1 Introduction

Computer software development has become very expensive in these United States, and there are a number of issues that high cost has brought with it. I suppose the most important of these is that the programming jobs are going overseas, along with most of our other manufacturing jobs. We can claim that we still do high-technology design, and the critical jobs of testing and integration. But these jobs, even if we did them well (which we don't always do), will also go overseas, along with the making of our beloved Levi jeans.

There is also a silent acceptance within the computer science community that the quality of American software is poor, to put it mildly. This is well known, even to our major publishing houses. They note that we write fewer and fewer ''readable'' books that our young and not so young practitioners can use. We forget that the academic preparation of young Americans is the foundation of their professional lives. It is hoped that a long and serious apprenticeship lies ahead of them, until they, too, become masters of their professional fields. A large portion of our publications and books is written just to publish, and not to teach the young professional. When I pick up foreign textbooks, I'm astonished at the pains their authors took to make them readable and truly useful to the reader. Foreign books are not written solely for one's academic peers; they are written to teach the reader.

We Americans are the great innovators and inventors, of course, but such jobs generally are restricted to a limited number of research houses, facilities, and universities. The earning of our daily bread as software professionals is becoming more and more difficult as the jobs are going away.

Yet, we *are* able to produce software of the highest quality, and at a lower cost, here in this country than anywhere else in the world. At present, based on my experience, the Federal Government pays between $350 and $800 per commented line of code for ground systems (the command and control software for aerospace and military applications). And for on-board flight systems—the software that is used in missiles, satellites, aircraft, spacecraft, instruments, and smart-guided weapons—the costs are much higher, ranging from $450 to $1,200 per line of commented code. Such systems are the very reasons for our global military superiority.

In contrast to these staggering costs, my colleagues and I here at the Jet Propulsion Laboratory have been able to reduce the cost of ground systems to about $10 per

The Cognitive Dynamics of Computer Science: Cost-Effective Large Scale Software Development, by Szabolcs Michael de Gyurky

commented line of code on our most recent systems. During the past 20 years, my teams and I have produced high-quality software at a cost ranging from $10 to $50 per line of commented code—depending on the mathematical complexity of the application, the availability of the technology in the marketplace, and the schedule to which we had to work.

1.1 THE RETENTION OF SOFTWARE JOBS

As a computer software professional, I am greatly concerned about the retention of software jobs within our borders. My colleagues are the very best in their fields, both in applications and systems software, and are pure technical innovators as well. They are highly paid and worth every penny they earn. I pay my programmers from $45 to $85 per hour, according to their years and levels of experience and skill qualifications (most of my computer scientists are contractors and consultants, as opposed to employees). Yet, we complete our tasks within the schedules and on the budgets we have committed ourselves to, producing a high-quality product at a cost far lower than anywhere else I know of in industry or in government.

Clearly, as a team, we are neither better educated nor more experienced than the rest of our colleagues throughout the country. So why is there such a difference in cost? Were it not for the fact that we are losing our livelihoods, this would be a moot point. But there are many ''software guys'' out there, including my colleagues, who are trying in vain to find jobs. They are worried and are asking me questions. We have professions for which we have invested large amounts of time and money, and we need incomes to pay our mortgages and raise our families.

So how does a project build software at $10 per line of code, while others build at $800 or more for the identical product? That is the essential question that this book is going to answer, together with:

What does it take to build quality software at a reasonable cost?

What are the major factors that influence the cost, quality, and development schedule in software design and implementation?

1.2 DEPTH OF EXPERIENCE

This book is a summary of my 32 years of experience in designing and building computer software: 7 years in the U.S. Army building software, and 25 years at the Jet Propulsion Laboratory. This period of accumulated experience also includes the information I gathered as a software management consultant to some of the finest and best U.S. corporations—indeed, the giants in our industry. The problems of quality, cost, and schedule have proven to be the same, end to end, wherever I was invited to help out. The phenomenon of schedule vs. cost vs. quality in computer science holds true for Germany, France, Holland, Denmark, and the rest of the European countries as well. I have no experience with Japan, China, or India

directly, but there is no reason to believe that their computer scientists and software professionals are any different from ours.

