# Computing Origins, Future, and Challenges

*By: Manal Helal, PhD*

## Abstract

In this article, I am trying to present my thoughts about my career in computing as I progressed through it from undergraduate, to industrial experiences, to postgraduate, to post-doctorate experiences, and academic lecturer to recent undergraduates and postgraduates. Computer science and engineering degrees fluctuate between high appreciation and great expectations for the future at one end, and lack of trust and low appreciation at the other end. I conclude that there is no ultimate answer to this paradox in the near future. However, I have enjoyed my career so far, and expect to continue enjoying these fluctuations and accept them all.

## Origins

Computing is an organizational behaviour and problem solving techniques practiced since the early ages of humanity to reduce the complexity of daily challenges. Computer Science (CS) finds its foundation in mathematics. The word “programming” is first used to describe a system of equations that describes the relationships between the factors that affect the phenomenal occurrences of an observed event. The objective of studying these systems is to analyse causes, predict outcomes, or just explaining them. This has been practiced in all domains, such as: physics, chemistry, environmental conditions such as earth sciences and astronomy, human cognition, psychology, medicine, politics and social change, among practically all aspects of life. When it was practiced manually, it was coined the term “scientific method”, which is the systematic collection of observations and analysis of the collected data mathematically and statistically. When it started to get partially automated or fully automated, it was coined the term “computing”.

We have all seen the circulation of several videos by world leading business executives explaining how the future will witness the death of various jobs, and more and more reliance on machine learning and robotics to replace humans. We have already witnessed the sizing down of several firms and employment problems in several sectors, particularly in banking, finance, administrative jobs due to Enterprise Resource Planning (ERP) computer systems and databases with various business intelligence modules, ATMs, web and cloud computing for remote availability among many other technologies that dramatically changed our lives. Even highly specialized jobs such as medical, pharmaceutical, agricultural and earth sciences among others have started relying on semi-automated robots to achieve higher accuracy, speed, remote presence, and remote sensing as examples only. This transformation has social and human impacts that are widely studied in various departments, and alternative education systems and employment opportunities should always be created to cope with the change. This will require curriculum updates, all learning objectives updated, transferable skills gained, because probably all currently known skills will not be needed in

the near future. Anticipating this future and swinging with it, is a serious challenge. These transformations of professions are discussed in this book [1].

As a child in the 80s, I used to sign “Architect” under my name. I was fascinated by buildings’ external and internal architecture. However, when I reached undergraduate education, having a British High School Certificate, and some administrative shifts in Egypt, delayed my enrolment in Egyptian national universities. The American University in Cairo (AUC) was the only choice such that I do not miss the academic year. At that time in the 90s, there was Mechanical Engineering and Construction Engineering only in AUC. The final skills a student would acquire in construction engineering secure jobs in sites of construction, rather than pure architectural drawings and design that was available in other universities. Since I enjoy office relaxed environments, I decided to major in CS. A friend, who majored in mechanical engineering, sarcastically told me that I majored for a profession that is easily practiced by other discipline graduates if they cannot find jobs in their fields of study, while she is the “engineer”. Amazingly, she was 100% correct as noted in this open question article [2]. This article lists a number of quotes that better explain my frustration and waves of over-confidence followed by waves of depressions. CS programming is practiced without specialized education since the early times. My father has a physics degree in the 70s, and practiced CS right after graduation. Same did all the early pioneers of the field, who come from mathematics, physics, various engineering disciplines, financial and accounting, as a non-inclusive list; they all practiced programming at various levels up to establishing the foundation of the discipline, such as Alan Turing and Dijkstra. Even in recent years, professionals from all other disciplines have practiced programming to implement their own ideas without hiring a CS professional. I have worked in Australia with medical doctors who built their own web solutions using php language on their own. The first undergraduate degree in CS in Egypt was granted to a class in the mid-80s; probably worldwide it was not taught separately for a long time as well after its inception in the 60s. After almost 40 years of graduates with Computer Science (CS) and Computer Engineering (CE) majors, many if not all computing aspects are still practiced by other disciplines for various reasons. Possible reasons for this are: to save the communication time of explaining the requirements to a professional CS or CE; freedom in attempting to implement own ideas of possible solutions; to save the cost of another hire; to maintain the secrecy and do not disclose details to others; to have a pay raise, or promotion by adding CS and CE skills to one’s resume.

