VILNIUS TECH
Vilnius Gediminas Technical University

Creativity studies
ISSN 2345-0479 / eISSN 2345-0487
2021 Volume 14 Issue 2: 444–460
https://doi.org/10.3846/cs.2021.14699

CREATIVITY IN COMPUTER SCIENCE

Piotr GIZA *

Department of Logic and Cognitive Science, Faculty of Philosophy and Sociology, Maria Curie-Skłodowska University, Pl. Skłodowskiej 4, 20-854 Lublin, Poland

Received 5 April 2021; accepted 29 June 2021

Abstract. The aim of this paper is to briefly explore creative thinking in computer science, and compare it to natural sciences, mathematics or engineering. It is also meant as polemics with some theses of the pioneer work under the same title by Daniel Saunders and Paul Thagard because I point to important motivations in computer science the authors do not mention, and give examples of the origins of problems they explicitly deny.

Computer science is a very specific field for it relates the abstract, theoretical discipline – mathematics, on the one hand, and engineering, often concerned with very practical tasks of building computers, on the other. It is like engineering in that it is concerned with solving practical problems or implementing solutions, often with strongly financial reasons, *e.g.* increasing a company's income. It is like mathematics in that is deals with abstract symbols, logical relations, algorithms, computability problems, *etc.*

Saunders and Thagard analyse rich experimental material from historical and contemporary work in computer science and argue that, as opposed to natural sciences, computer science is not concerned with describing and explaining natural phenomena. Now, I argue that there is a field of research in artificial intelligence (which, in turn, is a branch of computer science), called machine discovery, where explanation of natural phenomena, finding experimental laws and explanatory models is the primary goal. This goal is achieved by constructing computer systems whose job is to simulate various processes involved in scientific discovery done by human researchers, and help them in making new discoveries.

On the other hand, motivations that give rise to ingenious projects in computer science can be very strange and include curiosity, fun or attempts to be famous out of boring, stable life of a successful programmer in a big corporation. A good example is the phenomenon of open-source software, especially the development of the *Linux* operating system and its applications when, from economical point of view, *Microsoft* absolutely dominated the software market of personal computers.

Keywords: artificial intelligence, automated discovery systems, communication analogy, computer science, creative society, creativity, natural sciences, technology.

Introduction

Computer science is a very specific field for it relates the abstract, theoretical discipline – mathematics, on the one hand, and engineering, often concerned with very practical tasks

*Corresponding author. E-mail: *pgiza@bacon.umcs.lublin.pl*

of building computers, on the other. It is like engineering in that it is concerned with solving practical problems or implementing solutions, often with strongly financial reasons, *e.g.* increasing a company's income. It is like mathematics in that is deals with abstract symbols and logical relations.

The aim of this paper is to briefly explore creative thinking in computer science, and compare it to natural sciences, mathematics or engineering. In the paper I draw on ideas of the pioneering work with the same title, by Saunders and Thagard (2005) who analyse rich experimental material from historical and contemporary work in computer science to highlight various aspects of creative thinking in computer science. In section 1, I summarize their considerations about the nature and origins of problems, motivations and methods of computer science, and how they compare to those in engineering, natural science, and mathematics.

The paper is also meant as polemics with some of their theses, because in section 3 I point to important motivations, specific to computer science, the authors do not mention, and in section 2 I give examples of the origins of problems they explicitly deny.

Saunders and Thagard argue that, as opposed to natural sciences, computer science "[...] is not concerned with empirical questions involving naturally observed phenomena, nor with theoretical why-questions aimed at providing explanations of such phenomena" (2005, p. 171). Now, in section 2 I argue that there is a field of research in artificial intelligence (AI) (which, in turn, is a branch of computer science), called machine discovery, where explanation of natural phenomena, finding experimental laws and explanatory models is the primary goal. This goal is achieved by constructing computer systems whose job is to simulate various processes involved in scientific discovery done by human researchers, and help them in making new discoveries.

On the other hand, in section 3 I argue that there are important motivations, specific to computer science and absent in engineering and natural sciences, which Saunders and Thagard do not mention. These, sometimes very strange, motivations may give rise to ingenious projects in computer science and include curiosity, fun, or attempts to become famous out of boring, stable life of a successful programmer in a big corporation. I argue with an extended example of the phenomenon of open-source software, specifically, the development of the *Linux* operating system and its applications when, from economical point of view, *Microsoft* absolutely dominated the software market of personal computers (PCs).

Section "Conclusions" concludes and summarizes the main theses of the paper.

1. Computer science *versus* engineering, mathematics, and natural science

Saunders and Thagard (2005) claim that computer science similar to engineering because it is often concerned with building machines and designing complex systems. Like engineers, computer scientists use established techniques solve their problems, and the creativity mostly consists in development of new techniques. Moreover, like engineering, computer science is typically concerned with practical questions of how to accomplish certain technological tasks.

At the same time, computer science, like mathematics, and unlike engineering, is largely concerned with manipulating abstract symbols, especially when software engineering rather

than building new hardware is concerned. This, they say (Saunders & Thagard, 2005, p. 159), distinguishes computer science from other engineering disciplines to the effect that the complexity of the objects created is limited by the skill and imagination of the creator and not by the properties of the raw materials. They even compare a programmer to a poet, who "[…] builds castles in the air, from air, creating by exertion of the imagination" (Saunders & Thagard, 2005, p. 162).

On the other hand, in other engineering disciplines and natural science, especially experimental natural science, any serious research requires considerable funds. This lack of material and financial limitations in computer science allows for "building castles in the air" and, as I show in section 3, allows for the development of ingenious projects, on a large scale, which would never have been possible in other areas of science or engineering.

