DOCID: 4009810

~~TOP SECRET~~

NATIONAL SECURITY AGENCY
FORT GEORGE G. MEADE, MARYLAND

CRYPTOLOG

APRIL 1978

P.L. 86-36

EO 1.4.(c)
P.L. 86-36

~~THIS DOCUMENT CONTAINS CODEWORD MATERIAL~~

~~Classified by DIRNSA/CHCSS (NSA/CSSM 123-2)~~
~~Exempt from GDS, EO 11652, Category 2~~
~~Declassify Upon Notification by the Originator~~

~~TOP SECRET~~

DOCID: 4009810

~~TOP SECRET~~

CRYPTOLOG

Published Monthly by P1, Techniques and Standards,

for the Personnel of Operations

VOL. V, NO. 4 APRIL 1978

PUBLISHER **WILLIAM LUTWINIAK**

P.L. 86-36

For individual subscriptions
send
name and organizational designator
to: CRYPTOLOG, P1

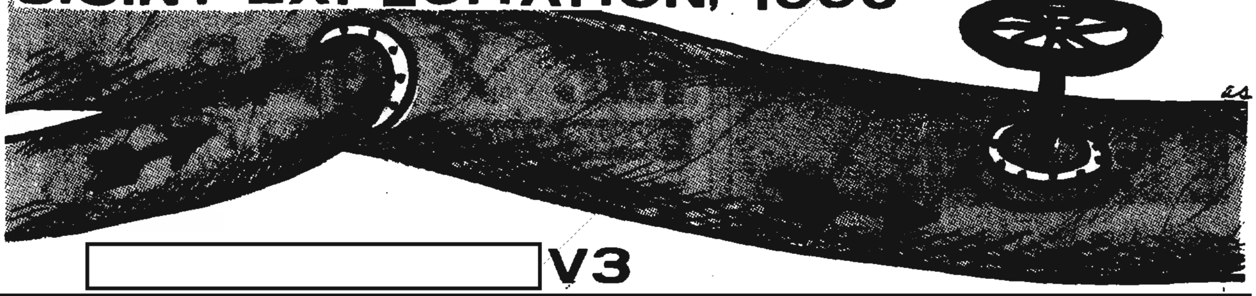
~~TOP SECRET~~

DOCID: 4009810

CONFIDENTIAL

P.L. 86-36

SIGINT EXPLOITATION, 1990



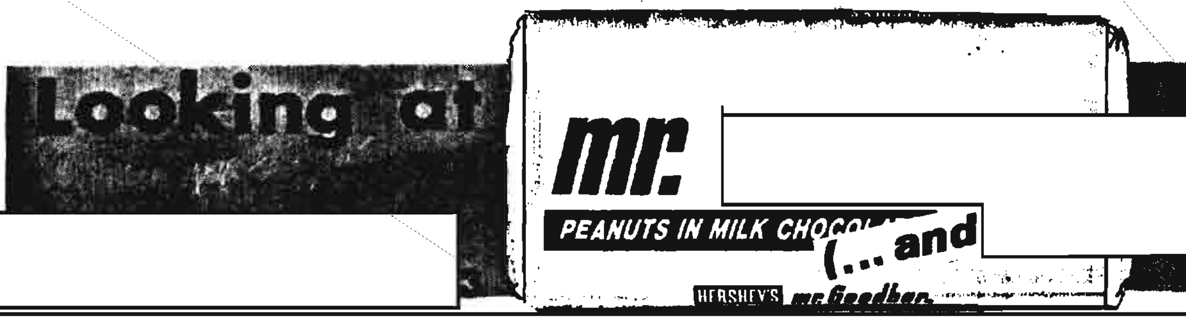
[redacted] V3

[redacted]

EO 1.4.(c)
P.L. 86-36

CONFIDENTIAL

EO 1.4.(c)
P.L. 86-36



[redacted]

[redacted]

P.L. 86-36
EO 1.4.(c)

EO 1.4.~~(c)~~
P.L. 86-36

[redacted]

P.L. 86-36
EO 1.4.(c)

~~TOP SECRET UMBRA~~

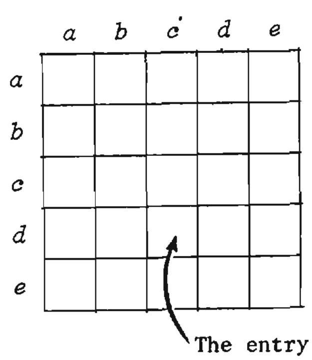
A LITTLE PEP TALK

R51

Over the past couple of years R51/CADRE has been experimenting with computer software for cluster analysis. Six programs, implementing nine clustering algorithms, have been acquired from outside the Agency and modified for use on CDC 6600/7600 computers.

These clustering methods are data-dependent, i.e, the model most suitable for a specific data set will depend on the form and nature of the data. The intent of the analyst must also be a consideration. (For example, does the investigator wish to find natural groupings in the data or to partition the data base into a predetermined number of categories?) It may be necessary to try several clustering methods on the same data to determine the most appropriate algorithm.

PEP-1 (*P*robability *E*valuated *P*artitions, version *1*) is one of the six clustering programs that R51 has adapted for Agency use. PEP-1 converts a set of input vectors into a matrix of pairwise similarities (or dissimilarities) (see Fig. 1). The choice of similarity (or dissimilarity) measure is an option of the program. The user may select any of the following measures: cross I.C. values, dot products, normalized dot products, Euclidean distances, and city block distances. The choice of measure must be guided by the nature of the real-world problem and the intent of the investigator. For example, does the analyst wish to match peaks and valleys in the vector profiles or, instead, match numerical values in corresponding components?



The entry in this cell measures the strength of the relationship between objects *d* and *c*. For similarity data, the higher the value, the stronger the association. For dissimilarity data, the lower the value, the stronger the association.

Fig. 1 Similarity Matrix on Five Objects

It should be mentioned that the analyst may construct his own similarity (or dissimilarity) matrix and then input the result into PEP-1.

PEP-1 orders the matrix entries from highest to lowest and replaces the numerical value in each cell with its rank order. (Tied entries receive the same rank order.) The PEP-1 algorithm can best be understood by regarding the entities to be clustered as vertices of a complete graph (i.e., an edge joins every pair of vertices) (see Fig. 2). The edge connecting any two vertices is identified with the entry in the similarity matrix which measures the strength of the association between those entities.

P.L. 86-36

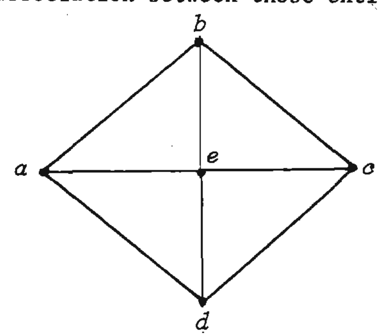


Fig. 2 A Complete Graph on Five Vertices

PEP-1 sequentially deletes edges from the complete graph according to the rank order of the entries in the similiarity matrix (the edge corresponding to the weakest association is removed at each step of the algorithm/ multiple edges are ridded in the case of ties)

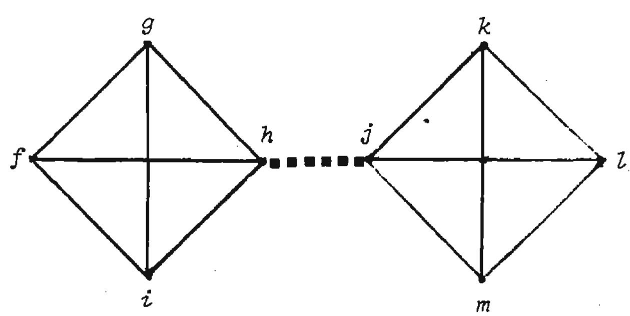


Fig. 3 Removal of the edge joining *h* and *j* produces a disconnected graph consisting of two clusters

until a disconnected graph is produced. At this point PEP-1 publishes the two partitions, the similarity (dissimilarity) value where the split occurs, the rank order of this value, and a "probability" that the disconnection could happen at random. The PEP-1 probability is merely an estimate (the true combinatorial probability is very complicated to compute) and is to be used only as an analytic aid to give a general impression of cluster validity. PEP-1 then reapplies the algorithm to each partition until the number of entities in a partition or subpartition is less than six.

