CONFIDENTIAL

SYSTEMS R & D CENTER
2670 Hanover Street
Palo Alto, California

February 10, 1965

MEMO TO: Dr. David Sayre - Research, Yorktown

SUBJECT: Some Ideas Concerning Software Strategy

Enclosed is the report you requested giving some of my random ideas concerning IBM's strategy in the software area.

The report is worded in the very general terms which one would use in giving a survey of the field to top management. Following your suggestion, I have emphasized what appear to be the basic problem areas and questions rather than technical accomplishments. I hope it will be adequate for your requirements, considering the short deadline.

I would appreciate being able to read the other consultants' reports sometime when they are available.

Harwood

Harwood G. Kolsky

HGK:bj
Enclosure

cc: Mr. John W. Luke, SR&D, Palo Alto
Dr. C. R. DeCarlo, Corporate/White Plains

IBM Confidential

SOME THOUGHTS CONCERNING A STRATEGY FOR IBM IN THE SOFTWARE AREA

Harwood G. Kolsky
Systems Research and Development Center
Palo Alto, California

February 10, 1965

INTRODUCTION

Approximately one week ago I was asked to set down as quickly and clearly as possible some thoughts which might be useful in forming an IBM strategy in the software area. The instructions given were deliberately meager, but requested that I emphasize what appear to be future trends and problem areas. I was also to give some of the questions the Company should be asking itself in this area, and perhaps my guesses as to the probable answers.

Of necessity the following comments are quite general, there not being time nor space to go into details. No claim can be made as to completeness nor to any accuracy beyond personal opinion. Without further apology, let us go on with the questions:

1. IS SOFTWARE REALLY IBM'S BUSINESS?

One of the first questions usually asked in discussions of management strategy is, "What business are we in?" The answers or opinions expressed in reply to this question (whether it is specifically asked aloud or not) frequently are the main reasons for approving or disapproving actions. This is particularly true in discussions concerning new business, diversification, research or development projects, etc. Quite often the answer may be, "Yes, the Corporation is in the _____ business, but your particular department or laboratory is not." This complicates the picture since the determination of the appropriateness of a given action is no longer unique, but depends upon a set of varying circumstances and opinions. The answer can be quite different, depending upon who within the Company is facing the question and how he views his mission.

The question as to whether IBM as a total corporation is in the software business must certainly be answered with a resounding "Yes!" The software extensions of our hardware, or the software bridge between our customers' applications and our equipment, is certainly a very integral part of our business. It is right in the middle of our main business arena. The quality of our software support in many fields is the only thing which really distinguishes us from some of our better competitors. Our continuing software support is often the one thing which makes the real margin

between us and price-cutting technical "scalpers", who are always ready to turn out a less expensive gadget provided they can quickly walk away and leave the customer with it.

Having stated that we are definitely in the software business, with no possibility of ever getting out of it as long as we are in the data processing business, the next question should probably be:

2. WHAT IS INCLUDED IN THE TERM "SOFTWARE"?

Right away the picture gets cloudy. There are certain items, such as compilers, report generators, etc., which people have come to expect as being always delivered with IBM machines. There is an increasing list of other items which are questionable as to whether they really fall within our definition of software or not, even if they may fall within someone else's definition. For example, recent technical meetings have had some of the following topics listed as software items:

Compiler Technology	Scheduling Algorithms
Simulation Techniques	Memory Organization
Computer Aided Design	Programming Languages
Coding Theory and Errors	Linguistics
Data Structures	Sorting Technology
Heuristic Processes	Real Time Computer Control
Self-Organizing Systems	Operating Systems
Data Reduction	Numerical Analysis
Hybrid Computation	Automata Theory
Commercial Data Processing	etc.

There are many other topics which can be classified as strictly applications-oriented, or strictly computer hardware organization topics, but many of the above straddle the line between hardware and software. An honest answer to the above question is probably, "Almost anything in the computer field has its software aspects." Being less evasive, one can identify a number of the items in the above list or in the classification list used by Computer Reviews as legitimate areas of software.

3. HOW MUCH EFFORT SHOULD IBM INVEST IN EACH OF THE IDENTIFIABLE FIELDS OF SOFTWARE?

Should IBM really expect to be a leader in each one of these fields? Are there certain of them which should be deliverately abandoned to our competitors or our customers?

The answers to these questions are, I believe, also clear. There is no area involving the extension of our equipment into any application area which we can afford deliberately to ignore or abandon to competitors.

