective computing (Picard, 1995). This interdisciplinary field consisting of computer science, psychology, and cognitive science interprets and evaluates human

effects in solving the software problem. Italy's largest telecommunication service provider Fastweb launched a digital transformation project in 2016 and used agile methodology for the project. The developers used affective agile design as a new agile model, and they defined user stories on the sprint scale (Pieroni et al., 2018). The evaluation of emotional feedback in the project, where many criteria of agile approach were applied, was realized by investigating whether the implementation of the sprints responded to user requirements in different releases.

To determine the expectations of team members developing different tasks in agile projects, Yu and Petter (2014) utilized shared mental models theory to describe the interactions with each other. This means that human-agent teams effect the success of the project with goal directed emotions. For example, the developer's high cognitive performance has a positive impact on the project, giving an important component of creativity. Here, the relationship between specific moods in the individual's particular working conditions and their creativity is evaluated. A positive relationship naturally leads to innovation. Moreover, the criteria that hinder creativity in any project are individual or organizational must be distinguished. In summary, innovation and the agile methods required for innovation mean the implementation of successful software products together with the creativity of the happy developers. The result of high productivity also means high code quality.

Studies on the happiness/unhappiness of people in the work environment are increasing in the software industry. Graziotin et al. (2014) have investigated the behavioral effects of software engineering on the software product by evaluating human aspects and happiness situations according to the developer's experiences (Graziotin et al., 2014). They found that emotion and core affect are important not only in software product development but also in interface design. Emotions are increasingly placed among the qualitative criteria of research in many studies. Russell (2009) explored problems such as culture, language, emotional behavior, and coherence (i.e., the subjects of emotion theory for software developers) and labeled them unhappiness. Thus, the conclusion of a person's neurophysiological condition in the form of happiness/unhappiness in the working environment is caused by positive/negative factors around the person. Today's agile-based software product development methods are also consistent with Russell's views. The time problem of the software product developed in small parts never ends. No matter how small the part of the product is developed, there will always be problems during delivery to the customer. Teamwork means that the person is constantly nervous. The responsibility of the entire team in agile work will put the individual in the race.

All of these can also be evaluated from the perspective of possible negative effects of the product developed as a result of the unhappiness of the people. Graziotin et al. (2017) conducted a quantitative analysis of unhappiness factors using not only data from software developers but also answers to questions from researchers and



The Role of Science, Technology, and the Individual on the Way of Software Systems Since 1968

students from the GitHub archive. Personal results affecting the performance of the

developed product were expressed as low cognitive performance, as low motivation, as inadequacy, and irregularity. On the other hand, process-oriented results were summarized as low productivity, low code quality, and as a complete cancellation of the written results. Various unhappiness indicators such as mental anxiety may cause team members to withdraw from the project and this would delay, or even end the project.

## CONCLUSION

As in Boehm's (2006) keynote speech at ICSE, this study naturally follows a Hegelian approach. When the success stories of the past are not utilized, it is difficult to reveal future products. However, benefiting from existing success stories does not mean that the target will be successful. This is why software engineering is a complex discipline and its problems are very difficult to solve. In his speech, Boehm adopts the definition of "engineering" as:

> The application of science and mathematics by which the properties of software are made useful to people.

Boehm's new definition of the word "engineering" in his speech has obligated this chapter to start with the explanation of the information technology concept. With the birth of electronic computers in the 50s, the contribution of computing to applications was labeled as information theory. In the 1970s, the impact of information theory on software products has been applied to many phases of the software life cycle as measurements. Although many of the measurements at that period were not empirically validated, they led to each development stage of subsequent software applications. Since measurements are fundamental for high-quality software products, there is a lot of research about this. For example, Horst Zuse (1991) has defined a number of complex software metrics that were unable to be described in the context of the chapter. On the other hand, Konrad Zuse's (Horst's father) main contribution to the computer world was that he invented the first automatically controlled universal computer in 1941 (Zuse, 1999). This machine, which has not been able to be used in public, begun the evolutions in the following decades.