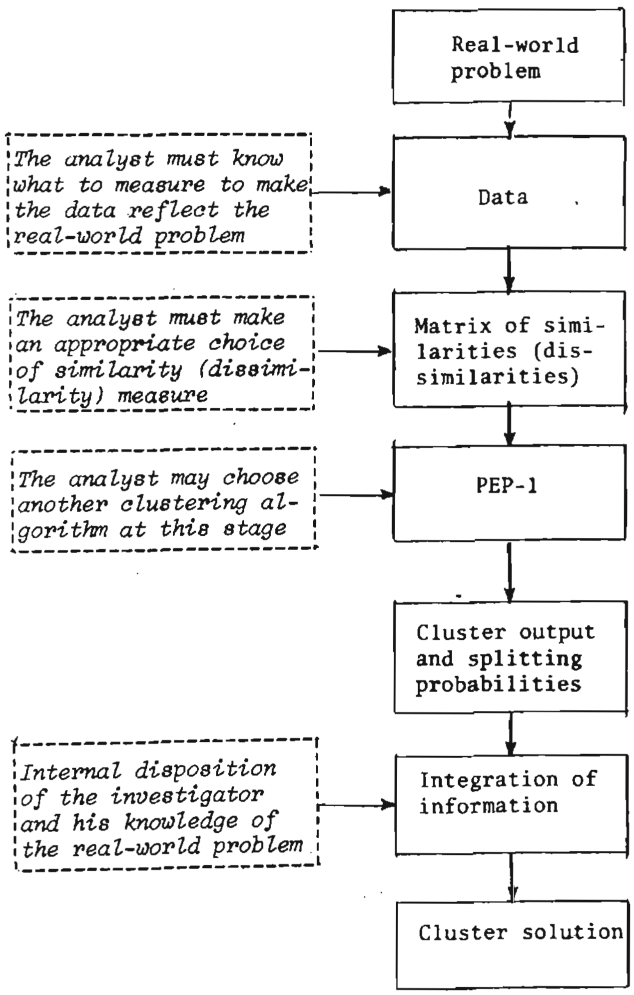


Fig. 4 The Clustering Process, Showing the Interaction (in Italics) of the Analyst and his Algorithm

It should be emphasized that the analyst is unable to prescribe in advance the number of clusters to be found by PEP-1 or the threshold levels at which the splittings occur.

PEP-1 is a reasonable choice for a clustering algorithm if the following conditions are satisfied:

- the analyst wishes to cluster a similarity (dissimilarity) matrix of his own construction or feels that he can select an appropriate measure to derive similarity (dissimilarity) data from vector input;
- the analyst seeks to determine natural clusters in his similarity (dissimilarity) data matrix;
- the analyst requires some numerical indication of the quality of the clustering produced by an algorithm;
- the analyst wishes only the rank order information in his similarity (dissimilarity) matrix and not the numerical values themselves to determine cluster structure;
- the analyst is aware that PEP clusters are not necessarily compact and may, in fact, be serpentine or sausage-shaped (see Fig. 5);
- the analyst wants an algorithm that will *not* be harmed by the presence of outliers (deviant observations) in the data;
- the analyst wishes to cluster fewer than 100 entries (fewer than 128 on the IBM 370/65).

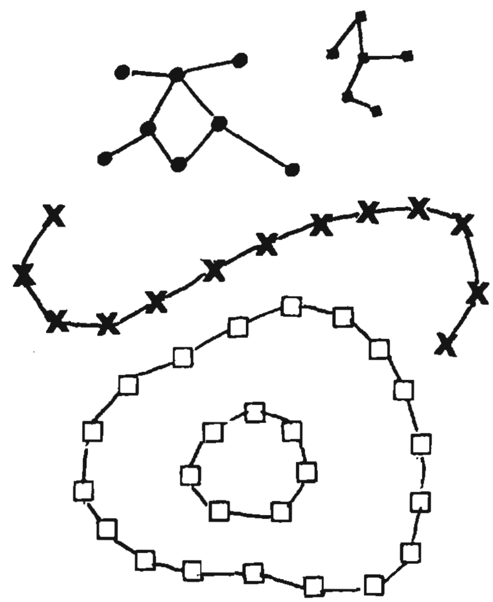


Fig. 5 PEP-1 clusters may assume a variety of shapes

If you have any questions about cluster analysis or about the possibility of applying clustering techniques to your operational problem, please contact either [redacted] (Chief, R51, 8518s) or [redacted] (R51, 8626s). (U)

[redacted]

P.L. 86-36

We gotta... ACCENTUATE THE NEGATIVE

P.L. 86-36

[redacted]

For some time now, the subelement manager has been engaged in a battle to defend his resources against reductions initiated for a number of reasons. Managers at all levels have been barraged by a series of statistics designed to aid in making resources-reduction decisions, to show the efficiency of collection resources, to determine the cost of product reporting, and to allocate the resource expenditures for each function within the subelement of interest. The usefulness of statistics as a tool for making these types of decisions has been widely accepted. The accuracy of the statistics is rarely questioned and many decisions are based solely on this data.

The current period of dwindling resources and rising inflation, with the concurrent review of intelligence expenditures, has created an increased dependence on statistical methods as a means of determining efficiencies of operation. Aided by a myriad of computers providing a capability to count and measure that was never before possible, statistics have emerged as the guardian of our expenditures, the protector of our resources, and the practically omnipotent champion of efficiency. Savings have been made and some very necessary trimming of fat has resulted. These are significant accomplishments and speak well of the systems we have created.

Statistics, by definition, is a branch of mathematics dealing with the collection, analysis, interpretation, and presentation of masses of numerical data. The compilation of statistics requires a collection of quantifiable data. Only those things which can be measured or otherwise quantified can be considered when constructing statistics. In measuring SIGINT production, the statistician must therefore look at such things as the number of COPES objectives satisfied, the number of minutes of copy by case, the number of products produced, the number of intelligence requirements satisfied, or some other similar quantifiable data.

This has caused great difficulties for some subelement managers when defending their resources and has caused us to overlook one of our greatest intelligence contributions, which I call *negative intelligence*. Negative intelligence is the intelligence gained from knowing that a target unit is performing normally or is perhaps inactive. [redacted]

Statistics, as most statisticians will admit, can be used to show whatever the statistician desires. Remember the old "Figures don't lie, but liars figure" adage? To guard against this, in our very important resources area, it is necessary to temper statistically suggested resource reallocations with considerations of the negative intelligence not reflected in the statisticians' figures or to devise a method of measuring negative intelligence contributions. Until we can do that, we must rely on the analysts to provide qualitative judgments through the subelement manager. The importance of these qualitative judgments cannot be overemphasized and they must be considered along with the statistical inputs. Granted this makes assessments subjective to a point and renders decision-making more difficult, since decisions cannot be made solely on a quantitative basis, these assessments nonetheless can provide the information necessary to protect that important negative intelligence data which may be irretrievably lost in a primarily quantitative evaluation. Remember that what we *don't* get is also important. In many cases we must accentuate the negative.

EO 1.4.(c)
P.L. 86-36

REMEDIAL SOFTWARE ENGINEERING

[redacted] T3214

Reprinted from *Proceedings: CISI Spring Conference, 23-26 May 1977* (Computer and Information Sciences Institute, National Security Agency: May 1977), S-217-661, 217 pp., SC. [redacted] paper was awarded First Place among the Conference Awards presented by Mr. Kermith H. Speierman, Assistant Deputy Director, Telecommunications and Computer Services, in the Friedman Auditorium on 23 June 1977.

This paper will describe Remedial Software Engineering, a systematic approach to identifying design deficiencies in software systems and implementing improved systems economically. I will combine ideas previously published by Yourdon, Metzger, and other authors to produce a methodology for dealing effectively with "problem" systems. I will give no detailed explanation of software design methodologies but will assume that the reader either is familiar with them or will consult the references for clarification where necessary.

Peter Prescription #36 asserts that rational action must be based on answering three questions:

1. Where am I?
2. Where do I want to be?
3. How do I know I am getting there?[1]

This paper will explain how to use proven tools and techniques of software design and project management to answer these three questions. It is based on my experience with some small- to medium-sized NSA systems where these principles have been applied successfully. I leave it to the reader to decide whether or not Remedial Software Engineering might be applied beneficially to very large projects.

I chose the term "Remedial Software Engineering' (RSE) for three reasons. First, to imply that its application would be to a system for which some pieces have already been created, but which needs further work to be made useful. Secondly, to distinguish it from any particular software design or project management methodology. And, thirdly, to imply some measure of orderliness in the craft of rescuing systems from disaster. Equally suitable titles for the discipline described below might have been "Creative Software Salvage" or "What to Do When Disaster Overtakes Your Software Development Project."

<u>*How Did I Get Here?*</u>

The premise of this paper is that the answer to Peter's first question, "Where am I?", is, "In charge of a software development project which is in deep trouble." A quick review of some basic causes of project failure will lay the foundation for the RSE methodology.

