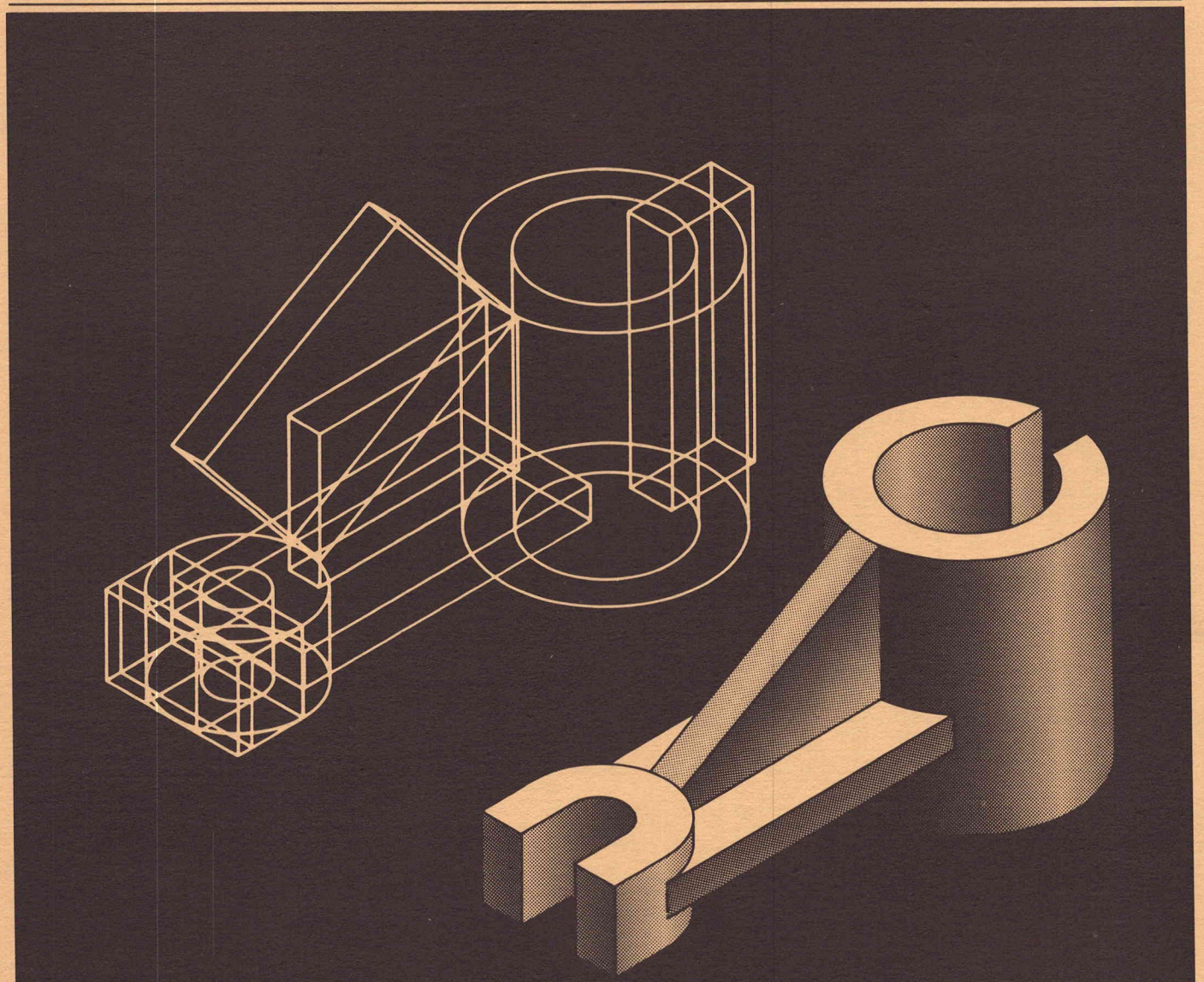
TEKTRONIX, INC. COMPANY CONFIDENTIAL

MAY 1980

TECHNOLOGY report

HARDWARE SOFTWARE FIRMWARE PROCESS ENGINEERING MATERIALS RESEARCH



TOWARDS ACCURATE GEOMETRIC-DESCRIPTION SOFTWARE

CONTENTS

Volume 2, No. 5, May 1980. Managing editor: Art Andersen ext. 8934 (Merlo Road), d.s. 53-077. Staff editor: Laura Lane. Cover and graphic design: Joe Yoder. Graphic illustrator: Jackie Miner. Composition editor: Jean Bunker. Publisher: Burgess Laughlin. Published by Technical Publications (part of Technical Communications Services) for the benefit of the Tektronix engineering and scientific community.

FEBRUARY PAPERS AND PRESENTATIONS

While providing recognition for Tektronix engineers and scientists, the presentation of papers and the publication of papers and articles contribute to Tektronix' technological leadership image.

The table below is a list of papers published and presentations given during February 1980.

The Technical Communications Services' (TCS) Engineering Support group's charter is (1) to provide editorial and graphic assistance to Tektronix engineers and scientists for papers and articles presented or published outside Tektronix and (2) to obtain patent and confidentiality reviews as required.

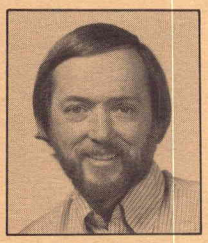
If you plan to submit an abstract, outline, or manuscript to a conference committee or publication editor, take advantage of the services that TCS Engineering Support offers. Call Eleanor McElwee on ext. 8924 (Merlo Road). □

TITLE	AUTHOR	PUBLISHED	PRESENTED
GENERAL:			
"Human Interface — A Learning Model"	Tom Bruggere	—	Human Interface Panel
SOFTWARE:			
☐ "PROWAY — A Local Network for Process Control"	Maris Graube	—	COMPCON
PROCESS ENGINEERING:			
☐ "Electrostatic Discharge Study of Solid-State Devices"	Herb Zajac	—	NEPCON
HARDWARE:			
"Display Technologies"	Jack Bennett	—	University of Wisconsin
☐ "Upgrading a TRS-80 to Terminal Status"	Jim Tallman	**Microcomputing**	
"An Electron-Bombarded Semiconductor Tube for High-Speed A/D Converters"	Ray Hayes	—	A/D Workshop
☐ "CRT Reigns Supreme for High-Speed Digitization"	Neil Robin	**Electronic Design**	—
"Computer-Based Approach to Measuring Dynamic Performance of A/D Converters"	Lyle Oaks	—	A/D Workshop
☐ "Taking the Heat Off Semiconductor Temperature Testing"	William Mark	**Electronics**	—

A box to the left of the title indicates that copies are available. For a copy of a paper or article listed here, photocopy this table, check the appropriate box, and mail to TCS, d.s. 53-077.

Name_____ **D.S.**_____

TOWARDS ACCURATE GEOMETRIC-DESCRIPTION SOFTWARE



Jack Gjovaag is a software engineer in Graphic Computer Systems Application Development, Information Display Division. Jack joined Tektronix in 1975 from California Computer Products. He has 15 years experience in computer graphics including automated cartography, graphic software development and computer-aided design and computer-aided manufacturing research and development.

Computer technology has not yet developed to the point where general-purpose automated factories are possible, but new advances are rapidly bringing this goal closer. The critical need of this emerging technology is better communication of geometric information between machines.

Automating a manufacturing activity is sometimes viewed as nothing more than configuring machines to process material without human intervention, such as with robots and NC (numerically controlled) machine tools. If this view were correct, then automated factories would now be common because such machines exist today.

In manufacturing operations geometric information is ubiquitous; it appears as descriptions of such things as raw-material shapes, shapes of objects to be manufactured, clamp and fixture shapes, machine-tool capabilities, and tool geometry.

Conventional manufacturing operations maintain geometric information mostly in drawings designed to communicate geometric information between people. To a computer, without either the human capacity for pattern recognition and the vast human storehouse of general knowledge, a drawing represents just a collection of lines. Thus drawings will not survive as the primary geometric-information storage medium in automated manufacturing.

DESCRIPTION OF PHYSICAL OBJECTS WITH THREE GENERATIONS OF SOFTWARE

There have been three generations of software for dealing with geometric information. Most of today's commercially-available software belongs to the second generation, but third generation packages will appear soon.

The first generation of geometric description software is computer-aided drafting software. It concentrates on capturing the information necessary to display a conventional drawing. This software is not adaptable to automated manufacturing.

Second generation software attempts to capture geometric information that is directly related to the object to be described. For example, a surface on a part may be described in three dimensions by a mathematical function rather than in two dimensions by way of the profile or **loft lines** of the surface. (Loft lines represent complex surfaces.) An edge on an object might be described as a segment of the curve of intersection of two three-dimensional surfaces rather than as a two-dimensional projection of the edge for a particular view of the object.

Second-generation object descriptions suffer from two significant problems: They are potentially incomplete and potentially invalid.

