Software development: Science or Patchwork?

by Robert L. Baber

Abstract

In the past, software development has been variously described as a science, art, craft, trade, racket, etc. Even today opinions vary considerably about how software should be developed and what qualifications are most appropriate for software developers.

In this paper a series of case examples illustrates the problematic state of affairs in the field of software development today. Many of these examples also illustrate the often overlooked potential and practical value of a mathematically oriented approach to solving software developmental problems.

The spectrum of possible software worlds of the future is characterized by its three extremes: an audacious (reckless), a backward (reactionary) and a celestial (radical) software future. The paths leading to each of these extremes and the prerequisites for achieving a significantly better future are discussed.

In this paper it is argued that the detailed specification, design and development of computer software is by nature an engineering field with mathematics as an important theoretical foundation. In practice, however, software development is too often treated today as patchwork. It is too often conducted by underqualified personnel instead of by professionally educated engineers. These are the major causes of the many problems and disappointments that we have been and still are experiencing in applying computer systems.

1. Prologue: the land of Moc

Imagine that you are living in an ancient country in the cradle of civilization. The time is about 2500 B.C. In the course of the years an active foreign trade has developed and several cities have been founded. A construction industry has been established in which professionally trained architects and civil engineers play an important role.

Suddenly and unexpectedly, a group of teachers of civil engineering develop a new technique for designing buildings. Using the new method, buildings can be designed and built which are much larger than those previously possible. Perhaps even more importantly, the new method reduces construction costs to about a tenth of their former levels.

As a consequence, the demand for buildings of all types increases very rapidly. The construction industry grows correspondingly. But new architects and civil engineers cannot be educated so quickly. Nor can the capacity of the civil engineering colleges be expanded sufficiently rapidly. Nevertheless, a

solution is found: New construction planners are trained in special short courses conducted by the suppliers of building materials. In this way the quantitative gap between supply and demand is bridged.

The application of this advanced technology is not without its dark side, however. Because the new 'three week wonders' (as the professionally trained designers contemptuously label their minimally trained colleagues) only superficially understand the theory underlying their designs, minor catastrophes are common. All too often the three week wonders rely on 'trial and error' (instead of theoretically based calculations) in order to determine if a proposed design will collapse or not.

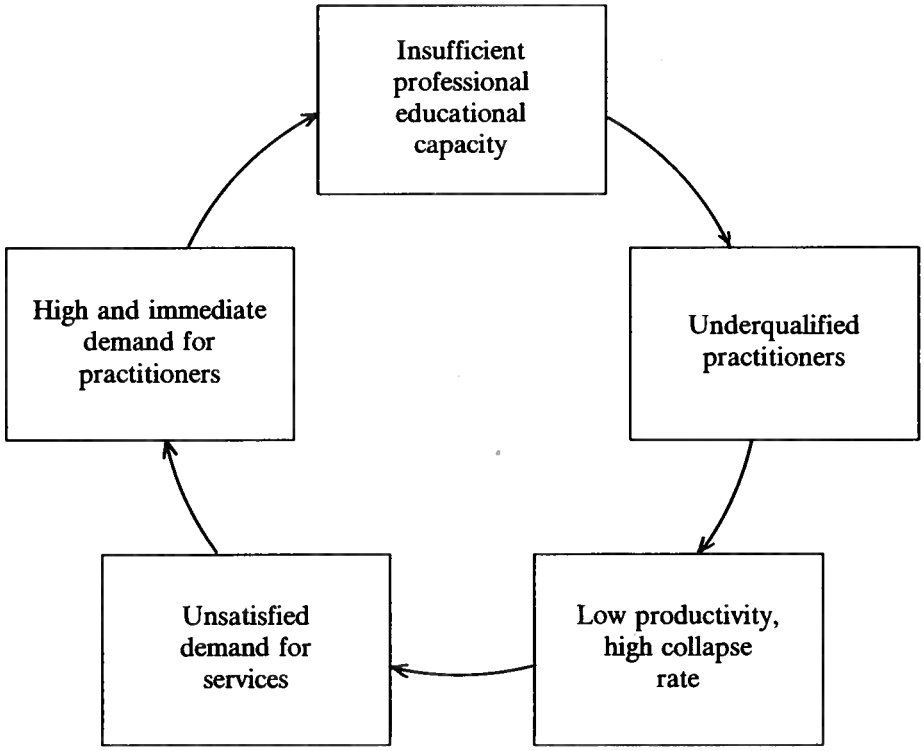

Fig. 1. The Mocsian vicious circle

The result of this 'trial and error' approach is not really surprising: about 30% of all newly designed buildings collapse during construction. As an eloquent professional once put it as he arrived at the scene of a particularly spectacular collapse, 'rubbish in, rubble out'. The rubbish was the design prepared by a three week wonder; the rubble, an incredible quantity of debris which shortly before had been a wondrous building under construction.