I do not intend this to be "just another book on software," as some of my colleagues in industry have remarked dryly, but a worthwhile book to read. The intention is to pass on my accumulated experience in computer science, as well as the relevant experience gained in the combat arms of the U.S. Army, which in my case was the Airborne Infantry. Now, there will be those who will say, "What can you learn about computer science and software in the infantry?" Nothing about computer software, it is true, but everything about management, leadership, understanding people, discipline, planning, organization of work and people, as well as seemingly trivial subjects like how to staff through and coordinate action items efficiently. All of these are essential to our work in computer science and to the development of computer software. This is especially true for large systems—those above 250,000 lines of code with high factors of technical complexity combined with a short schedule. The absence of these skills will increase the cost of the product tenfold or more.

1.3 THE SCOPE OF THIS BOOK

The scope of Chapter 16, "The Autonomous Cognitive System," is much too large for the subject to be covered in one chapter and must be dealt with in great detail on its own. It is an essential part of the overall scope of this book, however, because it interprets the work of the great cognitive philosophers from the software architect's point of view. I began to approach the works of Kant, Hegel, and Schopenhauer as a way of looking at computer science many years ago; I never had the time to write my interpretation down until now. When designing and building large software-intensive systems year in and year out, back to back, one does not have the luxury to spend time writing, except for the design-associated books specific to a particular project.

The main theme of this book is *high-quality*, *low-cost software*, *built on schedule*; this theme will be re-emphasized throughout. It is a teaching point, and teaching points are just that: They are there to keep the mind focused, specifically on low-cost, high-quality products, on schedule. A minor theme is the inhibitors, those issues that prevent us from building elegantly, fast, and cheaply. These issues are also deeply rooted in the Kantian human thought system (the architecture) and the human thought process (the information flow), the understanding of which will enable a person to at least grasp the "why," even if one can't do anything about it. Understanding and learning from mistakes—one's own and the mistakes of others—is the great teacher and leads to ever better products.

It became very clear to me over the years that the successful building of computer software at a reasonable cost was dependent on several important factors:

- Leadership
- Management
- Communication

- Organization
- Understanding software development standards, architectures, and methodologies

A project lacking in any of these factors will see an increase in the cost of its software products. This book will discuss each of the factors affecting cost, from a practitioner's point of view.

1.4 THE NATURE OF COMPUTER SCIENCE

Additionally, two very important issues need to be discussed that also bear directly on computer science and the cost of computer software.

The first of these is raised by the following questions: Where do computer science and computer software belong in engineering? Should software be the dominant part in an engineering effort, such as an aircraft or spacecraft project, or should it serve a supportive role? This issue of computer science and its placement is causing a rub throughout the industry. It is an issue created by the mindset and outlook on the part of individuals. It is a divisive issue, and the resultant conflicts cost the customer money.

The second critical issue is raised by the questions: What is computer science really? Why did we rename it ''information technology?'' What is the complete definition of computer software? What is the role of philosophy in computer science?[1] Is it an important role? Why?

All one needs to do is look at the important recent publications in our field. We find that there seem to be no answers to our questions. Our profession used to be called automatic data processing. That evolved into computer science, then into information technology, and finally it has branched off into artificial intelligence and neural networks. What this really means is that while we are pushing the state of the art, we have no agreement as to a unifying science, like biology and chemistry have. Engineers deal with tangible facts and theorems, yet we deal in abstractions. Our profession is a combination of art, science, and engineering. So what is it? Why is it so important that we find a unifying principle?

