The impossible takes a little longer

Lecture delivered on formally accepting the chair for
Industrial Complex Computer Systems
at the
University of Amsterdam
on the 18th of October 1996
by
drs M. Boasson

Introduction

The impossible was about to be achieved! Never before was the first version of such a complex[1] system, at least for the primary functions, considered flawless. With a probability of success above 95% and many years of experience, everything indicated a successful first mission. Notwithstanding the almost proverbial uncertainty over the correctness of complicated software, the European Space Agency was so certain of its case that it announced the launch with much fanfare. Moreover, four very interesting and, it is said, irreplaceable scientific satellites would be brought into orbit. And yet, already in the early stages of its maiden flight, because of a programming error the Ariane-V had to be destroyed. Sadly, we must once again conclude that producing truly dependable computer systems is still not possible. And worse yet, we fool ourselves by making a complicated calculation concerning the dependability of the software, and even pretend to understand the result! The fatal error in the software of the Ariane was notably one of a most common kind [8].

Problem statement

This debacle was only the latest in a long line of incidents in which programming errors played a main role. I will mention another four examples, that may help get an understanding of the severity and the scale of this problem, and of the extent to which contemporary society, dependent on computer systems, is threatened by it.

The American industry has already spent a many years working for the Federal Aviation Authority designing a new system for supporting air traffic control. The cost has already surpassed one billion dollars, but no working system is in sight yet [5].

The malfunctioning of the computer-controlled baggage handling system at the new airport of Denver has delayed the opening of the airport by over a year. While the airport was opened eventually, the system still does not function well and a

[1] Complex: consisting of many different and connected parts, not easy to analyse or understand; complicated or intricate [2].

significant fraction of the luggage is handled manually; and every now and then, suitcases and boxes get squashed [10].

The British Ministry of Defence spent many hundreds of millions of pounds before abandoning development of the Nimrod-AEW, a locally-developed radar surveillance airplane inspired by the American AWACS[2].

Because of sequences of user input not foreseen during the design, a decade ago a number of fatal accidents in a number of hospitals occurred with a computer-controlled radiation therapy machine [7].

Considering the nature of the problems it seems justified to ask what is wrong with the field, and what we could and should do to improve the situation. Such questions are of course not new, and already were raised 25 years ago. Much has been achieved since through scientific research, and yet we have not progressed meaningfully since the problem was first noted. This is a unique phenomenon in technology, and in this lecture I will attempt to provide a tentative analysis of this problem and, even more tentatively, suggest a direction in which perhaps a solution may be found.

But what subject are we discussing? I will try to give an impression of the kind of problems that designers of complex systems face by using a simple example and avoiding getting caught in technical details.

An example

Imagine: the owner of a famous restaurant, renowned for its quality, wants to improve its competitive position. All traditional measures have already been taken and based on a sense of the possibilities of modern information technology developed by reading popular science magazines, he suspects that automating part of the operation of his business will be helpful in achieving this. The sporadic but very disrupting problems caused by the unpredictable behaviour of his employees, such as sick leave, bad moods and employee churn, should be reduced by this as well.

A investigation of the kinds of problems occurring in the daily operation of the business that would be amenable to automation or support yields the following high-level overview (with apologies to the real restaurateurs, who truly understand these matters):

Firstly, a menu needs to be put together. That is to say, a partial specification of what the restaurant has on offer for its customers is made. This appears easier than it is in reality: it is not a given that all required ingredients are always available, prices can vary significantly and may cause the profitability of the business to come under pressure, the choices on offer may be unattractive to some customers, etc.

---

[2] No public documents exist for this project.

The aspects that perhaps allow for support are the relationship to the kitchen equipment and the related decisions to prepare or not prepare certain combinations of dishes, and the effects on policies for buying and stocking ingredients.

Secondly, it must be possible to make a suitable choice from the wine list for all reasonable combinations of dishes. In this the restaurateur is highly dependent on the providers of what in an industrial setting would be called a "product component". Even though the quality is not directly controllable, nevertheless guarantees must be made for that quality. An expert system could perhaps provide support, especially in the relationship with the menu. A database could in storing data on quality and reliability of suppliers.