Shannon's Theory was the most fertile theory in the 20th-century, and it was the blueprint of the computer world not only with digital representation but also with data compression (JPEGs, MP3s, and ZIP files). Although there are some points that remain obscure about Shannon's theory, it sheds light on the problems related to the information quantum theory. Additionally, Shor's (1994) algorithm was



difficult to solve with existing computers (Coles et al., 2018). This algorithm was exponentially faster at breaking public key encryption than the fastest non-quantum algorithm. It took more than five years to solve this problem with a 5-qubit IBMqx4 quantum computer since no traditional hardware was possible that utilizes the proven quantum speed-up. Everything from 2-qubit quantum computers designed in 1998 by a few universities and IBM to the evolution of the 128-qubit computer announced by the Rigetti Company in 2018 shows the success of quantum physics in the 20th

by the Ridetti Company in 2018 shows the success of quantum physics in the 20th century. Of course, especially Turing, and also Gödel, Neumann, Zuse and many other mathematicians' contributions will never be forgotten in the quantum world of tomorrow. Before Shannon's information theory was inadequate to solve the information-intensive complex problems of the 21st century, Feynman and Manin (1999) argued that in the 80s, quantum computers could perform operations that are out of reach by regular computers. Today, a non-scientific problem, a theory-based algorithmic music composition, has been designed by quantum information (Kirke, 2019). This is a success story of the computer arts world contributing to quantum computing research. Abstract mathematics and physics underlie the evolutionary and revolutionary developments of the computers and complex problems solved by them. Moreover, quantum computing will change cybersecurity because of its speed, security, responsibility, safety, and resistance when compared to traditional computers. The factorization time of mathematically complex large prime numbers will reduce to a matter of seconds. Data transformation with superposition and entanglement will offer information processing benefits such as improved random number generation. But this technology will also arm the hackers.

In software engineering, it is more accurate to evaluate the human criteria starting from undergraduate education. New engineers will shape the next ten to twenty years of research, so education programs need to start preparing students for future trends by using constantly updated course content. Students who learn the importance of lifelong learning will realize the integration of science and technological innovations in the planning of the digital world of the future. Thus, Moore's law, which doubles the computing power of computers every 1.5 years, will have an enormous amount of operating power in 2025.

The increasing need for integrated software and system engineering is important for working groups and individuals during the transfer process from traditional computers to quantum computers. The first rule of developing successful user intensive systems is the importance of continuous learning. Every business model based on user programming and their unprecedented capacities depend on the quality of the individual and on their satisfaction. Since all stakeholders of a developing software product are important, the related person makes risk-based decisions when managing these relationships.





Another important criterion that needs to be evaluated is the kind of methodology to develop for the software product. Software development methods used to solve complex real-world problems must be chosen according to the type of problem to be solved, the structure of the organization, and the capabilities of the product developers. Any agile development methodology (e.g., scrum, crystal, XP, and feature-driven development) can be chosen when the target of the team is to minimize risks. DevOpps deployment methodology is preferred when the strategy is centered on organizational change. This methodology enhances collaboration between the departments responsible for different segments of the development life cycle, such as development, quality assurance, and operations. Rapid application development is a condensed development process that produces a high-quality system with low investment costs. Embracing modern software development practices is what will

differentiate one company from the other and to move it forward.

There are many other improvements for the solutions of current and future software problems that were not investigated in this chapter. They will continue to be examined in chronological order with more detailed and broader research.

## REFERENCES


Alwen, J. (2018). What is lattice-based cryptography & why should you care. Wickr. Retrieved from https://medium.com/cryptoblog/what-is-lattice-based-cryptography-why-should-you-care-dbf9957ab717

An, K., Meng, N., & Tilevich, E. (2018). Automatic inference of translation rules for native cross-platform mobile applications. Proceedings of ACM MOBILESoft '18. Retrieved from http://people.cs.vt.edu/~tilevich/papers/inference-translation-mobilesoft2018.pdf

Auer, K. (2002). Extreme Programming Applied Playing to Win. The XP Series.