1.5 THE FUTURE OF COMPUTER SCIENCE

We in computer science are headed full-bore toward total autonomy. This means we need to build a true robot, one that can be sent into space. This would be an autonomous system that, in an intelligent fashion and with an intelligence similar to our human intelligence, could explore our solar system and go to places where we humans cannot survive but from which we can benefit. As a starter, for the building

[1] Schopenhaur, Arthur. *Die Welt als Wille und Vorstellung* (The World as Will and Imagination) Gesamtausgabe (Complete Edition). Deutscher Taschenbuch Verlag GmbH & Co. KG, München 1998. Zweiter Band (Volume Two), page 149: ''Every science has its special philosophy...''

of such a system, we need an architecture. We need a model that is rational, functional, and logical, and one that can be built. This poses a serious problem because it involves a paradigm shift in the way we think about our profession and how we work in it. Paradigm shifts are dangerous events, as we all know. Galileo was nearly burned for his ideas, and Bruno was. There is an inborn anger in some of us human beings when we are confronted with a phenomenon we don't understand. We either learn, or we are left behind.

I feel that here in the United States we have a few great issues at stake. One is retaining our lead in computer science, and in software programming jobs. Another is pushing the state of the art "through the looking glass." This *must* be accomplished, regardless of the opposition.

1.6 THE ESSENCE OF PHILOSOPHY

Why is the work of the great cognitive philosophers so important for the development of high-quality software at a reasonable cost? It is because they were the ones who researched how we humans perceive, think, decide, and act, and why we do what we do. *A thorough understanding of philosophy is therefore necessary to do what we do efficiently and thoroughly.*

The essence of philosophy comprises many factors:

- How we think
- How we organize our thoughts and our work
- How we contemplate
- How we pay attention and listen
- How we decide
- How we form value judgments
- How we communicate with others

Yet, above all, the essence of philosophy concerns itself with how we treat each other. To me, therefore, it is the greatest of all the sciences. Small wonder that many of the great philosophers were mathematicians and physicists. This discipline, philosophy, upon which all the sciences rest, is given scant attention in the curriculum of our universities and by those of us who earn our living in engineering and the other sciences.

The greatest teachers (and I have had the good fortune of having had quite a few) always used stories from their personal experiences to illuminate teaching points. As students, we were more attentive because the learning process was so much more interesting when it was related to historical events. The teaching of geometry and mathematics becomes exciting when related to Harpalus (a great Greek engineer and philosopher) and how he bridged the Hellespont for Xerxes and the Persian Army. Teaching and instruction become more memorable than watching and listening to someone writing equations on the blackboard. How did he build that 1.6-mile-long pontoon bridge between Abydus and

Sestus?[2] Why did he build it there? What was the outcome for Greece and Persia?[3] As a matter of interest, Harpalus can be justly referred to as someone we call today ''a defense contractor,'' as were Archimedes and Histiaeus of Miletus[4] in their day. The Jet Propulsion Laboratory started as a U.S. Army Ordnance Laboratory, during the Second World War. Thus, before NASA was established, JPL, too, started out as a defense contractor.

There is one more very important aspect to using personal experiences in illuminating (or ''adding substance'' to) an object in a philosophical sense. As you will see in the architecture of the human thought system and the human thought process (Figure 1), the role of ''experience'' is one of the dominant roles in Kantian philosophy. How we apply our acquired knowledge, learned in the classroom, and solve problems depend largely on the level of our personal experiences (Figure 8). We acquire personal experiences by living life, and learning personally, or by having someone tell us of their personal experiences in life. Thus, the personal stories I recount in this book are to add substance to an ''object'' of a design, idea, or concept,[5] not simply to fill space.

Science and history taught together make learning complex subjects far easier. This is a method that is also used in the Army to reinforce a teaching point and to provide the student with a reference to the application of a skill, especially in subjects like engineering and tactics. It so happens that Combat Engineering was among the many skills I acquired through schooling in the U.S. Army. I specialized in bridge design and construction, pontoon bridges and timber trestles being my favorites.

I use stories from personal experience also to illuminate issues of management, leadership, and the architectural design process. I have a habit of doing this in my seminars and classes in order to reinforce the teaching point I am making. After all, the role of experience is one of the dominant themes of the *Three Critiques* of Immanuel Kant.