Saunders and Thagard emphasize a strong mathematical component of computer science. They illustrate it with an example of Alan Turing's invention of a theoretical machine (now known as Turing machine) to solve David Hilbert's decidability problem, and the theory of the class of NP-complete, computationally intractable problems, as areas of research which belong both to mathematics and theoretical computer science (for an excellent discussion of these issues, also from philosophical point of view, see *e.g.* Rapaport, 2020, ch. 3, 8, 10).

Using numerous examples, the authors identify the nature and typical origins of problems and motivations in computer science. They argue that, like engineering, computer science (except for theoretical computer science) is concerned with questions of *how* to accomplish some technological tasks rather than *why* questions requiring explanation of natural phenomena, typical to natural sciences. As I show in the next section, this is not always true since the latter type of questions are also present in some highly creative branches of computer science.

Among sources of problems and motivations they list frustration with ineffective, boring and time-consuming existing solutions, both hardware and software, the pleasure of creating new computer programs or building new computers and, finally, commercial motivations in a company such as the desire to enter into a new market, to improve on an existing product and increase the income. To all this I would add important motivations, specific to computer science, like fun, curiosity, and boredom with monotonous life of a programmer in a corporation. I will illustrate it in section 3.

The authors also investigate, how computer scientists solve their problems and argue that, like all problem solvers, computer scientists use standard cognitive mechanisms like means-ends reasoning with rules, hypotheses formation, conceptual combination, and analogies to generate solutions to their problems and devote much space to the role of creative analogies in the development of computer science. They refer to another paper (Thagard & Croft, 1999) which investigates and compares problem-solving methods in scientific discovery and technological innovation and argues that, despite different kind of questions asked by inventors (also by computer scientists) and scientists, there is no reason to suppose that cognitive process underlying their solutions are different in the two domains. Accordingly, creativity in computer science seems very similar to creativity in the natural sciences and technology. Certainly, Thagard has much more to say about scientific problem solving in support of this thesis. Since the famous joint work written with other authors, *Induction: Processes*

of Inference, Learning, and Discovery (Holland et al., 1993), he investigates mechanisms of creative problem solving by humans, from initial conceptual analysis to numerous computer simulations and published the results in several books and numerous articles (see *e.g.* Thagard, 1992, 1993; Thagard et al., 2014).

2. The origins of problems in computer science and natural science

When comparing creativity in computer science to that in natural science, Saunders and Thagard (2005) observe that problems in natural sciences take the form of several kinds of questions: questions concerning mathematical relations between variables with observed values (*i.e.*, formulating empirical laws from data), more sophisticated, why-questions requiring explanation of observed empirical relations or questions (especially in biology) concerning explanation of the behaviour of complicated ecosystems.

They claim, that these sorts of problems do not occur in computer science:

> "Computer science is not concerned with empirical questions involving naturally observed phenomena, nor with theoretical why-questions aimed at providing explanations of such phenomena. Nor does it study naturally occurring mechanisms, but rather usually aims at producing new mechanisms, both hardware and software, that can provide solutions to practical problems" (Saunders & Thagard, 2005, p. 171).

I argue, however, that the matter is not always so simple as that, because there is a field of research called automated scientific discovery (or machine discovery), which is a branch of AI (which, in turn, is a branch of computer science) concerned with:

- Investigating natural phenomena;
- Formulating empirical laws and finding explanatory models;
- Postulating the micro-structure "hidden" behind observable phenomena.

These goals are achieved by constructing computer systems whose job is to simulate various processes involved in scientific discovery done by human researchers, and help them in making new discoveries. So in this highly creative area of research, the *why* questions concerning explanation of natural phenomena, typical to natural sciences, are on a par with questions (typical to engineering) of *how* to accomplish the task of creating successful machine discoverers.

A more thorough analysis of automated discovery systems would extend the scope of the present paper, therefore below I will only very briefly discuss some aspects of discovery systems to illustrate my point, redirecting the reader to my other publications on automated discovery (*e.g.* Giza 2002, 2006, 2018, Forthcoming).

The most famous research programme aiming at computer simulation of scientific discovery was developed by Herbert A. Simon and collaborators. The group created a number of discovery systems in physics, chemistry, and biology which discovered (or, rather rediscovered) many qualitative and quantitative empirical laws containing "theoretical terms". They also revealed the "hidden" structure of matter that could account for observable phenomena and, recently, some new systems were created, capable of searching for explanatory process models of complex phenomena, *e.g.* in ecosystems biology. Moreover, Simon's programme is the only one where researchers were interested in explicitly formulating deeper methodological assumptions involved in their work.

Although first publications of Simon on discovery systems go back to the 1960s, the fundamental methodological assumptions and most important results were described in a book he wrote with Langley, Bradshaw, and Zytkow (Langley et al., 1987). Philosophical and methodological assumptions of the approach presented by authors are best summarized with the following passage:

> "A hypothesis that will be central to our inquiry is that the mechanisms of scientific discovery are not peculiar to that activity but can be subsumed as special cases of the general mechanisms of problem solving. [...] Our method of inquiry will be to build a computer program (actually a succession of programs) that is capable of making non-trivial scientific discoveries and whose method of operation is based on our knowledge of human methods of solving problems" [...] (Langley et al., 1987, p. 5).

Using these methods, mentioned in section 1, and historical data about many discoveries in natural sciences, the group conducted the conceptual analysis of various aspects of scientific activity in several fields and created computer systems simulating processes, of discovery. They include, among others, finding problems, conducting experiments, formulating empirical laws, or discovering the hidden structure of matter.

