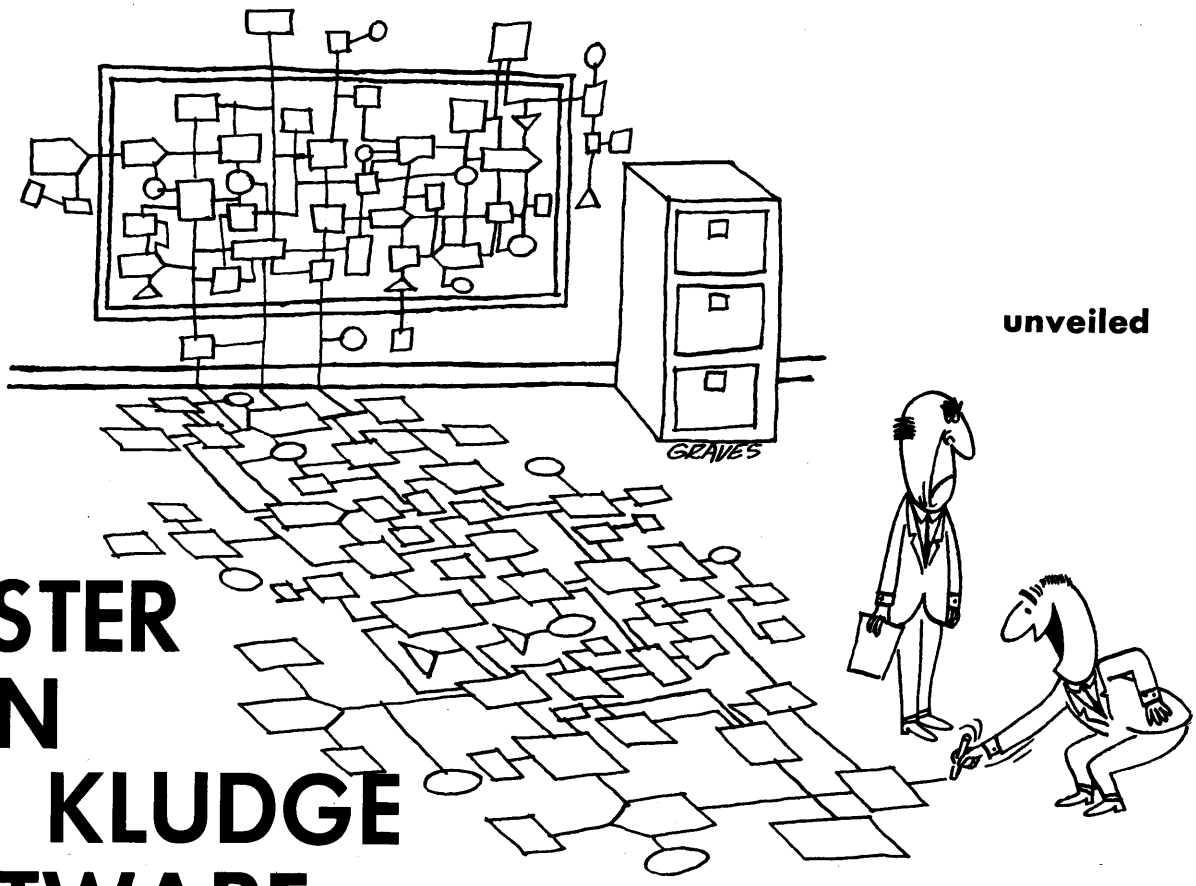

unveiled

THE MASTER PLAN FOR KLUDGE SOFTWARE

by Dr. MORRIS L. MORRIS & Dr. AUSTIN O. ARTHUR, Kludge Komputer Korp.

Previous writers in this series have attempted to show how the road to non-computing can be fostered at its foundations by:

(1) Engineering glitches into the hardware. With suitable ingenuity, these can be either new botches (discovered after the prototype is constructed, usually, and then firmly cast in concrete) or loving repetitions of what never did work right.