1.7 WHY AUTONOMY?

Why do we need a totally autonomous system, one that can act and decide on its own, without human intervention? The first thing that comes to my mind is the

[2] Herodotus. *The Peloponesian Wars*, Book VII, Chapter II. The Modern Library of the World's Best Books, Random House Inc. New York, 1942.

[3] J. F. C. Fuller. *A Military History of the Western World*, Volume I, Chapter I. Funk & Wagnalls Company, New York, 1954.

[4] Histiaetus and Thales actually built a 26-mile pontoon bridge over the Bosporus for Darius I ''The Great.'' It collapsed with tragic consequences to those on it. For details, read *The Peloponesian Wars*.

[5] Since Chapter 16 of this book culminates in my first thesis for an autonomous system, the reader must understand the correct technical definition of the terms *idea* and *concept* as these dynamic processes relate to the human thought process. The definitions of how these are formed are found in *Die Welt als Wille Und Vorstellung*. Arthur Schopenhauer. Drittes Buch. Seite 316. Deutsche Taschenbuch Verlag GmbH & Co. KG. Muenchen, Germany, 1998. (*The World as Will and Imagination*. Arthur Schopenhauer. Book Three. Page 316. German Pocket Books, Publishers. Munich, Germany, 1998.)

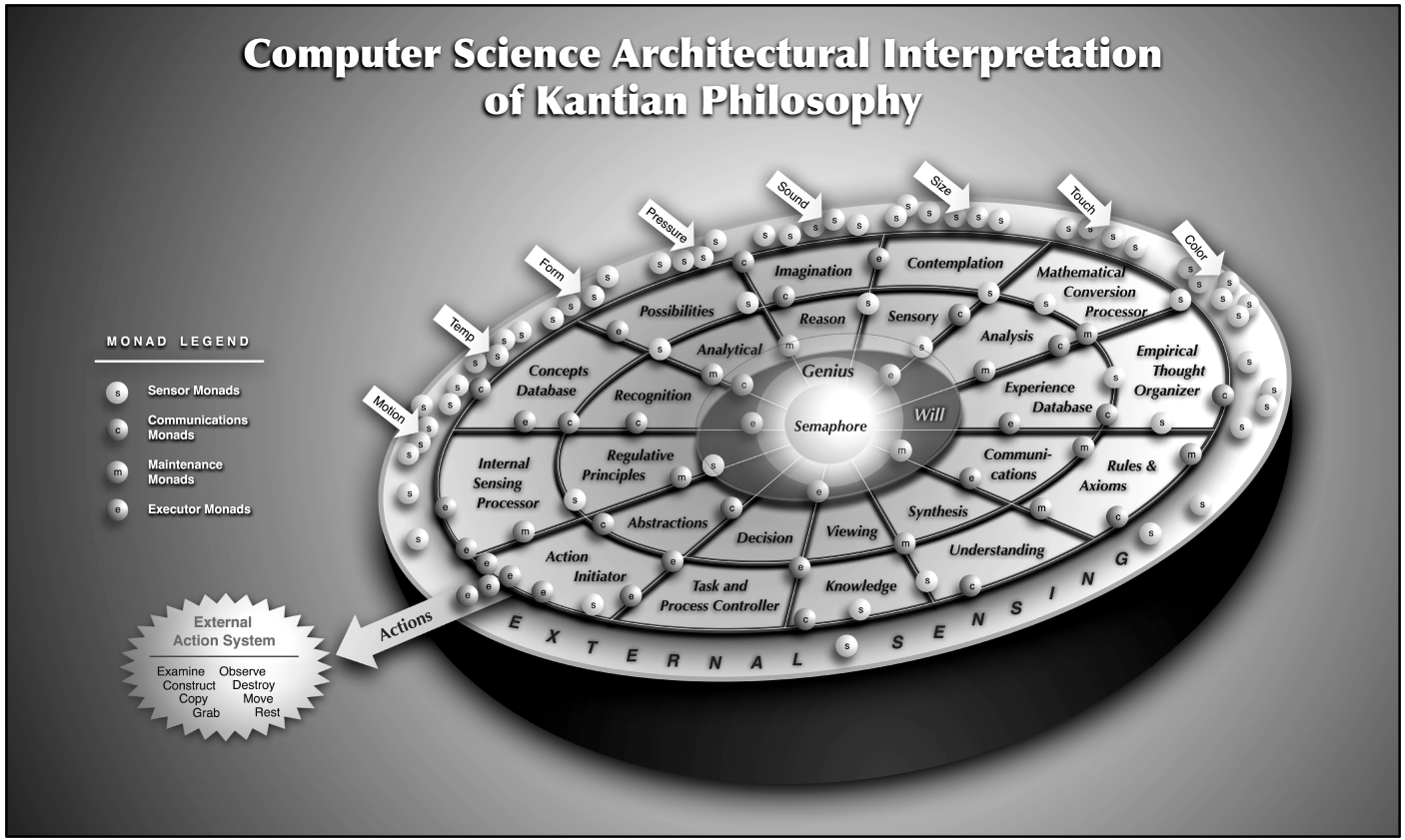

Figure 1. An initial impression of the functional organization of the human thought system.

exploration of deep space. All of space is hostile to human life, in fact, to all of biological life. Mars, Venus, and Jupiter, for instance, are all hostile environments, yet we must explore them because of the categorical imperative: ''I will, because I can.'' It would be infinitely easier to go to Mars if we had a crew of autonomous systems go ahead of the human crew. The autonomous systems would construct a fully life-supporting facility on Mars, test it, and provide the test data to the crew of the manned spacecraft prior to launch. There are also hazardous places on Earth, such as radioactive contamination sites like Chernobyl, the Hanford Reservation, and Oak Ridge that need cleaning up. Autonomous systems could perform the more dangerous jobs. Finally, there are the handling and disposition of biological and chemical agents, and the handling of unknown viruses and bacteria. All these tasks require autonomous systems that can learn, analyze, and decide on a logical course of action, with the human being still the suggesting superior, but not the controller.

