THE TRAINING NEEDS OF A SOFTWARE HOUSE

Mr. A. d'Agapeyeff

Computer Analysts and Programmers Limited,
CAP House,
14-15 Great James Street,
London, W.C.1.

Abstract:

The context in which the young graduate can expect to work as a professional programmer in a British software house is described. The in-house training methods and the attitudes inculcated are inimitably discussed for a particular organisation.

Rapporteurs:

Mr. D. Appleton
Dr. H. I. Scoins

The Chairman (Professor E. S. Page) in introducing the speaker referred to the fact that Mr. d'Agapeyeff was President elect of the British Computer Society, but said that one could not expect that this would in any way restrict the force with which controversial arguments would be presented. The speaker was not sure if he would be constrained, but suggested that age mellowed one a little. In any event, he was speaking for himself alone.

1. A dissent

Mr. d'Agapeyeff commenced by expressing his dissent on behalf of professional programmers from some of Professor Dijkstra's views expressed earlier in the day. Notwithstanding Professor Dijkstra's unsurpassed contributions to the art which were continually being exploited by the professionals, they could not regard programming as a branch of Mathematics or of anything else, except perhaps Communications Engineering. Moreover, the emphasis on puritanical correctness offended the speaker's own puritanism which insisted that, as professionals, they were the servants of the public. It was not particularly helpful to learn that a certain process took 3N as opposed to 2^N operations if N were 100,000. Programmers were nowadays liable to be dealing with problems which no individual alone could be expected to understand and solve, owing to the large number of people involved in an essentially dynamic situation. Nevertheless, the programming work must be done and neither elegance, beauty nor correctness were vital, only necessity. Computing Systems ought to be built more like engineered constructs, such as the telephone system, so that they never fail completely but admit a certain fallibility and therefore include effective recovery processes to maintain a service, despite the fact that not every piece of logic is perfect. If the consumer can be led to expect a sustained service from a computer system the job is well done, but if we have to wait until it can be done with absolute correctness, then, in many applications, it might never be done at all!

2. Software houses and their clients

In the United Kingdom, according to the Ministry of Technology, there are about 290 software houses with an average of ten people, but there are four large houses with over 200 staff each. Although expanding rapidly, they are miniscule relative to their American counterparts.

Broadly, they can be divided into those that are attempting to be truly professional technologists, those who act like a secretarial agency and those who trade, in that they tend to provide specific and possibly restricted services.

The largest users of computers and of software houses are commercial organisations. For example, the London branch banks are investing £30-40 million each on their computer systems with 2,000-3,500 terminals to be attached. Some software houses specialise in small coding jobs while others are involved with the design and implementation of such very large systems.

The computer manufacturers were traditional clients of software houses, requiring compilers for particular languages on particular families of computers. Similar clients now include terminal manufacturers and telecommunications companies, while the areas of application have grown to include the preparation of packages for dedicated systems (where the standard software was not sufficiently flexible) and the implementation of commercial and light engineering programs even for computer manufacturers!

The nationalised boards require programming for both industrial and for commercial work. The airlines are the second largest category of computer user in this country and have very large and successful systems which are the living proof of large systems that no-one fully understands but which work most of the time. However, the loss of a potential airline passenger owing to a program failure does not cause the same fuss as an error in an individual's pay packet!

Industry is not as big a customer as one might expect. Industrial automation, such as production control, is particularly difficult because these systems can currently only be partial, and must interface with a great variety of people, not all of whom are sympathetic to, or understand, computing. Such a system cannot be organised afresh, but must adapt to the existing organisation.

Scientific establishments bring in work varying from the process control of laboratory equipment to quite large systems such as the fashionable multi-access systems which remain technically onerous. Finally, central government, with minor exceptions, know too much about computers to patronise software houses!

The Chairman (Professor E. S. Page) in introducing the speaker referred to the fact that Mr. d'Agapeyeff was President elect of the British Computer Society, but said that one could not expect that this would in any way restrict the force with which controversial arguments would be presented. The speaker was not sure if he would be constrained, but suggested that age mellowed one a little. In any event, he was speaking for himself alone.