Thirdly, the stocking policy needs to match with a hypothesis on the behaviour of the customers that is based on experience. A top-notch restaurant can perhaps allow itself to occasionally fail at satisfying the desires of one customer, but stocking too much, especially when left unused for too long, is detrimental to the business. Simulation techniques, combined with methods for reasoning about uncertain situations appear to be the best aids for reducing these risks.

Fourthly, the salespeople enter into contracts with the customers. In this process, the partial specifications of the menu are (to the extent possible) completed, even allowing detailed changes in the offered dishes. One customer may for example dislike broccoli and request replacing that component of the product by something else. Another customer is in a hurry and asks whether it is possible to be done before a certain time, if necessary by simplifying some dishes. This activity depends heavily on human interaction, and the current state of the art does not seem to allow for automating this aspect. We may have Artificial Intelligence and even Cognitive Artificial Intelligence, but we are a long way still from Artificial Intelligentsia!

Finally, the kitchen needs to plan and perform the preparation of the ordered dishes. Here the following boundary conditions appear, among others: the amount of the equipment (ovens, kitchen machines, burners on a stove, etc.) as well as the availability of the personnel in the kitchen is limited, and yet, in certain cases strict requirements exist on the time between performing successive steps in the preparation of a single dish. Dishes prepared for customers dining together need to be delivered at the same time, even though the preparation times can vary greatly. Customers must not become impatient because of long waits between different courses; but the uncertainty in the time at which the preparation of a dish may start must not result in preparing it prematurely. Sadly, some customers, despite all efforts, will still be dissatisfied with the delivered product and request modifications: such requests have priority and send the carefully laid plans into disarray. On top of that, essential equipment can break down at critical moments, requiring improvisation; indeed, ensuring progress may at times even require accepting a reduction in quality. Obviously, a fully automated kitchen provides the perspective of a constant, high quality with minimal risk.

These issues are related in many ways to the difficulties of designing a complex industrial, computer-controlled system. There, too, it is often the case that it is

impossible to state in advance the expected load on the system; that a plan for a specific situation can become invalid at any point in time because of changing circumstances; that there may be insufficient resources to always complete all work on time and that progress can sometimes be more important than quality. Additionally, in practice there are the boundary conditions that a system must be delivered at an agreed-upon price and time. Both conditions are usually imposed without a deep understanding of the design problems to be expected.

Let us go over the steps that must be followed to obtain such a system, and do so following industrial practice.

Analysis

First of all, the scope of the automation must be determined, as must the requirements the system will have to meet. In this first stage of automation, it is decided to retain the traditional serving staff, so that the customer will not have to communicate with the system directly. The serving personnel however needs to be able to do that to inform the system of the dishes ordered by the customers. So too, the chef will need to work with the system, not only to "program" new dishes, but also to intervene when there is a crisis. Especially in this context, it will come as no surprise to you that the choice is made to use a modern user-interface based on menus. Furthermore, a choice is made for Bali, the hereby announced successor to Java, usable for real systems.

Then the equipment believed to be required is purchased, such as the computer-controlled kitchen equipment, robots, work stations, etc.; and only after that the work on the actual design of the system is begun. That such an early purchase is unwise is forgotten in the enthusiasm. Remember, it is still to be decided what characteristics and capabilities of the various components are minimally required for realising this complicated and business-critical system. The design task was difficult to begin with, the boundary conditions implied by this choice make it more difficult still.

Once it has been determined what exactly the system should do, the design of the system-as-a-whole can start, and in particular the design of the various software components. Systems design is not a simple task: many hundreds of components need to be designed to fit together as if they are pieces of a many-dimensional jigsaw puzzle, but also need to form a cohesive whole in their behaviour. The scale of such a development demands that a large number of designers work simultaneously, which in turn imposes strict demands on the design process. Because of this, it is nowadays common practice to require from a systems supplier that he can show the quality of the process followed. In a production environment where the various steps in the process do not require decisions to be made, the quality of the product will be directly related to the quality of the production process; in a design environment, the relationship is much less clear.