We may be a leader in a given field, or we may just barely be able to monitor certain activities. Perhaps we may have people who are participating in related topics bordering some field, and perhaps we are really outclassed, but we should never deliberately rule any of them out as areas which appropriately skilled IBMers must avoid. In other words, we should have experts working in as many aspects of the software field as we can afford. Where we cannot directly participate, we should at least have people who are monitoring activities. Of course, we cannot do other people's jobs for them, and we certainly cannot have all the key people in any field, but we should participate, contribute and evaluate work in most of them.

It also follows that there is a communication responsibility. In addition to knowing, such specialists owe it to the Company to make their knowledge available to all concerned by serving as internal consultants.

4. WHY ARE SOFTWARE PROGRAMS SO DIFFICULT FOR US TO DO?

This is a complex question. The basic reason is, of course, that the tasks to be accomplished are technically difficult. They require very great attention to detail and intimate knowledge of several demanding disciplines. IBM's systems usually must include great generality because of the widely differing demands of our customers. There is also the time-scale problem. We frequently have to do our systems first, against difficult schedules on new and unfamiliar machines. There are also some very serious management problems and personnel problems, which will be referred to later.

One of the fundamental characteristics of software programs which make them difficult is that they are never really finished. When a program is "completed" and released to the field, the work may just be beginning. If the program is well accepted and widely used, the problem of maintaining it and adding improvements to it becomes a very serious continuing effort. The influence of a good program can spread over great distances geographically and in diversity of application. Many systems programmers have found themselves "chained" to their successful programs. Keeping them updated, answering questions, incorporating new improvements, or even the effort of attempting to turn the programs over to someone else who is willing to maintain them, becomes a seemingly endless task.

The Corporation thus continues to "snow-plow" a larger and larger load of software efforts, because all of our past efforts remain with us as long as they are being used.

5. IS THERE REALLY A PROGRAMMER SHORTAGE? IF SO, WHAT SHOULD WE DO ABOUT IT?

It has become a truism that the main problems of software do not stem from the computers but from the people who program and use them. (One

can make a similar statement concerning almost every human endeavor.) The position can be stated more positively by saying that there is indeed a real problem in finding people who can do programming and do it well. Obtaining properly trained and motivated people in the right positions at the right times is one of our biggest problems. There has been a continuing shortage of advanced, experienced programmers for several years.

In all fairness, one should point out that some of the programmer shortage is synthetic. More often than we realize, the job promised and scheduled is probably impossible under the constraints established. The use of "programmer shortage" to explain the inability to do the job may be an acceptable reason, but it is not always the real reason.

There are three fundamental ways of increasing our supply of skilled programmers. The first is that of recruiting more and higher quality people into the ranks. The emphasis should be on bringing in people who are better in actual accomplishments or potential than the average of the people already on board. The second step is that of upgrading the programmers now on the job. Programmers are like any professionals, if they are not continuously challenged and encouraged to think about advanced problems within their fields, they will sink into ruts of doing the same job over and over again. Their skills will become more and more restricted and obsolete as the rapidly changing field passes them by.

The nature of the software field is that of change--change in equipment, change in notation and techniques, and change in fashions. (The computer field is just as subject to fashion changes as any other active human endeavor.) A programmer who does not keep up will in time be left behind or become narrowly specialized, unaware of the sweeping changes in other branches of his field. This "professional updating" problem is certainly an expensive one for the Company, but it is not unique to any one field. It must be attacked by the usual combination of education, seminars, rotation of assignments, etc.

The third way of improving our programmer skill level is that of removing people that are not up to the quality we wish to have. This is also more difficult than it may seem, because the natural tendancy is actually the opposite. Really good programmers--the ones who have really done well on some large system or application problem--have a strong likelihood of being promoted into positions where they no longer actually do any programming. They become managers of less skilled people, and spend all their time worrying about schedules and attending meetings. Conversely, the people who have not done very well on previous assignments tend to stay on in similar assignments. This is a universal problem, and is not necessarily bad for the individuals concerned, but it is a fact of life for the company trying to upgrade its programming manpower. This form of attrition of skilled programming personnel into management has been

more serious in the past than it should be in the future, simply because there will be many more skilled programmers in existence than there have been.

The programming field has tended to suffer on all three of the fronts mentioned. Because of the tremendous, sporadic demands for programmers, there is a tendency to bring in people who are not quite properly qualified. Secondly, the ones who are on board are never really encouraged to keep up with the latest developments of their fields because of the crises of the moment. And third, the really good people are the ones who tend to leave the profession, as quickly as possible.