Formulating empirical laws from data is the most successful and most frequently discussed by the commentators, area of the group's research. The most famous system of the group, BACON, was designed just for this purpose. Among numerous empirical laws known from the history of physics (re)discovered by BACON are: Kepler's third law of planetary motion, Black's law of temperature equilibrium including possible phase transitions, Ohm's law of electrical resistance, to mention only a few. The most famous of these, and most discussed by the commentators, is Kepler's third law of planetary motion: $\frac{R^3}{T^2} = const$, where R is the average distance of the planet from the Sun and T its period of revolution, and the most complicated one is the Black's law of temperature equilibrium, because it involves functions of many independent variables and the occurrence of nominal terms:

$$(c_1M_1 + c_2M_2)T_f = c_1M_1T_1 + c_2M_2T_2,$$

where: $c_1M_1T_1$ and $c_2M_2T_2$ are specific heats, masses, and temperatures of two liquids (*e.g.* water and mercury) mixed in a calorimeter, and T_f is the final temperature.

The successor of BACON, FAHRENHEIT (Zytkow, 1987), had improved procedures for determining the scope of laws and generation of experiments. Fahrenheit was able to discover more general forms of Black's law with phase transitions as well as their scope of application. It also postulated "theoretical terms" like melting heat, and had special algorithms for intelligent data gathering. Several years later (Żytkow et al., 1992) the system was enriched with new functions, such as the possibility of analysing the measurement error and its propagation, analysis of the repeatability of the results of experiments and finding maxima in the measurement data set, and successfully used in the electrochemical laboratory.

This is about discovering empirical laws from data, the area most noticed (and most criticized for inability of discovering new laws, previously not known by the scientists) by the commentators of the Simon programme.

Another, much less known but conceptually even more important area of research of Simon's group is discovering the "hidden", more fundamental, structure of matter that lies behind the empirical laws and could explain the observable phenomena, It addresses the second type of questions whose existence in computer science Saunders and Thagard (2005) explicitly deny in the passage quoted at the beginning of this section.

Systems created in this area of research include, for example, Langley's, Simon's, Bradshaw's and Zytkow's (1987) STAHL and DALTON, Rose's (1989) REVOLVER, or Fischer's and Żytkow's (1991) GELL-MANN. All of them postulate the structures not available by direct observation or measurement and, using mechanical, inductive methods, formulate, compare and justify hypotheses about these structures.

As I argue elsewhere (Giza, 2002, 2018), the most advanced and conceptually important of these systems is GELL-MANN. The program's task was to analyse elementary particle data available to physicists in 1964 and formulate a hypothesis (or hypotheses) about a "hidden", more simple structure of matter, or to put it in contemporary terms, to discover (or rather, rediscover) hypothetical, sub-elementary particles, called quarks. Its task was to generate quark models, which were to explain in the simplest way the huge and constantly growing number of particles discovered in the mid-20th century, previously considered elementary.

GELL-MANN is the first system which introduces attributes for postulated sub-elementary particles and performs an exhaustive search of all possible quark models to find the simplest model adequate to the data. Thus, we can talk not only about the search, but also about the justification of the specific hypothesis about quarks. Moreover, it dealt with a relatively new problem in theoretical elementary particle physics.

It must be emphasized that the system came up with a unique quark decomposition of elementary particle groups accepted by the physicists, proceeding in a radically different way, than human researchers actually did. As I argue in another paper (Giza, 2002) this coincidence constitutes additional strong evidence for the quark hypotheses. Using the terminology of an important distinction introduced by Cartwright (2002), scientists reasoned at the level of highly general, explanatory theories involving relativistic quantum field theory, and GELL-MANN came up with a quark model at the level of so called phenomenological laws, which describe properties of elementary particles.

GELL-MANN's successes encouraged further research in the field which resulted in new systems analysing reactions between elementary particles, making discoveries in the field of genetics, studying the structure of chemical reactions or studying, like GELL-MANN, the quark structure of elementary particles (Giza, 2006). All these systems have some common features: they build discrete models of hidden structure, postulating new properties or processes in various areas of research in physics, chemistry and genetics. Valdés-Pérez et al. (1993) emphasized similarities between these systems and proposed a common mathematical model describing the operation of systems revealing the hidden structure using matrix calculus, to represent discrete models and systematic analysis of the search space and the methods to limit the search with the help of knowledge from a given field of research, for example, conservation laws. The authors seemed to believe that their framework may in effect enable the creation of a unified discovery system capable of searching for many types of discrete models.

Langley, one of the authors of the joint book mentioned at the beginning of this section (Langley et al., 1987), returned, after a break in working on discovery systems, to research on a general system schema to model many aspects of the historical discovery process, but this time not in the field of empirical discoveries, as in the case of the BACON system, but in elementary particle physics. The BR-4 system (Kocabas & Langley, 2001) was created along the lines described above.

An important proof of the practical value of systems discovering the hidden structure is the fact that they appear more frequently in journals in specific fields of natural sciences than journals in AI (see *e.g.* Giza, 2006, p. 48; Džeroski et al., 2007; Langley & Arvay, 2019).

After Simon's and Żytkow's death in 2001, Langley, who joined Simon's group as a student of computer science and was the actual author of the BACON systems, continues with his collaborators, for almost two decades, work on discovery systems within a new framework called inductive process modelling (Langley et al., 2002; Bridewell & Langley, 2010; Park et al., 2010). This research falls under the third category of problems mentioned by Saunders and Thagard as not dealt with by computer science, namely explaining the behaviour of complex systems.

Inductive process modelling aims at modelling complex systems (like ecosystems in biology) in terms of processes and entities that explain the behaviour of a dynamic system. The authors illustrate their ideas with an example concerning formulating explanatory models of complex phenomena on the basis of time series data concerning the level of phyto- and zooplankton in aquatic ecosystems. Such process models consist of several processes described by ordinary and (recently) partial differential equations to explain spatio-temporal dynamics of the system (Langley & Arvay, 2019).