For example, it will not come as a surprise to anyone that making a good béchamel sauce is, apart from the quality of the ingredients, solely dependent on the quality of the process: careful stirring, control of temperature and a proper and timely introduction of ingredients guarantee an excellent sauce [11]. Such a process is therefore easily automated, without any negative effect on the quality of the product. Very different is, for example, the making of a painting, a design activity par excellence. Application of a process that results in a technically good result is indispensable for the durability of the painting, but by no means a sufficient condition for the creation of a work of art.

Designing a complex system resembles the creation of a work of art much more closely than it resembles pure production, and consequently there is no provable link between the design process and the quality of the resulting product. The essence of designing is weighing of and deciding among options, and doing so requires knowledge, insight and, sadly to often denied, intuition. The idea that the influence of these, for the quality of the final product essential, factors can be eliminated by prescribing in minute detail how the design process should be performed, is a fundamental error of reasoning. Perhaps there will come a day that specifications that are readable and understandable to domain experts can be mechanically translated into efficient programs, but for now this is but a vision of a far future. But even when that vision becomes reality, formulating these specification such that the system generated from it has the desired characteristics will still prove to be a non-trivial process. As so often in software- and nowadays also systems-engineering, here, too, the real problem is avoided: because we are incapable of measuring the quality of the various design decisions, we concentrate on the design process, in the idle hope that what holds for production will also hold for designing software and systems.

The desire to produce software of good quality by controlling the design process, must inevitably result in the process steps becoming very nearly meaningless. The minimal process step is roughly the production of one statement, but even that process still implies a design decision. Shrinking the processes to this level renders the describing of the relationships between these processes so complicated, that that will itself require a design process — and thus we sink deeper into the quicksand, rather than finding support. That this so-called process-programming has already gained a foothold in academic research is at best concerning. In this, it is remarkable that almost the entire western industry is conforming itself to the process-model, while at the same time making itself completely dependent on Microsoft, which consciously rejects it.

From these considerations, one must certainly not draw the conclusion that designing systems is exclusively a matter of intuition and luck. Just as the arts cannot exist without a well-developed technique — even Beethoven had to make a great many musically uninteresting exercises to learn the craft [3] — a system design of any quality can not be realised without a deep knowledge of the theoretical aspects of computer science and an understanding the management aspects of large-scale software development

One of the first steps is the creation of a so-called high-level design, for which the use of the latest fashion in technology is obligatory in 9 out of 10 cases. Today's infatuation with objects aims at formalising all concepts occurring in the system in a hierarchy of ever more detailed descriptions. Such descriptions (called objects) consist of both the variable aspects of the concepts and of the operations that may be performed on them, and that in a certain sense embody the characteristics of the objects. The power of the object-orientated model is in the implicit transfer of properties from a hierarchical level to the level below (the so-called inheritance mechanism) [9]. This is also the weakness of the model. In the system we are designing for automating the kitchen, we could for example create a class of objects that model all pots and pans used in the kitchen. Generic characteristics of all objects in this class are that they can contain raw materials and partial products, and that certain operations can be performed on the contents, such as stirring. More specific objects, such as pans, not only have these generic inherited properties, but also certain additional ones that do not hold for other subclasses, such as the possibility of heating up the contents using an external source of energy. Such a structure may seem logical and self-evident, but nearly always results in strange and unusable objects. For example: to what extent is a pressure cooker a pan? You can heat the contents from outside, but you can't stir in it! This type of problems can be solved, but the simplicity and elegance of the model are lost in the process.

In general, a hierarchical structure is better suited to analysis than it is for being used as a basis for a design. As a French colleague once put it: nobody would consider building a radio by first designing a generic receiver, and then specialising it for a certain frequency range by adding additional components. In software engineering such an approach is however very common, and this in itself is already a good reason to wonder whether it really is an engineering discipline!

Although object orientation can be very useful in certain cases, even for design purposes, this does not mean that all aspects of a system design should be approached in this manner. The software engineering community distinguishes itself from all other professions by taking what in principle is a useful idea, immediately generalising it and declaring it the ideal solution for all problems. That both industry and clients, and strangely also universities, then ascribe an almost religious faith to this latest philosopher's stone, can only be explained by a combination of incompetence and a lack of experience with the real problems.