One major source of trouble is the lack of any system design. If nobody understands what is being built, then it follows that nobody will be very successful in trying to build it. Another source of trouble is a design that will not satisfy the real requirements. In this case, the implementers know precisely what they are building. What they don't know is that their product will never perform useful functions for the customer without major remedial action.

A second cause of disaster has to do with the project plan rather than the system design. The system design is a detailed blueprint of all the elements of the system being built and their interrelationships. The project plan, on the other hand, is a step-by-step description of how the pieces of the system will be assembled, unit-tested, integrated, and tested as a system. Where the system design specifies precisely *what* will be built, the project plan details *how* it will be built. If a project plan is missing or if it omits key considerations, serious problems will result and will frequently manifest themselves as apparent hardware or software design problems. If a house-building plan calls for installing the roof before laying the foundation, it is almost inevitable that the house design itself will be attacked by the carpenters as being unworkable. The plan and the design will then be modified on an *ad hoc* basis to make possible the immediate tasks at hand. Usually there will be no distinction made between the plan and the design.

A third class of problems results from failure to follow the plan. For example, the system in trouble might have the wrong amount or the wrong kind of staffing even though the plan spells out exactly what staffing is required. Assigning 25 troglodytes to build a condominium is unlikely to produce a livable structure even though the cavemen are handed a set of detailed blueprints and an hour-by-hour description of the thousands of subtasks required. Assigning 4000 architects to the project, again with perfect design and planning documentation, might yield similar results.

Although Remedial Software Engineering is called for when development efforts run into trouble, there are other times when the application of the same principles might be beneficial. For example, if a functioning system is about to be upgraded to incorporate new hardware or software such as microprocessors, data base management systems, etc., the methodology detailed below might be an excellent preparation for the conversion. Similarly, a system may be producing acceptable results but only at an excessive software-maintenance cost; RSE could help reduce the costs. In short, use Remedial Software Engineering whenever the pain associated with your current software efforts seems to be unbearable.

In these introductory paragraphs I have

[1] Laurence J. Peter, *The Peter Prescription* (New York: Bantam Books, 1972), p. 159.

defined Remedial Software Engineering, stated its purpose, and suggested when its use would be appropriate. In summary, I quote the author of *Patterns of Problem Solving*, who said, "The most common difficulty in problem solving is not lack of information, but rather the failure to use information that the problem solver has."[2] The remainder of this paper details three phases of making systematic use of the information on hand to answer the three questions of Peter Prescription #36 and convert incipient software disaster into a productive system.

Where Am I?

First, gather the current requirements for the system you are seeking to build or modify, and design an "ideal" system against those requirements. Ignore for the moment the pieces of software you already have, even if they are "unit tested" and function perfectly. Use any structured design methodology of your choice and apply it rigorously. Various groups, some at NSA, have applied HIPO, Transform Analysis, Transaction Center Analysis, and Jackson Design methodology[3] with success. It appears as though the particular tool used is not as important as the fact that some technique is applied uniformly and rigorously across the whole system.

Fig. 1 is an abbreviated structure chart of a segment of a hypothetical system which must gather some data and display it on a terminal, among other things. Note that all functions are broken down into the smallest possible component parts and that all data flow and control are explicit. You can trace the flow of solid arrows to see where the data goes and how it gets changed along the way. Each module has a single function. The modules may be combined together in-line to form a single program, they might be a mix of separate programs and subroutines, or they might even be a mix of hardware, firmware, and software. The point is, the structure chart ignores all those considerations and concentrates on the elements of the problem solution only. Once they are understood, then intelligent decisions can be made separately about how to package the modules.

You may have to resist a great deal of pressure to skip this step. There are still some people lurking about NSA who will try to persuade you that time spent drawing pictures is wasted. (I suspect such characters pervade many institutions in the private sector, too, although I only deduce this from the work delivered by some contractors to NSA.) These people advise that we start attacking the problems directly, even though some symptoms may be half-recognized. Indeed it seems as though each of us tends to have a deep-seated urge to

[2]Moshe F. Rubinstein, *Patterns of Problem Solving* (Englewood Cliffs: Prentice-Hall Inc., 1975), p. 8.

[3]See: Edward Yourdon and Larry L. Constantine, *Structured Design* (New York: Yourdon Inc., 1975) for a discussion of the first three methodologies. See also: *The Jackson Design Methodology*, Infotech International Handbook (Pasadena, 1977).

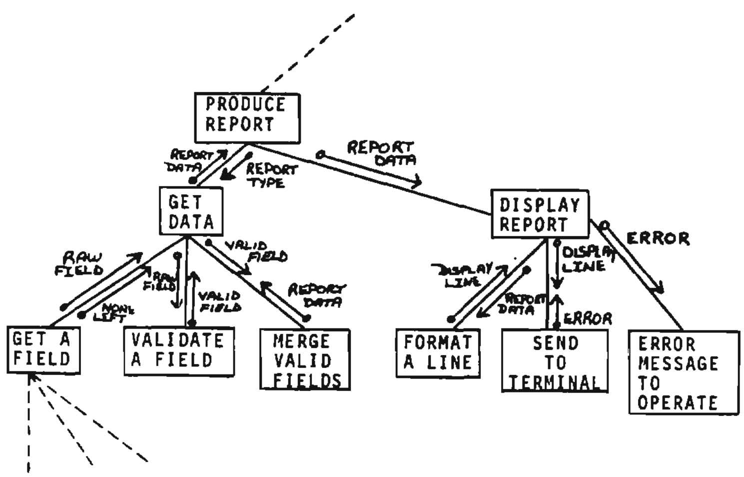


Fig. 1*

* This paper will use several of the Yourdon definitions and conventions. Among them are:

Solid-tailed arrows represent data.
Hollow-tailed arrows represent flags.
A box represents a module.
Arrows into a box interior indicate data flow into the middle of a module as opposed to through the module's entry point.

A module is any lexically contiguous set of computer instructions.
Atomic modules are the lowest-level modules.
A straight line represents one module invoking another module. The higher invokes the lower unless otherwise noted.

start modifying code too soon.

Just remember that design is cheaper than fabrication. It is an abstraction which can be manipulated easily without intervening distractions such as the hardware peculiarities, operating system vagaries, or programming-language oddities. If done properly, the structure of the design will very closely depict the structure of the actual problem to be solved. You will probably find that someone who has not had a hand in creating the mess you are correcting will be extremely helpful during this design phase. Being uncorrupted by the faults of the existing software design, he can give a more objective opinion of the "fit" between your "ideal" system design and the problem to be solved. The trick is to keep yourself and your team from corrupting him!

You must now document the existing system design using the same notation you used for the design of the ideal system. If you cannot document the design of the existing system, then you should decide to replace it in its entirety. In this case, you might be able to salvage some code from the old system modules as described below. This activity can be very dangerous, however, since you might preserve some of the major causes of your troubles. By saving only atomic modules and resisting the temptation to add code to them, you can protect yourself in most cases.

Now that you have two system designs, the old and ideal, lay them side by side and compare them. You should look for correspondence between the two designs. You might even have some modules from the old design that have already been coded and can be kept as is because they match the ideal design perfectly.

You are likely to have some modules that should be replaced because they are "corrupt" in design. The lack of correspondence will be seen as confused flow of control and inconsistent interfaces with various superordinate modules. The modules will also mix several functions without justifiable purpose.

Fig. 2 shows a possible representation of the existing design documented using the same conventions as the ideal design in Fig. 1. Notice that the data display portion of the system consists of a single module with the functions hopelessly entwined. This characteristic probably was due to a bad system design before the software implementation began, or perhaps the design was not understood or followed. When confronted with hopelessly corrupt modules such as the data gathering/displaying one in Fig. 2, you should salvage whatever atomic modules you can and then throw the remainder of the code away.

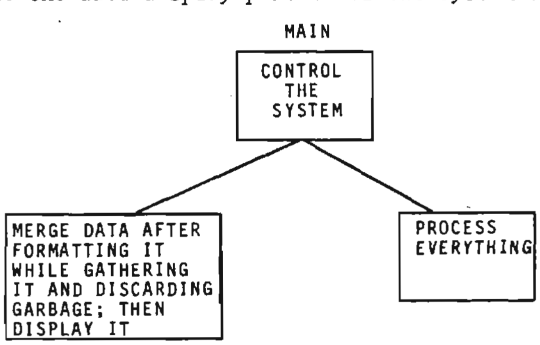


Fig. 2

Fig. 3 depicts how you might be able to extract the coding to construct your ideal "Send to Terminal" and "Format a Line" modules from the existing mess. Very likely, all the other functions are so dispersed and mixed up that you will be able to salvage nothing else.