Avanade. (2018). The intelligent enterprise: It's the way to predict and lead in your market. Retrieved from https://www.avanade.com/~/media/asset/thinking/intelligent-enterprise-pov.pdf

Bagozzi, R. P., Baumgartner, H., & Pieters, R. (1998). Goal-directed emotions. Cognition and Emotion, 12(1), 1–26. doi:10.1080/026999398379754

Bauer, F. L. (1987). An Interview with Friedrich L. Bauer. Retrieved from https://conservancy.umn.edu/bitstream/handle/11299/107106/oh128flb.pdf?sequence=1&isAllowed=y





Beck, K. (1999). Extreme Programming. The XP Series. doi:10.1109/TOOLS.1999.779100

Beck, K., Beedle, M., Cockburn, A., Fowler, M., Grenning, J., Schwaber, K., Sutherland, J., & Thomas, D. (2001). Manifesto for Agile Development. Retrieved from https://agilemanifesto.org/

Bennett, C. H., Brassard, G., Crépeau, C., Jozsa, R., Peres, A., & Wootters, W. K. (1993). Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. Physical Review Letters, 70(13), 1895–1899. doi:10.1103/PhysRevLett.70.1895 PMID:10053414

Berry, A., & Sigayret, A. (2004). Representing a concept lattice by a graph. Discrete Applied Mathematics, 144(1-2), 27–42. doi:10.1016/j.dam.2004.02.016

Binder, W., & Suri, N. (2009). Green computing: Energy consumption optimized service hosting. In SOFSEM 2009: Theory and Practice of Computer Science (pp. 117–128). Springer. doi:10.1007/978-3-540-95891-8_14

Boehm, B. (2006). A view of 20th and 21st century software engineering. In International

Conference on Software Engineering (ICSE'96), 12-29. 10.1145/1134285.1134288

Bohr, M. T. (2018). Logic technology scaling to continue Moore's law. IEEE 2nd Electron Devices Technology and Manufacturing Conference.

Borgida, A., Toman, D., & Weddel, G. (2019). On special description logics for processes and plans. DL 2019 International Workshop on Description Logics, 2373. Retrieved from http://ceur-ws.org/Vol-2373/paper-6.pdf

Borst, W. N. (1997). Construction of engineering ontologies for knowledge sharing and reuse (Ph.D. Dissertation). Institute for Telematica and Information Technology, University of Twente.

Brukner, C., & Zeilinger, A. (1999). Operationally invariant information in quantum measurements. Physical Review Letters, 83(17), 3354–3357. doi:10.1103/PhysRevLett.83.3354

Buxton, J. N., & Randell, B. (1969). Software engineering techniques report on a conference. Sponsored the NATO Science Committee.

Cao, L., & Park, E. H. (2017). Understanding goal directed emotions in agile software development teams. Emotions in Agile Software Development Team, 24. American Conference on Information System





Ceruzzi, P. E. (2012). A History of Modern Computing. The MIT Press. doi:10.7551/mitpress/9426.001.0001

Coles, P. J., Eidenbenz, S., Pakin, S., Adedoyin, A., Ambrosiano, J., Anisimov, P., . . .. (2018). Quantum Algorithm Implementations for Beginners. Retrieved from https://arxiv.org/abs/1804.03719

Collins, G. P. (2002). Claude E. Shannon: Founder of Information Theory. Retrieved from https://www.scientificamerican.com/article/claude-e-shannon-founder/

Deutsch, D. (1985). Quantum Theory. The Church-Turing Principle and the Universal Quantum Computer. Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences, 400(1818), 97–117. doi:10.1098/rspa.1985.0070

Dijkstra, E. (1976). A Discipline of Programming. Prentice-Hall.

Feynman, R. P. (1982). Simulating physics with computers. International Journal of Theoretical Physics, 21(6/7), 467–488. doi:10.1007/BF02650179

Flanders, J. (2009). Service station - More on REST. MSDN Magazine Block. Retrieved from https://msdn.microsoft.com/en-us/magazine/dd942839.aspx

Fowler, M. (2019). Refactoring: Improving the design of existing code. Addison Wesley.

Gil, Y. (2005). Description logics and planning. AI Magazine, 26(2), 73–84.

Graziotin, D., Fagerholm, F., Wang, X., & Abrahamsson, P. (2017). Unhappy developers: Bad for themselves, bad for process, and bad for software product. Proceedings of the 39th International Conference on Software Engineering Companion (ICSE-C '17), 362-364.

