Training Software Designers:

Lessons from a Development Project

Peter Freeman

July, 1975

Technical Report #70

This is a preprint of a paper to be presented in November, 1975 in Baghdad, Iraq, at the International Conference on National Planning for Informatics in Developing Countries.

ABSTRACT

A number of important considerations for training software designers are described and illustrated in the context of an actual training project. Suggestions for training software designers in developing countries are made on the basis of this experience. The paper also includes a description of the project and some general comments about software design training.

*This work was performed while the author was a UN Visiting Expert at the UNDP-supported International Computer Education and Information Center (SZAMOK) in Budapest, Hungary.

1. INTRODUCTION

Every computer program ever created involved some design. As programs and systems of programs become larger and more complex, the role of design becomes increasingly vital. The recognition of this fact can be seen in the increasing emphasis on design in textbooks (3,7), curricula (1,6), and professional meetings (2).

Training software designers is a difficult task under the best of circumstances and seems to be especially critical in developing countries. This stems from the fact that most software designers learn their skills by working alongside more experienced designers; when a country or an organization is just beginning to create a basis of computer personnel, this form of training is difficult to provide.

Numerous suggestions for training software designers can be made. Many seem like just good common sense when described, but are then found to be difficult to apply in practice. In this paper I will try to avoid this problem by illustrating the application of various ideas in the context of a particular training and development project on which I have been serving as an advisor. (These suggestions are also based on my experience teaching design to univeristy students.)

While this project is being carried out in a technically advanced country (Hungary), the organizational setting of the project has many of the characteristics found in developing countries. Thus, most of the suggestions made should be directly applicable by readers of this paper.

1.1 Three fundamental views

Underlying much of what follows are three basic premises. The first is that program design is different from programming, even though they are often closely associated. Software design involves specifying the functions to be provided and choosing logical structures (for both data and control) necessary to carry out the required functions. Furthermore, these decisions are usually made at several (interrelated) levels of detail. While such decisions are ultimately concerned with programs and may involve the choice of some detailed parts of programs (such as data representations), the design phase of program creation usually stops short of specifying each and every instruction. More detailed discussion of program design can be found in numerous places, including (3, 5, 7, 8).

The second premise is that one can only learn how to design by doing it. Reading papers and listening to lectures may prepare one for and expand one's conceptual understanding of design, but there is no substitute for actually doing design work. The obvious corollary is that one must design a number of systems and gradually build up expertise.

The third premise is that software design normally involves more than technical knowledge. Since it is often a group activity, management procedures, documentation techniques, and inter-personal psychological factors play an important role that must be understood and properly handled.

1.2 Central focus of this paper

I will concentrate on the software design aspect of systems, not on the total task of systems analysis, setting of organizational goals, coding, testing, maintenance, and so on. Many of the suggestions made may be applicable to training for these other tasks as well, but others can make those extensions if appropriate.

Also, it is important to note that I am addressing the problem of training software designers, not the problem of how software designers should do their jobs. Of course, designers must be trained to use the proper techniques and there are obvious extensions of what I say below to the activity of designing software. In addition to the other references given in this paper, one can consult such publications as IEEE Transactions on Software Engineering, Journal of Software Practice and Experiences, IBM Systems Journal, and various trade magazines (such as Datamation) for up-to-date information on software design techniques.

1.3 Organization of this paper

Having set the context and scope of my remarks, I will next describe the project to be used as illustration. I will then provide suggestions in three categories for training software designers: type of training and personnel, operational parameters, and technical content. Finally, I make some comments concerning training in developing countries.

2. THE REMOTE TEXT-EDITOR (RTE) PROJECT

This project was initially started in September 1974 and is still underway at SZAMOK. It has undergone several changes in personnel and direction, but these have little bearing on our concerns here. Thus, I will concentrate on the formal design phase of the project which has lasted from April to August 1975. During this time, the project staff has remained steady.

2.1 Project Goals and History

SZAMOK initiated this project for two purposes: 1) to provide better student access to the main SZAMOK machine, an IBM 370/145, and 2) to provide training in software design for SZAMOK staff members. It was decided to use a Videoton 1010 (R-10) computer as an interactive front-end machine to the larger 370. The R-10 will have eight video terminals, disk storage, an interactive text editor, and a remote-job-entry connection to the 370/145. This will permit students to prepare small job decks interactively, submit their jobs to the 370 via the communication link, and inspect their output on the video terminals after it has been sent back to the R-10 from the 370.

For various reasons, a number of decisions were made before the design phase started. These included: the RTE system will be built on top of the existing real-time monitor (RTDM) on the R-10; the well-known QED text-editor will be used; eight terminals will be supported; the system will be written in assembly language; and several other lesser decisions. Thus, when the design phase started, the task was well defined. (This is important, as it makes it clearer for the trainees what they must do.)