In my undergraduate studies, I started with command line programming, and in my final years, Graphical User Interfaces (GUI) was a recent advancement. I started with structured programming as the norm. Then in my final years, object oriented was a new paradigm for computing. Now it is the norm. Soon after graduation, web computing has become wide spread, and I had to self-educate myself on its foundations and keep up with its advances. Then various forms of distributed and parallel computing started to evolve. Sometimes the concepts remain useful while the names change, such as clusters, grids, and cloud computing all can benefit from distributed memory architectures (NUMA) and multicore as shared memory architecture (UMA). However, some other times the technology evolves and a mind shift is required to keep up, such as GPGPU. The dependence on libraries, operating systems, and even programming languages, could lead to a disaster of falling behind. None of what I was educated on in my undergraduate years exist today. However, the concepts do exist and help in understanding new emerging concepts.

## Is computing a science discipline or engineering discipline?

I have heard before the definition that science is studying the whole entities analytically until its tiniest atomic components are separated and understood independently, and the relationships between these components are identified. On the other hand, engineering is the study of various ways of assembling the separate components into various useful constructions to satisfy some objectives and requirements. In computing algorithms, this can be described as partitioning vs. agglomerative algorithms. Based on the ACM/IEEE-CS Joint Task Force on Computing Curricula 2013 (CS2013) [3], is divided into 20 knowledge areas, that are taught in 2 tiers in various levels, and choice of what is core and what is elective. These 20 knowledge areas are listed in Table 1.

**Table 1: Computer Science Knowledge Areas as recommended by The Joint Task Force on Computing Curricula 2013 [3]**

| # | Abbreviation | Knowledge Area | # | Abbreviation | Knowledge Area |
|---|---|---|---|---|---|
| **1** | AL | Algorithms and Complexity | **10** | NC | Networking and Communication |
| **2** | AR | Architecture and Organization | **11** | OS | Operating Systems |
| **3** | CN | Computational Science | **12** | PBD | Platform-based Development |
| **4** | DS | Discrete Structures | **13** | PD | Parallel and Distributed Computing |
| **5** | GV | Graphics and Visual Computing | **14** | PL | Programming Languages |
| **6** | HCI | Human-Computer Interaction | **15** | SDF | Software Development Fundamentals |
| **7** | IAS | Security and Information Assurance | **16** | SE | Software Engineering |
| **8** | IM | Information Management | **17** | SF | Systems Fundamentals |
| **9** | IS | Intelligent Systems | **18** | SP | Social Issues and Professional Practice |

The ACM/IEEE-CE Joint Task Group on Computer Engineering Curricula recommended in 2016, 12 knowledge areas for computer engineering degrees [4]. These are shown in Table 2.

**Table 2: Computer Engineering Knowledge Areas as recommended by The Joint Task Force on Computer Engineering Curricula 2016 [4]**

| # | Abbreviation | Knowledge Area | # | Abbreviation | Knowledge Area |
|---|---|---|---|---|---|
| **1** | CE-CAE | Circuits and Electronics | **7** | CE-PPP | Preparation for Professional Practice |
| **2** | CE-CAL | Computing Algorithms | **8** | CE-SEC | Information Security |
| **3** | CE-CAO | Computer Architecture and Organization | **9** | CE-SGP | Signal Processing |
| **4** | CE-DIG | Digital Design | **10** | CE-SPE | Systems and Project Engineering |
| **5** | CE-ESY | Embedded Systems | **11** | CE-SRM | Systems Resource Management |
| **6** | CE-NWK | Computer Networks | **12** | CE-SWD | Software Design |

The common knowledge areas between CS and CE are: Algorithms, computer architecture and organization, Networking and communication, operating systems, programming and software design, data security and encryption, software engineering and project management. We can see clearly that CS prepares students to various aspects of software development and problem solving. On the other hand, CE prepares students to hardware development as the infrastructure layer for software development. This makes both disciplines considered engineering disciplines, CS prepares for software engineering, and CE prepares for hardware engineering.