One of the questions a colleague and friend asked me was, ''What will we do if we build it, and it turns out that it doesn't like us?'' Good question! I never thought that possibility through, and certainly I have made many mistakes in the past, not contemplating unforeseen possibilities. For example, as a young U.S. Army Special Forces demolitionist in 1962, I used to train with atomic demolitions ammunition. I had to disassemble and reassemble all of the uranium components with only a lead apron, lead gloves, and a pair of goggles to protect me. What all that did to me physically, I have no idea; I was only 24 at the time, and nobody told me about long-term effects.

So yes, we can build an autonomous system. How we control it is a question we have to resolve *a priori.*

1.8 AN ARCHITECTURE FOR AUTONOMY

The final chapter of this book is the chapter on autonomous systems. From the architect's point of view it describes the system architecture at a level that we refer to in computer science as a Level I architecture. Every system, by force of imperative, must have the architect's vision of how the system will look from an initially subjective point of view.

The greatest problem posed to today's professionals who are contemplating building a truly autonomous system is the architectural design. The reason is that current thinking in computer science is cast into the concrete of hierarchical-sequential logic. In the context that AI and neural networks are attempting to solve the problems posed by autonomy, this approach is unfortunately a ''cul-de-sac,'' or dead end. Human beings who fall into the ''normal'' category have a completely dynamic, nonhierarchical, nonsequential, and nonlinear thought and reasoning process. Our ''thought system'' (our functional architecture), our ''external sensory input,'' and our ''thought process'' (data handling) are so dynamic that they cannot be expressed by using simple traditional methods. The traditional approaches to illustrating the architectures of functional relationships are hierarchical and linear; these use the ubiquitous block diagrams and boxes.