In particular, the following facilities had to be designed: QED, a file system, the remote job entry interface to the 370, a memory manager to take care of swapping edit-buffers to and from disk, an I/O manager to coordinate use of the I/O facilities of RTDM, a command language interpreter to provide the user interface at the terminals, a traffic controller to coordinate the flow of control between the various parts and to schedule allocation of processor time, and several smaller pieces.

The design phase lasted 4 months. The first month was spent in three activities: 1) refinement of requirements, 2) review of work already done and 3) continued work on implementation of a prototype system which had been undertaken as a familiarization project. The next two months were spent primarily working on the design of the RTE system, with some additional implementation work on the prototype system (particularly the QED module which will be the same in the final system). The fourth month was spent in putting the design documentation into the form of a case study for teaching and in preparing an implementation plan for the design.

In developing the design of RTE, we proceeded in a roughly top-down fashion, being careful at all stages to explicitly record our decisions and the alternatives we had considered.* That is, we first carefully detailed the requirements for the overall system. We then decided on an overall division of the RTE into major facilities (those named above) and further refined the

*It is suggested in (4) that recording the reasoning that led to design decisions will make the design more reviewable. In this instance, the design will form the central part of a case study to be used in teaching, making this especially important.

specifications for each of these parts. Then rough decisions concerning how each part would work (i.e. data representations and control structures) were made, followed by further decisions following from each of these high-level decisions. Once we felt we understood the developing form of a facility well enough, we represented the various programs of that facility in a metacode form (Algol-like control structures plus natural language statements of actions); this representation also was carried through two levels of detail. Additionally, external specifications and other descriptive material were prepared for each facility.*

2.2 Personnel of the project

The choice of people to be trained as software designers is a critical issue. This subject can be treated extensively, but my discussion here must be limited to a characterization of those participating in this project. In this way, the reader can at least obtain a feeling for the level of personnel on one project.

The design phase has been carried out by five staff members of SZAMOK with the assistance of the author.** I have served essentially as project leader, providing overall direction, leading group meetings that worked out the first levels of detail on the individual facilities, and carrying out the detailed design on one facility (the file system). My background consists

*The design methodology employed in a training project is of prime importance since it will form the model for future design efforts of the students. Its choice deserves much more careful consideration than space will permit here.

**Mr. Per Ofstad, UN Expert from Norway, worked with some members of the group at the start of the project; we benefited from his preparatory work.

of 14 years of work in computing (including the design of a system similar to RTE), research on design methods, and teaching of design and software engineering methods to advanced computer science students.

The SZAMOK staff members consisted of two regular members of the Software Development Department and three instructors assigned to the department on a temporary basis for job training. They can be roughly characterized as having strong educational and professional backgrounds but as being weak in actual programming and design experience. Table 1 provides a profile of each.

This dual characterization -- well-educated, but lacking in actual experience -- can be expected to be found in developing countries where machine time and/or opportunities for programming experience are in short supply. As we note below, this lack of programming experience caused problems and ideally should be remedied before design training begins.

2.3 Project evaluation

Until our design is implemented (which is just now starting) and until the staff members try to use their new skills in teaching or in other projects, it is impossible to evaluate it objectively. My feeling as a teacher is that the project personnel have benefited and are now capable of more than they were before -- they certainly understand better some of the difficulties in doing design. Their ability to deal with complex situations at a logical level, to analyze a problem and throw out unimportant factors, and to represent their design ideas has improved. They also feel the experience has been beneficial.

Table 1: Project Personnel Profile

	A	B	C	D	E
	(dept. head)	(instr.)	(dept. member)	(instr.)	(instr.)
Number of yrs teaching or working in computing	5	5	8	5	2
Number of machines whose structure is understood	5	5	4	5	2
Number of languages (reading knowledge)	8	3	5	5	1
Educational level	M.S. math/phy.	M.S. engrng.	M.S. engrng.	M.S. math/phy	M.S. engrng.
Prior software design experience	some	none	none	none	none
Number of programs written and debugged	<100	<100	<10	<100	<10
Largest program created (number of lines)	500	1000	200	700	200

3. IMPORTANT CONSIDERATIONS FOR TRAINING SOFTWARE DESIGNERS

There are many things to consider in planning and carrying out a training program for software designers. In this section I have tried to touch on those which can be illustrated from our experience on the RTE project. I have arranged them into three broad categories to facilitate comprehension.

3.1 Type of personnel and training

Clearly the first thing one must consider is what type of people to train and what basic form of training to provide them.