Continued on page 4

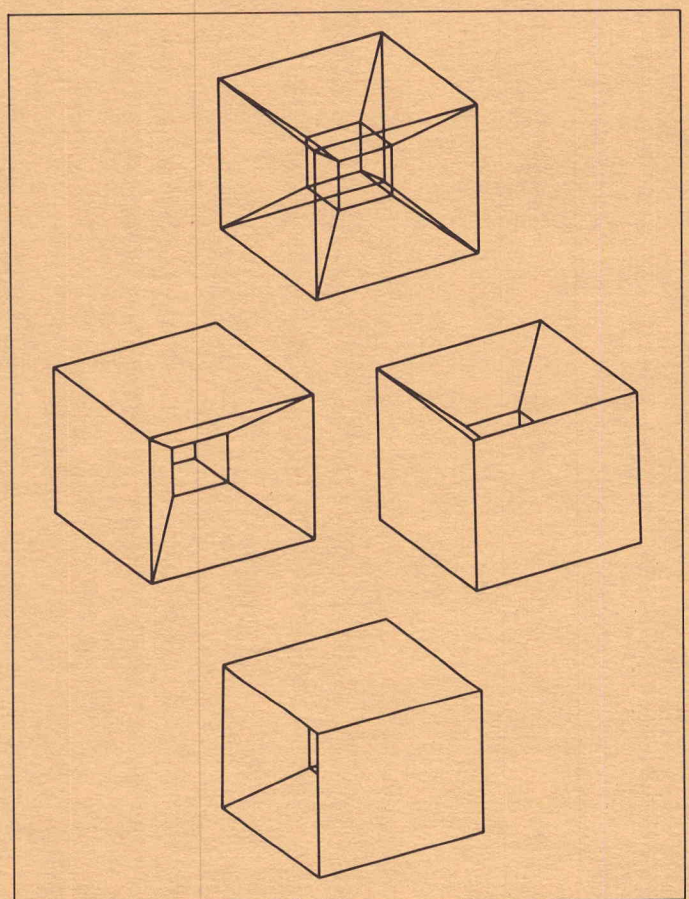


Figure 1. Illustration of an incomplete object (top) with three possible interpretations.

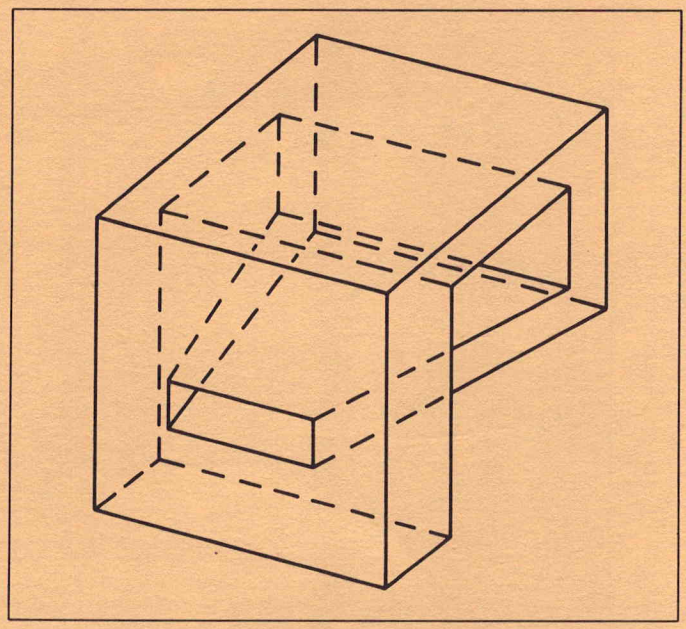


Figure 2. Illustration of an invalid object description (a Klein bottle).

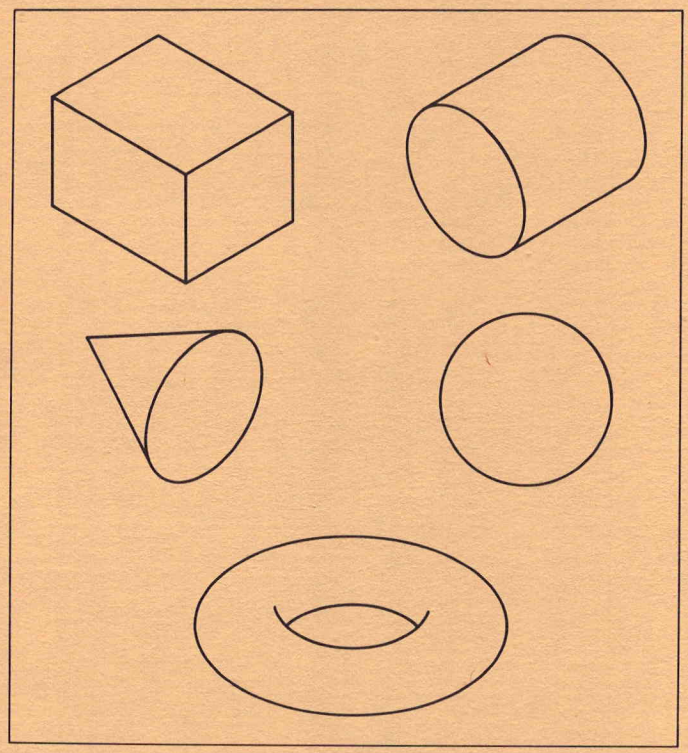


Figure 3. A set of elementary objects.

Continued from page 3

An **incomplete object description** lacks sufficient information to enable the software to determine all of an object's geometric properties. For example, the edge description of the solid object illustrated in figure 1, is incomplete because it does not allow the software to discriminate between three possible interpretations.

An **invalid object description** has no physically-recognizable counterpart. Figure 2 shows an edge representation of an impossible physical object: a Klein bottle.

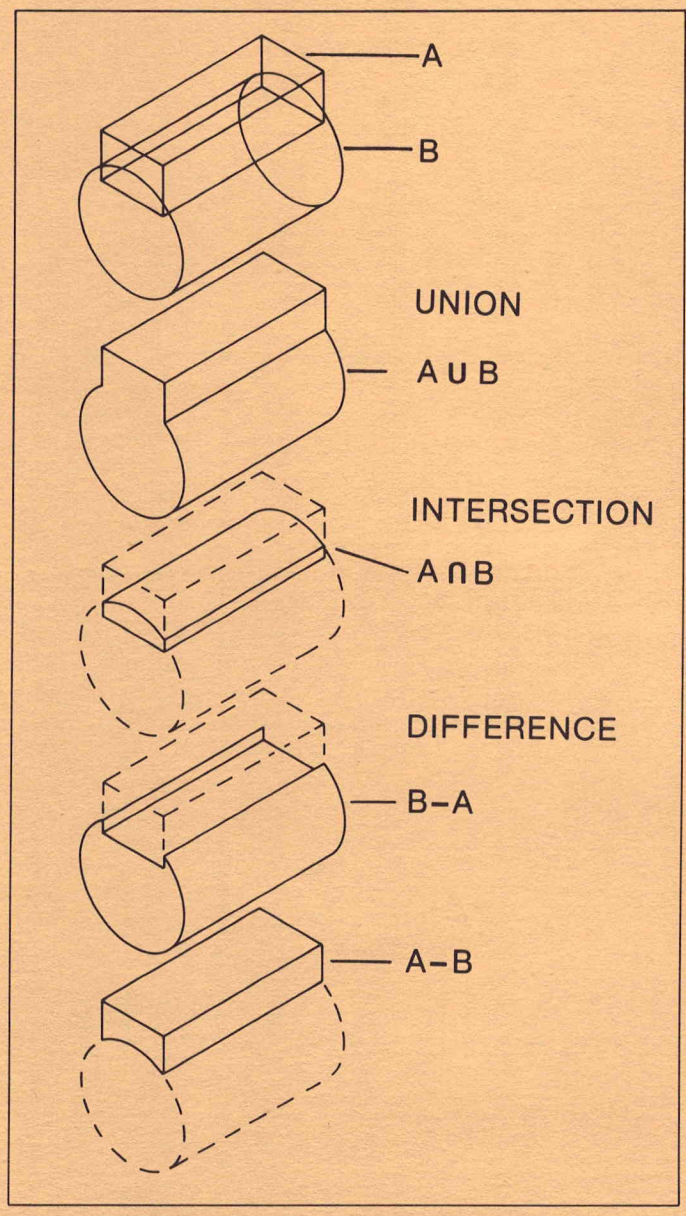


Figure 4. Object operators and their functions.

The goal of third-generation geometric-description software is to guarantee complete and valid object descriptions.

CONSTRUCTIVE SOLID GEOMETRY

A particularly useful scheme for describing physical objects with completeness and validity is called **constructive solid geometry** (CSG). The basic concepts of CSG have been known since 1964, but only recently have the concepts been thoroughly studied in the context of automated manufacturing and their theoretical foundations formalized.

In CSG, descriptions of objects are constructed from a set of **elementary objects,** used as "building blocks," and some **object operators** which combine the blocks in various ways. CSG reduces the problem of how to represent complex objects in a computer-readable form to the much simpler problem of how to represent elementary objects and combinations of elementary objects in computer-readable form.