To combat the problem of collapses during construction, elaborate testing schemes have been worked out. These do not solve the problems, of course, but they do tend to limit the negative consequences of the frequent collapses. The test procedures are expensive — causing as much as half of the total construction costs. Nevertheless, there is clearly a net economic benefit for the Mocsian society of using the new design technique.

Although the shortcomings of current design practice are widely recognized, this modus operandi appears to be firmly established. Prospective new designers prefer a short, cursory, practical training instead of a professional course of study lasting several years because they can thereby begin earning good money earlier. And most of the professionally trained architects and engineers, from whose ranks teachers could come, prefer a more lucrative professional practice over a career in teaching.

As a result, the fraction of professionals among the building designers in the land of Moc is declining. This is, in turn, causing a corresponding decline in the quality of Moc's building planners, a decline in productivity, a still larger gap between supply and demand and a demand for more building designers, even underqualified ones. Thus, as shown in fig. 1, the vicious circle is closed.

2. Moc Today

The story of the land of Moc is obviously pure fiction. Such a ridiculous story could never be true. Or could it? It is my thesis that the story of the land of Moc is true. The time is not 4500 years ago, but the present. The true story does not concern an ancient construction industry in the cradle of civilization, but our modern computer software industry in the most 'advanced' countries in the world.

Software development today is plagued by many — too many — unnecessary collapses. From time to time we read about some of these collapses. But many more go generally unnoticed because they occur so frequently that they cannot be considered unusual. Design by 'trial and error' (following the Mocsian motto 'try building it and see if it collapses') has become commonplace. In order to limit to some extent the effect of our failures we conduct an incredible amount of 'testing' (a euphemism for finding and correcting mistakes). Typically, 'testing' accounts for half of the effort expended in a software development project.

What has caused this situation? Advances in the computer area have been made so rapidly that we still have difficulty absorbing them — although computer systems are, after several decades, nothing new. Especially our

educational programs have not kept pace with developments on the computer scene. Our concentration on short term problems and solutions has prevented us from building a solid, optimum base for the long term.

Why has this state of affairs become so accepted? The computer is a fundamentally new tool so useful that even when sloppily applied by beginners and amateurs, the net benefit — after due consideration of the collapses — is still very great. We believe that the benefits realized are due to our own good efforts. In reality, we have, more or less by accident, stumbled onto a good thing. We could and should endeavor to make much more of this good thing in the future than we are now doing.

In my opinion software development is by nature an engineering discipline. The more successful developers of challenging software systems apply mathematical and scientific knowledge when specifying, designing and implementing their systems. They proceed in a systematic manner, whereby creative aspects of their work an be recognized — just as in already recognized engineering fields.

Unfortunately, an engineering approach is not characteristic of typical software developments today. The following examples illustrate the consequences of this deficiency. They also show some of the advantages of a mathematically and scientifically founded approach to software development. The examples are in chronological order.

- In the mid 1950's a course in computer programming was offered to advanced students at a particular engineering school. A sophomore applied to enroll in the course. After initial hesitation, the professor granted his request.
- Two years later, a course in computing was offered to freshmen. As many as 10% of the freshmen were expected to enroll. At least twice as many actually applied. The number of instructors available was hopelessly insufficient to meet the demand. It was assumed, of course, that this large gap between supply and demand for the education of software specialists was only a temporary phenomenon. Little did they know.
- In the early 1960's in one country's defense establishment, a series of programs printed a long classified report. Because of logical flaws in the programs' printing routines, a few pages would, from time to time, be printed which included the security legend at the top or bottom but no data. They could not be deleted from the report without introducing a discrepancy in the page count. It was necessary, therefore, to register and retain the 'TOP SECRET' blank pages.
- A major computer manufacturer installed several very large, fast computers. When it was discovered that the systems operated unreliably, many customers withheld their monthly rental payments. The manufacturer soon solved the hardware problems, but despite many corrections and revisions the operating system remained unreliable. The manufacturer's liquidity became very strained; some observers of the industry expected the company to become insolvent. Finally, large

infusions of loan capital enabled the supplier to remain in business while the errors in the system were being corrected.

- In a large information system, data relating to individual persons was indexed by the person's name. A method was required for locating data on a particular person even when the name was misspelled. This was to be accomplished by transforming the name into an abbreviation in such a way that typical misspellings transformed to the same abbreviation. Several rules for forming the abbreviations were to be investigated. A logically complex program was written to test the first rule. Whenever the programmer corrected one error, another appeared in its stead. After several months, the project manager asked a consultant for assistance.