More specifically, the authors state the task of inductive process modelling in the following way:

Given: generic entities that have properties relevant to the observed dynamics;

Given: generic processes that specify causal relations among entities using generalized functional forms;

Given: a set of entities present in the modelled system;

Given: observations for the continuous properties of those entities as they change over time;

Find: a process model that explains the observed data and predicts unseen data accurately.

In recent research, Langley and Arvay (2019) have reported a new approach to finding explanatory models called rate-based process models which allows for more accurate and more efficient model formulation.

To sum up: contrary to Saunders' and Thagard's claims cited at the beginning of this section, questions concerning mathematical relations between variables with observed values (*i.e.*, formulating empirical laws from data), more sophisticated, why-questions requiring explanation of observed empirical relations or questions concerning explanation of the behaviour of complicated ecosystems are addressed in computer science, because they are fundamental in a branch of it, called automated discovery.

Finally, something has to be said about machine learning, which lies at the heart of contemporary AI research and applications, the more so, as the new research programme on

inductive process modelling was developed in cooperation with scholars involved in logical programming, which has been the first machine learning research programme in the area (Giza, Forthcoming). Before proceeding further we have to note the important difference between automated discovery and machine learning: automated scientific discovery (or machine discovery) deals with discovering new knowledge formulated in language used by domain scientists. It is based on the general idea that heuristic search in problem spaces applies also to discovery tasks. All systems of Simon's group discussed in the previous section fall into that category, although recent systems of Langley and collaborators, also use machine learning techniques. On the other hand, the aim of machine learning is to train software in order to induce rules from individual cases that can predict outcomes in future cases.

Machine learning systems are also used to discover regularities and tendencies in scientific data. Historically, the first such research programme has been the Turing tradition which, according its important commentator of the programme, Gillies (1996), originates from his own work on a theoretical concept of the famous machine (now known as Turing machine), and the research on cryptography Turing conducted with Donald Michie, who later became the most famous researcher in the field of AI in England, United Kingdom and contributed to spreading out Turing ideas. The group was spread out geographically and mostly cooperated using the Internet, creating as Gillies calls it, its own intellectual tradition, whose two key features were the use of logic and the programming language *Prolog* and emphasis on practical value of created systems in industry, medicine, and science.

The group managed to create a number of systems capable of inducing, mechanically, inference rules for expert systems on the basis of training sets of individual cases qualified by experts. Some systems were directly applied to scientific investigations, specifically, formulating laws from data. These laws, however were limited to very low level empirical generalizations of statistical character without the component of "theory" or causal explanation (Giza, 2018). It is worth mentioning, however, that since the first joint paper (Langley et al., 2002) written by a prominent member of the Simon group, Langley, and two outstanding scholars from Jožef Stefan Institute, Slovenia[1], researchers of the two groups have, for almost two decades, been collaborating in creating process-model systems in which Langley and his staff from Institute for the Study of Learning and Expertise, California, United States (US) contribute advance search algorithms and scholars from Slovenia, their logic-programming methods in the spirit of the Turing tradition in machine learning briefly described in next section.

With time machine learning dominated research and applications of AI in many commercial domains, the methods and their efficiency dramatically changed, and in addition to logic programming they include numerous other algorithms like decision trees, Bayesian networks and, most of all, neural networks, however the main idea remained the same: learning from examples and generalizing to explore trends and tendencies in data.

As is well-known, machine learning techniques, especially deep neural networks, work best on large amounts of data that contemporary experimental science abounds in (for a up-to-date technical survey of deep learning methods and their actual and potential applications in scientific research see *e.g.* Raghu & Schmidt, 2020). Last decades witness the phenomenon

[1] An important research centre in Europe, where machine learning methods in the Turing tradition were developed since the very beginning.

of "data deluge" and gave rise to the advent of data science which deals with application of machine learning techniques to explore trends in big data. This is also true in science, where stress has moved from theory to data and some, like Jim Gray (Hey et al., 2009), a world-famous data scientist involved in numerous first-rank scientific research projects in various fields, described it as the new, "fourth paradigm" in science. Finally, he says, today we witness rapid development of data exploration (*Microsoft* eScience Research Group) which unifies theory, experiment, and simulation. Data is captured by instruments or generated by simulator, then it is processed by software, information (knowledge) is stored in computer and, finally, scientist analyse database files using data management and statistics. Scientists only get to look at their data fairly late in this pipeline, says Gray. The way science is done has changed, and there is no question about that. Machine learning systems are widely used to analyse large datasets produced by complicated instruments to detect trends and anomalies. However, this enthusiasm is a bit premature.

Already mentioned outstanding experts in both machine learning and discovery systems Džeroski et al. (2007) argue that the output of a discovery system should be communicated easily to domain scientists and that notations developed by machine learning researchers, like decision trees or Bayesian networks, differ substantially from formalisms typical to the natural sciences, such as numeric equations and reaction pathways. Most work on computational scientific discovery attempts to generate knowledge which is easily communicable to domain scientists, but communicability is a significant issue for standard machine learning methods, independently of their admitted successes.

Moreover, at the end of the 2010s some researchers started expressing their worries that after decades of research, machine-learning systems so successful in various fields, in science are still at best capable of managing large datasets and detect very low-level regularities in the data deluge characteristic of the "fourth paradigm". It turned out that in science, as opposed to commercial domains, standard "black box" data science methods may easily lose their predictive power in the face of new data.

Karpatne et al. (2017) distinguish two characteristics of knowledge discovery in science that prevent standard machine learning methods from reaching the level of success comparable to other domains. One is related to the fact that problems in science are typically ill-defined, or under-constrained and as such they typically do not provide many training samples and involve many potential variables. Second characteristic of problem-solving in scientific research, which obviously differs from problems in commercial applications, is that in science the primary goal is searching for interpretable models and theories to explain known phenomena and predict new ones while the standard black-box methods of data science merely aim at generating actionable models.