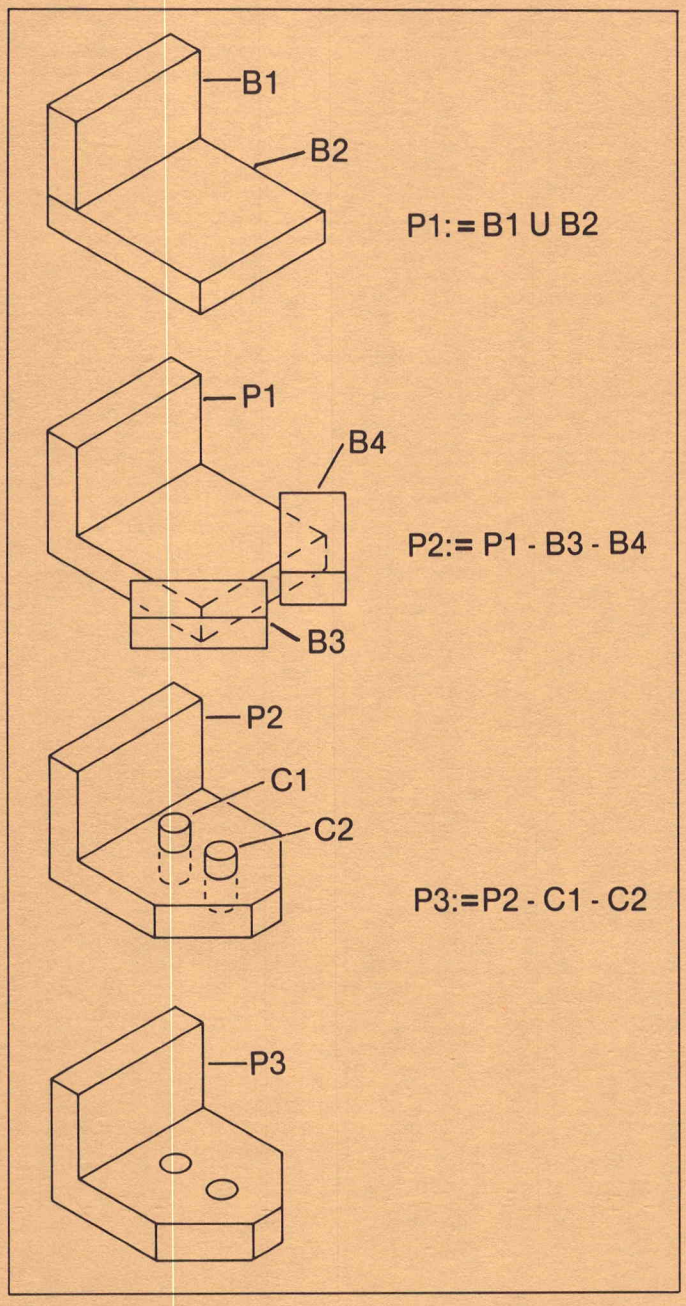


Figure 5. A sample part description sequence.

A useful set of elementary objects is illustrated in figure 3. Each elementary-object type is a prototype form where certain characteristics specific to each type are left variable. For example, the cube-type elementary object has variable height, length, width, and spatial position. An **instance** of an elementary object is produced by uniquely specifying all variable characteristics.

The object operators and their functions are shown in figure 4. These operators, which are similar to the familiar set-theoretic operators, combine instances of elementary objects; the resulting combination always is a complete and valid object description.

A CSG object description is easy to represent in computer-readable form as a binary-tree data structure. The leaf nodes correspond to instances of elementary objects and all other nodes contain object operators and are root nodes of subobject trees. Figure 6 shows the description tree for the simple mechanical part described in figure 5.

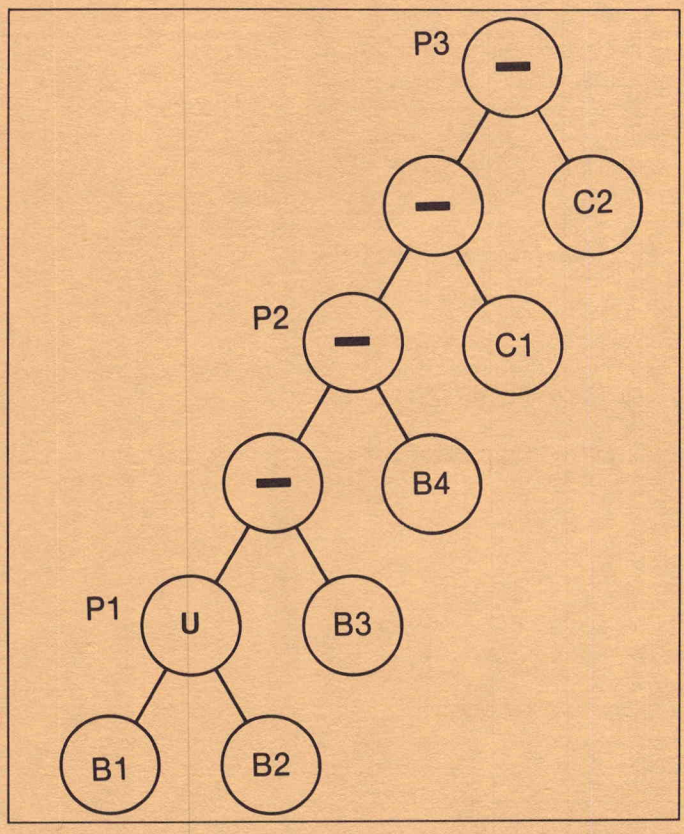


Figure 6. An object description tree for the part shown in figure 5.

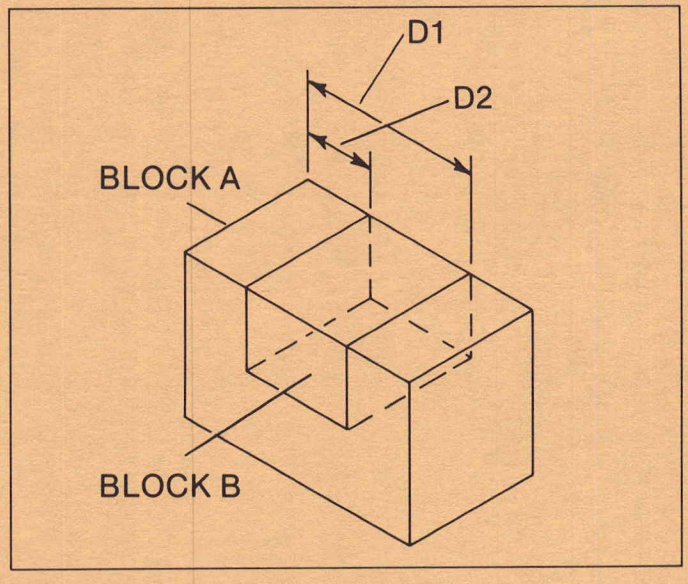


Figure 7. An example showing how dimensions control the size of an elementary object instance.

DIMENSIONS

An important question in the design of a CSG geometric-description program for mechanical engineering is what type of data the computer requires to produce an elementary-object instance from a prototype form, that is, how the parameters are to be assigned to elementary objects.

Continued on page 6

Not all methods of associating parameters with elementary objects are suitable or convenient. For example, we could choose to generate elementary-object instances by applying parameters such as scale factors, rotation angles, and translation values to prototype elementary objects of some standard size and position. This method is very simple mathematically, but it is not suitable for mechanical engineering because mechanical engineering and drafting generally use other parameters.

A more complicated, but nevertheless more appropriate, parameter-description scheme controls the shape and position of the individual mathematical surfaces in which the faces of the elementary objects lie rather than the size and position of the entire elementary object. Thus, for example, we may independently specify the positions of two opposite faces of a cube-type elementary object subject to the constraint that they remain parallel. In figure 7, the length of block B is indirectly determined by the distances Dl and D2. This parameter-description scheme enables production of instances of elementary objects from the same type of information now used by designers and draftpersons to describe shape and size, namely, functional dimensions.

Functional dimensions can be accommodated by representing nominal surfaces within the program as sets of points (x,y,z) that satisfy equations of the form: G(x,y,z,p1,...,pm,f1,...fn)=0. There is a unique function G (called a "generator") for each surface type in the description program. The parameters pl,...,pm are the dimensions that control the position (the axis of a cylindrical surface, for example) of the surface with respect to some coordinate system. The parameters fl,...,fn are the dimensions that control surface **figure** (the radius of a cylindric surface, for example).

CSG guarantees complete and valid descriptions when the elementary-object instances used in the description are complete and valid. This, in turn, is assured when all of the mathematical surfaces in which the faces of the elementary objects lie are well defined and have mutually-rigid spatial relationships. Tolerancing considerations prohibit any redundancy in the dimensions that establish rigidity of the surfaces. These requirements mathematically formalize a **properly-dimensioned object** (an object that is neither over- nor under-dimensioned). This formalization makes it possible to include tests in a CSG geometric-description program to detect and prohibit improper dimensioning.

TOLERANCES

A description of an object to be manufactured must include not only the description of the size and shape of the object in ideal terms, that is, its nominal description, but also information about how far a manufactured part's dimensions may vary from this nominal description and still be acceptable. This tolerance information, when applied to a nominal object description, produces a description of a **class** of parts, all of which are functionally equivalent and can be used interchangeably.