3.1.1 Minimum qualifications are essential. Software design is an intellectually demanding task for which not everyone is well-suited. A good general education and a familiarity with computing in general is necessary; the ability to deal with a large number of details and to separate them into manageable groupings is critical; first-hand experience with programming and the use of software systems is indispensable.

In the RTE project, as noted in Table 1, the personnel were well-educated and familiar with many computing concepts. This background made it easy for them to deal with new concepts that arose in the design as well as to carry out complex tasks (subparts of the design) without much direct supervision.

Their lack of programming experience, however, seriously detracted from their performance as designers. When considering design alternatives, they did not have the intuition of an experienced programmer which would have permitted them to make such decisions more easily. Further, since some of them were not fluent in a higher-level language, they were not comfortable at first with the design representation we used. Finally, their lack of actual programming experience seems to be related to their difficulty in dividing the total task into manageable pieces and in shutting out of their minds unneeded details. (The concept of subroutines in programming, which an experienced programmer will have used extensively, encourages this form of thinking.)

On the basis of this project and my teaching experience, it seems one could safely get by with somewhat less educational background than the RTE project members had and, perhaps, with less professional maturity. However, significantly more actual programming experience is clearly needed.

3.1.2 Actual design work is necessary. Designing can be mastered only through practice. This is true in most technical areas and is certainly true in software design where the techniques used are often informal. In effect, it is an art which cannot be fully explained and which must therefore be learned by doing it under the direction of someone who already knows the art. (One explanation of this is the fact that design involves making decision involving many different factors whose interrelationships cannot be explicitly stated. Thus, one cannot learn a simple rule that will enable one to design, but instead must learn by experience what factors to consider.)

The amount of design work in the RTE project was just about right. There was enough of it (and enough time) so that people could try different approaches, discovering as they did which worked best; yet, it was small enough that the task did not seem impossible. Also, the implementation (the real test) can be done in a reasonable amount of time so that the designers can see precisely how their ideas work out. It is worth noting that much of the implementation work will be done by the designers (now working as an implementation team), which can be a very valuable carry-on project if people implement parts other than those they designed.

3.2 Operational factors

Even if one starts with personnel with good background and organizes a training project that includes ample practical experience, many factors associated with carrying out the project can make it a success or ruin it.

3.2.1 Projects must be realistic. The goals of a training project should be realistic in extent--that is, not too small and not too large; a real need for the systems being designed should exist--phony projects, made up just for training, make it difficult to motivate people; the work conditions of the project should be as much like actual design conditions as possible. It is especially important that projects be realistic in terms of their actual demand (see 3.3.4 below) and that they be broken up into relatively small segments so that people can see more tangible form of progress every week or 10 days.

The RTE project is a good example of the right kind of project. The need for the product existed independently; it was a large enough task to provide ample opportunity for training, but not impossibly large; the technical content was challenging, but not too difficult. The work conditions were not ideal from the standpoint of designing (see 3.2.3 and 3.2.4), but were consistent with the future conditions the people involved must work under.

3.2.2 Experienced and strong project leader needed. The central theme of design training must be apprenticeship. Thus, the presence of an experienced designer is essential; but, further, this experienced designer must also be able to lead the trainees and guide them in useful directions, both organizationally and technically. It should be noted that not all good designers are also good project leaders.

In the RTE case, the lack of experience internal to the SZAMOK organization in this respect was solved by bringing in outside assistance (the author). This is often an excellent solution, but must be approached carefully to insure that the "expert" has the right design and leadership qualities in fact, as well as on paper. One problem in this project was that a project leadership structure existed before I arrived; initially I served primarily as an advisor only, leaving the actual direction to staff members. This did not work out well. When we later changed so that I took a more active role in project management, the difference in design progress could easily be seen.

3.2.3 Full-time involvement necessary. Design tasks usually require a large amount of effort. A "critical-mass" effect operates so that if the task is worked at only 1 or 2 hours per day, the chances are great that the work will never be finished. It is essential that the trainees (and leader, too) have no duties or responsibilities and be able to concentrate full-time (6 to 8 hours per day) on the design task.

This proved to be a major problem in the RTE project. Three of the five trainees were instructors relieved of most of their normal teaching duties on a job-rotation plan designed to provide them experience in the technical areas in which they teach. Yet, a number of small, but time-consuming activities remained for them (consultations with former students, planning of future teaching duties, etc.). Further, normal administrative interruptions took a large amount of time. Finally, the simple business of living (housing, transportation, income) took a high percentage of the trainees's time. The net result was a good bit less than full-time involvement; this seriously comprised the success of the project.