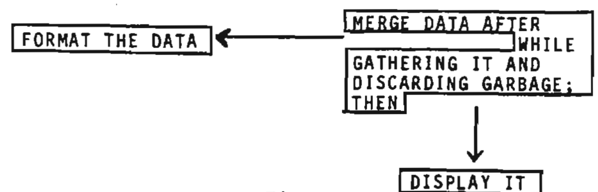


Fig. 3

You might also find some modules in the existing system that are moderately corrupt but have been performing satisfactorily for some time. These represent a real problem, particularly if you find yourself in the same tight resources situation as the rest of the world. You will be tempted to keep them and thus corrupt your ideal system design. Indeed, you might have no choice but to keep at least some of them because of the lack of time, money, or people.

Where Do I Want to Be?

You have already documented where you would like to be in a utopian world when you designed an "ideal" system. But now you must reconcile your ambitions with the reality of limited resources. You must answer Mr. Peter's second question by defining revisions to the software system that make the best possible use of the software already implemented as well as new software to produce a reliable and useful system.

If you keep any poorly-designed modules, you must recognize that you are doing so and realize the possible consequences. The cost of maintaining those modules will be excessive for the life of the system. They might contaminate the rest of your system if you do not take positive steps to prevent it. They might not even be functioning as well as you think; perhaps they just have not yet had sufficient opportunity to fail. They might come back to haunt you at a critical point in the project.

A more subtle danger in using admittedly corrupt modules is that members of your team might perceive a lower set of standards for the software project than you intend. Actions speak louder than words. If you retain substandard modules from the old system, make sure that every member of your team understands that you know they are substandard, that you have weighed the risks, and that you kept the modules for very specific reasons. Emphasize that the modules are substandard and that you will accept no new modules of similar quality.

You should insulate all substandard modules from the rest of the system with interim interfaces that conform to all the standards of good system design. Figs. 4 and 5 show how this can be done. Module 3 is typical of modules that can be found in many systems. It has multiple entry points; its logic depends on internal flags set by external modules; it gets its data inserted into it by external modules; and it returns its results by placing data directly into the middle of the invoking modules. Let us suppose that module 3 has been stable for a considerable length of time under extremely severe testing conditions and further suppose that to replace module 3 would take 12 man-months. Let us also suppose that you want to replace module 2, a one man-month job, and retain modules 1 and 3.

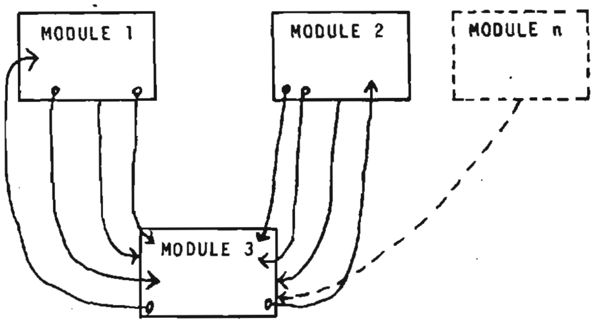


Fig. 4

Fig. 5 depicts the proper way to insulate modules 1 and 3 from the rest of your system. The interim interface module has single entry and exit points, and conducts its external communications (data and control) explicitly. Note that the new version of module 2 uses the interface and follows all the rules of proper intermodule communication, but that modules 1

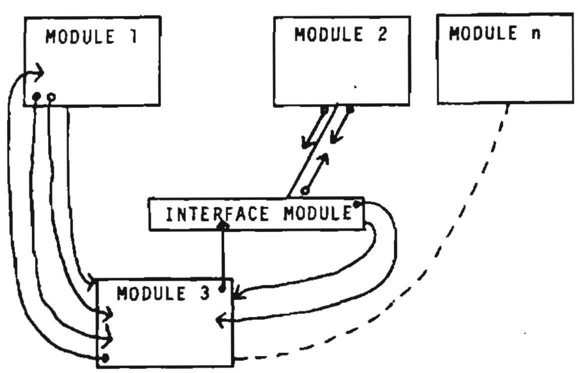


Fig. 5

and 3 are unchanged. You are now in a position to redesign at your leisure module 1 and any other modules which invoke module 1. You can also deploy new modules which invoke module 3 through the interface without compromising their design because of deficiencies in module 3.

After all the invoking modules have been converted, then module 3 itself can be reconstructed. Fig. 6 depicts this last step. The interface module has become permanent with the assumption of controlling functions. An array of subordinate modules has been created with each submodule performing exactly one of the functions from the original module 3.

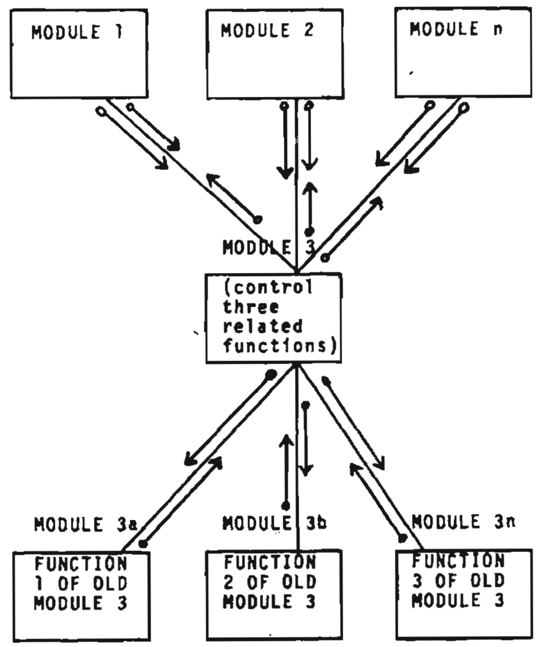


Fig. 6

Another activity of designing is to incorporate the best commercially available products wherever possible, rather than reinventing the wheel. If a portion of your system design calls for modules to manage data, consider using a commercially available data base management system. There are now several on the market which have given many customers satisfactory results[4]. By using one of them, you can concentrate your own personnel on solving the customer problems rather than the computer problems. Consider also some of the newer hardware technology that is available. For example, some of your modules might be best implemented on microprocessors. When you complete step 2 of your remedial software engineering, you will have a complete design plan for your system including how each module will be implemented.

---

[4]See: Herbert L. Gepner, "User Ratings of Software Packages, *Datamation* (December 1976), p. 108.

How Do I Know I Am Getting There?

You have compared the existing design to an "ideal" system based on the current requirements and devised a design for the system you will implement. Now you must plan the implementation effort and continually compare the actual progress to the plan to answer the last of Peter's questions.

Managing a Programming Project[5] is an excellent reference for use in the project-planning phase and I will not repeat in this paper everything Metzger says in his book. Several points deserve special emphasis, however, and Metzger touches on them only lightly.

It is essential to match your resources to the task or vice versa. Remember the cave dwellers building the house? Assign the right number of people with the appropriate skills to the remedial software effort. Consider augmenting your work force by contracting some pieces of the system to the private sector if you have a surplus of money and a shortage of people, or if some particular skills are unavailable to you directly. I know of one project, not at NSA, where the configuration management function was contracted successfully.

If you cannot staff your effort properly, then scale down your ambitions and undertake to deploy a more limited system. If it is impossible to undertake less and still produce a useful and reliable system, then you should cancel the project. To do otherwise would be a waste of taxpayer dollars. One project manager at NSA recently mused as he stood in the shambles of an overly ambitious and understaffed project, "If you are not big enough to take the bull by the horns, then the best you can possibly hope for is to stay a few steps ahead of the beast as you run in circles around the ring." He was right, only if you accept his assumptions that his size and boldness were the only variables. But the bull could have been either trimmed down in size or eliminated. Shrinking the bull would increase the project manager's relative size and executing the bull would allow the manager to tidy up in a leisurely manner before regrouping for the next fray.

Having achieved the right match-up between the project size and staffing, make sure you provide your workers with all the tools they need. Dick Brandon, president of ACT-Brandon, recently said, "I am amazed today that we are working with 1975 hardware, using 1971 software, and managing as though it is 1960 -- and that we are trying to automate an organization with a structure designed in 1944."[6] Mr. Brandon could have been studying NSA when he said that.