A geometric-description program must provide tolerancing adequate for completeness and validity of toleranced parts. **Completeness** in this context means that the toleranced part description describes a unique class of parts. **Validity** means that this part class is not empty (that it is possible for some physical object to satisfy the description). To fulfill these requirements, it is necessary to first mathematically formalize tolerancing theory and then to embody theory in algorithms.

Two tolerancing schemes are in wide use in industry: **conventional** (plus/minus) tolerancing and **modern** (or geometric tolerancing).

Under conventional tolerancing criteria, a manufactured surface is acceptable if it is **accurately** described by the generator that describes its nominal counterpart (it has negligible form errors) and with values p1',...,pm', f1',...,fn' that are "close enough" to the values p1,...pm,... f1,...,fn that describe the nominal surface. The needs of conventional tolerancing are therefore served by simply providing a facility for associating tolerance zones with nominal surface dimensions.

Modern tolerancing overcomes some of the practical problems of conventional tolerancing (such as the underlying assumption that all errors in the form of a manufactured surface are negligible) and, in doing this, appears fairly complicated. For the purpose of our geometric description program, however, we can group modern tolerances into two categories: tolerances of form and tolerances of position.

Form tolerances specify the allowable variation of surfaces from their theoretically exact forms. Constraints such as flatness, straightness, and roundness are of this type. Each type of form tolerance can be represented in our description by an appropriate mathematical relation: $Ef(S',f1,...,fn) \leqslant Tf$, where S' is the set of points comprising the manufactured surface and Tf is the form-tolerance value. The function Ef generates a measurement-of-form error independent of whatever coordinate system from which the set of points S' was measured.

Position tolerances control perpendicularity, parallelism, angularity and true position. They express the permissible variation in the position of a feature with respect to a specific reference frame called a **datum system.** Datum systems are defined from combinations of nominal surfaces on a part. Dimensions can be thought of as coordinates with respect to these systems; for example, three non-parallel plane surfaces on a part serve to define a three-dimensional datum system from which point positions may be specified with three numbers. Position tolerances can be similarly represented by mathematical relations of the form: $Ep(S',p1,...,pm) \leqslant Tp$. Ep is a position-error function and Tp is a position tolerance. The points of S', in this case, must be measured with respect to the appropriate datum system.

APPLICATION

The computerized object-description scheme this article outlined is a key component of an automated manufacturing system. The properties of completeness and validity ensure that such descriptions are sufficient for manufacturing needs but they do not restrict how the descriptions can be used.

Use of a computerized object-description scheme must be approached independently for each manufacturing activity that uses geometric information. Design analysis, process

Continued on page 7

technical standards

LETTER-SERIES STANDARDS

Many Tektronix Letter-Series standards are still valid and applicable. Tektronix Standard 062-4055-04 (Directory Standards) lists the Letter-Series Standards; 062-4055-02 lists the Letter-Series Standards by subject matter. These standards comprise what is left of the Volume 1 Standards; the Technical Standards group will be reformatting these standards to part-numbered standards as they complete work on them.

Technical Standards doesn't maintain these standards unless a specific change is brought to our attention. We will thoroughly review these standards when we reformat them; in the meantime, please call any inconsistencies to our attention (ext. 241, Town Center).

STANDARDS UNDER REVISION

Technical Standards is reviewing the following Tektronix standards to see that they serve Tek's needs. Send information for revision or update of these standards to Technical Standards, d.s. 41-260 (Town Center).

062-1706-00 — Converting Fractions to Decimals

062-1707-00 — Rounding Off Decimal Numbers

062-1732-05 — Component Mtg. E-Series Details

062-1732-07 — Component Mtg. H-Series Details

062-1732-08 — Component Mtg. M-Series Details

062-1860-00 — Product Safety Standard, X-Radiation

062-2318-00 — Circuit Board Standard, Marking Adherence and Solder Flux Cleaning

062-2319-00 — Switch Standard, Cylindrical Cam Switch

062-2801-04 — Forming and Shaping Asbestos Materials (Call Jerry Turnbaugh, ext. 7777, Beaverton)

062-2843-00 — Drafting Standards, Drawing Scale

062-2846-00 — Drafting Standards, Glossary of Terms

062-2851-00 — Cable Standards, Glossary of Terms

062-2877-00 — Cable Standards, Wire and Cable Color Coding

NEW STANDARDS

To borrow or order copies of standards, call ext. 241 (Town Center).

ANSI B1.13M — **Metric Screw Thread** — M Profile

MIL-R-83407 — Amendment 1. General Specification for **Relays, Reed** (Mercury Wetted).

MIL-S-22885/7 7B — Amendment 1. **Illuminated, Push-button Switches,** 4-Lamp, 0.75 Square, 7.5 Amperes, Electromagnetic Interference Shielded.

MIL-STD-1562B — Superseding MIL-STD-1562A. Lists of Standard **Microcircuits.**

MIL-C-81706 — Int. Amendment 5. **Chemical Conversion Materials** for Coating Aluminum and Aluminum Alloys.

EIA-RS-186-9E — Standard Test Methods for Passive Electronic **Component Parts.** Method 9: Solderability.

MIL-C-17E — Supplement 1. General Specification for Flexible and Semirigid, Radio Frequency, **Cables.**

UL-1418 — Implosion-Protected **Cathode-Ray Tubes** for Television-Type Appliances; revision pages for pages 9, 10 dated November 4, 1977.

MIL-C-3965F — Supersedes MIL-C-3965E. General Specification for **Capacitors,** Fixed, Electrolytic (Nonsolid Electrolyte), Tantalum.

MIL-C-39012B — Supplement 1B. General Specification for **Connectors,** Coaxial, Radiofrequency.

MIL-STD-889B — Revision Notice 1. **Dissimilar Metals.**

IPC — **IPC Technical Review,** January issue, featuring part one of a two-part technical paper, "Condensation Soldering Technology."

UL 913 — Revision page for Standard for **Intrinsically Safe Apparatus and Associated Apparatus** for Use in Class I, II, and III, Division 1, Hazardous Locations.

MIL-W-5086C — **Copper or Copper Alloy Wire,** Electric, Polyvinyl Chloride Insulated. Amendment 1.

Continued on page 10

Continued from page 6

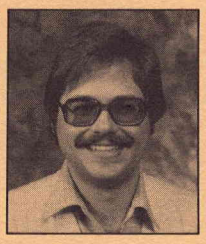
planning, NC programming and inspection, for example, have different information needs. Much research remains to be done on the information needs of each activity before complete automation of any of them is possible; realistically this automation is five to ten years away.

However, shorter-term benefits can be realized by a gradual application of third-generation geometric-description software to the machine-to-machine communication process. Significant application programs can be developed with available technology that, while still requiring humans in the processing loop, are nevertheless generally superior to those supported via second-generation geometric software.

FOR MORE INFORMATION

For more information call Jack Gjovaag, ext. 3548 (Wilsonville). □

THE ADVANTAGES OF DECISION TABLE PROGRAMMING



Harald Philipp is a design engineer in Frequency Domain Instrumentation and a member of the Engineering Activities Council. In FDI he has been responsible for the design of a synthesized timebase and firmware. Hal joined Tektronix from the National Bureau of Standards in 1978. At NBS he developed custom instrumentation for high-energy fusion research projects. His BSEE is from Michigan Technological University.

All Tektronix programmers seek coding techniques that (1) promote coherent, unambiguous code; (2) provide clear and concise documentation in parallel with or ahead of code development; (3) simplify walkthroughs and debugging during development; and (4) simplify making and debugging program changes during development. **Decision-table programming** possesses these qualities, yet does not sacrifice code size or execution speed.

DECISION TABLE

A table of all contingencies that the programmer should consider when describing a problem, together with actions to be taken for each contingency.

A decision table is a programming element that is completely self-contained. A decision table incorporates the equivalent of many IF--THEN--ELSE-ELSEIF program lines, organized in a simple, clean, orderly tabular format. Given a set of initial conditions, a decision table determines which set of actions will occur as a result of those conditions.

ADVANTAGES

A distinct advantage to decision-table programming is that it forces engineering discipline on what has otherwise been considered an art.

Where decision-table programming has been used it has produced dramatic results. Bell Telephone made extensive use of decision tables in their ESS-4 switching system where they were able to achieve productivity levels of 25 documented, debugged lines of code per programmer-day. The resultant code was essentially error-free and extraordinarily simple to maintain, even though the programmers were not selected for special ability.

My own productivity has been about 17 documented and debugged lines per day, but I was limited by the absence of a decision-table compiler. Complete debugging of my 1200-line program took seven days. The program has run error-free for more than three months. Coding and documentation were completed six weeks ahead of schedule.

Figure 1 describes the advantages of decision-table programming over other techniques during the various stages of the software life cycle.