Once all concepts and operations have been captured in objects, the design process can proceed to the next step. In it, a representation is chosen for each of the objects that allows it to be transformed, automatically or not, into an executable program. For a complex system with a large number of concepts, such as our restaurant system, this results in a collection of a very many, mutually dependent program components. Although all functional requirements should in principle be satisfied, we are still a long way from a functional system. Indeed, the real problems are only starting now. The first problem will likely be the discovery that the programs do not fit in the available (prematurely acquired) machines. Once that problem has been solved by, on the one hand, buying additional machines, and, on the other hand,

revising parts of the programs, usually requiring adding water to the object-wine[3], problems with the timing and utilisation of equipment will surface. Some examples illustrating the kind of problems are: because of an unexpected instability of the planning function, in exceptional situations the system attempts to put two pans on the same stove simultaneously; preparing dishes sometimes takes too long without any discernible cause, giving rise to customer complaints (the technical term starvation is very appropriate here!); kitchen-robots often enter each other's working area and cause deviations from the plan, if not worse. Solving these problems is difficult, in part because an exhaustive description of the interactions cannot be given because the set of combinations of recipes the system has to deal with is not known beforehand, and in part because of the followed design methodology. The consequence of the latter is that the global state of the system is almost impossible to relate to the actual situation in the kitchen, and moreover is almost impossible to analyse. In addition to that, an important aspect is that the system was considered too large and complicated for a single designer, and that it was therefore designed by a large team. The result is that no one understands the system in its entirety in sufficient detail to explain the observed phenomena and to prevent as-yet unobserved problems through reasoning. A sufficient solution for this problem would require reconsidering a large fraction of the design decisions taken, but fear of delaying delivery beyond the agreed-upon date causes the management to decide there is insufficient time left do to so. Instead, the design team is grown, with the well-known consequence that, though two know more than one, progress is slowed by an explosion in communication, misunderstandings and incompatible ideas. It is self-evident that these circumstances do not improve the quality of the system.

The restaurateur's legitimate question about the probability of occurrence of errors that would be annoying to his customers, is answered by showing a complicated calculation, indicating that that probability is less than one in 10^{10}. This is an excellent value, and for example the norm for control software in civil aviation. Unfortunately, nobody can give a reasonable interpretation of what this means in practice. The difficulty is rooted in a lack of knowledge of where the errors are located (they would have been solved otherwise) and any statement about the occurrence is therefore completely arbitrary. The only thing that can be stated is that the system appears to work well in a very limited number of cases. An error can however have occurred during the testing period, remained unnoticed, and yet at a later time cause serious problems.

With great effort[4] a working system is delivered that appears to meet the requirements formulated by the client. That not all requirements could be met had

---

[3] A literal translation of the Dutch "water bij de object-wijn doen", for which I have failed to find an appropriate idiomatic translation that is equally appropriate in this context.

[4] The original uses the idiom "met stoom en kokend water" — with steam and boiling water — posing a similar problem to this translator.

already become clear during design, but explaining to a client that he has fundamentally impossible requirements is unpleasant, and uncommon in this business. After all, the client can't show that the system does not meet all requirements under all possible circumstances anyway. It is much easier (and superficially considered more lucrative) to ask for some extra time and to hope that the pathological cases do not occur (you understand: the impossible just takes a little longer …). In defence of the system designer, it should be remarked that out of ignorance, clients can have the wildest expectations. In political circles in The Hague it was recently stated that road pricing should be introduced quickly, because surely being able to send men to the moon and back means that road pricing can't pose technical problems! Independent from the question whether the problems involved in it can be solved with the current state of the art, this line of reasoning shows gross ignorance and a complete lack of understanding. Similarly, the growing enthusiasm about the almost limitless possibilities of a public-transport chip card very facilely ignores whether the correct functioning of such systems can be guaranteed.