Some of the tools that some managers give too little emphasis to are training, testing mechanisms, and standards. It is unfair to everyone involved to ask a newly hired college graduate or recently transferred computer operator to design, code, and test a piece of software for a computer he has never seen, in a language he knows nothing about, to solve a problem he only vaguely understands. Instead, he should be trained in the language he is to use; he should be handed a copy of the standards his documentation, software, and test plans are expected to adhere to; and he should understand the system design and project plan documentation. He should be working from a detailed functional description of the module or modules he is to produce. The functional description will include detailed descriptions of all interfaces between his software and the rest of the system. The system should have built-in trace and audit trail features to facilitate debugging and testing. Given these tools, you can safely leave the detailed design, coding, debug, and unit testing to the individual's programming team.

One last point I would like to add to Metzger's discussion of planning the project is to consider carefully your system implementation strategy. Harlan Mills of IBM has made some thought-provoking points about the working relationship between computer people and their customers.[7] He suggests that rather than disappearing for several months of coding and testing as soon as you have gotten the customer to sign a detailed requirements document, you should demonstrate another real piece of the system at least once a month. This would greatly reduce the understanding gap that frequently widens between the customer and the system implementers on many software development projects. It will help insure that misunderstandings which occurred during the requirements definition phase are surfaced early, before the whole system is coded and tested. Mr. Mills' arguments tend to be reinforced by my experience with several NSA projects and they suggest some guidelines for your implementation efforts.

Getting There

You have done your homework and rolled up your sleeves to begin the real work. Since your system design has been thoroughly tested in the abstract and you have a detailed project plan and you have even scattered all the right tools about your workers' domain, not much can possibly go wrong. Right?

Wrong!

[5] Philip W. Metzger, *Managing a Programming Project* (Englewood Cliffs: Prentice-Hall Inc., 1973).

[6] "The Computer As Villain," *Datamation* (April 1976), p. 14.

[7] Robert L. Patrick, "Software Engineering and Life Cycle Planning, *Datamation* (December 1976), p. 79.

Remember, you are beginning your remedial software engineering effort at a low point on the project. Everyone has been working for 3 years and seems to be no closer to putting a system into production now than when they started. Morale was low before you walked in, but now the requests for transfer are outnumbered only by the volume of help-wanted ads being clipped from the newspaper. You will have to manage the remedial project better than any other project you have worked on.

Metzger covers most aspects of managing the remedial software effort in his chapter, "The Programming Phase."[8] If you heed his advice on organizing your effort, dealing with changes, monitoring and controlling the work, and carrying the water for your subordinates, you will be in pretty good shape. Since the premise of this paper is that the project is already in the middle of the programming phase (or even the test phase) a few additional points should be made.

First, you are likely to encounter great resistance to change. You will find that some optimistic members of the project team are caught up in the "light at the end of the tunnel" syndrome. They will insist that the successful completion of the system is in sight; just a few more corrections and a little extra effort will save the system. Of course, they have been saying this for months and have been reporting the system as 95% complete for about 50% of the elapsed project time. They are expending all of their energies on trying to make whatever code they have work. They never stop to seriously consider that their system design might be unworkable, either from basic flaws present from the outset or because the design has been seriously corrupted during the implementation.

The trick is to convince them that things are really as bad as you know they are. Sometimes a suspension of coding/debugging/testing and the assignment of a design analysis task culminating in a written report will do the trick. Some people will even see the light if you brief them on your own analysis of the existing design. Sometimes, though, you might have to simply pull rank on the individuals and direct them to abandon their design in favor of your revisions. The result could be subordinates who are convinced that you sabotaged their good efforts just as they were about to bear fruit. You will not have their commitment to the revised system and their support will be less than enthusiastic. It would probably be better to transfer to another project anyone who has such an enormous emotional investment in the unworkable design that he refuses to listen to reason.

You may also find that many of your programmers have become obsessed with creating "sophisticated" code to the detriment of the system. This is a common problem, stemming in part from historical pressures to use every possible trick to cram many functions into the fewest memory locations possible. It might also stem from individual recognition that the current system design will never work and the "part of a successful team" reward will never materialize. The individual then internalizes his rewards by challenging himself to be even more clever in his production of code that will run faster in less core and do more work. The trouble is that nobody, including the original creator, will be able to understand the code in 3 months when it has to be modified to allow for a requirements change.

The solution to this problem is twofold. Orient every member of your team to the project objectives and build commitment from all to the development of a system that works. Also establish and enforce programming standards for the project. By ensuring that each individual understands the long-term benefits of never using part of an executable instruction as a constant, never using self-relative arithmetic, never hard-coding variables, avoiding GOTO's, etc., it will become easier for you to enforce the standards. Be sure to reward those who produce quality code as well as redirecting those who spend forever making a seldom-executed subroutine run faster, violating standards in the process.

Remedial Software Engineering: Summary

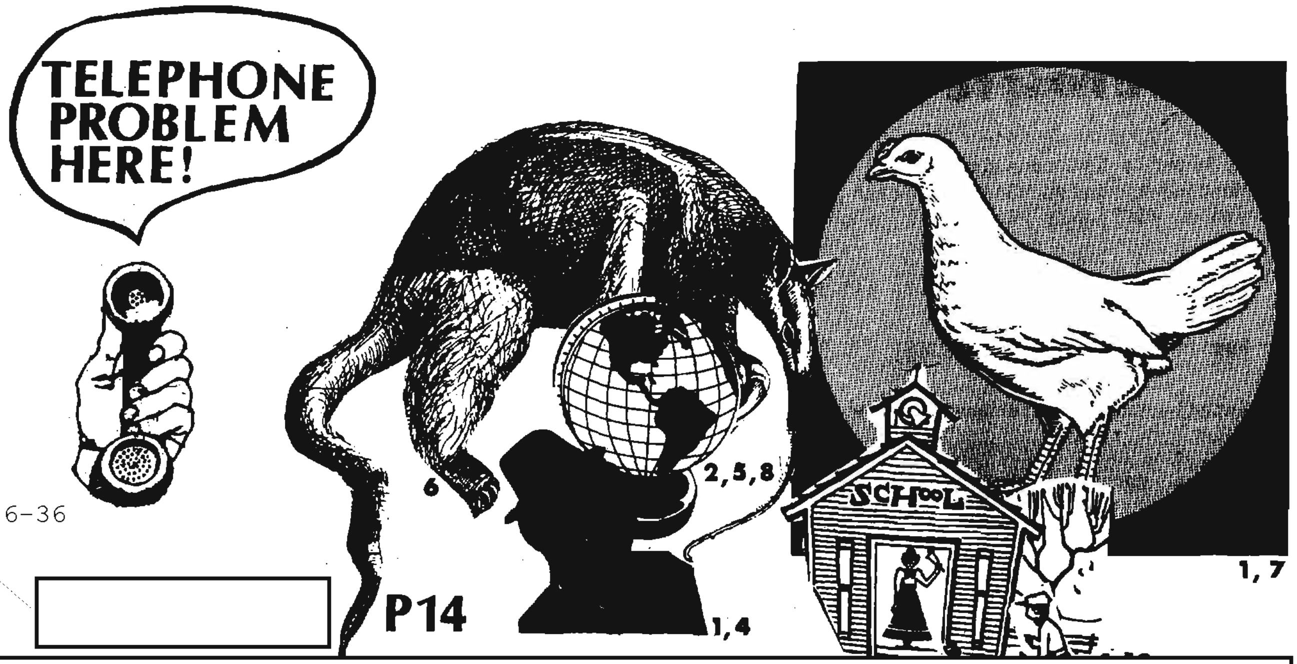
Remedial software Engineering is the systematic application of the best available software design and project management techniques to bring about desirable changes in software systems. RSE uses readily available information to analyze the current state of the system and implement an improved system.

First, the remedial software engineer designs an "ideal" system to satisfy the requirements of the customers and documents the design of the existing system with the same design notation. Next he compares the two designs and evaluates discrepancies between them. Then he resolves the discrepancies by designing a revised system which combines elements of the existing system, new elements, and some temporary elements. Lastly, he manages the remedial project paying particular attention to the "people problems."

By using a Remedial Software Engineering approach to problem systems, the project manager will make rational choices among explicit alternatives, lay the groundwork for productive team efforts, and bring the expectations of management, programmers, and customers into harmony with reality. He will have the management tools required to produce a quality product in a reasonable time.

[redacted]

[8] Metzger, pp. 69-112.



P.L. 86-36

[redacted]

[redacted]

EO 1.4.(c)
P.L. 86-36

NSA-crostic No. 13

The quotation on the next page was taken from an NSA publication. The first letters of the WORDS spell out the author's name and the title of the work.