The authors make an attempt to develop a general framework called theory-guided data science which they consider an emerging paradigm in data science applications in scientific discovery by introducing scientific consistency as an essential component (along with training accuracy and model simplicity) for learning generalizable and scientifically interpretable models.

The problem seems to me reminiscent of problems encountered by early machine learning systems in the Turing tradition, despite the fact that contemporary systems and methods,

largely using deep neural networks, are way more advanced and effective. Results obtained by such systems must be interpretable in terms of theories and domain knowledge, otherwise they are not reliable and robust and do not defend themselves in the face of new data.

Coming back to Saunders' and Thagard's claims cited at the beginning of this section, wide use of machine learning methods of data since for analysing massive amounts of data in contemporary experimental sciences, seems to run counter their claim that computer science is not concerned with empirical questions involving naturally observed phenomena. However, much remains to be done in order for data science methods to provide better generalizability of models by anchoring data science algorithms with scientific knowledge and by producing scientifically interpretable models.

3. Peculiar motivations and sources of problems in computer science

Motivations that give rise to ingenious large-scale projects in computer science, not mentioned by Saunders and Thagard (2005), can be very strange and include curiosity, fun or boredom and attempts to be famous out of boring, stable life of a successful programmer in a big corporation. A good example is the phenomenon of *Open Source* software, especially the development of the *Linux* operating system and its applications at the time, when, from economical point of view, *Microsoft* absolutely dominated the software market of PCs. To explain this, I have to start with a very brief history of operating systems in order to highlight the context in which highly creative work motivated by somewhat unusual reasons, specific to computer science, may arise.

Operating system is the absolutely crucial piece of software without which hardware is quite useless and whose tasks are so diverse as detecting and managing various pieces of hardware, managing the running processes in random-access memory, resolving conflicts in access to the resources, managing network connections, and creating an interface between users' programs and hardware (see *e.g.* fundamental work on operating systems, Silberschatz et al., 2008).

When, at the turn the 1960s and 1970s Massachusetts Institute of Technology, Cambridge, Massachusetts, US and *Bell Labs*, Murray Hill, New Jersey, US created a new operating system, called *Unix*, it was soon run on an *IBM PDP-11* machine. Simultaneously, the *C* programming language was created and *Unix* was rewritten in the new language. The purpose of the project was to create a unified (hence its name) operating system that could be run, at least, on many types of *IBM* machines and, with time, also on different architectures. Before that time each hardware architecture required its own operating systems laboriously written mostly in assembly language. Soon *Unix* dominated the mainframe computers and, with time, the Internet servers throughout the world and became the most complicated, stable, reliable, and expensive operating system (Silberschatz et al., 2008).

On the other hand, within a few years after the advent of PCs in the 1980s the software market for the new hardware platform was almost entirely dominated by products of *Microsoft*. The first operating system of *Microsoft* designed for PCs was, strangely enough, a version of *Unix*, called *Xenix* (Allan, 2001). Soon, however, it was replaced with disk operating system, more suitable for the mass user, and then with subsequent versions of *Microsoft*

Windows. Several important companies, like *IBM* with its *OS/2*, tried first to cooperate, then to compete with *Microsoft* and finally gave up.[2] Others, like *Apple Inc.* created its own version of a *Unix*-like system on their *Macintosh* PCs, but they only managed to hold about 10% of the market. Moreover, with their *New Technology File System* and network functions of subsequent *Microsoft Windows* editions, *Microsoft* managed to eliminate from the market the ex-champion of local networks, *Novel Writer RT* with its *NetWare* system. Adding to this the increasing popularity of *Microsoft* utility software with their *OfficeSuite* in the first place, no reasonable company could even dream about competing with *Microsoft* on the PC market, without the risk of loosing all invested money or, in the worst case, going down and even bankrupting.

Nevertheless, in April 1991, a 21 years old programming genius, Linus Torvalds, a student of computer science at the University of Helsinki (UH), Finland, created the first versions of the *Unix*-like kernel for the *i386* PC. He did it mostly for fun (see *e.g.* his autobiographical book, *Just for Fun: The Story of an Accidental Revolutionary*, written jointly with David Diamond (Torvalds & Diamond, 2001)) and, partly, because he was not entirely satisfied with a minimal, academic version of *Unix* called *Minix*. Actually Torvalds first created a terminal emulator for dialup connections from his home to the university main computer, then he created the shell and ported the open source version of *GNU Compiler Collection*. This made him relatively independent of the *Minix* installation in further work on the new system.

It must be emphasized that Torvalds was definitely not motivated by financial reasons and, as he himself admits (Torvalds & Diamond, 2001, p. 91), he would probably have stopped his project by the end of 1991[3] if it had not been for two things that happened around August, 1991: first, when experimenting with the newly created system, by chance, he seriously damaged his *Minix* installation and instead of re-installing the system he did not like too much, he decided to start writing a better *Unix*-like system, with its most important part, the kernel.

Second, people started to send him feedback as a reply to his first post from August, 1991 on *Minix* newsgroups, in which he informed that he had started to work on a new *Unix*-like operating system a couple of months earlier, and that it was nearly ready. This feedback essentially changed his attitude to the new system and was highly motivating: he realized that he was creating something interesting and useful to hundreds of users throughout the world (by the end of 1991) who downloaded it from the ftp server of the UH. But money, was definitely of no concern to him, maybe just a little, to the extent to which his friends announced on the Internet the collection of money to pay off Torvalds' new PC *i386* that he bought in instalments on January 3, 1991, both as a Christmas and birthday gift. In fact, instead of money he preferred postcards from people around the world, who started using the new system and considered it valuable.