After several years of very satisfactory use of the system (the pathological and unsolvable problems happily never resulted in catastrophic events) a desire to extend its features and capacity surfaces. In buying new equipment, careful attention is paid to the compatibility of the specifications with those of the equipment to be replaced, to ensure the existing and field-proven software will continue to function without requiring modifications. Only some isolated parts of the system need to be adapted as a result of extending the capacity, such as some functions for planning the use of equipment, while extended automation merely requires adding some new functions.

The restaurateur is confident that the modernisation will be trouble free, considering the satisfactory experience with the system and the pleasant cooperation in resolving some teething problems. When moreover it turns out that the supplier has obtained the ISO 9000 certificate, and additionally has gone up on the CMM ladder [4], both guarantees of a good control of the process, the modernisation surely will go smoothly. Prominent guests are invited to the festive re-opening of the establishment. Because time is limited once again, the system can not be tested extensively, but in light of the large amount of re-use and an unshakeable belief in the correctly executed processes, this is deemed not needed. Can you imagine the anger, frustration and disappointment of the owner when the preparation of a significant fraction of the dishes fails completely during the gala dinner? Afterward, analysis shows that while one of the new stoves does have exactly the same interfaces as the replaced one, the heat-conduction to the pan is much more efficient in the new one, and consequently, that the cooking times on which the automation is based should have been adjusted! To the system designer the cooking time is simply a configurable parameter, to the chef the correct setting of it is part of the delivery, if he considers this at all, and so nobody made a mistake. This is, though differing in the details, very similar to what went wrong with Ariane-V. Re-use of existing software components is a good cause, but the lack of a clear relationship between the characteristics of such components and the requirements

imposed on them by an application, mean accidents from incorrect use can easily occur.

How then?

The situation I have sketched, though not incorrect, is perhaps too pessimistic. Although a significant fraction of the industry has something akin to a religious faith in the power of the fashionable process-oriented beliefs and still seemingly remains convinced of the advantages of a mechanical over a creative approach, and although academic research usually restricts itself to mono-disciplinary research questions, there are indications that — at least technically and scientifically — the time for a different approach has arrived[5].

Achieving meaningful improvement in the situation requires, not only a number of technical and scientific innovations (which will be addressed later), but also a cultural change in industry and in academic research. The industry needs to become more conscious of the fact that designing complex systems is difficult and not something that can be done by machines, or by humans that have almost been reduced to machines. On the contrary, only the best, most inventive and creative minds are, if well-educated, capable of really solving the problems. The creative and somewhat chaotic design process in which decisions need to be made at many different levels of abstraction simultaneously, is ill-suited to the traditional methods of management. The designing of a complex system is first and foremost a technical problem, to which the non-negligible management aspects should be subordinate. Despite this, often the existing management structure is taken as a starting point for the design process to be followed. It is remarkable in this context that despite the enormous changes between the running a company 50 years ago and today, the management structure has hardly changed. The preoccupation with management aspects means that industry does not always provide a working environment in which designers function best.

The academic world should come to understand that, in addition to fundamental and application-oriented research, there exists multi-disciplinary research restricting itself to a certain class of applications with properties specific to that class, and in which extraordinarily interesting and very challenging questions need to be answered. Results of such research are however generally more difficult to get published, because the mechanism of peer-review favours a mono-disciplinary culture, one in which multi-disciplinary research is considered as neither fish nor fowl. On the one hand, there is therefore a need for finding other mechanisms for ensuring quality, and on the other hand, the traditional way of measuring productivity of research, in terms of numbers of publications, needs to be reconsidered.

---

[5] See, e.g., the Senter-funded project ORKEST ("Onderzoek naar Randvoorwaarden voor de Konstruktie van Embedded Systemen" ["Research into the boundary conditions for the construction of embedded systems"]) in which three universities are cooperating with industry.

It would have been great had the government decided to prefer the founding of a Technological Top Institute in the field of Embedded Systems over that of a similar institute in so vague and little fundamental, but fashionable, field as Telematics. Such an institute could have have functioned as a catalyst for the necessary cultural changes in addition to generating useful knowledge. As an aside, the decision making process for the proposed Technological Top Institutes are a beautiful but sad illustration of how ignorance can cause a reasonable process to generate bad results [6][12]!