3.2.4 Proper support must be given. People working at a complex activity require two kinds of support in order that they may concentrate on and carry out the task. First, they must feel that their work is important, that their employer values it, and that the result will be valuable to themselves as well. Second, the physical arrangements necessary must be present: proper office conditions, freedom from distractions, documentation of equipment, reproduction and typing facilities, and machine time if required.

The first type of support was not always present in the RTE project and this presented problems of motivation. It must be recognized that organizational support is perceived by people not only through official statements but even more through organizational actions. The second type of support was a special problem in this situation and seriously effected progress. For example, normal turnaround time on Xeroxing a piece of paper was at least 7 days; since the physical objects produced by a design consists of pieces of paper, and since it is often necessary to distribute copies of working documents to all project members, it can be seen why this lack of support was so critical.

3.3 Technical content of the training project

When one considers the various rules of good design that should be taught to potential designers and the principles underlying the system being designed (for instance, operating system principles), we find a multitude of factors. In what follows, I concentrate on several points important to a wide range of projects.

3.3.1 Emphasize the difference between programming and design. Programmers typically approach a design task as though it were the same as a programming task. Likewise, people trained in systems analysis may approach a software design strictly from the viewpoint of overall organization. Software design requires a special blend of programming and system design knowledge. It is essential that trainees be constantly reminded of this and shown by example how to create a software design.

The need for this was evident in the RTE project. Some trainees had only experience with assembly-language programming and found it quite difficult to think about the functional requirements of a module and its interconnections to other modules without thinking about the details of implementation. When the staff members approached the design task from a programmer's standpoint (worrying about the smallest details of data representation and control flow from the very start) they had great difficulty in solving the larger problems of overall system design. As they gradually became accustomed to thinking about parts without considering their implementation in details (via stating functional specifications and gross implementation decisions) they found they were better able to see how the various parts of the system should fit together. This, of course, was essential to stating a correct design.

3.3.2 Use good conceptual models. A design of any size is normally complex enough that it is impossible to keep all of its details straight without some aid. Some type of abstractions (conceptual models) are needed so that one can consider important details while ignoring irrelevant ones. Typically, different models for different parts of the design may be needed.

For example, in the RTE system we used the process concept to aid in determining the interconnections between parallel activities and a logical model of lines of text in designing the file system. Without the process model, it was impossible to correctly determine the proper connections between pieces. Once we represented the system in this way, we could see more clearly what was needed. Likewise, the file system design was made easier by divorcing the local requirements for operations on lines of text from the particular physical characteristics of the R-10 disk.

3.3.3 Design representation is important. The way in which the final design is stated is quite important, of course. (That is, it must be well documented.) But, it is also quite important to make designers aware of the need for good working representation of the design. It is quite impossible to think out all the details of a design and only write down the results. Further, much of the activity in a design consists of evaluating proposed solutions and changing decisions already made. The use of good representations for the design and its attendant information as it progresses is thus quite important.

In the RTE project we found it useful to use two types of representation, augmented by the usual tabular presentations of data structures. As we made initial decisions, we wrote them down in a carefully numbered scheme. We tried to think through one topic at a time, collecting all the decisions concerning that topic together. At that stage, the decisions were represented simply as short, natural-language statements. After refining these decisions to the point where we felt we understood the particular point, we would start to represent actual programs in the form of a metacode (described above in 2.1). This provided us a very explicit representation of our design for the various parts without getting bogged down in many of the extraneous details involved in actual programs. Further, we often used metacode to represent our concepts of the overall system structure and interconnections. For example, this permitted us to represent in a single notation the interaction between hardware interrupts and software signals. As noted in our discussion of conceptual models, the difference in our ability to deal with the developing design was markedly improved when we used these representations.

3.3.4 Build on existing knowledge of trainees. Learning to design is not an easy task. If the trainees must also at the same time learn new concepts of hardware of software, then their task is made much more difficult. Further, their attention is diluted so that neither their grasp of the new concepts or the training in design is as effective.

In the RTE project this effect could be seen quite easily. Those staff members with some knowledge of operating systems could more readily deal with the problems of representing their design in a proper fashion. Likewise, those with command of a high-level language were better able to deal with the metacode representation.

3.3.5 Explicitly deal with decision making. One of the ways in which design differs from programming is in the existence of many difficult decisions to be made. It is an important part of the training to sensitize people to the need for identifying critical design decisions, gathering information relevant to them, and making the decisions only at the appropriate time. If this is not treated explicitly, then one of the important differentiations between design and programming may be lost.