Graziotin, D., Fagerholm, F., Wang, X., & Abrahamsson, P. (2018). what happens when software developers are (un)happy? Journal of Systems and Software, 140, 32–47. doi:10.1016/j.jss.2018.02.041

Graziotin, D., Wang, X., & Abrahamsson, P. (2014). Software developers, moods, emotions, and performance. IEEE Software, 31(4), 12–15. doi:10.1109/MS.2014.94

Hartnett, K. (2019). A new law to describe quantum's rise? Retrieved from https://www.quantamagazine.org/does-nevens-law-describe-quantum-computings-rise-20190618/. Quanta magazine.





Health, N. (2018). Moore's law is dead: Three predictions about the computers of tomorrow. TechRepublic. Retrieved from https://www.techrepublic.com/article/moores-law-is-dead-three-predictions-about-the-computers-of-tomorrow/

Herman, A. (2019). The quantum revolution is coming, ready or not. Forbes. Retrieved from https://www.forbes.com/sites/arthurherman/2019/03/12/the-quantum-revolution-is-coming-ready-or-not/#3680ef55265a

Istrail, S., & Marcus, S. (2013). Alan Turing and John von Neumann - their brains and their computers. In E. Csuhaj-Varjú, M. Gheorghe, G. Rozenberg, A. Salomaa, & G. Vaszil (Eds.), Lecture Notes in Computer Science: Membrane Computing. CMC 2012. doi:10.1007/978-3-642-36751-9_2

Jensen, R. W. (2014). Why do we still have software development problems? In Improving Software Development Productivity: Effective Leadership and Quantitative Methods in Software Management. Prentice-Hall.

Kagermann, H., Helbig, J., & Wahlster, W. (2013). Recommendations for implementing the strategic initiative INDUSTRIE 4.0. Final report of the Industrie 4.0 Working Group. Forschungsunion.

Karypis, G., & Kumar, V. (1998). A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing, 20(1), 359–392. doi:10.1137/S1064827595287997

Kirke, A. J. (2019). Applying quantum hardware to non-scientific problems: Grover's algorithm and rule-based algorithmic music composition. International Journal of Unconventional Computing, 14(3-4), 349–374.

Lardinois, F. (2019). IBM unveils its first commercial quantum computer. Retrieved from https://techcrunch.com/2019/01/08/ibm-unveils-its-first-commercial-quantum-computer/

Lemos, A. L., Daniel, F., & Benatallah, B. (2016). Web service composition: A survey of techniques and tools. ACM Computing Surveys, 48(3), 1–41. doi:10.1145/2831270

Lodwich, A. (2016). Understanding Error Correction and its Role as Part of the Communication Channel in Environments composed of Self Integrating Systems. Retrieved from https://arxiv.org/abs/1612.07294

Lombardi, O., Holik, F., & Vanni, L. (2016a). What is Shannon information? Synthese, 193(7), 1983–2012. doi:10.100711229-015-0824-z

Lombardi, O., Holik, F., & Vanni, L. (2016b). What is quantum information? Studies in History and Philosophy of Modern Physics, 56, 17–26.





Manin, Y. I. (1999). Classical computing, quantum computing and Shor's Factoring Algorithm. arXiv:quant-ph/9903008v1

McClure, R. M. (2001). The NATO Software Engineering Conferences. Retrieved from http://homepages.cs.ncl.ac.uk/brian.randell/NATO/Introduction.html

McHugh, O., Conboy, K., & Lang, M. (2012). Agile practices: The impact on trust in software project teams. IEEE Software, 29(3), 71–76.

Messier, D. (2019). China's quantum satellite establishes photon entanglement over 1,200 km. Retrieved from http://www.parabolicarc.com/2019/06/19/china-quantum-satellite-establishes-photon-entanglement-1200-km/

Moore, G. E. (1965). Cramming more components onto integrated circuits. Electronics (Basel), 38(8), 114–116.

Mulder, H. (1994). The Standish Group Report Chaos. The Standish Group.

Musser, G. (2015). Spooky Action at a Distance: The Phenomenon That Reimagines Space and Time--and What It Means for Black Holes, the Big Bang, and Theories of Everything. Scientific American.

Naur, P., & Randell, B. (Eds.). (1968). Software Engineering: Report of a conference sponsored by the NATO Science Committee. Garmisch, Germany: NATO.

Neilsen, M. A., & Chuang, I. L. (2000). Quantum Computation and Information. Cambridge University Press.

Newell, A., & Simon, H. A. (1972). Human Problem Solving. Prentice-Hall.