On both technical and scientific fronts, much is still to be done. Although computer science has made much progress in many subjects over the past 25 years, there are still many unanswered questions and unsolved problems in realising complex systems. Even if the suggested cultural change will take place, and multi-disciplinary, application-oriented research gains respectability, it will still take many years of research before one can speak of a mature discipline. Without any pretence at being exhaustive, I will try to adumbrate the areas in which research is required to at long last found a true engineering discipline. It is necessarily a simplified version of the areas of research, if only because the various areas are intimately related, and these relations do not lend themselves well to expressing them in a linear summation.

Research

The starting point for each design must necessarily be the specification of the desired characteristics of the system. Considering that, in principle, only the future user of the system to be constructed knows what purpose it serves and what is expected of it, such a specification must relate to the concepts that are meaningful to this user. The current, and in itself certainly useful, research into methods of specification primarily concentrates on the designer of the system, and such a specification may therefore be considered a first step in the design. However, for the future user of the system, such a specification is utterly unintelligible. A formalism that is understandable and workable for a domain expert, allowing him to capture his requirements completely and unambiguously, would consequently be a major step forward. Not only can such a specification of requirements be used as a starting point for the design, it can also play an important role in negotiating contracts and proving conformance to specifications.

The designing of systems is the main thread of the research. One of the most important requirements that must be imposed on the design process is the avoidance of introducing unnecessary complexity to what is already such a difficult problem: the construction of a system that exactly meets all the requirements. A system of extensive functionality and with stringent performance, availability, reliability and extensibility requirements is an incredibly complex structure, in which structure and abstraction are virtually the only means of remaining in control. Unfortunately, the usual method of design considers the structural aspects as subordinate to the functional. The consequence is that there is no uniform structure and that the connection between each pair of functional components must

be formalised, and thus implemented, separately. As it would also have been possible to first design a global architecture within which the functional components need to fit, the current method leads to avoidable and hence unnecessary complexity. The research should therefore focus on the question whether different architectures are desirable for different application domains, and what properties those architectures should have for the systems in these domains to be realised as easily as possible. In the study of the desired properties of an architecture, the relevant characteristics of the application domain should be taken into account. Instead of striving for, e.g., an architecture that ensures complete consistency between distributed components, one should investigate to what degree of inconsistency in the given domain can be tolerated and how a potentially weaker requirement in the realisation may be beneficial. By the way, such an investigation requires an originality of thinking that is stimulated far too little in the current educational system.

Abstraction is indispensable in controlling complexity but not all abstractions are equally useful, and for different purposes different abstractions may be desirable. However, in software engineering, it is an inviolable truth that data must be hidden and meaning can be attached only to operations. This is seen as a prerequisite for successful design. Without exception, however, the designs of the enormous failures I mentioned at the beginning of this lecture all relied on data abstraction; at the very least one should question the usefulness of the methods used for designing such systems. Although data abstraction is useful in certain circumstances, especially for embedded systems the data is a reflection of reality and the functions model relationships between quantities [1]. When designing embedded systems, it is therefore advisable to choose an abstraction that makes precisely the data visible and lets the relationships between the different data elements play a minor role. Both the failure of the Ariane-V as well as the catastrophic reopening of our hypothetical restaurant might have prevented this way.

There is a strong but poorly understood interaction between the chosen architecture and the approach to the design problem. For example, the use of existing components depends to a much greater extent on the architecture chosen than on the components themselves. In addition, reuse requires that each component can be fully specified without being dependent on the characteristics of other components. A design strategy that leads to such components not only has the advantage that the components can be useful in all kinds of applications, but also allows the system properties to be derived from the properties of the individual components and those of the supporting and connecting architecture. Although there are still insufficient mathematical tools to carry out such derivations on an industrial scale, there is good hope that technologies can be developed that will enable this to be achieved in the foreseeable future.

Another aspect of this interaction is the extent to which system properties can be realised by measures of a strictly structural nature, solutions entirely within the specific application components, or by a combination of both. Much of the current research simply assumes that strict separation is possible, which in turn simplifies

the problem statement and allows universal solutions. In practice, it invariably turns out that such solutions are not useful because the solutions found for different aspects are incompatible. Research will therefore have to focus more on developing methods that make it possible to investigate the global properties of approaches that are compromises. Gaining an understanding of which combinations of requirements are achievable is an extremely useful by-product.