(2) Marketing the resulting Kludge with the same techniques used to peddle deodorants and cigarettes. The analogy is painfully accurate.

(3) Applying time-tested rules for maintenance which operate faithfully to minimize uptime. These rules are a sort of check list for the Kludge Fixer so he can save time by listing excuses by the numbers.

Now, brief reflection quickly reveals that the above approaches are *only* the foundation. The real approach today has far greater scope, since it strikes at the heart of the matter – software. Everyone knows that software is *the* thing. The planners at Kludge Komputer Korporation are keenly aware of this trend. Having pioneered so well in the fundamentals, they can be expected to excel here too. What follows is the basic outline followed at KKK. The Master Plan is the result of years of study and represents the ultimate in software systems for users of the famous line of Kludge hardware.

Kludge software master plan

The cardinal Commandment of any software development program is "Announce it first, worry about producing it later."

The first and most important Edict is, "Always start with fresh programmers." This is a simple rule and its raison d'être is obvious. Any programmer who has used or designed another system has been adulterated or biased and such bias (sometimes referred to as experience) may well influence him in the task he is to perform.

Edict 2. "Never let your fresh programmers talk to the programmers who designed the last software package, if this can be avoided – but at all costs never let any software programmers talk to the people who design, sell or maintain the hardware." This rule needs no explanation.

Edict 3. "Never let the software specifiers talk to the software implementers." This form of warping young programmers' minds is to be avoided like the plague. Software designers always seem to have the weird idea that they better than anyone else, know how to implement their designs.

Edict 4. "Never let a software group know that there are other groups." A weaker form of this rule is also useful, "Never let one software group know what the others are doing." The stronger form of this rule tends to create very high morale or esprit de corps brought about by a feeling of exclusiveness. In the weaker, but more pragmatic form, the effect is acquired by implanting the idea that all of the people in all of the other groups are a bunch of inexperienced poopheads.

Edict 5. "If you must document the software, be sure that the documentation is done by a wholly separate group, preferably technical writers who are not too knowledgeable about computers and programming." Many benefits accrue from this approach, but the greatest one is the high regard with which your documents will be copied. Worry not about their accuracy; users are more adaptable than you think. The proof of this is found in the great number of installations still operating Kludges with *no* documentation.

Turning our attention now to the design phase of software (predicated of course upon the above personal practices) the following rules have served not only the Kludge

Komputer Korporation but many of its fellow companies for many years.

Rule 1. "Every software package must have a monitor, irrespective of the size of the Kludge." Without a monitor to occupy between 25 and 50% of the available fast store (and at least ¼ of the backup storage) the users' programmers will become careless. They will eventually discover that one can trade time for space – which leads to reduced utilization and thus, rental income. An appropriately designed monitor on the other hand can be depended upon to *waste* a minimum of 10% of the available time just searching the system tapes.

Rule 2. "All error messages must be as coy and ambiguous as possible." The object here is to make the console operator (and, later, the programmer) unstable. For example, what could be simpler than "AN IMPOSSIBLE ERROR HAS OCCURRED ON AN UNSPECIFIED UNIT WHILE EXECUTING AN UNIDENTIFIABLE PROGRAM." It should be pointed out that the creative souls who can compose such wonderful phrases are rare beasts, indeed, and when found should be coveted and nurtured.

Rule 3. "There should be more phases in the monitor than there are jobs or programmers in a given shop." This is the only realistic settlement to the ever raging argument between the one-phase and three-phase proponents. Besides, how else can you provide the capability for clobbering programmer A's phase p results while running programmer B's phase q interlude if everyone knows what is to happen in each phase? And what better way to keep an operator guessing – you see he is much less likely to call for the KF's under these circumstances – another form of saving not to be discounted.

Rule 4. "Every software routine should have a snappy acronymic name." For instance, we at Kludge Korporation called our Kludge Monitor System KLUMSY; the Kludge assembler, KLAP; and the alternate version, KLAPTRAP, required for a Kludge with traps (working or not).