Nguyen, A. T., Nguyen, H. A., Nguyen, T. T., & Nguyen, T. T. (2014). Statistical learning approach for mining API usage mappings for code migration. In Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering. ACM. 10.1145/2642937.2643010

O'Connor, J. J., & Robertson, E. F. (2003). John von Neumann. Retrieved from http://www-history.mcs.st-andrews.ac.uk/Biographies/Von_Neumann.html

OMG. (2011). Business Process Model and Notation (BPMN), Version 2.0. Retrieved from http://www.bpmn.org/

from http://www.bphil.org/

Ore, O. (1948). Number Theory and its History. Dover Publication.

Picard, R. W. (1995). Affective Computing, M.I.T Media Laboratory Perceptual Computing Section Technical Report No. 321. Retrieved from https://affect.media.mit.edu/pdfs/95.picard.pdf

Pieroni, A., Scarpato, N., & Scorza, M. (2018). Affective agile design: A proposal for a new software development model. Journal of Theoretical and Applied Information Technology, 96, 68–79.

Raattkaine, P. (2015). On the philosophical relevance of Godel's incompleteness theorems. Revue Internationale de Philosophie, 234, 513–534. Retrieved from https://www.cairn.info/revue-internationale-de-philosophie-2005-4-page-513.htm#

Randell, B. (2018). Fifty years of software engineering or the view from Garmisch. Paper presented at the meeting of the International Conference on Software Engineering (ICSE), Gothenburg, Sweden.

Richards, M. (2015). The challenges of service-based architecture. Retrieved from https://nofluffjuststuff.com/magazine/2015/10/the_challenges_of_service_based_architecture

Robinson, H., & Sharp, H. (2005). The social side of technical practices in extreme programing and agile processes in software engineering. Academic Press.

Rodriguez, N. D., Cuellar, M. P., Lilius, J., & Calvo-Flores, M. D. (2014). A Survey on Ontologies for Human Behavior Recognition. ACM Computing Surveys, 46(4), 1–33. doi:10.1145/2523819

Royce, W. (1970). Managing the development of large software systems. Proceedings of IEEEWESCON, 8, 328-338.

Russell, J. A. (2009). Emotion, core affect, and psychological construction. Cognition and Emotion, 23(7), 1259–1283. doi:10.1080/02699930902809375

Saxson, W. (1995). Dr. George Stibitz, 90, inventor of first digital computer in '40. Retrieved from https://www.nytimes.com/1995/02/02/obituaries/dr-george-stibitz-90-inventor-of-first-digital-computer-in-40.html

Shannon, C. E. (1948). A Mathematical Theory of Communication. The Bell System Technical Journal, 27(4), 623–656. doi:10.1002/j.1538-7305.1948.tb00917.x

Shannon, C. E., & Weaver, W. (1949). The Mathematical Theory of Communication. Urbana University of Illinois Press.

Siegel, E. (2019). Could an Incompleteness in Quantum Mechanics Lead to Next Scientific Revolution? Retrieved from https://www.forbes.com/sites/startswithabang/2019/04/17/could-an-incompleteness-in-quantum-mechanics-lead-to-our-next-scientific-revolution/#6dd6e2756ecb.

Stein, N. L., Trabasso, T., & Liwag, M. (1993). The representation and organization of emotional experience: Unfolding the emotion episode. In M. Lewis & J. M. Haviland (Eds.), Handbook of Emotions (pp. 279-300). Guilford Press.

Thackray, A., Brock, D. C., & Jones, R. (2015). Moore's Law- The Life of Gordon Moore, Silicon Valley's Quiet Revolutionary. Basic Books.

Timpson, C. G. (2004). Quantum Information Theory and The Foundations of Quantum Mechanics (PhD thesis). The Queen's College.

Turing, A. M. (1936). On Computable Numbers with an Application to the Entscheidungs problem. Retrieved from https://www.cs.virginia.edu/~robins/Turing_Paper_1936.pdf

Vincent, J. (2019). IBM's new quantum computer is a symbol, not a breakthrough. Retrieved from https://www.theverge.com/2019/1/8/18171732/ibm-quantum-computer-20-qubit-q-system-one-ces-2019

Wilde, M. M. (2019). From Classical to Quantum Shannon Theory. Cambridge University Press. Retrieved from http://www.markwilde.com/qit-notes.pdf

Xia, H., Chen, Y., Gao, H., Li, Z., & Chen, Y. (2010). Concept Lattice-Based Semantic Web Service Matchmaking. In International Conference on Communication Software and Networks (pp. 439-443). IEEE Explore.