This month's NSA-crostic was submitted by guest NSA-crostician David H. Williams, P16

DEFINITIONS	WORDS
A. Feels clumsily	141 160 126 53
B. Intellectual nourishment (3 wds)	76 229 118 204 189 133 176 150 38 86 212 65 103 52
C. Russian man in the street	69 58 127 112 61 207
D. Inside scoop	8 55 163 46 186 200 95
E. Line from film by Word R, cited in Bartlett's Quotations, 14th ed. (5 wds)	172 214 20 203 109 205 88 217 32 159 134 153 121 40 105 4 51 188
F. Industrial city in NE Ohio	9 142 223 221 71 21 60 26 12 168
G. Emphasizes	183 149 31 177 165 231 119 16 78 74 93
H. Pressure group	24 139 106 135 50
I. "To be ------- is some danger," *Hamlet*, Act III, Sc. iv (2 wds)	11 83 77 132 115 224 199
J. "... of their appointed ------"	64 90 1 161 192 34
K. Prolific motion picture director (2 wds)	44 30 79 218 39 181 73 94 17 227 56 100 146 48 110
L. Reinforcing	213 37 144 173 230 138 129 98 208 171 123 2 96
M. Erases	131 84 36 13 120 155 222
N. Another line by Word R cited in Bartlett's Quotations	196 175 91 104 18 5 209 137 187 68 164 41 219 47 122 201 75 148 193 10 156 99
O. Gemstone	107 67 182 152
P. Eluder	195 14 116 206 102 166 33
Q. Word K's fiftieth film, 1966 (2 wds)	54 143 210 125 85 22 42 185 57 226 180
R. Star of eleven films ("Every Day's a Holiday," "Night after Night," etc.), author of eight of them (1893-)	170 80 162 97
S. Vicinity	113 45 147 92 82 63 15 128

T. Question asked of the cook at the monastery's fish-and-chip stand (5 wds)
220 49 59 101 211 202 72 28 108 62 136 114 3 174 89 228 23 151

U. Process of producing images by reaction of light
6 194 66 198 111 216 27 81 158 130 157

V. Intermittently (3 wds)
35 140 25 179 154 215 169 191

W. Answer to query in Word T (5 wds)
70 124 7 43 184 117 190 145 178 197 225 87 29 19 167

1 J	2 L	3 T	4 E	5 N	6 U	7 W	8 D	9 F		10 N	11 I		12 F	13 M	14 P
	15 S	16 G	17 K		18 N	19 W	20 E	21 F	22 Q	23 T	24 N		25 V	26 F	27 U
	28 T	29 W	30 K	31 G	32 E	33 P	34 J		35 V	36 M		37 L	38 B	39 K	
40 E	41 N	42 Q	43 W	44 K	45 S		46 D	47 N	48 K	49 T	50 H	51 E	52 B	53 A	
54 Q	55 D		56 K	57 Q	58 C	59 T		60 F	61 C		62 T	63 S	64 J	65 B	66 U
	67 O	68 N	69 C	70 W	71 F		72 T	73 K	74 G	75 N		76 B	77 I	78 G	
79 K	80 R	81 U	82 S		83 I	84 M		85 Q	86 B	87 W	88 E	89 T	90 J	91 N	92 S
93 G	94 K	95 D	96 L		97 R	98 L	99 N		100 K	101 T	102 P	103 B	104 N	105 E	
106 H	107 O	108 T	109 E	110 K		111 U	112 C	113 S		114 T	115 I	116 P	117 W		118 B
119 G	120 M	121 E	122 N	123 L	124 W	125 Q		126 A	127 C	128 S		129 L	130 U	131 M	
132 I	133 B	134 E	135 H	136 T	137 N	138 L		139 H	140 V		141 A	142 F	143 Q	144 L	
145 W	146 K	147 S	148 N	149 G	150 B	151 T	152 O		153 E	154 V	155 M	156 N	157 U		158 U
159 E	160 A	161 J	162 R		163 D	164 N	165 G	166 P		167 W	168 F	169 V	170 R	171 L	
172 E	173 L	174 T	175 N	176 B	177 G	178 W	179 V	180 Q	181 K		182 O	183 G	184 W		185 Q
186 D		187 N	188 E	189 B	190 W	191 V	192 J		193 N	194 U	195 P		196 N	197 W	198 U
199 I		200 D	201 N	202 T	203 E	204 B		205 E	206 P	207 C	208 L		209 N	210 Q	211 T
212 B	213 L	214 E	215 V		216 U	217 E	218 K	219 N	220 T	221 F		222 M	223 F	224 I	225 W
226 Q	227 K	228 T	229 B	230 L	231 G										D.H.W.

(Solution next month)

¿CONOCES BIEN LA GEOGRAFÍA?
Tony Melzer, D5*

Esta vez queremos que tu dés no sólo el nombre de cada país sino el de su capital también.

	País	Capital
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		

Las respuestas se hallarán en el página 21.

Tanteo	
0-8	¡Muy malo!
9-12	Talcualillo
13-15	Bastante bueno
16-17	¡Muy bueno!

* Cartógrafo y autor del texto en inglés; traducido en español por Esteban Vermás.

C.A.A. NEWS By W.E.S.

May 1978 Presentation Planned

As of this writing (February 1978), it *looks* pretty firm that our featured speaker for May will be our #1 Member (see reproduced membership card). Watch for announcements about day and time.

COMMUNICATIONS ANALYSIS ASSOCIATION
THIS IS TO CERTIFY THAT
VADM B. R. Inman, USN
is a member in good standing of this association for the year ending 31 December 19 78
1 '78
President

(U)

What to do for members on PCS?

CAA members who transfer to overseas jobs are continued on our membership rolls "free" (no dues) for the duration of their PCS tour overseas. One of the things the CAA Board will be wrestling with this year is how we can extend the aims and goals and benefits of the CAA to people who are "off campus." We want to talk with those of you who have ideas about this. Have you been overseas and "out of touch"? What do you think the CAA can do to help people who are now in the situation you were in? (U)

Meet our new president!

Our new president is David W. Gaddy, a Cryptologist in D5. He earned his BA in U.S. History at the University of North Carolina and an MS in International Affairs at George Washington University. [redacted]

His cryptologic experience has included assignments as linguist, bookbreaker, reporter, and supervisor on a wide variety of problems and targets in Southeast Asia. He was selected for the Armed Forces Staff College, Norfolk, Virginia, in 1967, and for the National War College in 1971.

Since his return from the War College, he has held various senior planning positions, including a tour at the Pentagon (NCR Defense). He is currently Deputy Chief, Intelligence Community Affairs (D5).

[redacted]

[redacted]

He is certified as a Linguist and Special Research Analyst. ~~(C-CCO)~~

...and our new president-elect!

Our new president-elect is Francis X. Porrino, Chief of A1, Office of Operations and Control. He has served in the cryptologic community since 1951, first as an enlisted man in the Air Force Security Service, then in several capacities as a civilian (since 1955). [redacted]

He has performed at all levels of management through Office Chief. [redacted] has attended the National War College, has served on the staff of the NCRDEF, and has served as Chief, A1. Also, over the course of his NSA career, he has completed studies at the University of Maryland, earning a BA degree in Government and Politics, and has served overseas in the Office [redacted] He has also been awarded the Civilian Meritorious Service Award.

EO 1.4.(c)
P.L. 86-36

He is certified in TA and SR and has been a member of the SR Career Panel. ~~(C-CCO)~~

P.L. 86-36

How to become a member

CAA dues are $1.00 per year. You can get membership applications from any of the people mentioned in these CAA news items. Send the completed application and $1.00 to Tim Murphy, B09, 3W114. Then just sit back and watch your in-basket fill up with news bulletins and announcements about all the many and varied activities and programs of CAA. (U)

Communications Analysis Association:

President	David Gaddy	3247
President-elect	Frank Porrino	5879
Secretary	[redacted]	
Treasurer	Tim Murphy	3791
Board members	[redacted]	

I'm running out of space! I wanted to give a biographical sketch on our newest CAA Board member, [redacted] and mention our Program Committee chaired by [redacted] Can't wedge them in. I'll try next month!

ata

american translators association

Dear President Carter:

September 14, 1977

The following letter was sent on September 14, 1977 to the President of the United States by Royal L. Tinsley, Jr., President of the American Translators Association.

Dear Mr. President:

Please accept my congratulations and sincere appreciation for your encouraging attitude toward the study of foreign languages and cultures as expressed in your letter of 29 June 1977 to Congressman Dante Fascell concerning the Helsinki Final Act. Let us hope that the foreign-language teaching profession has learned its lesson from the 1960s and will make better use of future support by the Federal Government. I am a dedicated teacher of German (and sometime Russian teacher), but it is not as a teacher of foreign languages that I am writing this letter; you will no doubt receive enough letters from my colleagues in the profession.