A commonly known fact among Egyptians in the 90s is that the basic difference between engineering degrees that qualify for engineering syndicate membership is the study of descriptive geometry subject. This subject is what develops the engineering sense and ability to project 3D surroundings on 2D papers from any perspective and for any parameterized change in scene contents and arrangements in objects, lights intensity, source and angle, viewing angle, and any possible involved parameter. This is studied in CS graphics courses that can be further studied theoretically in the field of computational geometry. The engineering sense is empowered when this is practiced from scratch and preferably manually by hand on a piece of paper. However, the existence of software tools that is programmed to perform this automatically is what reduced the engineering sense in recent graduates. The existence of these automation packages in practically every discipline is a major obstacle in education. Training students on using tools to start professions of applying these tools repeatedly to new datasets, will require continuous training to use new tools, and probably mind shifts to new emerging technologies later on. Besides, this approach prevents graduates from eventually participating in the generation of these software packages at the end.

## What, when, and where

Computing education gives us a number of questions to address. The answers to these questions will keep changing over time, and all possible answers can be valid for a particular purpose. These questions are: what computing topics to teach, and to what depth, and when to start teaching them, and finally in which department or discipline? Programming basics are being taught now in high schools. Moreover, computing subjects are taught in one or more courses in many other undergraduate disciplines, for example: mathematics and statistics, mechanical and civil engineering, finance and accounting, biology, chemistry, and physics. I also know that a political science college does offer CS courses as a major or minor for future politicians. I failed to understand the motivations for this establishment, but there is probably a reason politicians need to design software and hardware on their own. The presentation of CS as a service to other disciplines, created space for interdisciplinary education that was coined the term “computational science”. Computational science refers to the use of computers, algorithms, networks and high performance computing to do computational simulation, scientific experimentation, geometry, mathematical models, numerical analysis, and visualization to address computing problems in the sciences and engineering. These problems often model real-world changing conditions and spans various domains such as: geographic information systems (GIS) and earth sciences and observation, hydrodynamics (fluid dynamics, reservoir modelling, global ocean/climate modelling), atmospheric science (seismology), chemistry, magneto-hydrodynamics, astrophysics, environmental studies, nuclear engineering, biology, economics, materials research, medical imaging, animal science, structural

analysis, solid mechanics, civil engineering related transport phenomena, and other physical field problems in many science and engineering firms and laboratories [5].

These interdisciplinary sciences can be studied in three models. First, the specialized computational models can be studied within their specialized departments as elective or final year's specialization. This will add the burden of deciding how many courses from the fundamental CS, and up to which level of depth. Second, CS students study a few fundamental models that span a number of disciplines, and acquire extensive system analysis skills that enable CS professional to analyse any given data set for any defined objective. The second approach is now labelled as data science. The third approach is to start new degrees that specialize in the interdisciplinary domain, such as bioinformatics. The third approach assumes the problem will remain open area of research for long, and continues application of current methods to solve existing problems will require waves of graduates over a number of years. Often this is not the case. We see lots of problems started as an area that attracts funds and scientists, and soon becomes totally solved with off the shelf libraries solving it and parameterized to all possible variations. This leaves very tiny window for innovation in the same direction, and will replace human graduates with these automated systems.