[2] *IBM*, the original producer of PCs, at the beginning of the 1980s signed arrangement with *Microsoft* concerning a new operating system for its hardware. Fortunately, *IBM* was one the leading computer hardware manufacturers in the US so the defeat was not so severe.

[3] For Torvalds, creating the elements of a new system which he needed, was a challenge and fun. However, when the basic goals of a project were achieved, there remained laborious work of debugging the code, and Torvalds, at least initially, lacked motivation for this sort of boring, mundane activity.

Coming back to *Linux*[4], since from the very beginning, its source code was made available free on the Internet, its history has been one of collaboration by many programmers and users from all around the world, cooperating almost exclusively over the Internet.

The first publicly released *Linux* kernel, version 0.01, dated May 14, 1991 had very limited functionalities: it had no networking support and could run only on *Minix* filesystem.

However, the next milestone version, *Linux* 1.0, released on March 14, 1994 after three years of rapid development of the *Linux* kernel code, was already much more matured: it had full networking support (including device drivers for numerous Ethernet cards), it supported its own filesystem, much enhanced compared to that of *Minix*, virtual memory support and a wide range of hardware support.

As Silberschatz et al. (2008, p. 802) emphasize in a chapter devoted to the case study of *Linux*: in its early days, *Linux* development revolved largely around the central operating system kernel – the core, privileged executive that manages all system resources and that interacts directly with the computer hardware. We need much more than this kernel to produce a full operating system, of course. It is useful to make the distinction between the *Linux* kernel and a *Linux* system. The *Linux* kernel is an entirely original piece of software developed from scratch by the *Linux* community. The *Linux* system, as we know it today, includes a multitude of components, some written from scratch, others borrowed from other development projects, and still others created in collaboration with other teams.

Indeed, any *Unix*-like operating system has four basic components: kernel, shell, filesystem, and utility programs, and, as I mentioned earlier, Torvalds originally created the shell and started working on the kernel and (after crashing his *Minix*) the filesystem. With time, he concentrated on the kernel (with other programmers of so called kernel group) and other components of the system were developed in cooperation with numerous programmers from the Internet. Starting from the first kernel which only implemented basic services, the *Linux* system quickly developed to include most *Unix* functionalities.

This cooperation of many users and programmers gave rise to the advent of a completely new phenomenon on an unprecedented scale: free, open source software which developed into numerous projects like free operating system *Linux* with various distributions, its graphical desktops and windows managers, utility software, server software, web authoring tools and many others.

Actually, the idea of free, open source software was invented already in the 1980s by the founder of Free Software Foundation, Richard Stallman, who was also the author of the free software copyright licence, the GNU General Public License (GPL), according to which free software may be distributed. However, it was only with the advent of *Linux* and its GPL licence (since 1994) that it really got its impetus.

The basic principles of open source, in the case of an operating system, say that the source code, the programming instructions underlying the system, is free and available to the public. Anyone can improve it or change it, but those changes have to be made freely available. When a project is opened up, says Torvalds,

[4] The name *Linux* was given to the new system by Torvalds' friend, Ari Lemmke, who created for it the directory tree *pub/OS/Linux* on the *ftp.funet.fi* server, before Torvalds decided to give it a name and to make it available to the community.

> "[...] there is rapid and continual improvement. With teams of contributors working in parallel, the results can happen far more speedily and successfully than if the work were being conducted behind closed doors" (Torvalds & Diamond, 2001, p. 226).

In one of the final chapters of his book, under a significant title "Why Open Source Makes Sense", Torvalds calls this phenomenon the best method of creating and improving the highest quality technology ((Torvalds & Diamond, 2001, pp. 225–234). The advantage of open source over close, proprietary code methods used by software corporations is, as in the case of *Linux*, that

> "Instead of a tiny cloistered development team working in secret, you have a monster on your side. Potentially millions of the brightest minds are contributing to a project, and are supported by a peer-review process that has no, er, peer" (Torvalds & Diamond, 2001, p. 227).

In section 1 of the present paper I mention an important point made by Saunders and Thagard (2005) which should be restated here: like engineering, computer science is frequently concerned with building machines and designing complex systems. However, unlike engineering, the complexity of the objects created is limited by the skill and imagination of the creator and not by the properties of the raw materials. To this I would add technological facilities, housing base, and financial background without which projects of comparable importance and scale as *Linux* and its accompanying software, would never have been possible in engineering sciences, unless supported by large, innovative companies.

Having described the context in which peculiar motivations like fun and curiosity could give rise to the open source movement which involved millions of users and developers around the world, however, we must try to answer one important question: how is it possible that so many good programmers are willing to contribute to the development of free software for absolutely no money and what motivations could they have for their hard, creative work?

It is very interesting, how Torvalds himself tries to explain this phenomenon, which at least at first sight, seems to run counter basic principles of psychology, economy and even common sense: one of the least understood pieces of the open source puzzle is how so many good programmers would deign to work for absolutely no money. A word about motivation is in order. In a society where survival is more or less assured, money is not the greatest of motivators. It is been well established that folks do their best work when they are driven by a passion. When they are having fun. This is as true for playwrights and sculptors and entrepreneurs as it is for software engineers. The open source model gives people the opportunity to live their passion. To have fun. And to work with the world's best programmers, not the few who happen to be employed by their company (Torvalds & Diamond, 2001, p. 227).

He also claims that open source programmers are, to a large extent, motivated by the esteem they can gain in the eyes of their peers by making solid contributions, that they want to impress their peers, improve their reputation, elevate their social status. Open source development gives programmers the chance (Torvalds & Diamond, 2001, p. 122).

