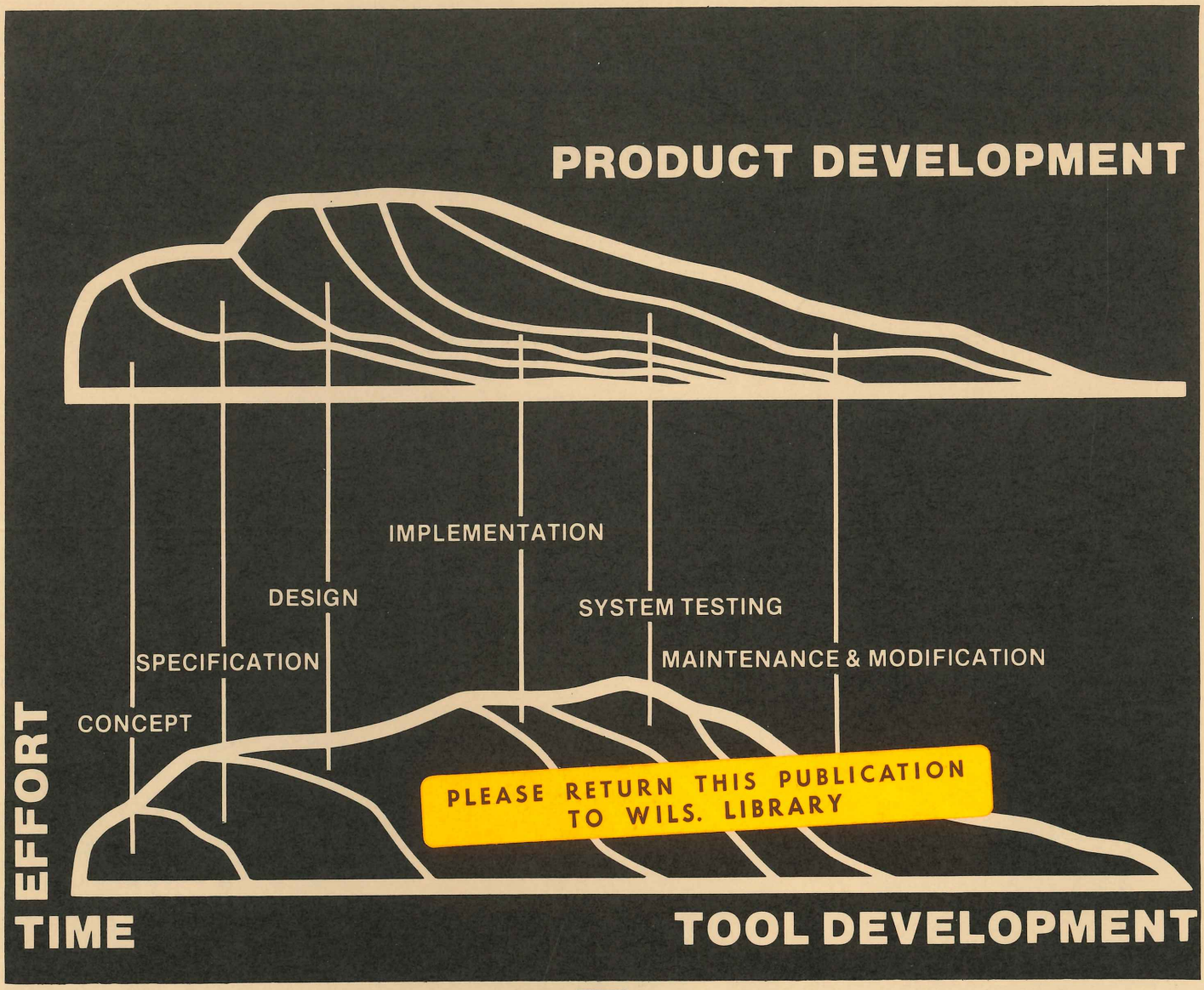
Published
August 1979

FORUM REPORT

COMPANY CONFIDENTIAL



Managing Firmware Through Its Life Cycle

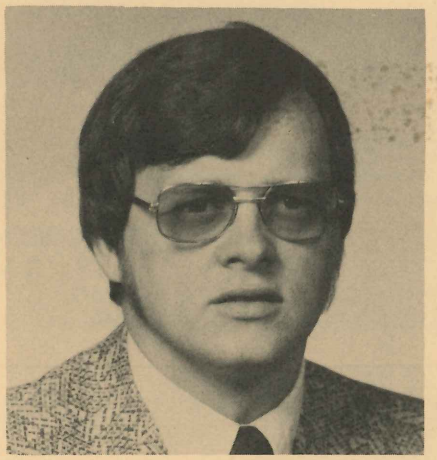
Managing Firmware Through Its Life Cycle

The Engineering Activities Council provides Tektronix engineers and scientists with a forum in which to present directly, to multiple levels of management, what engineers and scientists themselves consider important in technology.

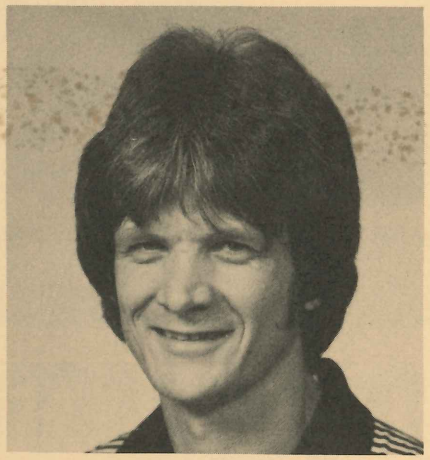
Forum 13, "Managing Firmware Through Its Life Cycle," was held in April 1979 in the Technical Center (building 50) auditorium.