1. A dissent

Mr. d'Agapeyeff commenced by expressing his dissent on behalf of professional programmers from some of Professor Dijkstra's views expressed earlier in the day. Notwithstanding Professor Dijkstra's unsurpassed contributions to the art which were continually being exploited by the professionals, they could not regard programming as a branch of Mathematics or of anything else, except perhaps Communications Engineering. Moreover, the emphasis on puritanical correctness offended the speaker's own puritanism which insisted that, as professionals, they were the servants of the public. It was not particularly helpful to learn that a certain process took 3N as opposed to 2^N operations if N were 100,000. Programmers were nowadays liable to be dealing with problems which no individual alone could be expected to understand and solve, owing to the large number of people involved in an essentially dynamic situation. Nevertheless, the programming work must be done and neither elegance, beauty nor correctness were vital, only necessity. Computing Systems ought to be built more like engineered constructs, such as the telephone system, so that they never fail completely but admit a certain fallibility and therefore include effective recovery processes to maintain a service, despite the fact that not every piece of logic is perfect. If the consumer can be led to expect a sustained service from a computer system the job is well done, but if we have to wait until it can be done with absolute correctness, then, in many applications, it might never be done at all!

2. Software houses and their clients

In the United Kingdom, according to the Ministry of Technology, there are about 290 software houses with an average of ten people, but there are four large houses with over 200 staff each. Although expanding rapidly, they are miniscule relative to their American counterparts.

Broadly, they can be divided into those that are attempting to be truly professional technologists, those who act like a secretarial agency and those who trade, in that they tend to provide specific and possibly restricted services.

The largest users of computers and of software houses are commercial organisations. For example, the London branch banks are investing £30-40 million each on their computer systems with 2,000-3,500 terminals to be attached. Some software houses specialise in small coding jobs while others are involved with the design and implementation of such very large systems.

The computer manufacturers were traditional clients of software houses, requiring compilers for particular languages on particular families of computers. Similar clients now include terminal manufacturers and telecommunications companies, while the areas of application have grown to include the preparation of packages for dedicated systems (where the standard software was not sufficiently flexible) and the implementation of commercial and light engineering programs even for computer manufacturers!

The nationalised boards require programming for both industrial and for commercial work. The airlines are the second largest category of computer user in this country and have very large and successful systems which are the living proof of large systems that no-one fully understands but which work most of the time. However, the loss of a potential airline passenger owing to a program failure does not cause the same fuss as an error in an individual's pay packet!

Industry is not as big a customer as one might expect. Industrial automation, such as production control, is particularly difficult because these systems can currently only be partial, and must interface with a great variety of people, not all of whom are sympathetic to, or understand, computing. Such a system cannot be organised afresh, but must adapt to the existing organisation.

Scientific establishments bring in work varying from the process control of laboratory equipment to quite large systems such as the fashionable multi-access systems which remain technically onerous. Finally, central government, with minor exceptions, know too much about computers to patronise software houses!

3. Programming — a definition

To avoid confusion, it is appropriate to define programming, and so distinguish between what is meant herein by a 'programmer', the unidentifiable 'systems analyst', and the almost mechanical 'coder'. Although the definition of Duncan and d'Agapeyeff was given in 1967, it is still valid to say:

> Programming is all the work directly involved in the implementation of an identified application together with advice on the exploitation of available computing facilities.

Thus, the professional programmer necessarily combines a knowledge of the possible with that of the desirable to produce the essential compromise which is contained in the best advice.

The professional is necessarily involved to a greater or less extent with the two broad categories of technical and managerial activities. The personnel also range from the back-room boys, who should never meet a client, through the few who claim to cover the two facets, to the professional managers without whom no profits can be made.

4. Technical activities

Both business and scientific problems need analysis. The young graduate is liable to be over-confident and will often imply that he knows better than the client how to run the latter's business. It is vital to understand the instructing client and his business; how he thinks, what pressures he is under, what his aims are. Business people are often much more tolerant of, and patient with, bright young men than are scientists; but the programmer must be prepared to go outside his own field and immerse himself in seeing someone else's problems from that other person's point of view.

Although working with scientists is always exciting, it can be a struggle, particularly when they have themselves learnt, say, Fortran and think they know it all but don't quite understand why they have fouled up their project. There are now a number of establishments where programmers from software houses are attached to specific research groups, supervising the programming and the associated computer runs.

Design is rather difficult compared with analysis, for one is only too aware of the incomplete specification of what is to be done, and yet the job must be done. There are different ways of organising the work of design such as using separate design teams or using part-timers as sounding boards. It is clear that one cannot afford to design in an iterative manner down to the coding level, but must first correctly fix the overall strategy. It is not clear how this should best be done.