6. WHAT IS THIS ISSUE OF "PROFESSIONALISM" WHICH SEEMS TO PLAGUE THE FIELD?

This is a topic which has been a "hot one" among our systems engineers for some time. This is another complicated issue which goes back to the conflict between the goals of a research professional and the goals of business. We should admit frankly these are disparate goals and that we should never expect anything to exist other than a dynamic shifting balance from day to day.

IBM needs people in many fields who think of themselves primarily as professionals. Particular professionals who recognize and admire the worth of individual accomplishments--particularly those of advanced or novel areas. A professional views his own work as being important when it can be identified as his. The respect of one's colleagues is important to a professional, as well as the traditional academic rights, such as the right to publish, the right to follow ideas where they lead regardless of previous intentions or promises, etc. These are always in conflict to the sharply defined deadlines and specifications characteristic of commercial software activities. Another proposal or another compiler is always due on a tight schedule. There is also a large amount of repetitive work, such as doing the same job over for a different machine configuration.

These types of jobs have to be done in our business, and quite often they can only be done by persons who are also the sensitive professionals. The answer to this problem seems to be one of equilibrium. The professional software specialist must expect to meet the deadlines on important tasks, if he is willing to take IBM's paycheck to do so. A professional who wants to work in a field that is as fraught with deadlines as is computing, must expect to encounter them in his work. On the other hand, the Corporation must realize that it is dealing with qualities which cannot be measured by accounting procedures. Quite often in professional activities, one real expert may be worth a whole room full of dullards, and he should probably be paid accordingly.

Universities have faced this problem for years and have worked out a

reasonably good balance. A professional faculty member has his time divided between required "production-type" work--teaching classes, serving on committees, etc.--and free research time. The university takes the attitude that as long as the employee meets his formal classroom and other administrative duties, he is free to do as he pleases with the rest of his time; provided, and this is an important point, he continues to do research work which is recognized as high quality in his field over the years.

7. HOW CAN THE CORPORATION TELL WHEN ITS SOFTWARE PEOPLE ARE REALLY DOING A GOOD JOB?

This is the general problem of measurement, one of the most controversial questions in the software field. How does one measure the output of a software group? Or the output of a particular programmer, for that matter? This is a subject on which there has been a great deal of discussion and study.

All the standard accounting-type procedures, such as measuring the number of debugged instructions produced per day, etc. are for the most part very misleading if used to imply quality. Attempts to raise "productivity" as measured by any such accounting procedures usually result in an increase in the quantity being measured, and a decrease in quality of the output. A cutting of corners in one area can cause serious mop-up effort at a later date. The maintenance effort or the bad reputation which results from such false economy, can far outweigh the benefits.

This doesn't mean that it is impossible to measure the productivity of a good software group or individual professional. Any professional in a given field can usually tell whether another is really doing good work or not. The same is true of research work in general. There is no real difficulty in telling whether a given physicist is doing good work, or whether a given mathematician's publications are solid. The problem comes rather in who is doing the measuring and for what purpose.

Accountants, or business managers who tend to use accounting-type procedures for measuring everything, have difficulty in assessing whether a given scientist's work is worthwhile or not. This results in attempts to use the number of papers, or the number of lines of code written as measures of quality.

In other cases, managers resort to some form of the "jury system". In it, a group of people are asked to vote on the quality of the technical job in question. This has certain desirable features, but it will work only if the members of the jury are professionally qualified, and if they are unbiased. Obtaining unbiased, impartial opinions within a business-professional atmosphere is almost impossible. Anyone in the business who knows enough about the given topic to really judge it is practically

always already involved in the effort, or is a member of a rival group which stands to gain by the other's demise.

The problem of measuring is thus not in the accumulation of statistics, but in who is doing the measuring and for what purpose. A program may be very good from the programmer's view of technical excellence, but may be a net loss from the business point of view if it cannot show any influence on the marketplace. The solution seems to lie along the academic lines mentioned in Paragraph 6--some jobs should be clearly laid out and closely checked, others should be long-term investigations measured only by professional excellence.

8. WHAT IS THE OPTIMUM SIZE FOR A PROGRAMMING GROUP? CAN LARGE GROUPS REALLY DO GOOD WORK?

This is another question to which there is no exact answer. Good work has been done by large groups and good work has been done in similar areas by quite small groups. A larger number of people can certainly do a bigger job than a smaller number, but the payoff is probably less than linear. The smaller group can "stay loose" and make changes during the development of a system by mutual consent, whereas a large group must resort to formal change procedures and documentation.