## What is the end product?

If the end product is a software or hardware product, then this is a CS or CE contribution, even if the problem domain is in another discipline. Eventually an off-the-shelf library, toolbox, APIs, web service, special purpose hardware, or any form of delivering the end product can be generated. Once the product is finalized, well documented, and parameterized for re-use and customization, the enrichment of knowledge discovery in the target domain is what is practiced by the domain experts such as the use of Stata, matlab and R to perform traditional statistical analysis for a given dataset. Pre-processing the dataset could be a very challenging task using various data structure transformations, dimensionality transformation, feature extraction from text, images, audio, video, MRI images among many types of datasets. If the level of data pre-processing qualify for a knowledge contribution, it can be a credit to CS and CE professionals.

On the other side, the simple re-use of APIs can be using the same fundamental programming constructs that CS and CE professionals are trained on. These include but not limited to: library interfacing and inclusion, loops, selection statements, function calls, variables types and logical sequencing of these statements (algorithms and problem solving techniques). The level of proficiency manifested by all these programmers from other disciplines can vary. Some can simply use spread sheets built-in equations, Office APIs to automate a computation, simple Stata/matlab/R code, or even visual basic complete projects. Some can be advanced to the level of use of external databases, web programming, parallel, distributed and cloud computing techniques. No matter how advanced the programming done by non-CS and non-CE professionals, if the outcome has enriched the knowledge in another domain by simple application of existing CS and CE modules, then I think this programmer is still labelled as end-user of software and hardware products. The fear lies in educating CS and CE students to simple re-use of existing products through customizations only without being able to contribute to their production and advancements.

## Computer Science Crises

All the discussion above describes a case that was described as CS identity and education crises. The crises is the consistent need to update the curriculum from huge possible choices, and the reduction in the number of interested students to qualify to the job market that needs increasingly more of STEM (Science, Technology, Engineering and Math) professionals generally and CS and CE specifically. The description of the crises have been addressed from a psychological perspective and as a matter of perception as in [6]. Dijkstra in [7] questions the depth of hard science education in the computing degrees and the influence of business oriented education. More math courses need to be added to CS, such that graduates can contribute to the theory of algorithms and deserve the title "scientist" indeed. In any curriculum for a degree in CS, a minimum of 5 mathematics courses are required to accredit the degree by many organization. However, even 10:15 courses will not be enough to turn computer scientists and engineers into mathematicians. The independent study of math courses in later postgraduate studies for computer scientists and engineers usually becomes overwhelming as postgrads discover that they should have learned these topics earlier to facilitate the understanding of end products that they have focused on. This perception is what is sometimes described as "math envy" [6]. I actually describe it as scientific curiosity to learn the theoretical foundations to the studied end applications. This curiosity is useful for the further advancements of these algorithms. Postgraduates usually panic to be overwhelmed with math, and they should address this knowledge by comprehending the highest levels of abstraction first that covers the intuition and the applications. Then deeper levels of understanding can be developed only if needed and when needed. After all, CS and CE professionals will always be busy expanding these models to perform for bigger datasets, newer hardware technologies, further automation and features additions, and there might be limited opportunities in their career to need deeper understanding of these math models.

The question of business influence has become unavoidable. We have seen advancements in CS and CE coming from major industry companies such as: IBM, Intel, AMD, Apple, Google, Microsoft, Amazon, and many more. Education should try to generalize, and use case studies from the detailed product specifications offered by these companies. Research as well sometimes capitalizes on advancements offered by these companies to benchmark, compare, and develop theories from these products. It is very difficult to prevent the influence of business and industry on computing education, while in fact major advances in this field have been developed in research and development departments in industry. I think CS and CE are applied disciplines where there is no fine line between academia and business oriented industry. However, academic handling of industrial advancements can abstract the business objectives into scientific curiosity and underlying theories behind the end products.