A distinction can usefully be made between the software for a batch processing project, and that appropriate to the organisation of a large system of programs and machines which must never fail completely (i.e. an on-going computing service). The traditional approach in operating systems assumes that a queue of independent programs is waiting to be run on a computer, and if an error occurs the proper action is to terminate the current program.

However, such an attitude is inappropriate in a large on-line business system using a network of inter-dependent programs. In this case, programs must not be automatically thrown off the computer; only the data which caused the fault. Such projects may well require special scheduling software and special recovery monitors.

The activities of the implementation and the coding of smaller jobs are less interesting, and naturally fall to the newcomers. It is important hackwork where mistakes can only be minimised by careful disciplines and good project control.

Software facilities or tools are worthy of special consideration and are liable to be ignored by young graduates. They may be general tools such as compilers, editing facilities or information retrieval systems; or specific tools which have to be developed to assist in the programming and coding of a particular project. Recently the speaker had been involved with a facility in which a file was maintained on the computer giving a complete description of the software system being developed. Retrieval facilities allowed the consequences of any change in specification (e.g. to a data element) to be rigorously followed through to every relevant module and subroutine.

5. Managerial activities

At CAP, young graduates are given some training in the management techniques of marketing and selling by specialists in those areas, including some from outside the company. The general approach is that it is no good being an inarticulate and frustrated back-room boy unable to sell his ideas or techniques to clients or colleagues. Selling techniques have been well received in general although some trainees felt them to be unfair to the unsuspecting customer.

Proposals and reports, however, tend to be very poor. An incredible amount of senior management time is spent in trying to get the young men to express their ideas sensibly, logically, clearly and with some sense of the priorities of importance. There seems to be no solution to this very worrying problem.

Project management is of increasing importance. It may not be obvious at University that, even in this country, teams of up to 100 people may be working on one implementation and medium-sized groups of 10-20 people are quite common. A professional programmer must be prepared to work as a member of a team and there is no room for the complete individualist in a software house.

In personnel matters it is relatively easy to take account of the immediate requirements of juniors but new managers seem to find it difficult to appreciate the necessity of worrying about the long-term needs of career development in their staff. Again, an outside lecturer can help understanding.

Business policy has to be impressed on new staff, and old, from time to time. A charitable tendency will be detrimental to their own bonuses, and to that of top management!

6. General attitudes

The foregoing sets out in crude form the context in which a young professional programmer can expect to work. An understanding of the service aspect is of prime importance and the idea of strategic design is next.

The preparation of strategic documents before one gets down to flowcharts is too frequently ignored, to the detriment of the final product. The crucial part of a design may be how a large file or data base and the records within it are partitioned and how this fits in with a sensible access

system which may have to be developed independently as the standard I/O system is possibly quite inappropriate and time-consuming. Often the minimisation of the corruption of information, after a machine fault, is of vital importance. It is in this kind of context that strategic designs must be produced, and must be examined at the start and not half way through the project.

Along with teamwork goes supervision, control and a proper sense of professional responsibility. In too many installations, today, the unchecked ideas of a single individual are implemented without adequate supervision.

In CAP no one is allowed to go too far on his own and it is taken as natural and essential that strategies are discussed and reviewed. Intellectual honesty is the foremost of professional values. Making debating points in discussion may make you unpopular, but the capital crime is to fail to admit you don't know, when appropriate, or to attempt to cover up a doubtful situation. This last, knowingly to omit to report a danger or weakness is the fastest way to get yourself fired.

7. The initial training course

When the graduate first arrives at CAP, he is put on a course which appears to be primarily concerned with some programming language, often COBOL. The overt reason for this is to make him a useful member of the company as soon as possible; a comparatively short course in the language allows him to join project teams as a coder.

However, the course is really a vehicle to enable the company to inform him about the company itself, its practices in computing and indeed, to some extent, about computing generally. If you like, it is a kind of gentle brainwashing. The graduate frequently has a very false idea as to the nature of computing. It may come as a surprise to him to learn that it is not primarily concerned with calculation but with administration and management communication, mostly applied to the business area. He may be startled to learn that some of the most advanced software developments are taking place in such organisations as banks and airlines.