  The consultant noticed that a transformation of names was homomorphic to (and could therefore be represented by) a finite automaton with a small number of states. He suggested expressing each abbreviation rule in the form of a transition table and writing a single table driven program to simulate the automaton defined by the table for each rule. In the attempt to construct the table for the rule already programmed, a logical inconsistency in the rule was discovered. After the rule was corrected, this approach led quickly to success.

  In this example the method of 'trial and error' could never have led to a successful result, because the programmer had been trying to program an unprogrammable (logically inconsistent) task. In addition, the conventional programming approach, even if successful, would have been very inefficient, involving several complicated programs instead of one smaller, simple one and a few tables, each one page long.
- One company in the new world decided in the mid-1960's to implement a truly integrated information system. About 5 years later, around the turn of the decade, shortly before the planned implementation of the system, the project was abandoned. Only a few small parts of the system could be salvaged and used productively. Several higher level managers left the company, some willing, some not. The loss was estimated to be about $ 10 million. In the following years, the electronic data processing pendulum swung to the other, conservative extreme in this company. Many economically justifiable electronic data processing applications were overlooked or intentionally rejected. The opportunity cost of these potential applications was never estimated but was certainly substantial.
- In the early 1970's the old world was generally considered to be some years behind the new in matters concerning electronic data processing. Unfortunately, this was not the case when it came to creating spectacular collapses. Shortly after the collapse outlined above became known, almost the same occurred in a European country. In both cases unsatisfactory communication and lack of mutual understanding between the developers, users and management as well as exaggerated optimism were important reasons for the expensive failures.

- In another firm a software system for sales forecasting was designed and implemented. Several programs, which were obtained from the computer's manufacturer, had allegedly been successfully used earlier in other companies. During implementation, errors were discovered in several of these programs. One of the errors represented a fundamental oversight in the mathematical analysis of the statistical forecasting model upon which the program was based.
  Earlier, the occurrence of such errors was assumed to represent a transitory phenomenon. Slowly one began to recognize that this was not the case. The fact that a program had been used successfully by others could no longer be interpreted as evidence of its correctness.
- An inventory control system was developed to satisfy the needs of a particular company. For this system, a set of formulae was developed which could be solved only by iterative approximation. Although preliminary tests gave no indication of potential problems, one member of the team was concerned that the iterative method might not always converge or that it might sometimes converge to an undesired solution. A lengthy mathematical analysis showed that no such problems would arise in the case at hand, but that a particular kind of non-linear interpolation in a table was required in one part of the computation.
  Another member of the development team was concerned about the possible consequences of the unavoidable delays in the man-machine communication in this batch system. The desired behavior of the system was formulated as a specification of an automaton. (See Baber [2], pp. 166-167, for further details.) During the necessary discussions between the system's designers and future users, a number of possible sequences of events came to light which no one had considered before and which forced the users to think through in detail what they really wanted from the system. The resulting specification was implemented successfully and employed for many years.
- A data base system was planned as the basis for an order processing system. After detailed investigations were conducted, the software system and the data organization were selected. From the outset, some members of the project team were concerned about the system's response time, but initial analysis indicated that this would probably not be a significant problem.
  During *implementation* of the system another — insoluble — problem arose: As the data base was built up, it was noticed that the time required to load additional data increased very rapidly. It became clear that just loading the complete data base would require a *year* or so of computer time. The response time also became problematic, although it was less serious. The project was abandoned in the implementation phase, *after* design and programming were complete, and the history of software development was enriched with another collapse.
  This failure could have been avoided. Without undue difficulty, one

could have determined the time complexity of the required functions using the selected data organization — before expending any significant developmental effort. Probably, no member of the development team even knew about the concept of computational complexity. A bent for quantitative analysis was apparently also lacking, for even without knowledge of complexity theory one could have estimated the time characteristics of the data base system.

- While designing a relatively straightforward application system, a system analyst was forced by arbitrary limits in the system software to make use of linked list techniques in organizing a data file. During the programming phase difficulties arose because the programmers, among them a graduate of computing science, had no experience with linked lists.
  This example shows the value of a theoretical background: without knowledge of linked lists, the analyst would have been unable to solve the problem at hand using the given computer system. This example also illustrates that today's computing science education leaves much to be desired, for the graduate was unfamiliar with a method of data organization which is basic, well known in professional literature and often used in system software.
- An experienced free lance programmer contracted to write a particular program for a software house. The program was to run on a machine with which the programmer had no prior experience. After reading the manual, he still had some questions regarding commonly used input-output functions. He turned to a specialist for that machine for clarification. The specialist could not answer his questions. The programmer asked the specialist how he resolved such problems. The specialist replied, 'I just keep trying until something works.' The programmer was not satisfied with his advice. He used only those commands which were unambiguously described in the manual. His program was perhaps not so elegant as it might have been, but it worked reliably. The same could not be said for the specialist's programs.
  Although the 'trial and error' method often leads to unsatisfactory results (or even to no results at all), it is still used again and again.
- The manager of a production planning department asked an electronic data processing specialist with a mathematical inclination to investigate the feasibility of optimizing a regularly recurring task of his department. The goal was to schedule the production of a particular product so that the amount of scrap generated would be minimized. The specialist investigated several mathematical methods such as integer programming. He concluded that all of these methods were infeasible in the given situation due to their computational complexity. During his investigation, however, he discovered a heuristic algorithm which usually gave optimum results. He programmed this algorithm on a small microcomputer which was purchased solely for this application. His system was used successfully for a long time.