## Research Methods

Research methods define the scientific approach to contribute to the body of knowledge. Three research methods are usually discussed: empirical, exploratory, and constructive. The empirical method is the most famous knowledge contribution coming from attempts to explain a phenomena or a dataset. This dataset is collected over a number of years, about a particular condition such as a disease, climate change, purchase patterns, customer behaviour, learning efficiency through analysing student outcomes and millions of other datasets. These datasets are available in many

databases and data repositories in many companies, web servers, university research projects, and practically everywhere. The traditional empirical research methods uses qualitative (when data are text, images, audio) and quantitative methods (when data is numerical or categorical) to describe using statistics packages the causes of an event occurrence, the prediction of an outcome, the correlation between factors, prove or negate a hypothesis. These datasets can scale up to form a big data problem. In these cases, traditional sequential off the shelf solutions might fail, and CS approaches become needed to migrate an existing solution using emerging technologies such as high performance computing, multithreading on multicore programming, GPGPU processing, mobile and web computing, cloud computing, migrating to distributed databases using any of the recent advances of NoSQL, visualization using graph databases and other data visualization techniques, and the list is endless.

The knowledge contribution can be in the form of identifying problems and their specifications such as in the exploratory research. Another form of contribution is to construct a solution for an existing defined problem. Since CS is often presented as a service to other disciplines, identifying the proper research methods and scientific methods to address a defined problem in another discipline is often confused with traditional empirical research methods. I believe that constructive research methods are more suitable to CS and CE, since their main knowledge contribution is the construction of software or hardware. The constructed product is either a breakthrough design based on no previous generation, or an optimization of existing construction to enhance the performance or add features. Since there is huge number of solutions that claim different features and performance promises, another knowledge contribution of CS and CE is benchmarking all these products to identify the suitability of each of them to various new problems, and to identify short comings that are not defined by the original authors, or simply confirm authors' claims.

There are two approaches in interdisciplinary computational problems. First, a solution is already designed, and the main objective for CS contribution is either one or all of the following: 1) the scaling up for larger datasets, 2) achieve speed up, or 3) higher accuracy in producing the results. Second approach is when the problem is defined in the other discipline, and CS is used to design a solution from scratch, by means of simulation, visualization, prediction. This endeavour requires iterative approach of revising results and fine tuning the solution until an established solution is documented as off the shelf package parameterized and ready for reuse.

As a person who was offended by being on a profession that can be practiced by anyone who cannot find a job in their field of study, I always try to quantify the CS contribution and resolve the paradox of why it is becoming that important, while being trivialized at the same time. I find the first approach of implementing a solution already designed by the other discipline, is when the CS contribution can be described as a "coding" exercise that doesn't qualify for a research contribution. Academic honesty requires acknowledging any contribution in terms of time and effort spent on all steps to final results. This means "coding" is not as trivial as it sounds when the "software" output is the main contribution of the research project.

Quite often there is a misunderstanding between having an already designed solution that only needs "coding", and having only a structured list of requirements and images of possible output, with no plan of the steps from analysis, design, implementation, testing, up to verification to reach this output. I find several research projects think they only need "coding", while they do not have

any step achieved in any software engineering model. In many cases, other disciplines need CS and CE researchers to design the solution from scratch, and only guide the requirement specification step, and the verification of results step. Since defining a problem and its specification is a major body of knowledge contribution, the chance of trivializing other disciplines contribution practically does not exist.

## Computational Grand Challenges

The computing grand challenges have been updated over the years by various institutions worldwide [8]. In the 60s and up to the 80s the grand challenges were focused on high performance computing. A number of important computational science problems have been identified such as: climate change, human genome, among others. The human genome project has been considered completed, and a new biological focus has been shifted towards assembling the tree of life and understanding biological systems. The trend of large scale computation such as in Big Data analytics continue to make high performance computing various models as a grand challenge. Data visualization have been proposed as well, machine learning and robotics, CS and CE education continuous advancements, and workforce development [9].