The initial course also prepares the graduate and cushions him from certain other shocks. Project discipline may be something new. Instead of a gentle university tutor, who is interested in the student's ideas and viewpoints, he is likely to encounter an overworked, short

tempered project manager who requires a specific piece of work to be done and who may not be too sympathetic to the graduate's idea of how the whole project should be re-designed!

Once a man gets on to projects, the criteria for his subsequent success in CAP is fair but a little brutal. Group managers and team leaders have a certain freedom of choice in the membership of their teams. The best men are, therefore those in constant demand while their weaker brethren are to be found sitting around the office with nothing to do. To be excluded from a given team once may not be significant, as personalities still count for a lot in a business in which dealing with people is so important. Yet, to be excluded two or three times could well imply that the man should re-consider his professional career, for ability to cooperate with others as one of a team is or prime importance.

8. Other training features

The young graduate is regarded as an apprentice, for although a limited amount can be taught by exercise in the classroom, it is only by sitting alongside experienced people actually engaged on a job that design aspects and the practical problems which arise on projects can be learnt.

Every member of a team tends to be involved in two types of meeting. There are project meetings, usually 'in-house' on a weekly basis in which any difficulties arising on the project, technical snags and so forth can be ironed out. These meetings are also used to ensure that every man on the team understands the nature of the project and the part he is playing in it. The other type of meeting is normally held monthly with the client management, and deals with progress, bottlenecks over machine time and the like, and all the other considerations which affect the delivery date. For junior members this tends to focus attention on the object of the exercise - a working system which somebody is actually going to use. It is in these meetings that the graduate learns the importance of business administration and accounting on the average job. Even on so-called 'scientific projects' he finds that although numerical analysis may be a crucial item it is often only a small part in the data handling and the treatment of output. There are, of course, many other types of meeting held internally on an irregular basis. There are those designed to create a proposal to a client for the carrying out of some implementation. Here the emphasis is on estimating, on anticipating the latent ambiguities in a specification and on forecasting

the inevitable foul-ups which are always likely to occur somewhere, probably in the least expected place. Here, any tendency towards brash over-confidence in what an individual can actually achieve is briskly corrected. The apprentices learn caution and the hallmark of a professional: promise only what you can be sure that you will deliver.

There are also special studies to consider some new development in computing or software. One of the happy advantages of a reputable software house is that it can often persuade a client to pay for it to learn about some new advance, on the premise that, with its experience, its staff will be quicker in gaining a proper understanding than those of the client. Here the graduate is on more level terms with senior members, but he may be surprised at the effort they are prepared to put in, over weekend and otherwise, to digest a new manual quickly and thoroughly. Finally, there are seminars. These are used to tackle some new problem or to reach a decision on some question whose solution is not known. They are a form of brainstorming, usually under some very senior chairman to keep overall control. The aim is to focus experience from different backgrounds and different projects on to some particular project. Junior members are encouraged to attend in order to gain experience, but it also helps them to display their ideas. Talking for the sake of hearing one's own voice or to make superficial debating points may be roughly treated.

9. Problem areas

There are many problems which arise over training in a software house. Some of which have, as yet, no satisfactory conclusion. These include:

1. Second-year despondency.
2. The 'not invented here' factor (NIH factor).
3. Over-optimism.
4. Rotation on long projects.
5. Design compromises.
6. Packaging.

9.1 Second-year despondency

Many companies have a high rate of turnover of graduates who have been with them for about two years. This restlessness arises from a despondency over the length of time it takes to become a real professional in programming, a desire to be regarded as a more complete expert enjoying the

status of a title - like Systems Analyst - and, of course, the possibilities of greater financial rewards elsewhere. Although working in a computer user's installation may be less appealing, as there is little formal training and only a limited type of work, this disadvantage is often offset by salaries which are excessively high for half-trained people.

9.2 NIH factor

This term refers to the tendency to re-invent techniques already well established. To some extent, life in a university seems to encourage working over old ground and it is very difficult to persuade graduates to accept strategies and methods which are popular with management simply because they are known to work. It is also a real problem for the professional to keep abreast of developments in this rapidly changing field.

9.3 Over-optimism

This is the Achilles' heel of all computing. Users may ruin valid applications by excessive ambition. Programmers tend to feel that they can readily knock off some program on the assumption that everything will turn out right. In a sensible software house there is little pressure to persuade people into promising early dates. On the contrary, management tends to add safety factors and provisos. On the other hand, when a man has promised a date, enormous pressures may be applied to ensure he keeps to it. A constant flow of slippages is an invitation to managerial severity.