Yu, X., & Petter, S. (2014). Understanding agile software development practices using shared mental models theory. Information and Software Technology, 56(8), 911–921. doi:10.1016/j.infsof.2014.02.010

Zuse, H. (1991). Software complexity measures and methods. Berlin: Walter de Gruyter. doi:10.1515/9783110866087

Zuse, K. (1999). Konrad Zuse. Retrieved from https://www-history.mcs.st-andrews.ac.uk/Biographies/Zuse.html

## KEY TERMS AND DEFINITIONS

Agile Methodology: It is an exciting and fascinating approach to software development. The agile process breaks a larger software project into several smaller parts that can be developed in increments and iterations. The purpose is to deliver a working software program quickly to the customer. The agile hierarchy in an agile team is based on competence, not authority. All members constantly explore ways to add more value to the customer. When compared with traditional methods, agile

projects qualify that they are better in terms of overall business value, higher in quality, less costly, more productive, and show quicker time-to-market speeds. Lean, Kanban, Crystal, Extreme Programming (XP), Dynamic Systems Development Method and Feature-Driven Development are a few of the agile methodologies.

Emotional Labor: Emotion in software engineering has recently gained attention in both the research community and industry. Work performance and team collaboration that are the fundamental criteria of today's software development approaches depend on the positive emotional effects.

Hegelian Approach:This approach depends on dialectics. The dialectical method is a discourse between two or more people holding different points of view about a subject but wishing to establish the truth through reasoned arguments. Hegel (1771-1831) defined four concepts: (1) everything is finite and exists in the medium of time, (2) everything is composed of contradictions, (3) one force overcoming its leads to crisis, and (4) change is periodic without returning to the same point.

IncompletenessKurt Gödel wrote the incompleteness theorems in the 1930s. They are still valid as the subject of several of today's discussions and will continue to be effective in the future. These theorems show that any logical system consists of either contradiction or statements that cannot be proven. Therefore, they help to understand that the formal systems used are not complete. On the other hand, the trend of mathematicians in the 1900s was the unification of all theories for the solution of the most difficult problems in all disciplines. Before Gödel defined any system as consistent without any contradictions, and as incomplete with all or some disproved statements, Hilbert formulated all mathematics in an axiomatic form with Set Theory. Gödel's theorem showed the limitations that exist within all logical systems and laid the foundation of modern computer science. These theorems caused several results about the limits of computational procedures. A famous example is the inability to solve the halting problem. A halting problem finds out whether a program with a given input will halt at some time or continue to run into an infinite loop. This decision problem demonstrates the limitations of programming. In a modern sense, this means that it is impossible to build an excellent compiler or a perfect antivirus.

Information Theory:This term, srcinated in Shannon's work, which started the digital age. There is not a single definition accepted in the literature and over time different definitions have been made in different studies. Generally, information theory lies at the core of understanding computing, communication, knowledge representation, and action. Like in many other fields of science, the basic concepts of information theory play an important role in cognitive science and neuroscience.

Ontology: In computer science, ontology is a formal representation of the knowledge by a set of concepts within a domain and the relationships between those concepts. The domain is the type of objects, and/or concepts that exist and their properties and relations. The creation of domain ontologies is the fundamental of

the definition of any enterprise framework. Every information system has its own ontology. Software agents can communicate with each other via messages that contain expressions formulated in terms of an ontology. It is important to use an ontology for the connection with the user interface component. The user browses the ontology to better understand the vocabulary and to formulate queries at the desired level. Application programs containing a lot of domain knowledge may not explicitly store information in the database for various reasons. In this case, it is possible to constitute a knowledge base by an ontology.

Software Conceptualization. During the elicitation of software requirements analysis, the customer's requirements are defined by visual diagrams or formal specifications (e.g., by the logic of first-order propositions, at the high level). An abstract solution closer to the customer side is realized. The design phase of the software visualization and the functional analysis of the previous development step are carried out in detail. Conceptualization at the design phase is closer to the implementation of the product.