Again, a combination of theoretical knowledge and a sense of what is needed in practice seems to be associated with success.

- In the early 1980's a world wide transportation company contracted with an important computer manufacturer for the development of customized software for an integrated fleet operations and accounting system. Two years later, shortly before key elements of the system were to be installed, the supplier announced that it had discontinued the project and would not deliver the application software.
  After the collapses in the old and new worlds, this was bound to happen sooner or later.
- An error in a computerized defense control system resulted in a false report of an enemy attack. Interceptors were launched. Fortunately, the error was discovered in time.
  After many years we are still building errors into our computer systems. But with time the potential consequences of these errors are becoming more serious and irreversible. They are not so comical as the 'TOP SECRET' blank pages of years before.
- An experienced software system designer developed a software system for a management game for a client. The structure of the game required that decisions be made by each team of players (representing competing companies) for successive time periods. To fit into the environment within which the game was to be played, it was necessary that the computer program could be interrupted at several points within each time period and restarted later at that or an earlier point in the game. In order to enable this mode of operation, the players' decisions were stored in separate files by team and by time period; data specifying the state of the game were stored in a 'run control file'.
  The designer, convinced that the logic of the interactive control program would be fairly simple, began to write it without planning its structure in detail. After several false starts and several hours wasted, he decided to start all over and do the job properly. He began by specifying the assertions (logical conditions) which the values of the various variables defining the state of the game would have to satisfy at all times. Supported by these eight assertions (program invariants) and knowing the intended 'variant' action — advancing to the next time period as soon as the prerequisites (implied by the invariants) were established — he was able to write the control program in a straightforward manner. As he originally expected, the program turned out to be logically simple. His successful approach took less time than he had already wasted on the false starts.
  If a mathematically oriented approach can yield such benefits even in the case of small, logically simple programs, how can one afford *not* to take such an approach when developing complex software systems?

- Computer courses are being introduced into secondary schools in many countries. The better prepared teachers of these classes took one or two courses in computing subjects in college; many have had no formal training in computing. Teachers of other subjects are required to have completed a much more extensive training in their fields (cf. mathematics, languages, history, sciences, etc.). Why is informatics treated in this very different — even irresponsible — way?
- A seminar on advanced topics in programming methodology for software developers and research scientists was arranged. The number of applications greatly exceeded the expectations of the organizers and, of course, the planned capacity. Many qualified applicants had to be rejected. After a quarter century a very considerable gap between supply and demand for computing science education still exists (cf. the course offered to freshmen at the engineering school above).
- Because of an error in software an unmanned space ship was lost in deep space.
- In the field of informatics at universities in the Federal Republic of Germany the ratio of students to faculty has grown to 100:1 [12]. Potential students are interested in studying this subject and companies are very interested in employing all graduates. Still this great — probably unprecedented — discrepancy between supply and demand for informatics education prevails. While the Federal Republic of Germany may be an extreme example in this regard, this problem is by no means a uniquely German one. Neither is this a new problem which has suddenly appeared. One could and should have seen it coming. Many *have* seen it coming for many years.

What is wrong with the current situation?

Because the net benefits resulting from applications of computer systems are so great, one might ask: 'What is really so bad about the current state of affairs? We are producing large quantities of software which is of considerable value to its users. As long as this situation prevails, we do not really have a problem.' There are, however, a number of serious negative consequences of our current approach to software development:

- The consequences of errors in software are becoming more costly, more dangerous and more irreversible.
- Major and unnecessary failures, mishaps and attendant losses occur too frequently.
- The total cost of our software systems (including the costs associated with failures and their consequences) are unnecessarily high.
- We are obtaining much less benefit from computer technology than is potentially possible.
- A large gap exists between supply and demand in the software market. This gap represents an economic loss (opportunity cost).