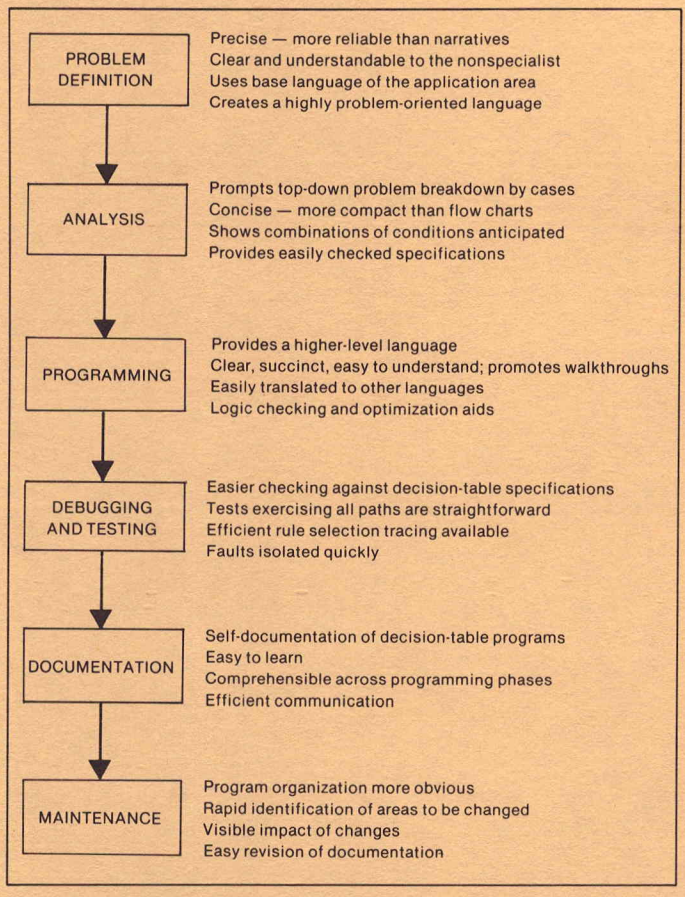


Figure 1. Advantages of decision table use by programming phase (from *Decision Table Languages and Systems*, Metzner and Barnes, Academic Press).

Aside from their ability to structure program definition and implementation, decision tables are a powerful management tool; design reviews and structured walkthroughs become straightforward. A decision-table format is a clear, concise method of communicating to management the status of a project during all phases.

Most decision-table languages are implemented in a host language, such as Fortran, Cobol, or PL/I. In some cases

Continued on page 10

GLOSSARY

Array processors — High-speed arithmetic processors used as "add-ons" to conventional processors. Due to their parallel and pipelined architectures, array processors are extremely efficient at performing the multiple computations required for decoding and executing decision tables.

Associative memory — A memory device where memory locations may be identified by specifying all or part of their contents. This type of memory is used where extremely fast data look-up is required. They can be used to perform rapid decision-table rule checking.

Decision table, limited entry — A minimized primitive decision table, where impossible combinations of conditions are eliminated and where combinations of conditions that result in identical actions are combined. Usually, programmers may use standard Karnaugh-map reduction techniques to minimize a primitive table.

Decision table, primitive — A decision table containing all possible combinations of conditions, together with actions to be taken under every such combination.

Multiprocessing — The use of two or more processors to simultaneously execute a given program, or possibly to execute two separate but related programs, in order to speed up the execution of a problem.

State-transition programming — A technique used to program state machines, where at a given interval of time the code can be considered to be in a particular 'state' or condition. An advance to a new state is achieved by consideration of the present state along with tests of selected variables. The technique is particularly useful for specifying decision-table linkages.

Stub — The left-hand portion of a decision table. For an example, see figure 2.

Table splitting — A technique whereby a single large decision table may be broken up into smaller tables that, in sum, are equivalent to the large table. This method can greatly reduce the total number of table entries.

Top-down programming — A programming technique that begins with the specification of all inputs, outputs, and activities of the program. The programmer then develops the overall structure of the program along with tight specifications of subfunction inputs, outputs, and activities. Contrast with bottom-up programming in which a programmer first specifies and writes subfunctions, and then fits them together later.

Walkthrough — A detailed examination of a small portion of a design or program by a team of programmers for the purpose of finding errors, discussing alternate techniques and code, and solving problems. Decision tables greatly simplify this task.

HOW IT WORKS

In decision-table programming, computation results or status information are checked in a table to determine the sequence and flow of actions. The programmer performs this checking during the development phase, as the code is written. During the execution of this code, the processor performs an identical function in the checking of stored tables. In general, a program would contain numerous decision tables.

Each decision table can decode a multi-way branch from the decision variables. Further, a table can contain minimizations to reduce entries. Tables also contain lists of actions to be taken after a branch. Figure 2 is such a table, describing the options for a traveler who wishes to fly to Boston. The condition stub documents the decision variables; the action stub documents the possible outcomes or actions. Figure 2 expresses conditions as Boolean true-false possibilities and indicates the resultant actions by an "x." This table is minimized: it does not express all of the eight possible combinations of conditions. This "limited entry" table has been reduced from a larger, primitive table. Such minimizations can usually be performed with conventional Boolean reduction techniques.

Each table can be accessed from another table; after the execution of an action sequence, a table can provide an unconditional branch to another table. Each table can have as many different such branchings as there are rules. Figure 3 shows a table which references five other tables.

Unfortunately, decision-table programming has not gained wide acceptance because it is unsuitable for bottom-up programming. As top-down analysis and programming gain acceptance, decision-table use should increase.

		RULES				
CONDITION STUB	WEATHER IS FAIR	T	F	-	-	-
	PLANE SEAT RESERVED	T	T	F	T	F
	HOTEL ROOM RESERVED	T	T	T	F	F
ACTION STUB	TAKE THE PLANE	X				
	CANCEL PLANE SEAT		X		X	
	TAKE THE TRAIN		X	X		
	TRY AGAIN TOMORROW				X	X

- = DON'T CARES
X = ACTIONS

Figure 2. Trip-to-Boston decision table — limited entry form (from *Decision Table Languages and Systems,* Metzner and Barnes, Academic Press). If the conditions are: the weather is *not* fair, the plane seat *is* reserved, and the hotel room is reserved, then the traveler will cancel the plane seat and take the train.

	TABLE B	RULES				
CONDITION STUB	A =	ϕ	ϕ	≥ 2	1	
	$B > C \cdot (A)^{3/2}$	T	F	T	F	ELSE
	$C = \phi$	T	T	F	F	
ACTION STUB	$J = \phi$					1
	$J = 1$	2		1		
	$B = C^2 + J$		1	2		
	L = TRUE	3		3		
	CALL SUBC	1	2	4	1	
BRANCH STUB	TABLE K		X			
	TABLE D				X	
	TABLE F			X		
	TABLE A					X
	TABLE B	X				

Figure 3. The table above is a decision table with sequenced action columns and a branch stub (a "stub" is the left-hand portion of the decision table).

Consider the situation where, prior to the execution of this table, A = 0, $B \leq C \cdot (A)^{3/2}$, and C = 0. A decision table interpreter scans the rules from left to right until a match is found (second rule). Following this, the computation $B = C^2 + J$ is performed, and subroutine 'SUBC' is called, in that order. Upon completion, the processor skips to table K, another table in the program.

The 'ELSE' rule is normally for error trapping, in case no match is found in the other rules.

Continued from page 8

they are written as preprocessors to a host language. It is also possible to write your own, as an interpreter, to augment the host language. As a last resort, you can use CASE statements, as in PASCAL or TESLA, to approximate a decision-table format. In my own situation, I found TESLA to be quite suitable for this. The beauty of decision tables lies in their consistency. In a decision-table language there is only one language construct to learn, and variances among decision-table languages are minimal.

THERE ARE SOME DISADVANTAGES BUT . . .

Decision tables have some pitfalls: (1) some programmers claim the tables lack flexibility, (2) tables for lengthy control structures can be cumbersome, (3) tables are not well suited for specifying and designing mathematical algorithms, and (4) improperly minimized tables may contain rules which conflict, causing ambiguity in table meaning.

A recently discovered technique that overcomes most problems splits a large decision table into smaller, linked tables thereby reducing or eliminating ambiguity. Table splitting also reduces table entries by an order of magnitude, or more. This splitting technique is so simple to implement that one wonders why it was not discovered long ago. Table linkage can be specified with state-transition techniques.

It is true that lack of flexibility (pitfall number 1) in decision tables severely restricts personalized programming "style"; the user is forced to conform to the given format. But this objection is the link to understanding the many advantages of forced uniformity. It is precisely this uniformity that makes table programming so predictable, repeatable, and serviceable. The methodology places a very tight lid on tricky, synergistic, bottom-up code. And where table-driven programming fails in mathematical algorithm construction, it is a simple matter to write those sections of code in an ordinary high-level language and link them in module form to the tables. These modules can be called by reference as an action in any table.

MULTIPROCESSING TOO . . .

The possibilities for multiprocessing are striking. Rule checking and execution of action sequences could be split easily among several processors. Associative memories and array processors also adapt to decision-table programming quite well, due to the matrix nature of the table format. Decision-table advocates have noted that manufacturers could easily add microprogrammed table decoding instructions to processors. And the programming process itself is readily adaptable to entry and modification by graphic display systems, rather than conventional text editing.

Because a decision table is primarily made of data elements, not syntax elements, there is no reason why a piece of code could not modify itself during its own operation. One simple application of this is to have the code dynamically restructure table content order to enhance operating speed by keeping statistics on branching probabilities. To reduce overhead, this restructuring could be done only during code development. Minimal additional code would be required.