Despite the fact that Torvalds remarks that money is not the greatest motivator for programmers, he notices that as a result of rapid development of high-quality open source software, it gains momentum in the world economy, and its developers earn considerable recognition as potential employees. Companies instruct their human resources departments

to search credit lists, typically included in open source software, to determine who is making multiple contributions. So

> "Open source hackers aren't the high-tech counterparts of Mother Teresa. They do get their names associated with their contributions in the form of the 'credit list' or 'history file' that is attached to each project. The most prolific contributors attract the attention of employers who troll the code, hoping to spot, and hire, top programmers" (Torvalds & Diamond, 2001, p. 122).

So dreams of doing interesting things and being a bit famous instead of boring, stable life of a successful programmer in a big corporation may be truly motivating for many high-class professional programmers with established career. No wonder that quite a lot of them, after the *Linux* project matured in the mid of 1990s, joined the open source community.

To give a very short, typical example, which, at the same time is a sort of success story:[5] Donald Becker was a programmer at National Aeronautics and Space Administration where he worked on Beowulf high performance computing clusters. However, he became famous in the 1990s, for creating Ethernet software for the new *Linux* operating system – kernel drivers and utility software for numerous network cards (Sterling, 2001, pp. 69–70). In effect, he became the Chief Technology Officer of *Scyld Software*, a private supplier of high performance computing software based on Beowolf clusters, which became one of the leading cluster solutions in the *Linux* world.

Conclusions

I conclude by summarizing the major theses of the paper. As stated in section 1, I agree with Saunders and Thagard (2005) that problem solving methods in computer science do not differ from those used in engineering and natural sciences – the authors support this thesis with thorough investigation of numerous examples and, in addition, refer to research on various aspects of creative problem solving Thagard has conducted for many years. I also agree with their opinion that computer science resembles both engineering and mathematics: it is like engineering in that it is concerned with solving practical problems or implementing solutions, often with strongly financial reasons, it is like mathematics in that is deals with abstract symbols and logical relations.

This is not to say that I completely agree with all of their theses. In section 2 I give examples of the origins of problems the authors explicitly deny, and in section 3 I point to important motivations in computer science the authors do not mention.

I do not completely agree with the authors' thesis that, like engineering, computer science (except for theoretical computer science) is concerned with questions of *how* to accomplish some technological tasks rather than *why* questions requiring explanation of natural phenomena, typical to natural sciences. In section 2 I show that the latter type of questions is of the main concern in the field of automated discovery, a highly creative branch of AI (which, in turn, is a branch of computer science). Automated discovery deals exactly with

[5] The author of the present paper joined the *Linux* community in the mid of 1990s, as an application tester and passionate *Linux* server system administrator, starting to implement *Linux*-based, open source solutions on servers, desktops, and terminals. He knows this example from his personal, professional experience.

investigating natural phenomena, formulating empirical laws, finding explanatory models and postulating the micro-structure "hidden" behind observable phenomena.

Saunders and Thagard investigate typical sources of problems and motivations in computer science, like frustration with ineffective, boring and time-consuming existing solutions, the pleasure of creating new computer programs or building new computers or commercial motivations in a company to increase the income. In section 3 I discuss additional, rather peculiar motivations, specific to computer science that they do not mention, like fun, curiosity, and boredom with monotonous life of a programmer in a corporation. These motivations, together with specific lack of material and financial limitations in computer science which allows for "building castles in the air", may result with the development of top-quality projects on a large scale, which would never have been possible in other areas of science or engineering. I illustrate my argument with an extended example concerning the development of the *Linux* operating system and its applications.

References

Allan, R. A. (2001). *A history of the personal computer: The people and the technology*. Allan Publishing.

Bridewell, W., & Langley, P. (2010). Two kinds of knowledge in scientific discovery. *Topics in Cognitive Science, 2*(1), 36–52. https://doi.org/10.1111/j.1756-8765.2009.01050.x

Cartwright, N. (2002). *How the laws of physics lie*. Clarendon Press/Oxford University Press.

Džeroski, S., Langley, P., & Todorovski, L. (2007). Computational discovery of scientific knowledge. In S. Džeroski & L. Todorovski (Eds.), *Lectures notes in computer science: Vol. 4660: State-of-the-Art-Survey. Lecture notes in artificial intelligence. Computational discovery of scientific knowledge: Introduction, techniques, and applications in environmental and life sciences* (pp. 1–14). J. G. Carbonell & J. Siekmann (Eds.). Springer-Verlag. https://doi.org/10.1007/978-3-540-73920-3

Fischer, P., & Żytkow, J. M. (1991, 25–27 October). Discovering quarks and hidden structure. In Z. W. Ras, M. Zemankova, & M. L. Emrich (Eds.), *Methodologies for intelligent systems*, *Vol. 5: Proceedings of the 5th International Symposium on Methodologies for Intelligent Systems* (pp. 362–370). Elsevier Science Publishing Co.

Gillies, D. (1996). *Artificial intelligence and scientific method*. Oxford University Press.

Giza, P. (2002). Automated discovery systems and scientific realism. *Minds and Machines, 12*, 105–117. https://doi.org/10.1023/A:1013726012949

Giza, P. (2006). *Filozoficzne i metodologiczne aspekty komputerowych systemów odkryć naukowych*. Wydawnictwo Uniwersytetu Marii Curie-Skłodowskiej w Lublinie.

Giza, P. (2018). Sign use and cognition in automated scientific discovery: Are computers only special kinds of signs? *International Journal of General Systems, 47*(3), 193–207. https://doi.org/10.1080/03081079.2017.1414209

Giza, P. (Forthcoming). Automated discovery systems, machine learning and data science: New developments, current issues and philosophical lessons. *Philosophy Compass*.

Hey, T., Tansley, S., & Tolle, K. (Eds.). (2009). *The fourth paradigm: Data-intensive scientific discovery*. Microsoft Corporation.