Rule 5. "Before letting pragmatic aspects interfere, be sure that the things that count are taken care of. SEE THE SALES DEPARTMENT FIRST AND THE COPY WRITERS IMMEDIATELY." As we all know, if a Kludge can't be sold on its hardware merits alone, (and it can't) the software must carry the burden.

A Kludge cannot subsist on a monitor alone. Thus it behooves the Kludge software suppliers to provide working languages for the Kludge. The more the merrier. Since the list of OK languages changes from season to season, and position within the list is not constant (or computable), one had best consult one's marketing people to find out which ones have strongest current motivational appeal. Within these bounds, the guiding principles of Kludge languages follow.

Principle 1. "Hop aboard all of the current band wagons." If FORTRAN is the current best selling gee-wiz, write (promise) a better one. Change the name slightly so that you don't lose your identity, but not so much so that you can't tell who whelped it. Thus, Kludge Korporation's version of FORTRAN will be called KLUDGTRAN.

Principle 2. "Always release preliminary undebugged versions of the translators, compilers, generators, assemblers, etc." Why NOT? Let your customers debug the things. Why should you spend your programmers' time and operate a machine with all that awful overhead? If your customers want it badly enough, they'll check it out for you. (Unfortunately, the day is gone when you could get him to design and implement it, too.)

Principle 3. "No 'preliminary' or 'field test' version of any translator should be compatible with the monitor or any other translator." The savings realized by the elimination of coordination and liaison would amaze you. This also prevents future coordination and liaison because each "field test" version gets too deeply imbedded for anyone ever to want the final version anyway.

Principle 4. "Join and actively support any and all government sponsored and international magic language generation efforts." This lets you know what the opposition is thinking. But be careful: contribute only those ideas which you know are impossible to implement on the competitors' hardware, or will at least make it look bad. Always be ready with a claim to have a working version of whatever language is the current vogue.

Principle 5. "Never, NEVer, NEVER write a decent or useful training manual for any system." Remember, you may be taking the bread out of the mouth of some striving young author.

Principle 6. "If you must supply an assembler, do it uder duress." The best ploy to use to get out of this one is to tell the customer he never ever needs to know the basic language of the machine. All of his problems can be solved with the New International Magic Language Number 6.5. Well, our version of it anyway. As we all know, magic languages are the thing, and you can't sell a Kludge without magic.

We must now turn our attention to software maintenance; an area too often ignored.

Tenet 1. "Each program should be on a separate tape with its own unique format." This is logical. You wouldn't want the corrections for one system to be acceptable to another, would you?

Tenet 2. "Corrections should be distributed at such a rate (empirically determined) as to keep the users from inundating you with requests for additions and changes to the system." It seems best to protect our programmers as much as possible from new and different ideas. Besides, we've always done business this way.

Tenet 3. "Distribute new versions of each system as soon as the previous one is showing signs of being checked-out." This guarantees job security for a very large segment of the programmer community known as "System Programmers."

Tenet 4. "Whenever a new system is proposed or implemented, refuse to continue maintenance on some other (any other) existing system." How far can *you* make a rubber band stretch? We've already provided for keeping our programmers busy!

Tenet 5. "Never let the programmers who implemented the system maintain it." It has been found over the years that many system programmers acquire a certain attachment for their own code and refuse to consider sullying it with corrections. Thus, the only solution is to have some other programmer do the appropriate surgery (preferably a brand new one, fresh from the university) in the form of absolute binary patches.

In closing, we should like to point out that we of the Kludge Komputer Korporation have managed to keep abreast of the competition in software as well as hardware by having a loyal claque. Which brings us to the last Commandment:

"FORM A USERS GROUP," whence springeth all that is worthwhile. Wine and dine them, buy their loyalty and in every way possible make sure that *your* users are satisfied and happy. Channel their desires appropriately. Hold meetings at least twice a year and see that all of the attendees get smashed (we at KKK sometimes pick up the tab) and go home generally feeling loved, wanted and appreciated by all.

the bitter end