The requirement to subdivide jobs among a large number of people puts heavy emphasis upon the formatting of programs in terms of the data transmission from one part of the system to another--frequently from one machine to another. It is only by putting this emphasis on data formats that the controlling parts of a program can be isolated. For example, this enables a person working on one part of a program to assume that all data will be presented to him in a certain format under certain conditions. The conditions themselves can also be considered as the values of a formatted condition-table. His program then has the task of generating certain output data which is to be put in given formats for other programs to use. This technique allows many people to work on really large programs without requiring each to know exactly what goes on inside all of the other sections of the program. This technique of formalizing data formats and the separation of the logic from the data in software systems should make much larger and more complicated systems possible in the future. This trend will be helped greatly by the development of higher level descriptive languages, and better techniques of documentation.

There is the danger that such a compartmenting approach may not result in a very efficient program. This is particularly true if the standard formats are not "natural" to the problem involved. An individual writing programs in one "compartment" may spend much effort converting data from a given input format to that which is more convenient, not realizing that the person who was furnishing the data had it in that form originally, but spent considerable effort converting it to the standard format. Thus, a transformation and its inverse were performed for no reason at all.

A great deal of the efficiency of the approach depends on how carefully the original structure has been thought through. The best way is to go through an iterative design, in which the whole system is reworked several times by the same people. Very few planners are capable of the insight required to lay out a set of formats and data flow conditions in the most efficient way the first time.

The tendency to write out volumes of programming specs before ever putting an instruction down on paper is a symptom of trying to avoid the iterative design and come out with a "one-pass" development schedule. This is an expected mode of operation when a very large effort must be made in a relatively short time. "Man-decade" or "man-century" jobs which are being done in a year or two are almost doomed to come out this way.

On the other hand, a small group also has its dangers. The main ones are the individuals may have "blind spots" or tricky methods which may not be good for the project at hand. Surprisingly enough, intelligence and lack of experience can often go together on a particular job. There is also a serious problem in sustaining a program after it has been written by a few isolated geniuses. The cleverer they have been, the harder the documentation and maintenance becomes.

9. WHAT DO YOU THINK IS THE MOST SERIOUS PROBLEM IN THE SOFTWARE AREA TODAY?

If I had to vote on the most serious basic problem in the software area today, I would pick the general problem of documentation. This is a real problem to users and to originators alike, although it is perhaps more apparent to the users. Anyone who has tried to fight his way through our manuals trying to find out what happens in some obscure particular case, usually ends up completely frustrated or resorts to experimentation on the computer to find out what really happens.

Documentation of programming systems, of applications programs, and of the computers on which they are based, are all almost without exception very difficult to read. If they are written with a real attempt to describe exactly what will happen under all combinations of circumstances, they become almost unreadable and legalistic. If they are written to be readable and give concepts only, then they are useless for answering detailed questions.

Certainly the problem of keeping systems documentation updated is a tremendous drain on the Company's resources. It is not only costly in materials and distribution expense, but a large number of people are required to fight this almost hopeless battle of keeping all our manuals and write-ups current and consistent. At the receiving end, a user almost needs a librarian to keep track of the series of changes and modifications received every month. The problem is equally serious

when users do not receive the changes, so that they are relying on outdated manuals unknowingly. It is also a fact of life that the documentation is always the most fragmentary and inaccurate just when it is needed the most--on new systems, or during large modifications.

10. WHAT CAN BE DONE TO SOLVE THE DOCUMENTATION PROBLEM?

The detailed description of either the hardware or software of a computer system is not a task for which ordinary English prose is particularly well suited. One of the main premises of COBOL, for instance, is that one can describe a complicated commercial program in stereotyped English phrases which "anyone can understand". My feeling is that this premise is probably fundamentally wrong. Complicated things are complicated and require complicated descriptions. If they appear simple, it is because they are not really being described in detail.

Of course, one can and should describe a given system at many different levels of detail, but this is one of the basic problems of documentation --How many levels can we afford to write, and how do we keep them consistent?

A simple answer to the documentation problem is not an easy one to come by. One of the goals of the ALGOL movement has been the documentation of algorithms for use in applications programs. In this goal, it has been fairly successful. ALGOL presently serves as an international medium for transmitting ideas in a fairly concise form between professionals who use it--mostly in university circles. Similarly, the FORTRAN programs which have been made available through SHARE and other organizations have helped a great deal in transmitting ideas with a high level of precision. Indications are that this approach can be carried much further.

A much better descriptive language, or perhaps a set of descriptive languages, seems to be close at hand. Certainly we can hope that the new programming language, NPL, will serve as a natural successor to FORTRAN, ALGOL, and others of that level--although this is probably wishful thinking. Vested interests in all fields are hard to displace.