## Ethical Considerations

The IEEE code of ethics [10] states 10 points about ethical considerations to commit to while practicing any of the engineering and technological professions. Some of these points are difficult to quantify and set strict boundaries to their practice to separate an ethical from an unethical practice. Point number seven states the importance of honest criticism, evaluation of credit, and acknowledgement of the contribution of others. Since computing is presented as a service to all other disciplines, the evaluation of the credit to CS and CE professionals might always be a problem. There is no Noble prize for computing disciplines for probably the reasons discussed in this article.

The blame of this lack of appreciation does not go totally to non-computing disciplines only. My first employer was a pharmaceutical company, and the chemist CEO has strongly expressed his lack of confidence in all computing professionals. Later on, I managed to impress him and gave him some good projects to reduce this lack of trust. I believe the problem is collectively caused by some of the computing practitioners. The ethical practice in machine learning is discussed in [11], identifying three avenues for cheating: data, algorithms, and results. Not only machine learning, same considerations apply to all software products; on data input manipulations, processing data incorrectly, and misinterpretations or visualization of results. I have attended the lecture about research reproducibility [12] and find it very much relevant to all research outcomes not only in social and political research. The open source initiatives and availability of data on web repositories are examples of the main achievements in the last decade of technological advancements. The standards of data sharing, and processing documentations could be the answer to define what is ethical and what is not.

## Conclusion

There is no doubt that computing professions will survive and flourish for a long time to come. Incredible dependence on the technological products these professionals develop has been

witnessed over the last 6 decades from all disciplines. The dependence on technology will increase and is expected to replace currently performed manual tasks for higher speed, accuracy, and ubiquities. The computing education has definitely evolved from unrecognized, perceived with lack of trust and appreciation, to very much needed and appreciated, but disputed about distribution of credit. The distribution of the computing education from high schools, to various depths in various higher education departments is inevitable. People will always want to run their own projects independently. However, I believe the need will always exist for deeper knowledge of how the hardware and software are developed from scratch, to develop reusable, parameterized, generalized, complete packages with all possible features included for different types of end user choice of applications. This is why CS and CE education need to focus on the problem solving, analytical thinking and continuous learning and transferable skills, rather than specific products or technological detailed education that might not exist in a few years.

## Bibliography


[1] R. E. Susskind and D. Susskind, *The future of the professions: how technology will transform the work of human experts*. Oxford: Oxford University Press, 2017.

[2] H. S. Nwana, "Is Computer Science Education in Crisis?," *University of Cambridge*.

[3] ACM Computing Curricula Task Force, Ed., *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. ACM, Inc, 2013.

[4] Joint Task Group on Computer Engineering Curricula, *Computer Engineering Curricula 2016 - CE2016. Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering*. Association for Computing Machinery (ACM) IEEE Computer Society, 2016.

[5] A. B. Tucker, Ed., *Computer science handbook*, 2nd ed. Boca Raton, Fla: Chapman & Hall/CRC, 2004.

[6] Douglas E. Comer, "The Identity Crisis In Computer Science." https://www.cs.purdue.edu/homes/dec/essay.cs.identity.crisis.html.

[7] E. W. Dijkstra, "On the Cruelty of Really Teaching Computer Science," *Commun. ACM*, vol. 32, no. 12, pp. 1398–1404, Dec. 1989.

[8] R. Allan, "Computing Grand Challenges," CSE , CSE-HEC , STFC, Oct. 2010.

[9] "A Report of the National Science Foundation Advisory Committee For Cyberinfrastructure Task Force on Grand Challenges," National Science Foundation, Mar. 2011.

[10] "IEEE Code of Ethics." IEEE, https://www.ieee.org/about/corporate/governance/p7-8.html.

[11] A. Venkateswaran, "Ethics in Machine Learning," *Towards Data Science Sharing concepts, ideas, and codes.* 22-Jun-2017.

[12] L. Corti, "Transparency in Social Science Research & Teaching," presented at the The Data Dialogue: Time to Share, The University of Cambridge, 09-Aug-2016.