- Negative consequences and losses caused by errors in software are often shifted unfairly onto persons who are not responsible for those errors, who have no control over the situation and who cannot protect themselves from the consequences of such errors.
- Software products are often characterized by disappointing quality. Errors in software are often of a rather simple nature.
- Productivity in software development leaves much to be desired.
- Scarce resources are wasted in unsuccessful projects.
- Unreliable systems frustrate and confuse laymen and shake their confidence in applications of computer technology.

Why do we have these problems?

There are several different but related causes of the problems outlined above:

- The transfer of knowledge and experience from one generation of software practitioners to the next is not functioning satisfactorily. This applies to both practical and theoretical knowledge. In other words, our educational system is not achieving its objectives.
- Only a small fraction of software developers has a command of the scientific knowledge of computing science as well as of the application area for which they develop software. Educational programs were and are quantitatively and qualitatively insufficient.
- A great communication gap exists between the practical and theoretical worlds of informatics. This statement applies to a certain extent to all areas of applied science, but it is particularly true of software development today. Many practitioners do not understand the basic language of the theoretician (= mathematics) and many theoreticians exhibit little or no interest in the problems and work of the practitioner.
- 'Trial and error' appears to be a widely accepted method for designing and developing software.
- Software developers rely much too heavily upon 'testing' instead of getting their designs correct in the first place. Especially in this regard does a mathematically oriented approach and way of thinking offer considerable promise for improvement.
- The frequency and consequences of errors in software systems suggest a lack of sense of responsibility on the part of the developers (and sometimes of the users). Perhaps the developers are mentally fully occupied with the technical details of their designs. Perhaps this is due, in turn, to an inadequate understanding and command of those technicalities.
- The proper place of informatics in the academic world is still unclear. Particularly when one compares the position of informatics in universities in different countries does the full variety of organizational forms become apparent: as part of mathematics, electrical engineering, business or economics or as an independent department in a faculty of natural science, engineering or economics, with or without smaller branches in the various departments representing the fields of application of computer systems.

- Economic forces induce potential professors of informatics to become practitioners instead. Potential students are induced to become poorly prepared 'system analysts' and coding technicians today instead of good software engineers tomorrow. Income and success in the short term are overemphasized; potentially greater income and success in the longer term future are largely neglected.

Our software situation today is, it seems, very much like that of the Mocsian construction industry. The most unsettling aspect is that — just as in Moc — a stable vicious circle has arisen (fig. 2).

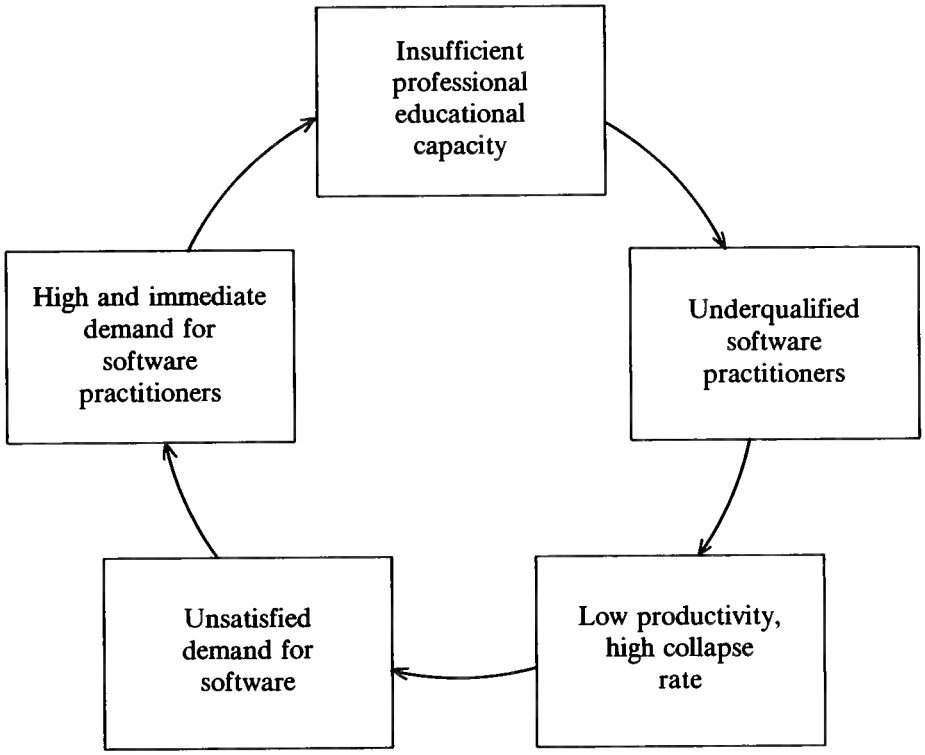

Fig. 2. Our contemporary vicious circle

3. Software development tomorrow?

The nature of the software world of the future will be determined by many factors, the most important of which are our own choices and decisions regarding what sort of a future we desire. I will not, therefore, attempt to forecast the characteristics of our software future. Instead, I will outline the alternatives among which we can — and, explicitly or implicitly, will — choose.