9.4 Rotation of long projects

Large-scale projects are tending to take longer and longer to implement. This may not be too bad for the senior staff who will become personally involved in the project success, but the junior teams may not have a sufficient understanding of the aims and importance of the project to share that involvement. They may feel themselves to be tiny cogs when 30 or 40 people are engaged in a project. It may also be difficult to give them fresh things to learn, particularly during long-drawn-out debugging phases. Thus, it becomes important to rotate staff, which is never easy owing to the time required for familiarisation and special training in the details of a new implementation.

9.5 Design compromises

These have been discussed earlier, but it is worth repeating that compromise is the essence of design and an inevitable part of the work, since no design will ever be perfect. It will often be better to produce a slightly

inadequate program on time than to wait upon further refinements and polishing. A more subtle difficulty arises from the realisation that when a user department puts some work on a computer, the options open thereafter, in terms of possible changes, may be substantially reduced. The programmer must, therefore, attempt to anticipate the constraints which may only be implied by the user, and he must try to construct a framework of generality in which the design is embedded.

9.6 Packaging

In addition to the need to make the finished produce look good to the user, it is essential to create software components as packaged entities which could ultimately be transferred from one computer system to another, be maintained by other people and, preferably, be exploited in new situations. It is all very well writing beautiful binary programs but they are usually non-portable and rapidly become useless. An inducement to better packaging and proper documentation can come from an insistence that any member of the team should be able to understand most of the component parts.

10. Conclusion

The speaker felt sure that a number of those in the audience would have bricks to throw, but emphasised that, although he was occasionally upset by what students were not taught, and might suggest additional material, there was no doubt that first class people were coming out of the universities. It was pleasant to record that despite all the alleged troubles of the younger generation, the present graduates were nearly all well balanced people and had the additional merit of being very, very bright.

11. Discussion

Professor Perlis began the discussion by suggesting that the kind of trainee Mr. d'Agapeyeff wanted in his Company was, in fact, one who had been taught by Professor Dijkstra.

Mr. d'Agapeyeff agreed that anyone whom Dijkstra had taught would have learned much of what he himself regarded as important, and that he would welcome such people because they would have learned appropriate tools and an excellent approach to programming; because indeed they were taught programming and not numerical analysis for instance. He intimated that, while he could only speak for the European software houses, he believed that some of his requirements were shared by other people.

Professor Bauer emphasised the importance of training young people in such a way that they can be expected to use their own initiative to keep up with the development of their field for 30 years. This meant that they should learn the scientific aspect even if they were not going to work as scientists and that to give them only the present status was insufficient.

Mr. d'Agapeyeff agreed that it was the general method which is important, but said that this was best given by examples in actual practice.

Professor Hoare felt that the training costs implied by what CAP did with their trainees could not be compared with the fees paid by University students. Professor Page quickly intervened to point out that the average cost per student in advanced education in the United Kingdom is about £1,200 per student-year plus about £500 for his maintenance.

Professor Michaelson suggested that what the software houses themselves were doing was merely short-term training and that they were providing no basis for the future, but Mr. d'Agapeyeff disagreed. It was his view that they were encouraging and teaching their trainees to think.

Professor Michaelson then asked why the software houses recruited only graduates, to which Mr. d'Agapeyeff replied that it was the best way to ensure a reasonable proportion of successful trainees. He pointed out, however, that after a year in the software house the degree taken by the trainee was immaterial.

Dr. Hartley asked whether Mr. d'Agapeyeff agreed with the idea of giving all undergraduates a course in computing science.

Mr. d'Agapeyeff deplored the impossibility of obtaining a degree in computing science without having studied a significant amount of numerical analysis. The real necessity was to teach programming and business studies courses.

Professor Perlis suggested that the time was now past when numerical analysis was a necessary part of a university computing course, and Professor Michaelson pointed out that the British Computer Society professional examinations still required a knowledge of numerical analysis.

Mr. d'Agapeyeff blamed the latter position on the fact that the setting up of the examinations had been largely carried out by academics. He anticipated some changes in the near future.

Reference

Duncan, F. G. and d'Agapeyeff, A. Computer Journal, Vol.9, p.229, 1966-67.