It is in this arena of thought that the great classical German cognitive philosophers (e.g., Immanuel Kant, Arthur Schopenhauer, and Georg Wilhelm Friedrich Hegel) have laid the groundwork for us. When the collective "genius" of these three men is combined with the current state-of-the-art in the technology of computer science, we are looking at total autonomy as being within our grasp. Whether it is for the good of mankind or not is an entirely different question.

This still leaves the problem of how we are to express, illustrate, and articulate the architecture of an autonomous system. We must be careful at this point not to confuse software and hardware, and their equivalent in the human body and the human brain. The human brain is the equivalent of the computer hardware, but we are interested in the software that resides in the brain, the human cognitive system!

We would be using the human cognitive system as the model, but it must be done in such a manner that will be understood by those who will be building it. At this point the tools, processes, and methodology required to express the architecture become critically important, because if it is faulty or too complex, the engineering team will get lost in the details.

So, what is the alternative to the hierarchical and linear approach, to block diagrams and boxes? I personally have used spheres and circles in illustrating my designs during the past 30 years. I have expressed software architectures in this way because the systems I have designed and managed the development of were one-of-a-kind, unique, and mostly large systems built on constrained schedules with capped budgets. I have had a difficult time expressing these systems using the hierarchical approach and did not do so unless forced by my management. More often than not, they did not understand the context diagrams and architectural expressions anyway. The surprising thing was that my GDSS Design Team understood the concept, instantly, when I presented it to them in January 1986. As time went on, and my successes were followed by more successes, I was allowed to follow my methodology. I use Leibnitz "circles" and Euler "spheres"[6] to express the relationships between the functional attributes of the "human thought system" and the "human thought process." I also use them as a general approach toward the articulation and illustration of the functional relationship between computer software segments or subsystems.

1.9 OTHER NOTES

I would also like to bring the matter of the footnotes to your attention. Where I use references to foreign publications, it is only because that is the language I have read

[6] *Beitraege zur Berichtung bisheriger Missverstaendnisse der Philosophen*. Johann Michael Mauke, Jena, 1790. Theil von Abhandlung Nr. IV: *Ueber das Verhaeltniss der Theorie des Vorstellungvermoegens zur Kritik der reinen Vernunft*, S. 277–294. Gerhard Karls, Universitaet Tuebingen, Germany.

Contributions to the Commentary on the Misunderstandings of the Philosophers. Johann Michael Mauke, Jena Germany, 1790. Part of Essay No. IV, about the relationship of the IMAGINATION (with all of its attributes) to the *Critique of Pure Reason*.

them in. This is particularly true of the works of Kant, Schopenhauer, and Hegel. They are my main references and have had the greatest influence on my approach of how I manage and how I develop software. I translate the titles of the books for your benefit, but I cannot go back and reread the works in English for the benefit of this book; that would take years, and I'd never get finished.

My use of the German language in the original and in footnotes has its origins in my early college days. I started to use German books and publications for the clarification of technical and science problems that I didn't fully understand in my English textbooks, in particular, chemistry and engineering. This use of material in its original language became a habit and has remained with me throughout my professional career. I certainly would not have found the linkage between the classical philosophers and computer science had I not read their works in the original and developed an enthusiasm for them as a means of relaxation. While going about my business of designing software architectures and then managing their development, I suddenly realized that an autonomous system was doable and achievable in the near future.

As mentioned earlier, I use stories from personal experiences to illuminate issues of all the subjects covered in this book. I have acquired the habit of doing this from the teachers and instructors who were most effective in teaching me. I do this in my seminars and in my lectures to reinforce a teaching point I am making. There are many colleagues and acquaintances who cannot use personal experiences related to the constructs in computer science. These are friends who simply do not have the benefit of experience in the fields addressed here, or have only limited experience, having spent most of their career in the classroom teaching. They understand my style, however, and enjoy my references to practical and real situations.

Finally, the chapter on autonomy will be followed up with a volume purely dedicated to that subject, developed to a Level I architecture, enabling those who are interested and understand the systems concept to start the process of requirements and detailed design.