The set of possible software worlds of the future form a continuum characterized by its three extreme points. Each possible software future can be thought of as a convex combination of these three extremes, called Future A, B and C below. The point representing our software future will be determined by the average level of professional competence achieved by software practitioners and by the complexity of the applications attempted (fig. 3).

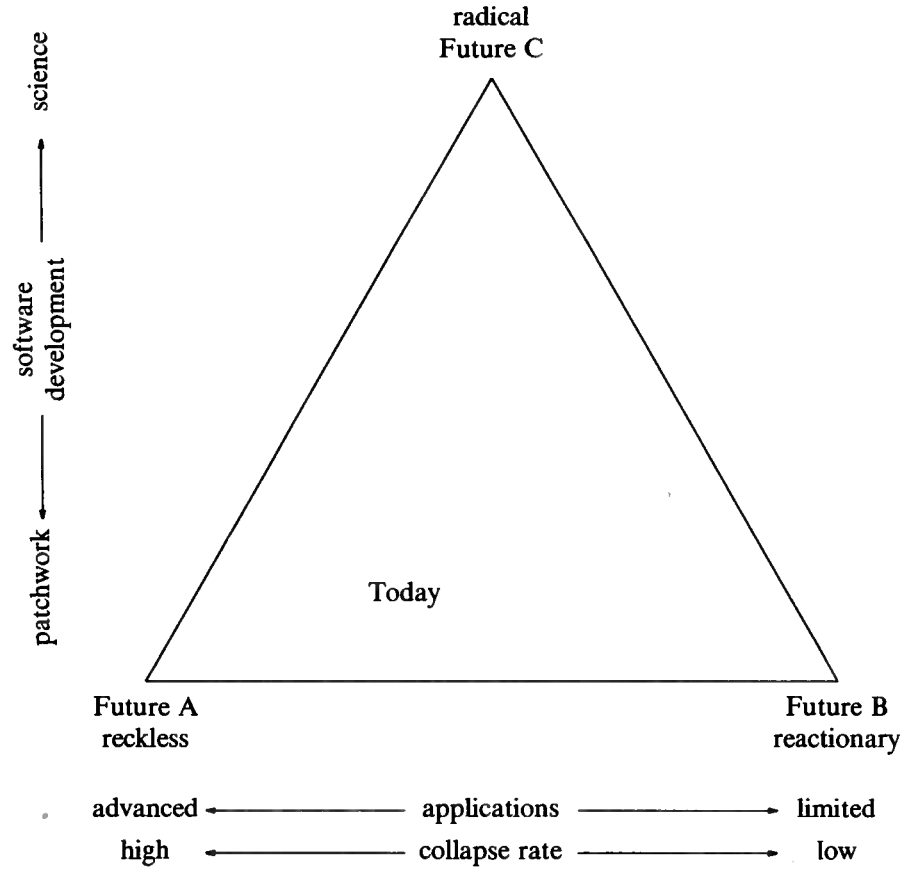

Fig. 3. Possible software futures

In the audacious Future A, much is attempted, capabilities are limited and major failures are frequent. In the backward Future B, strong pressures are present to restrict computer applications to those within the limited capabilities of the software practitioners and their customers. These systems are usually successfully realized, but of course much is left undone. In the celestial Future C, the competence of the average software practitioner has been developed to such a high level that even very complex applications are normally implemented without major difficulty or problems of a fundamental nature. The extreme Future A can be simply described as a reckless future; Future B, reactionary; and Future C, radical.

Today we are between Future A and Future B, somewhat closer to Future A (see the case examples in section 2 above). Many trends and developments in data processing, especially the advent of the microcomputer, will result in considerable pressure to apply computer technology much more extensively throughout society in the next decades [5,10]. This will push us even closer to the point representing Future A. If we decide to develop our professional software capability much more extensively in the future, we can deflect our path away from Future A and toward Future C. If we do not decide to do so, the catastrophic collapses characteristic of Future A can be expected to give rise to a wave of public reaction leading us to Future B.

Let us look at each of the futures A, B and C in more detail:

The reckless, audacious Future A

A software corollary to Parkinson's Law and the Peter Principle characterizes Future A particularly well: Software developers will conceive and try to build ever more complicated systems until the limit of their ability to cope with complexity is exceeded.