SUMMARY

There are many advantages, and a few disadvantages, in using decision-table programming. In summary:

(1) Decision tables are a unified high-level software program element. They are easily written and modified. They communicate software design efficiently and are well suited for structured walkthroughs.

(2) Decision tables are highly deterministic, due to their rigid format; disadvantages of this rigid format are offset by inherent predictability and serviceability.

(3) Mathematical algorithm computations are awkward to implement with decision tables. In compensation, it is easy to imbed ordinary code into a decision-table program.

(4) State-transition programming techniques readily adapt to the decision-table methodology as an assist to the programming process.

(5) Decision tables are an elegant solution to the problem of programming of multiprocessor architectures. In addition, decision tables are suitable for use with advanced array processors, associative memories, and graphic display systems.

FOR MORE INFORMATION

For more information call Hal Philipp, ext. 7240 (Beaverton). □

Continued from page 7

DOD-B-24507B(SH) — Amendment 1. Military Specification **Battery Cells,** Storage, Silver-Zinc Alkaline Type (for NR-1) Metric.

NBS PB-294 845 — Guide to **Technical Services and Information Sources** for ADP Managers and Users.

NBS PB-293 487 — **CMOS/SOS Test Patterns** for Process Evaluation and Control: Annual Report. March 1 to November 1, 1978.

NBS Technical Note 958 — Four Versatile **MIDAS Compatible Modules.**

ANSI/NMA MS4-1972 (R1978) — National Standard **Flowchart Symbols** and Their Usage in Micrographics.

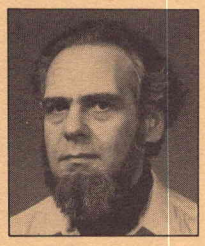
NMA MS 301-1979 — Micrographics Systems **Flowcharting Template.**

UL 544 — Revision pages for Standard for **Medical and Dental Equipment.** Second Edition. □

ENGINEER IV PROFILES

Engineer/Scientists IV's and V's serve as technical resources both inside and outside the company. To increase their visibility to the Tektronix' technical community, *Technology Report* is publishing a series of profiles of these individuals.

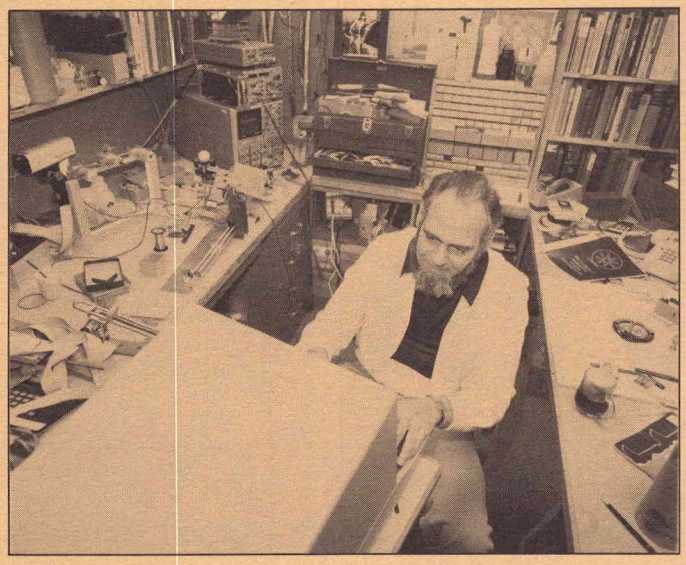
JON MUTTON



Jon C. Mutton, Technology Development Group, Information Display Division, ext. 3722 (Wilsonville).

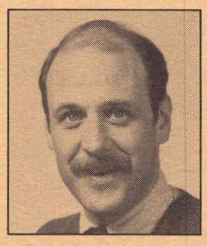
Jon C. Mutton is an engineer specializing in the technologies of graphic input and man/machine interfaces. Jon also acts as a hard-copy technology consultant here at Tektronix. He joined Tektronix in 1964 and helped pioneer direct-view storage tubes. Working in an advanced development group within what was then Display Device Development put Jon in the right place at the right time — this group was one of several which combined to form the Information Display Division (IDD) in 1965. His past projects contributed to the 4501 Scan Converter, 4551 Light Pen and most of the hard-copy units Tektronix has marketed. He is presently in the Technology Development Group of IDD.

Jon's degree is in physics from Portland State. He has published papers in his field: "A Hard Copy Device for Digital Video Signals" in the 1972 **SID Digest of Technical Papers,** "The Role of Display Terminal Copiers" in several editions of (1972-1976) **the Proceedings of Institute for Graphic Communications,** and "Hard Copy — Big Business for Tektronix" in **Technology Report,** January 1980.



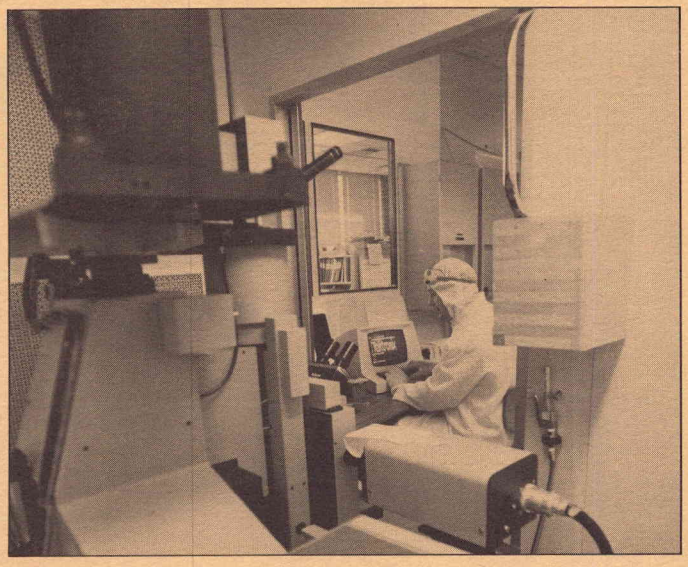
Jon holds the following patents: No. 3,594,608, "Electron Beam Display System Having Light Detector Pen With Associated Sampling and Memory Circuits" and No. 4,148,042, "Electrographic Copier With One-Piece Belt and Styli" (jointly with Pete Unger). □

DELMER FEHRS



Delmer Fehrs, Monolithic Circuits Development, ext. 5528 (Beaverton).

Delmer Fehrs is an integrated circuit process engineer in the Monolithic Circuits Development Branch of the Component Development Group, part of Tek Labs. He is responsible for the development of photolithographic processes. IC photolithography is the dominant technique by which images, generally quite small, are transferred to precise locations on a silicon wafer. The process forms a photosensitive resist layer which is then exposed to the image. The image is then developed and etched into the wafer. Finally, stripping the resist layer completes the process of photolithography, which accounts for about 35% of the manufacturing process of integrated circuits.



Before joining Tektronix about two years ago, Del worked in photolithography for Bell Laboratories in Allentown, Pennsylvania, in the production of high-density random-access memories. He also designed optomechanical systems for automated pattern inspection. Before Bell Labs, he studied the absorption of alkali metals and halogens by refractory metal surfaces at the Research Branch of the NASA Electronics Research Center in Cambridge, Massachusetts. The object of the studies was to find low work-junction surfaces to be used in direct thermionic conversion of heat to electricity.

Continued on page 12

Continued from page 11

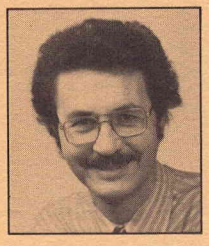
Recently, Del has been improving photo processes and starting up and characterizing new lithographic equipment to meet the accelerating demands for higher circuit densities. His first project at Tek involved upgrading of photolithographic production processes for the super high freqency circuits (SHF III) used in the 1 Ghz bandwidth 7104 oscilloscope.

Del has recently written one paper and co-authored another on photolithography: "Linewidth Control in Contact Lithography" in the **IEEE Transactions on Electron Devices** and "An Empirical Approach to Linewidth Control in Projection Lithography" in the **Proceedings of the 1979 Kodak Interface Conference.** He has also written various papers on laser-scanning systems and on absorption of alkali metals, halogens, and other electronegative elements upon surfaces of refractory metals.

After graduation from Grants Pass High School in Grants Pass, Oregon, Del went east and "hung around MIT for a sufficient time to obtain S.B., S.M., and Ph.D. degrees, in what is best described as engineering science, from the Department of Mechanical Engineering."