A final — at least in this lecture — aspect of the relationship between architecture and design, is the extent to which intrinsic parallelism in an application can provide guidance for the use of parallel and distributed processing when constructing a system. The current design strategies make but very limited use of these techniques and arrive at the necessary parallel processing on the basis of other considerations, considerations that are generally unrelated to the application domain. This usually results in avoidable complexity and performance problems.

A complementary research direction consists of constructing mathematical relationships between the formally specified system requirements and the architecture to be used. Although this is but a dream at the moment, it should in principle be possible to choose or specify system components based on such relationships. Such relationships will one day also make it possible to prove that a system meets the requirements. Only then will the foundation be laid for an actual engineering discipline. Finally, to achieve the desired situation in which system design is more engineering than art, it is necessary to achieve metrics that allow the quality of individual design decisions to be assessed.

Education

There is also a lot to be done on the educational side. The cries of distress from the business world for the educational system to better address its needs are in my belief both misinterpreted and expressed without good cause. A well-trained computer scientist of sufficient ability will be able to adapt in a very acceptable time to the process and procedures of a new working environment, provided that he has been adequately trained in applying the theory to problems from fields other than computer science itself. In current education there is too little attention for this due to a lack of time. The ever-increasing pressure on students and the associated impoverishment of the curriculum, are a result of budget cuts that ultimately cost society more than the marginal costs of a somewhat longer and more balanced study. Businesses should learn that it is more useful for new employees to be able to work at different levels of abstraction than to know a few tricks that happen to be fashionable at any moment. Matching educational goals to business needs in my view means that students must have a good and broad theoretical basis, to frequently be contact with applications and, above all, that they must gain the flexibility needed to apply different methods for different types of problems. The field is too young and immature, and therefore too sensitive to fashion, to give more than superficial attention to any so-called universal solution, such as the following examples from the recent past and current practice: Programming Environments, Client Server Architecture, Object Orientation, the Internet, and Java, to name but

a few. All interesting and useful concepts in themselves, but none of them even remotely suitable for building complex systems, as demonstrated by a number of failures. Therefore, business is not served with new employees who, fresh from the university, expect to solve the problems simply by application of the latest fashion.

I see it as my task to make a contribution in bringing applied computer science to maturity. To this end, instead of making false promises and raising unreasonable expectations, we must be able to objectively determine what is possible and what is impossible. To the extent possible, when searching for solutions a good computer scientist makes a primarily creative use of the many theoretical and practical achievements of computer science. I intend to make my contribution in the following way:

Firstly, to provide students with insight into the problems involved in the construction of large and complex industrial systems, and in relation to that, demonstrating the possibilities for finding solutions. One of the most important means of doing this is enabling students to experiment with various possible and impossible solutions to problems from actual industrial practice. Students can not be challenged enough in this. The time required for gaining this practical experience should not be found through a weakening of the theoretical basis, but in reducing the time spent on superficial skills from current software engineering.

Secondly, through research contribute to reducing the enormous gap that still exists between knowing what needs to be built and realising a system that is doing exactly and exclusively what is required. The fact that achieving this goal requires a substantially different approach and entails entering untrodden paths on the one hand increases the challenge, and on the other hand will also inevitably lead to entering dead ends. As in all other engineering disciplines, it will eventually prove to be mathematics that provides the basis for a solution. For such research it is of the utmost importance to learn from experience, and so there must be a large amount of experimentation and evaluation. Here is a unique opportunity to connect research and education.

Thirdly, make it clear to fellow computer scientists, managers in industry and politicians that they have been chasing a will-o'-the-wisp, that more of the same only leads to more of the kind of problems we already know, and unfortunately also that there is no silver bullet in this field either. The main message may well be that designing complex systems requires a multidisciplinary approach, and that other possible solution strategies exist in part because of this, ones which at least have not yet proven themselves to be useless.