Our approach to this in the RTE experience was to point out the existence of critical decisions as they arose. (This is one of the places the need for an experienced designer as leader is most evident.) Once we had clearly stated the decision to be made, we tried to think of as many different possible solutions as we could. We would list these and for each write down a brief evaluation of it as a possible solution. As suggested in(4), this permitted us to go back and review our decisions in depth at a later time.

3.3.6 Don't forget the organizational aspects of design. Software design very often requires the cooperation of several people working together in a group. This type of activity is usually new to people who have just been programming. It is important to help people deal with this mode of activity during the training project. The need for readable working documentation, the necessity of disciplined working meetings, management techniques for assigning work loads, the importance of interpersonal working relationships, and so on should all be dealt with carefully.

The RTE project members had worked together prior to this project so that some of these concerns had already been dealt with. However, in the area of assigning workloads an important lesson was learned. Initially, broad areas of responsibility were assigned and specific tasks within those areas were not explicitly assigned. This did not work well since the trainees were not accustomed to producing working documentation, external specifications and the other intermediate products of a design. When I began providing more detailed assistance on how to organize the task, progress improved. Later, the project members were able to subdivide the work more appropriately themselves.

4. SPECIFIC SUGGESTIONS FOR DEVELOPING COUNTRIES

There are a number of factors to consider in training software designers besides those mentioned above. These are chosen because of their importance and because they illustrate problems often encountered in developing countries. Thus, each point should be evaluated with this in mind.

Beyond these specific points, there are four general points that should be carefully considered by those dealing with computer technology in developing countries.

4.1 Keep software design in proper perspective.

There is sometime the tendency on the part of those interested in software to build an extensive capability to design compilers, operating systems, file systems, and numerous other types of software. This is often unneeded since many such systems are readily available.

On the other hand, the tendency of those whose personal interests are more along the lines of organizational structure is sometimes to assume that software design is the same as programming. As a result, little attention is paid to the problem of building some design capability.

In short, the needs of each country or organization must be carefully evaluated. Enough design expertise should be developed to permit a reasonable amount of independence of others, but not more than is required by the jobs to be done.

4.2 Don't try to skip essential steps.

It is often tempting to hurry past or even skip over important steps. For example, it might be felt that a well-educated person with an understanding of, but no experience in, programming should not worry about the lack of experience and be trained as a software designer. As we pointed out above, this would be a mistake.

Building a technological capability in a society is a slow process and skipping stages can often lead to trouble. This is not to say that the process always takes the same time. Hopefully we can learn from the mistakes of those who have already tried it and thus improve the development process. (That is the point of this paper, after all!)

Finally, it is worth remembering that the technologically advanced countries with much experience in utilizing new technologies, are only now starting to have much success in the area of training software designers. Many of the failures known so well in our field were due in part to people skipping logically necessary steps in training.

4.3 Carefully choose techniques.

As with most human affairs, there are competing concepts dealing with software design and the training of those who practice it. Alternatives should be carefully evaluated with respect to the needs, people, and resources of each situation.

4.4 Get good advice.

Just as techniques, advice must also be carefully chosen. All advisors are not equal, either in capability or purpose. Likewise, no piece of advice (including this one!) is universally applicable!

5. SUMMARY

After setting the context of this brief discourse, I described a particular software design training project. I then drew a number of lessons pertinent to training software designers and illustrated them in the setting of the RTE project. Finally, several general suggestions were made for training in developing countries.

ACKNOWLEDGEMENTS

This paper could not have been written without the members of the RTE project group: Zoltan Szekely, Andras Kelen, Sandor Majoros, Sandor Zold, and Mary Koos-Hutas. The advice of A.G. Dale and Per Ofstad was most helpful.

REFERENCES

1. Ashenhurst, R.L. (ed). "Curriculum Recommendations for Graduate Professional Programs in Information Systems", Communications of the ACM, May, 1972.

2. Brown, R.R. "1974 Lake Arrowhead Workshop on Structured Programming", Computer, IEEE Press, New York, October, 1974.

3. Freeman, Peter Software Systems Principles: A Survey Science Research Associates, Palo Alto, 1975.

4. Freeman, Peter "Towards Improved Review of Software Designs", Proceedings 1975 National Computer Conference, AFIPS Press, Montval, New Jersey.

5. Hice, G.F., W.S. Turner, and L.F. Cashwell, System Development Methodology, North Holland Publishing Company, Amsterdam, 1974.

6. Intergorvernmental Bureau for Informatics. "An International Curriculum for Information Systems Designers", Rome, no date

7. Ledgard, Henry F. Programming Proverbs, Hayden Press, Rockelle Park, New Jersey, 1975.

8. Simon, H.A. The Sciences of the Artificial. MIT Press, Cambridge, 1969.