He is the co-holder of two patents: No. 3,879,131, "Photomask Inspection by Real Time Diffraction Pattern Analysis" and No. 3,944,369, "Optical Comparator System to Separate Unacceptable Defects from Acceptable Edge Aberrations." □

JON MORRIS



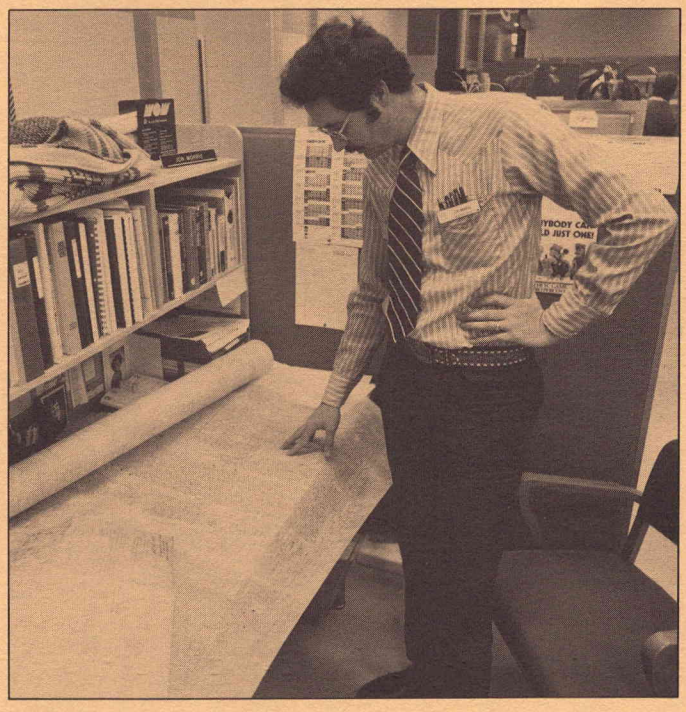
Jon J. Morris, Product Engineering, IC Manufacturing, ext. 5613 (Beaverton).

Jon J. Morris is a group leader in Product Engineering, part of Integrated Circuits Manufacturing (ICM). He's been with Tektronix for over two years, during which time he has helped transfer NMOS and CMOS wafer fabrication processes from engineering to manufacturing. He has also helped transfer MOS/LSI and high performance digital/analog products from Tek Labs to ICM and developed new standard MOS test keys used in production wafers. A test key is an IC containing various-sized transistors, resistors, capacitors, line-width-spacing structures, and other devices built into the wafer for controlling and measuring the electrical characteristics of the wafer fabrication process.

As group leader for MOS/LSI products and horizontal bipolar products in ICM, he is responsible for introducing and then sustaining in manufacturing a group of key semiconductor devices used in a variety of Tektronix products. Some of these products are: the 4050 Series Graphic Computer Systems, the 492 Spectrum Analyzer, the 1410 Series Television Generators, and the 5000/7000 Series Oscilloscopes.

Prior to joining Tektronix, Jon was a project leader for MOS/LSI design at Electronic Arrays; there his products were 16K EPROMs, 8-bit microprocessors, electronic organ circuits, and calculator chips. At Electronic Arrays, he also managed engineering services. Earlier, Jon was a design engineer at the Singer Corporation.

Jon holds a degree in electronics engineering from the California State University at Fresno. He has published a technical report, "Power Losses in an Extra High Voltage Transmission Line" in the **Pacific Gas and Electric Company Technical Digest.** While at Electronic Arrays, Jon developed over 50 proprietary ICs and several proprietary design techniques.



Jon makes some observations based on his twenty years in electronics, including military service: "I've found that any project, no matter how simple or complex, is only as successful as the ability of people to work together. Teamwork is the key, that's why we need to maintain and promote the Tek Spirit."

Jon describes what his group produces as "products" although they are not listed in the **Tektronix Catalog.** He likes to think of them as part of a hierarchy of Tek products, "one building on others," to produce the final product we sell to the outside world. In this sense cables, transformers, subassemblies, and other components are all *Tektronix products* because we make them and we sell them to "our customers" — other Tektronix organizations. □

PLANNING FIRMWARE PROJECTS

Jack Grimes is an engineering group manager in Graphic Computer Systems. He managed the firmware development for the 4051, learning the hard way much of what this article is about: the process of planning a firmware project. Jack has been with Tektronix nine years, working first on the 31 Calculator. He holds an M.S. and a Ph.D. in electronic engineering with a minor in computer science. Jack is an adjunct professor of computer science at OSU and teaches in the TEK-OSU program. He is also doing part-time graduate work in psychology.

This article discusses planning firmware projects at Tektronix. Because firmware is "software in ROM," most of the discussion applies equally to software projects. *Forum Report 13* and the October *Technology Report* described the other major firmware-project phases: staffing, implementation, verification, and production. For copies, call Technical Communications Services, ext. 8934 (Merlo Road).

The planning phase for firmware projects has four subphases: defining requirements and results, selecting development tools, preliminary scheduling, and estimating costs.

DEFINING REQUIREMENTS AND RESULTS

The speaker at a training session that I attended summarized the need for requirements by saying, "If you don't know where you're going, then any road will take you there." The most important part of planning firmware projects is the vital first step of determining where you are going and what *results* are required. The planner must specify these results accurately and concretely.

Many of us find it easier to specify activities rather than results. There are two popular management techniques that can help: **MBO** (management by objectives) and **RIO** (responsibilities, indicators, and objectives). These techniques both focus on results (sometimes called objectives or goals) and on *their measurement*. The ability to measure results is an excellent test for their quality.

The hardest part of defining requirements and results is stating them in terms of customer wants and needs. Today, our products are too diverse and firmware-based products are functionally too complex to use the "next bench technique" (assuming the customer is like the person at the bench next to yours). This is especially true in the Information Display Division where we have few traditional instrumentation customers. One of our allies for determining customer requirements is, of course, the marketing organization. Just as engineering is responsible for implementing the product, marketing is responsible for identifying customer requirements. If you don't have a set of written customer requirements to test the EIS against, how do you know anyone will buy the product when it is complete? Or, how many they will buy and why?

As our products acquire sophisticated functions (firmware based), you need psychological information about customers: Are customers willing to learn how to operate a newer, more complex device, or must the product function exactly like an existing product? And, most important of all, you need increasingly more complete information about customers' applications, rather than broad statements such as "troubleshoot computers, build instrumentation for measuring geophysical data," or "test high-performance semiconductors."

The planner's goal, then, is the specification of requirements for the product in terms *meaningful to customers*. In a sense, the product specification should lead directly to a sales brochure or a preliminary data sheet that the customer will need to make a "buy" decision.

Identifying requirements, features, or results often produces a long list. That is good. The longer the list, the more customer-oriented value is likely to result. However, a long list is usually a problem because of the time and resources needed to implement it. The next step is to prioritize features to create a fall-back position that you can use when the inevitable resource limitations occur. Prioritization also helps resolve conflicting requirements.

A useful technique for describing features is specifying the requirements in measurable terms; use nouns and numbers to describe results. Terms such as "fast" and "powerful" are difficult to quantify and are not useful requirements terms.

SELECTING FIRMWARE DEVELOPMENT TOOLS

In firmware development, most people associate tool selection with selecting a programming language. In fact, once you have selected a programming language you have selected perhaps one-fourth of the necessary tools and have completed the easiest part of the tool-selection process. Further, the name of the programming language (whether it is PL/M, or assembly language, or PASCAL) is only part of the lanquage selection task. After language selection you should ask: Is the programming language resident on a small development system such as our 8000 series? Or, does it run on a host system such as the Cyber system or the DEC 10 to generate code for the product that the firmware is being developed for?

Other significant questions revolve around what *debugging tool* you are going to use in the selected language. This is especially important when you are using a high-level language such as PASCAL. The PASCAL compiler often generates assembly language when high-level debugging tools are not available. Then the programmer has to take what should be an unnecessary step: figure out what the

Continued on page 14

compiler generated in assembly or object code in response to the high-level commands. The programmer must debug the program at a more primitive level than at which it was written — a considerable extra effort (about 2:1). A *symbolic debugger* is the proper companion development tool for a PASCAL compiler; it enables the user to interact with the program at the source level, avoiding debugging at a more primitive level.

Whether the programming language runs in a resident environment or on a host machine, the firmware designer usually needs a *linker* and *library support tools.* The importance of linker and library support increases dramatically when the number of people working on the project reaches three to five. At this level it is usually efficient to develop modules individually and then link them together as a final step before testing the combined efforts. The host computer and most resident systems provide library support tools which enable the designers to update, merge, and control changes to source programs.

Resident *development-support hardware* deserves special attention. It may seem expensive to spend as much as $20,000 for a resident development system for one firmware designer, but the importance of providing this powerful tool cannot be overemphasized. The firmware development process is so labor-intensive that, when analyzing the capital cost of the your project, you should discover that development-hardware or multiple-development systems, even at $20K each, are a good buy for many projects. This is particularly true where the program is not large and you don't need the more powerful resources of the Cyber or DEC 10 System; or, where timesharing access to a host computer significantly affects the firmware designer's productivity.

In thinking about tools, you might consider this: Last year 250,000 Americans bought 1/4" drill bits. Not one of those people wanted a 1/4" drill bit, they all wanted 1/4" holes!

PRELIMINARY SCHEDULING

This section discusses scheduling when insufficient scheduling information is available. The most valuable kind of preliminary scheduling information is historical data. Within Tektronix we have historical data for all sizes of projects and people are quite willing to share that information.

Normally we underestimate the time needed to do things which have many or complex interactions. This understanding is not compensated for by experience; it is an inherent, systematic error caused by our poor intuition concerning conditional probabilities. You should allow for the **two-times-scheduling factor.** (Unscheduled tasks and unexpected problems seem to add up to 50% of the project time; we all go to unscheduled meetings, get sick, and receive "short," higher-priority, unrelated jobs!)