Acknowledgements

Having almost reached the end of my lecture, I would like to take the opportunity to thank those who made it possible that I can now stand here, as far as possible in chronological order — as any other order might lead to incorrect suggestions. Let

those I forget or skip be comforted by the thought that I am very aware of the relativity of my current position, and especially of my own share in achieving it.

My *parents* have had more influence and have been an example to me more than anyone else. In particular, I want to mention the never-ending concern for our development; the absolute honesty in their thinking and actions under all circumstances; the stimulating care for our mental and physical well-being and the loving guidance in sometimes difficult circumstances.

Anner Bijlsma taught me, like no other, that attention to detail should never be at the expense of attention for the whole. Thanks to his broad vision, great interest and unmatched artistry, I have learnt that much of what is true in music also applies to technology and science. This insight is an enormous enrichment.

Antje I cannot thank in this way. Those who know her even a little, know that without my wife nothing would have become of me.

Max Ceuleers, as head of the department at which I worked at the time, allowed me to investigate alternative methods for constructing complex systems, often against the opposition of the company. Without the trust and support of such an incorruptible and honest person as Max, my development as a researcher would never have taken this flight.

Bob Hertzberger I first met as a pioneer in the use of the Motorola 68000. The collisions that are almost inevitable between two such quirky spirits did not happen, and a long-lasting and fruitful collaboration grew, eventually leading to my appointment.

Edsger Dijkstra has tolerated me for many years as a member of the Eindhoven Tuesday Afternoon Club as a result of a happy coincidence. The care practiced by Edsger in the study of fundamental issues has been a major formative influence on my development as an computer scientist.

Herman Driessen, as technical director of Signaal, took the courageous decision to use the systems architecture I invented, revolutionary though it was in industrial practice, for a new generation of products. Despite the understandable and not insignificant resistance to this decision, he never lost his confidence in the solution, and I am pleased to note that Signaal has done well by this decision.

Colleagues at Signaal and Philips, with whom I have been privileged to work for the last 25 years, as well as colleagues at the various Dutch and foreign universities with whom I have had intensive contacts, have had an undeniable influence on my thinking and acting.

The *Management of Signaal* decided more than three years ago to offer the, now formally occupied, chair to one of the Dutch universities and asked me to do what undertake the required actions. My gratitude concerns both the choice of the chair and the fact that they have allowed me to choose the University of Amsterdam.

Finally, the *Board of the University* deserves my very special thanks for my appointment in this particularly engaging and challenging position. I will do my utmost to justify the trust invested in me.

I thank you for your attention and would like to end with the observation that achieving the goals outlined within the time allotted to me should be considered unattainable and that the impossible will also take a little longer in this case ...

I have spoken.

Referenties

[1] M. Boasson, Control Systems Software, IEEE Transactions on Automatic Control, vol 38, n 7, pp 1094-1106, 1993

[2] Oxford Dictionary of English, Oxford University Press, 2003. Ed. Catherine Soanes, Angus Stevenson.

[3] E. Forbes, Thayer' s Life of Beethoven, chapter VIII, Princeton University Press, 1967

[4] W. Humphrey, Managing the Software Process, Addison Wesley, New York, 1989

[5] Joch, How software doesn't work, Byte dec. 1995

[6] H. van der Laan (Chairman), Technological Top Institutes, Report on the Scientific Assessment at the end of Phase I, KNAW, 1996

[7] N. Leveson, C. Turner, An investigation of the Therac-25 accidents, IEEE Computer, vol 26, pp 18-41, 1993

[8] J.L. Lions (Chairman), Flight 501 Failure, Report by the inquiry board, ESA, 1996

[9] B. Meyer, Object-oriented Software Construction, Prentice Hall, 1988

[10] M. Shaw, Status and Prospects for Software Engineering, in M. Shaw, D. Garlan, Software Architecture, Prentice Hall, 1996

[11] C.J. Wannée, Kookboek van de Amsterdamse huishoudschool, page 172, H.J.W. Becht, Amsterdam, 16e printing

[12] Economic-Technological Assessment of Proposals, Final Report, PA Consulting Group, 1996