Holland, J. H., Holyoak, K. J., Nisbett, R. E., & Thagard, P. R. (1993). *Computational models of cognition and perception. Induction: Processes of inference, learning, and discovery*. J. A. Feldman, P. J. Hayes, & D. E. Rumelhart (Eds.). The Massachusetts Institute of Technology.

Karpatne, A., Atluri, G., Faghmous, J. H., Steinbach, M., Banerjee, A., Ganguly, A., Shekhar, Sh., Samatova, N., & Kumar, V. (2017). Theory-guided data science: A new paradigm for scientific discovery from data. *Association for the Advancement of Artificial Intelligence Transactions on Knowledge and Data Engineering, 29*(10), 2318–2331. https://doi.org/10.1109/TKDE.2017.2720168

Kocabas, S., & Langley P. (2001). An Integrated framework for extended discovery in particle physics. In K. P. Jantke & A. Shinohara (Eds.), *Discovery Science. DS 2001. Lecture Notes in Computer Science*, vol. 2226. Springer. https://doi.org/10.1007/3-540-45650-3_18

Langley, P., & Arvay, A. (2019). Scientific discovery, process models, and the social sciences. In M. Addis, P. C. R. Lane, P. D. Sozou, & F. Gobet (Eds.), *Synthese library: Studies in epistemology, logic, methodology, and philosphy of science. Scientific discovery in the social sciences,* Vol. 413 (pp. 173–190). O. Bueno (Ed.-in-Chief). Springer Nature Switzerland AG. https://doi.org/10.1007/978-3-030-23769-1_11

Langley, P., Sánchez, J. N. J., Todorovski, L., & Džeroski, S. (2002, 8–12 July). Inducing process models from continuous data. In C. Sammut & A. G. Hoffmann (Eds.), *ICML '02: Proceedings of the Nineteenth International Conference on Machine Learning* (pp. 347–354). Sydney, Australia. Morgan Kaufmann Publishers Inc.

Langley, P., Simon, H. A., Bradshaw, G. L., & Zytkow, J. M. (1987). *Scientific discovery: Computational explorations of the creative processes*. The Massachusetts Institute of Technology. https://doi.org/10.7551/mitpress/6090.001.0001

Park, Ch., Bridewell, W., & Langley, P. (2010, 11–15 July). Integrated systems for inducing spatio-temporal process models. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, Vols. 1–3 (pp. 1555–1560). Atlanta, Georgia, United States. Association for the Advancement of Artificial Intelligence Press.

Raghu, M., & Schmidt, E. (2020). *A survey of deep learning for scientific discovery.* https://arxiv.org/pdf/2003.11755.pdf

Rapaport, W. J. (2020). *Philosophy of computer science*. University at Buffalo/The State University of New York.

Rose, D. (1989, 26–27 June). Using domain knowledge to aid scientific theory revision. In A. Maria Segre (Ed.), *Proceedings of the 6th international Workshop on Machine Learning* (pp. 272–277). Ithaca, New York, United States. Morgan Kaufmann Publishers, Inc. https://doi.org/10.1016/B978-1-55860-036-2.50076-X

Saunders, D., & Thagard, P. (2005). Creativity in computer science. In J. C. Kaufman & J. Baer (Eds.), *Creativity across Domains: Faces of the muse* (pp. 153–168). Lawrence Erlbaum Associates, Inc., Publishers.

Silberschatz, A., Galvin, P., & Gagne, G. (2008). *Operating system concepts*. Wiley.

Sterling, Th. (2001). *Scientific and engineering computation series. Beowulf Cluster Computing with Linux*. J. Kowalik (Ed.). The Massachusetts Institute of Technology Press. https://doi.org/10.7551/mitpress/1556.001.0001

Thagard, P. (1993). *Computational philosophy of science*. The Massachusetts Institute of Technology.

Thagard, P. (1992). *Conceptual revolutions*. Princeton University Press. https://doi.org/10.1515/9780691186672

Thagard, P., & Croft, D. (1999, 17–19 December). Scientific discovery and technological innovation: Ulcers, dinosaur extinction, and the programming language *Java*. In L. Magnani, N. J. Nersessian, & P. Thagard (Eds.), *Model-based reasoning in scientific discovery*. *Proceedings of the International Conference on Model-Based Reasoning in Scientific Discovery* (pp. 125–138). Springer Science+Business Media, LLC. https://doi.org/10.1007/978-1-4615-4813-3_8

Thagard, P., Findlay, S., Litt, A., Saunders, D., Stewart, T. C., & Zhu, J. (2014). *The cognitive science of science: Explanation, discovery, and conceptual change*. The Massachusetts Institute of Technology.

Torvalds, L., & Diamond, D. (2001). *Just for fun: The story of an accidental revolutionary*. HarperCollins Publishers.

Valdés-Pérez, R. E., Żytkow, J. M., & Simon, H. A. (1993, 11–15 July). Scientific model-building as search in matrix spaces. In R. Fikes & W. G. Lehnert (Eds.), *Proceedings of the Eleventh National Conference on Artificial Intelligence* (pp. 472–478). Washington, D.C., United States. Association for the Advancement of Artificial Intelligence Press.

Zytkow, J. M. (1987, June 22–25). Combining many searches in the FAHRENHEIT discovery system. In P. Langley (Ed.), *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 281–287). University of California, Irvine. Irvine, United States. Morgan Kauffman Publishers, Inc. https://doi.org/10.1016/B978-0-934613-41-5.50032-5

Żytkow, J. M., Zhu, J., & Zembowicz, R. (1992, 12–16 July). Operational definition refinement: A discovery process. In *Proceedings of the Tenth National Conference on Artificial Intelligence* (pp. 76–81). San Jose, California, United States. Association for the Advancement of Artificial Intelligence Press.