Before using this magic factor-of-two you must determine whether or not the factor-of-two has already been included. If it has, that's fine. If it hasn't, it should be included; then tell your manager it has been included when the optimistic preliminary schedule goes into a product proposal.

Staffing must be a factor in scheduling. You should not plan on adding experienced people that are not already on board and assigned to the project. Very experienced people are extremely difficult to hire. For more on this difficult aspect of firmware development, see Bob Edge's article in the **Forum Report 13** (available from Technical Communications Services, d.s. 53-077).

Programmer *productivity* has been shown to vary from person to person by 10:1, to as much as 30:1. Productivity varies with the number of interactions that are required between the people designing firmware and others on the project, including other firmware groups and people within your group. A rule of thumb is that three-to-five firmware people are the maximum that can efficiently work on a project or major subsystem.

Programmer Productivity

Type of Program	Instructions Per Year
BASIC Applications	10,000
FORTRAN Translator	5,000
Operating System	1,500

Table 1. The type of programs generated, alone, accounts for a 7:1 variation in programmer productivity.

Where very few interactions occur, such as in writing application programs in BASIC, a single individual can produce 10,000 instructions per year of designed, debugged, and documented code. With more interactions, such as in developing a FORTRAN translator, productivity is about 5,000 instructions. With many interactions, such as in development of operating systems programs, productivity drops to about 1,500 instructions. The type of programs generated, alone, accounts for a 7:1 variation in programmer productivity. These factors are clearly very important to consider when making preliminary estimates of needed resources.

THE TWO-TIMES SCHEDULING FACTOR

In designing and developing complex products, preliminary estimates are often low by a factor of two. Frederick Brooks recounts the experience of an aerospace company in his book **The Mythical Man Month.** In this particular case, an experienced team produced a carefully estimated schedule for several hundred subtasks using a PERT-chart.

When the project began to slip and it became apparent that the estimates were low by a factor of two, the project manager asked the members of the team to carefully log their time each day. The conclusion from this study was that about only 50% of the average work week was used for the subtasks identified on the PERT-chart. The other 50% of each week went for machine downtime, higher priority and short unrelated jobs, meetings, paperwork, company business, sickness, and personal time; all very reasonable items to expect on a project — but not scheduled!

ESTIMATING COSTS

Figure 2 shows that the engineering development cost balance shifts between firmware and hardware as relative product complexity increases. We cannot define complexity very well so the graph is mostly conceptual; The best complexity measure we have is the number of circuit boards (hardware) and the number of lines of source code (firmware).

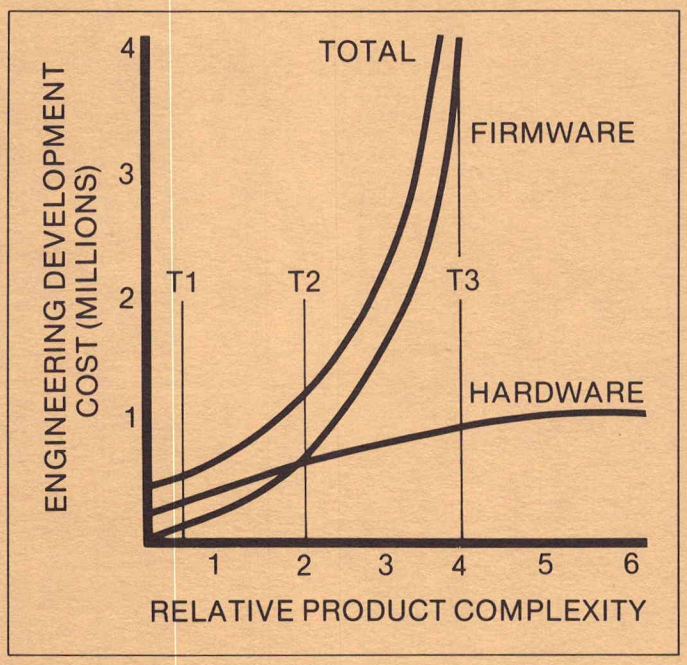


Figure 2. This graph approximates the relationship of the complexity of selected products (T1, T2, T3) to development costs. This relationship is covered in the text of this article.

The graph shows that hardware development costs approach a million dollars and then level off. Manufacturing costs, which are loosely tied to hardware development costs, also increase and then level off. Our pricing strategy causes the selling price to be a multiple of the manufacturing cost. Since we sell most products within a relatively narrow price range, this indirectly places a limit on hardware development costs.

As product complexity increases, the product acquires more computer-like characteristics and more product features move into firmware. The reason that increasing complexity encourages the increasing use of firmware is that the *selling price is not coupled tightly to the amount of firmware.* The manufacturing cost of firmware is quite low relative to that of hardware since firmware cost is determined primarily by the cost of read-only memory. The *development cost* of the firmware, however, goes up dramatically with product complexity.

On one hand, more firmware is an advantage because higher functionality results with a modest increase in manufacturing cost; but on the other hand it is a substantial disadvantage because the engineering development cost of the firmware increases rapidly. A large amount of firmware also requires many people, powerful tools, and lots of support resources. All of this is inconsistent with developing a product quickly.

In figure 2, T-2 represents the complexity of the 4051 project which was completed in 1975. T-1 represents a more traditional and less firmware-intensive project. T-3 probably represents the largest project that Tektronix ever could and would undertake.

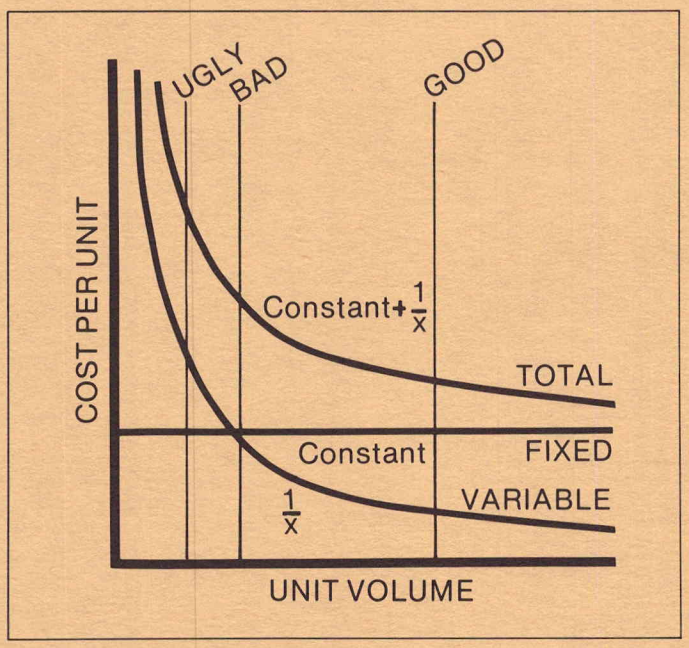


Figure 3. This graph shows how costs per unit relate with production volume. Such a graph is used to predict how a project will ultimately fare financially while you are still in the planning stage.

To estimate cost impacts, a simple model can be used. Figure 3 is a simple graph that shows how costs per unit

Continued on page 16

interact with production volume (and profits). It will provide some insight as to how a project will fair financially before you make a more complete analysis as part of the project proposal. The vertical axis is linear and represents manufacturing cost per unit. The horizontal axis is also linear and represents the production volume expected.

The curve labeled *variable* refers to the cost per unit (because it is amortized over the life of the product) that varies with production volume. These costs include product development, marketing, and manufacturing start-up, that is, the total cost which represents the development process. In this simple model, all this cost is assumed to occur prior to production. Therefore, the cost per unit produced is simply a 1/X shaped curve.

The curve labeled *fixed* represents the cost per unit which is constant as a function of the volume produced. In this category, the manufacturing cost is the dominant item. Examples of minor fixed costs are warranty costs and sales commissions. *Total* is the sum of the two curves.

The three vertical lines, in figure 3, indicate "the good, the bad and the ugly." Products are "better" as they achieve higher volumes because then total cost is dominated by the manufacturing cost of the hardware, costs that the customer readily accepts as representing tangible value. In contrast, the variable cost represents something that customers don't care about, in that they often see this as the "cost of Tektronix doing business." Because the price is normally dominated by the fixed cost (the manufacturing cost) in most of our pricing algorithms, clearly a product will be more profitable when its overall start-up cost is lower relative to the revenue generated.

Products with very low manufacturing costs, such as application software, are difficult to price for a given percentage profit, because they are dominated by the development costs.

The model represented by figure 2 is built into several financial analysis programs that are used to estimate project profitability for product proposals.

SUMMARY

When you are in the planning stage, specify requirements.

A good plan includes only measurable results. Planners should prioritize requirements in order of their importance to customers. In addition, requirements should be specified from the customers perspective in concrete and measurable terms.

Recognize the value of tools. Treat them as a capital investment. Justify them on the basis of productivity and on the match between the cost of the tools and the magnitude of the project.

Implementation costs should be based on historical data within the company. People are generally willing to have you benefit from their experiences.

Finally, do a preliminary financial analysis to provide an approximate (and often illuminating) sneak preview of upper management's reaction when you present your proposal.

FOR MORE INFORMATION

For more information, call Jack Grimes on ext. 3558 (Wilsonville). □