Rather, as President of the American Translators Association I am deeply concerned about the future of the translating profession in this country. Although the study of foreign languages is critically important for the welfare of the U.S., no one can learn more than a few languages, and most will do well to acquire minimal competence in even one foreign language. Anything the individual learns about all the other cultures must ultimately be learned through the medium of translation.

You are more aware than most Americans, I am sure, that the United States is no longer the world leader in several scientific and technical fields. This is no catastrophe, in itself, for surely every nation should have the right to excel at something. I think you will agree, however, that unless our American scientists and technicians can keep abreast of the latest developments in their respective specialties there is, at the very least, undesirable duplication of effort and waste of time, energy, and money. Translation provides the only feasible means by which we can keep up with developments in all areas of the world.

Reprinted by permission of "The ATA Chronicle," newsletter of the American Translators Association, P.O. Box 129, Croton-on-Hudson, N. Y. 10520.

In recent years there has been much duplication and waste even in the vital area of technical translation. Cover-to-cover translations of foreign technical journals often provide translation of worthless material along with the crucially important. There should be a screening process for each discipline whereby leading experts in the field would select important articles and books for translation by competently trained translators.

The latest developments in science and technology are now being published in more than 70 languages of some 100 countries. Even though English still accounts for about 45% of this material, and another 35% is published in Russian, German, and French, some 20% appears in the less-well-known languages. The information value of Japanese publications has increased remarkably over the past few years, while Hungarian has become very important for electronincs, Swedish for metallurgy, power engineering, and communications systems, and Finnish for woodworking, pulp and paper technology, icebreaker construction, etc. Polish, Czech, Bulgarian, and Rumanian technical literature is also becoming important, as is Spanish and Portuguese. When an agreement is finally reached that will permit us access to the technical literature of the People's Republic of China we shall have several decades of development to study, and there is less than a handful of people in the United States today who are competent to translate technical Chinese.

The number of translators in the Federal Government has been steadily reduced over the past several years. The shocking state of affairs in government translation was described by Richard S. Relac in a report in *The Federal Linguist*, Vol. 6, No. 1-2 (Summer 1974). In what was presumably intended as an economy measure, government translators were progressively phased out by elimination of jobs and by refusal to replace personnel lost through death or retirement. Many simply resigned because they were given GS ratings below those specified for the positions and were denied promotions. Government translation needs were supplied to an increasing degree by contractors who received the work on a low-bid basis. Despite claims to the contrary by government procurement officers it can be documented that this procedure resulted in an incredible waste of money for a large number of marginally useful or completely worthless translations. Most professional translators either refused to work for the ridiculously low rates paid by the contractors or, in some few cases of which we are ashamed, they simply did not spend the time and effort necessary to do a professional job. The contractors were forced to recruit students, immigrants, war brides, etc., in order to meet their commitments. Some of these people did indeed become competent translators in time, but at a staggering cost to the taxpayers!

Mr. Lester C. Benefeld, Jr., Director of the Foreign Language Research Branch of the U.S. Army Foreign Science and Technology Center, was questioned during a meeting at Georgetown University in 1975 about the quality of this contract translation. Mr. Benefeld answered that most of these translations would never be used, and that it was sufficient, in any case, if the Army's scientists and technicians could deduce from the equations, tables, and graphs what the article was about! One wonders why the government should have paid even low rates for translation that apparently was not needed.

"Shocking state of affairs in government translation."

In July of this year an official of the U.S. Bureau of Reclamation in Denver stated that there are government scientists and technicians who honestly believe that their Russian and German counterparts are incapable of writing understandable reports of their own experiments because every translation these American technicians have tried to use has been virtually incomprehensible! Competent professional translators do not turn Russian and German technical writing into incomprehensible English, and the Bureau of Reclamation asked the American Translators Association to hold a workshop for translators so that the Bureau could explain the types of materials it needed translated. Through this workshop the Bureau found several professional translators who could provide competent translation services for reasonable compensation.

This false economy move of the Federal Government has also had its negative effects on foreign-language education in the colleges and, especially, in the high schools of this nation. Granted that the foreign-language teachers were slow to respond to the desire of students to learn foreign languages for practical use rather than as an entrée into the literature of these languages, government policies must also share a large part of the blame for the decline in foreign-language study in this country during the past few years. Back in 1973 President Nixon's budget wiped out support by the National Defense Education Act of some 4500 courses in more than 80 languages and in all foreign areas -- at a savings of less than the cost of one F-111 fighter-bomber! [Marshall D. Schulman, *The New York Times*, April 6, 1973, p. 39.] With the Federal Government eliminating career opportunities and foreign-language programs in the schools, indicating by its own example that foreign-language study was of little or no importance, is it any wonder that young Americans have no desire to study foreign languages?

The United States is unique among the large industrial nations of the world in its attitude toward translation. There is practically no training available for translators in this country. Most of our professional translators are either self-trained through years of trial and error, or they received training in European schools for translators and interpreters. According to the Relac Report, there were no more than a half dozen translator trainees in the entire U.S. Civil Service in 1974. Prior to 1970 the only training programs were at Georgetown University and at the Monterey Istitute of Foreign Studies. Since then some 25 or 30 colleges and universities have initiated one or more courses in translation techniques, but the only viable, comprehensive training programs that have been added for technical translators are those at the University of California at Santa Barbara (French, German, Spanish), Carnegie-Mellon University (French, German, Russian, Spanish), Stanford University (German), the University of Puerto Rico (Spanish), St.-Mary-of-the-Woods College (French, Spanish), and the Rose Hulman Institute of Technology (German, Russian).

"No reliable statistics exist for the amount of translation in the United States."

Moreover, most of the students in these programs (all of them at the Rose Hulman Institutute) are not training to become translators -- they are future scientists and engineers who want to learn how to read foreign research in their own fields. (They will still be able to read only one or two foreign languages, however.) Of a total 162 students enrolled in 5 of these programs as of October 1976, approximately 60 hoped to become translators. The same 5 programs graduated 12 students in 1975, at least 10 of whom were employed in non-translation jobs or in positions where their language skills were of only minor importance.

In contrast, Canada has 8 training institutions for translators and interpreters, and Western Europe has over 40. There is already a critical shortage of competent technical translators in this country, and if the Bilingual Courts Act ever becomes law there will be utter chaos with every Spanish-speaking janitor pressed into service as a court-interpreter. In a courtroom situation where property, freedom, and even lives may depend upon accurate translation of not only words but of culturally significant gestures, tones of voice, facial expression, etc., we cannot afford to have untrained people acting as interpreters. Translating and interpreting require training and experience, NOT just bilinguality.

"A national disgrace"

No reliable, comprehensive statistics exist for the amount of translation performed annually in the United States, the amount spent for translation, the relative importance of various language combinations and of the different scientific and technical fields that use translations, the number of full-time and part-time translators in the country, etc., etc., etc. On the other hand, periodic surveys of almost every other aspect of business provide current statistics on how many hamburgers are consumed annually in Kalamazoo or how many pickles one can expect to sell in Atlanta. Since 1970 I have contacted the Bureau of the Census, Department of Commerce, Congressmen, Senators -- anyone and everyone I thought might persuade the Federal Government to collect such data along with the myriad other statistics they are compiling continuously -- all to no avail. How can we plan training programs intelligently, how can we predict how many translators will be needed in 10 years, for example, and in which languages and technical fields, without reliable data concerning the present state of the profession? The American Translators Association simply does not have sufficient funds to gather such statistics.

The Soviet Union has these vital statistics. The All-Union Center for Translation of Scientific and Technical Literature and Documentation in Moscow reports that its translation output doubled between 1970 and 1973, that it increased by 150% between 1973 and 1975, and that the 1976 output was 30% greater than in 1975, totally some 350 million words. The All-Union Center is only one of several agencies in the Soviet Union. The Chambers of Commerce and Industry translated 230 million words in 1975, Intourist translates about 17 million words annually, and the Central Research Institute for Patent Information a little over 7 million words. This was a total of more than 500 million words in 1975, and other, smaller departments, agencies, and research institutions in the Soviet Union are reported to have translated several times that amount. [V. N. Gerasimov, *Translation News*, Vol. 7, No. 1 (April/May 1977), pp. 1-11.]