Despite our software problems such as those outlined in section 2 above, there is today among broad segments of society considerable optimism and confidence in our computer based future. Futurologists predict wonderful things (see e.g. Evans [5]). There is, however, a great danger that this 'wonderful' future will not turn out as hoped, but instead as follows:

- Because of an error in software in an air traffic control system, four jumbo jets collided over Paris one cloudy morning. All 1631 passengers and crew members as well as 1000 people on the ground were killed. The ensuing traffic jam blocked the area for two days.
- After an extended period of financial difficulty, one of the largest companies in a medium sized European country went into bankruptcy. The detailed reasons were not determined. Only one thing was really clear: the company's information and communication systems were in such a state that the company had become unmanageable.
- A thirty story building collapsed during construction. The design was reexamined in detail. Mistakes in the calculations of stresses in critical structural elements were discovered. These calculations had originally been done by computers whose programs contained errors.

- At the turn of the millennium 1999-2000, business in the computerized economies of the world all but collapsed. For bills rendered in 1999 but due in 2000, overdue notices were issued already in 1999, charging interest for some 99 years. After January 1, 2000, many systems failed to issue overdue notices for amounts due in 1999 but not yet paid. The cause: computer software and data bases which provided only 2 digits for the year.

When such incidents occur sufficiently often, a public reaction can be expected. Depending upon whether the reaction is directed against computer applications as such or against the inadequate capabilities of the software developers, pressure will arise which will push us in the direction of Future B or Future C.

The reactionary, backward Future B

If Future B comes to be, software related situations something like the following may become typical:

- A law was passed which placed very severe restrictions on computerized air traffic control systems. Particularly onerous were requirements for high levels of public liability insurance coverage. Shortly after passage of the new law, a governmental agency announced that a newly completed system would not be implemented. The costs of fulfilling the new legal requirements were so high that the system could no longer be economically justified.
- Members of the interest group 'Ban the Computer' discovered that applied research was being conducted on a computerized vehicle guidance and control system for automobile highways. They organized a massive two day 'sit in and lie in' strike on all major roads within 50 km of the headquarters of a company which was funding the project. Threats of violence were directed at the project's chief scientists and engineers.

The radical, celestial Future C

In comparison to Futures A and B a description of Future C seems a bit dull, because everything functions well, as planned:

- A computerized, fully automated air traffic control system which had operated uneventfully and error free since its installation several years earlier suddenly discovered that two aircraft were on a collision course, only 45 seconds flying time apart. The emergency subsystem automatically took control and guided them safely apart. Shortly thereafter, the system informed the (human) flight monitors on the ground and on board the aircraft involved about the incident. The cause was identified as a traffic volume which exceeded the system's design specifications by 34%.
- A computerized control system for a nuclear reactor discovered an operator's error and issued an appropriate warning. Two successive actions recommended by the system were overridden by the human operator. The system was finally compelled to take over control and shut

down the reactor. In the ensuing investigation, the operator stated that the chain of events took place so fast that a human was incapable of making rational, considered decisions effectively. All concerned agreed that safety regulations should be revised to require fully automatic control over such potentially dangerous processes.

- A study of the software industry showed that the 'debugging' and test phase of typical larger software development projects accounted for approximately 10% of the total effort, compared with some 50% twenty years earlier. Of 2500 projects included in the study, 3 were unsuccessful. The report also stated that 'practicing software developers are now more highly educated than ever before. The average coding technician has completed a three year technical training program; the typical semi-professional software designer, a four year university course; and every software engineer, at least a five year university course.' The study also found that all software engineers *regularly* read professional articles available through the various on-line professional literature services. These services were used regularly by 83% of the semi-professionals and by 31% of the coding technicians.

4. The path from today to tomorrow

Who must do what to achieve a better software future? Everyone directly concerned with designing and developing software as well as everyone involved in their education and training must take active steps to break the vicious circle at several points simultaneously. An attack at one point only would have no lasting effect; the positive feedback in the vicious circle would counteract any such perturbation and would serve to maintain the status quo.

It should be apparent that academic institutions must play a key role in bringing about a significant improvement. Especially in the early stages must academic institutions play a leading role in building a sufficiently widespread and qualitatively adequate base upon which software engineering can be founded. Success here is a necessary (but not sufficient) condition for a major and lasting improvement in the practice of software development.

Already in the 1940's Norbert Wiener described computer programming (he used the term 'taping') as 'a highly skilled task for a professional man of of a very specialized type' [14]. Edsger Dijkstra said much later 'Programming is one of the most difficult branches of applied mathematics' ([4], p. 129). There is no path leading to a mastery of this field in practice which does not pass through a good tertiary education in general and mathematics in particular.

We have not yet chosen our path to our software future. Do we want to proceed first toward Future A and then to be deflected to Future B? Or do we want to take the perhaps more difficult but certainly more promising path to Future C?