Personally, I place a lot of stock in the future of the Iverson notation. It is a very well designed descriptive notation for recording in very precise form exactly what is done in a given program or computer mechanism.

Iverson notation has been very successful in describing the various levels within a computer. The recently published description of the System/360 in Iverson notation was a real tour-de-force, and may well mark the beginning of a new standard of computer documentation. Experiments show it

equally good for problem descriptions. This field of powerful descriptive languages and notations is one in which IBM needs to maintain a continuing effort, because the potential payoff is so great.

One possible solution to a phase of the documentation problem would be to develop self-documenting programs which will themselves document other programs. There have been a number of experiments involving successful flow-diagram-drawing programs, and other forms of self-documenting programming languages. This development is another which IBM should support internally from our own point of view, because we stand to gain most from it. Programs which can translate from older, less well-documented versions to newer systems should also be pushed. These have been quite successful in particular cases in the past. A major effort to aid customers to convert to System/360 by such automatic or semi-automatic methods should be high on our list of software projects. If the conversion can also result in improved documentation, then there will be a much bigger incentive to make the conversion.

11. WILL USERS REALLY FOLLOW IBM TO OUR NEW PROGRAMMING SYSTEMS?

The Corporation has probably become "gun shy" from listening to its very vocal critics at some installations. The answer is that the majority of our customers will indeed follow our lead, provided our Systems Engineers can break trail for them on a local program-by-program basis.

One of the problems which any computer user faces every day is that of making changes in an error-free consistent fashion. However, systems programmers themselves are much more willing to make changes in formats, procedures, notation, and syntax, than the programmers who are applications oriented are willing to follow these changes. Applications programmers and users generally are willing to learn new procedures and notations if there is an obvious benefit to them to do so. However, they become very unhappy if such changes happen too often, even if they are beneficial. Users will generally rebel if they are expected to follow the week-by-week gyrations of the systems programmers. Their patience is very short when it comes to making changes for the benefit of the system only.

This resistance to change on the part of applications programmers should be listed as one of the basic problems of software systems. This does not mean changes will not eventually be accepted, nor does it say anything about the magnitude of the changes. A small change may be more infuriating than a really big change. Users are more likely to make infrequent large changes rather than many small changes. It may, for instance, be easier to encourage users to switch to NPL than to get them to reprogram to another brand of FORTRAN. For example, many FORTRAN II programs are still being written (or at least being updated) rather than being converted to FORTRAN IV. Human factors seem to favor large changes possessing large enough advantages to overcome the inertia of habit and experience.

12. HOW IMPORTANT WILL TIME-SHARING AND REAL-TIME SYSTEMS BE TO IBM?

Although they are often discussed together, these days real-time systems and time-sharing systems are really quite different concepts.

Real-time systems imply the collection, analysis, and presentation of results directly from experiments or processes while they are going on. They may or may not include elements of feed-back control.

Time-sharing is also an important new development which allows a number of different programmers to use a single computer system by means of separate consoles. Such a system is a real-time system of sorts in that it interacts with the human programmers in real-time, but the intent and details are quite different from process control real-time systems.

It seems to me that much of time-sharing's present importance arises from the fact that it is the fashionable thing to do. Everyone seems to be rushing to emulate Project MAC, because it is "dernier cri" in computing. A more evolutionary approach based on extending the transmission of data to queues for a more traditional batched-processing monitor seems likely to give much better economics for the average user. Professor Miller of Stanford has ideas for implementing such a system which sound very promising.

Of course, IBM should follow the fashion designers and develop one or more time-sharing systems, but it should put more emphasis on systems which show better utility and allow for easier evolutionary development and modification.

One thing we can be sure of is that real-time systems of all sorts are going to become increasingly important all through industry, including management data systems.

One of the big problems which faces the software strategist in this area is that the borderline between hardware and software becomes much more complicated and difficult to distinguish. The computer is no longer an isolated device under the control of computation center personnel. The measuring devices connected to the physical process being monitored, the analog-to-digital converters, the interrupt or sampling system, are now really inside the computer system like the proverbial camel in the tent. Questions of reliability of the equipment are quite serious, but the reliability of programs is just as important--and even harder to define. What happens is that large trade-offs become possible between hardware and software in such real-time control systems. The programmer must understand engineering or, perhaps more importantly, the engineer must understand programming to achieve the best systems solution.

###

H. R. Hoberg
Feb 10, 1965