Moreover, since 1975 all translations in the Soviet Union have been registered. We have a National Translations Center at the John Crerar Library in Chicago that makes a valiant effort to eliminate duplication and reduce costs by making existing translations available for approximately the cost of duplication, or by directing requestors to the proper source for the desired translation. Unfortunately, National Science Foundation funds for the NTC were cut off a few years ago and the NTC staff was reduced from 11 to 4. Over the past four-and-a-half years the NTC has deposited or reported availability of 85,891 technical translations, but even more were deposited with or reported to the NTC -- the staff of four simply could not process any more. The NTC estimates that no more than half the technical translations made in the U.S. (and probably considerably less than half) are reported to them, due at least in part to lack of funds and staff to make the existence of the NTC known to everyone involved in the production and utilization of translations. This, Mr. President, is a national disgrace.

Although I have concentrated on technical translation because that is the critical field, commercial translation (banking reports, correspondence, contracts, bills of lading, etc.) is playing an increasingly important role as American business tries to compete in the world marketplace. Nor should we forget the importance of translation in making literature and drama from other cultures available to the American public. The only training program for literary translators in the U.S. at this time is at the State University of New York at Binghamton.

In your efforts to encourage the study of foreign language and cultures I hope you will also remember the neglected but very important role of the translators of America -- in science and technology, in business and commerce, and in the humanities. In closing I would like to offer my services as an individual and the cooperation of the American Translators Association to the Commission proposed by Congressman Simon.

Respectfully,

Royal L. Tinsley, Jr.
President of ATA
Associate Professor of German
University of Arizona
Tucson, Arizona 85721

Letters to the Editor

To the Editor, CRYPTOLOG:

The note at the end of [redacted] article "A Linguist Looks at the 'Tube'" (CRYPTOLOG, March 1978) states that the article was "untouched by human hands." Whenever I read or hear such statements, I start looking for human intervention. I remember, for example, machine-translation output that used to be advertised as "completely un-postedited" that would prove to have hyphens appearing mysteriously (but correctly!), or to have proper names output in capital letters (such as EYZENKHAUER) even though the names had been "unrecognized" by the computer.

From reading the article, I get the impression that the right-hand justification is done by adjusting the spaces between the words in each line, without bothering to use any complicated hyphenation rules. But I see (last line, left column, page 11) "organi-" and (line 6, right column, page 11) "justifi-." Who put in those hyphens and when?

Hy Fennwatcher

The author of the article replies:

The word processing system used for my article accomplishes right-margin justification by inserting a maximum of two spaces between words and works best on normal lines of 60 to 80 characters.

Because of CRYPTOLOG's short lines, the word processor occasionally was unable fully to perform its right justification on my article and, in fact, left a very ragged margin in the paragraph that Hy Fennwatcher mentioned.

Therefore, to improve the appearance of the paragraph, my helper interactively (i.e. on the CRT screen) hyphenated "organization" and let the computer realign the rest of the paragraph. Seeing that the results were acceptable except for the last three lines, she then hyphenated "justification" in the third-from-last line and let the computer redo the last two lines.

This intervention would not have been necessary for a page of standard width, but it is a good example of the ease with which text can be revised on a CRT-based text handling system. The alterations did not require retyping of the article or even a page.

[redacted] G92

(U)

To the Editor, CRYPTOLOG:

Yes, Art, CRYPTOLOG ads *do* get results. In a quarter-page filler in the January 1978 issue I listed some "new and improved aids for bookbreaking." Within a week after the issue hit the streets, the persons listed as contact points were bombarded with requests for information or copies of the programs. It pays to advertise in CRYPTOLOG!

An unsolicited testimonial from a satisfied subscriber -- V. V. (Bookbreaking and Cryptolinguistics Coordinator, P16) (U)

Solution to NSA-crostic No. 12

(*CRYPTOLOG*, March 1977)

Sam[uel S.] Snyder, "[*Influence of U.S. Cryptologic Organizations on the Digital*] *Computer Industry*, National Security Agency, Fort George G. Meade, Maryland, May 1977, 36 pp., S-217,130, (U).

"One of the earliest commercial high-speed line printers came about because of NSA support, based at least partly on experimentation and initiatives by Agency engineers. Likewise, one of the first practical character-sensing machines received early [contract] encouragement from NSA."

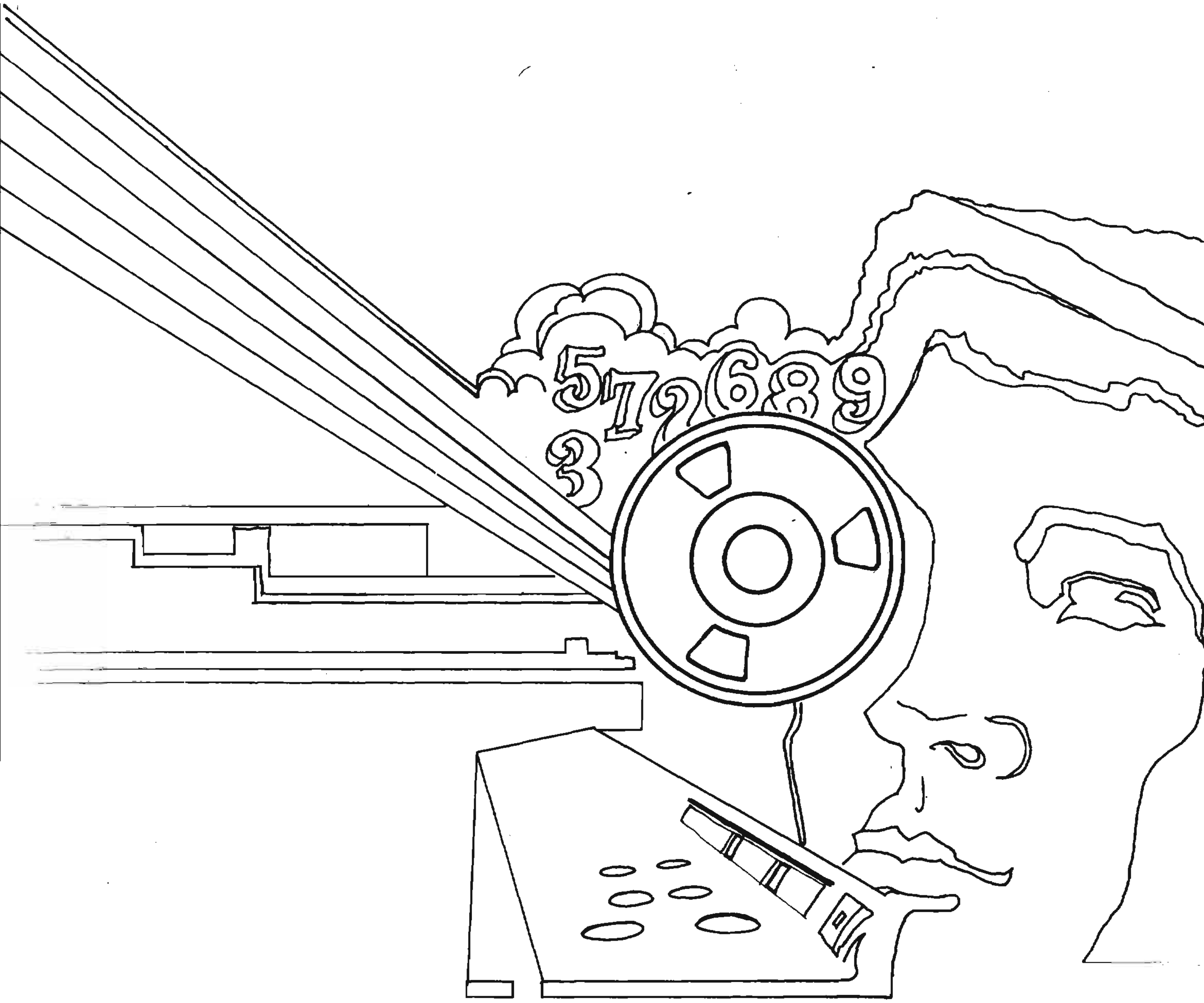
(U)

Solución a "¿Conoces Bien la Geografía?"

(p. 18)

1.	Honduras	Tegucigalpa
2.	Nicaragua	Managua
3.	Costa Rica	San José
4.	Panamá	Panamá (Panama City)
5.	Colombia	Bogotá
6.	Venezuela	Caracas
7.	Guayana (Guyana)	Georgetown
8.	Surinam	Paramaribo
9.	Guayana francesa (French Guiana)	Cayena (Cayenne)
10.	Ecuador	Quito
11.	Perú	Lima
12.	Brasil (Brazil)	Brasilia
13.	Bolivia	La Paz/Sucre
14.	Paraguay	Asunción
15.	Chile	Santiago
16.	Argentina	Buenos Aires
17.	Uruguay	Montevideo

(U)


572689
3