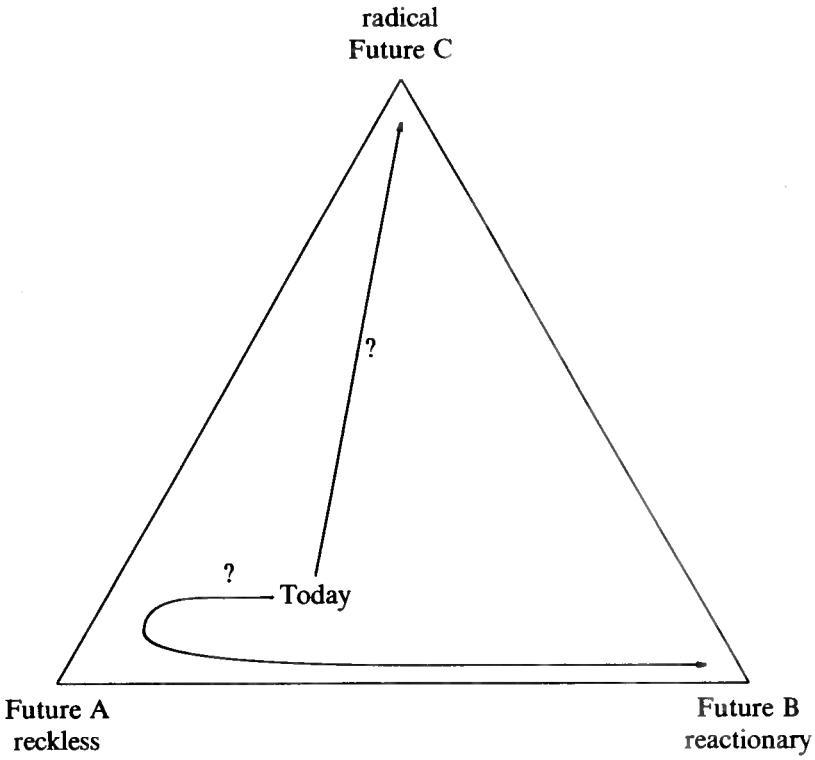

Fig. 4. Possible paths to the future

Literature

[1] Baber, R.L., "Informatikausbildung in der Bundesrepublik Deutschland: Der zukünftige Preis des gegenwärtigen Nichthandelns", *Informatik-Spektrum*, **6**(1983), 1, p. 36.

[2] Baber, R.L., *Software Reflected: The Socially Responsible Programming of Our Computers*, North-Holland Publishing Co., 1982.

[3] Buxton, J.N. & Randell, B. (eds.), *Software Engineering Techniques*, Report on a conference sponsored by the NATO Science Committee, Rome, Italy, 27th to 31st October 1969, NATO Science Committee, Brussels, 1970.

[4] Dijkstra, E.W., *Selected Writings on Computing: A Personal Perspective*, Springer-Verlag, 1982.

[5] Evans, C., *The Micro Millennium*, Washington Square Press Pocket Books, New York, 1981.

[6] Fairly, R.E., *Software Engineering Education: Status and Prospects*, Proceedings of the Twelfth Hawaii International Conference on System Sciences, Pt. I, Western Periodicals Ltd., North Hollywood, California, U.S.A., 1979, pp. 140-146.

[7] Fleckenstein, W.O., "Challenges in Software Development", *Computer*, **16**(1983), 3, pp. 60-64.

[8] Ganzhorn, K.E., "Informatik im Uebergang", *Informatik-Spektrum*, **6**(1983), I, pp. 1-6.

[9] Gesellschaft für Informatik e. V., "Numerus Clausus in der Informatik — hochwertige Arbeitsplätze von morgen durch unzureichende Lehr- und Forschungskapazitäten von heute gefährdet?", Press Notice, *Informatik-Spektrum*, **5**(1982), 2, p. 126.

[10] Haefner, K., *Der 'Grosse Bruder'*, Econ Verlag, 1980.

[11] Haefner, K., *Die neue Bildungskrise*, Birkhäuser Verlag, 1982.

[12] Krüger, G., "Zur Situation der Informatikausbildung an den Universitäten der Bundesrepublik Deutschland", *Informatik-Spektrum*, **5**(1982), 2, pp. 71-73.

[13] Naur, P. & Randell, B. (eds.), *Software Engineering*, Report on a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 October 1968, NATO Scientific Affairs Division, Brussels, 1969.

[14] Wiener, N., *The Human Use of Human Beings: Cybernetics and Society*, Doubleday, 1954.

Address of the author:
Landgrav Gustav Ring 5
D-6380 Bad Homburg v.d.H.
Federal Republic of Germany