Speaking at the forum were: Doug Bingham (Copier/ Plotter Product Development), Bob Edge (Graphic Computing Systems Advanced Development Firmware), Jack Grimes (Graphic Computing Systems Engineering), Rick Potter (Digital Service Instruments Software Engineering), and Rik Smoody (Graphic Computing Systems Evaluation).

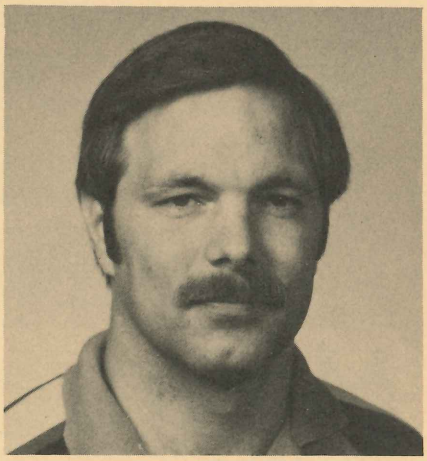
The chairpersons for Forum 13 were: Dave Armstrong (Digital Accessories Design), Lynn Saunders (Scientific Computer Center, Software Engineering), and Jim Tallman (7000 Series Electrical Engineering).



Dave Armstrong



Lynn Saunders

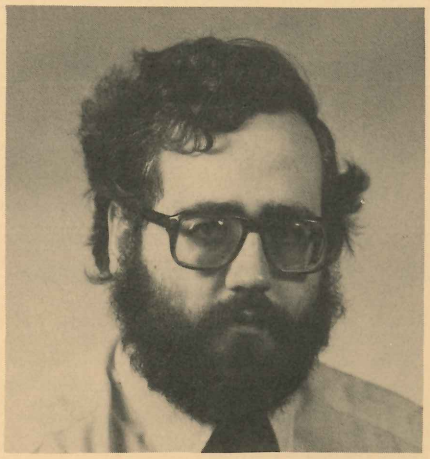


Jim Tallman

CHAIRPERSONS

HIRING AND PLACING PEOPLE...

A project in itself



Bob Edge, GCS Advanced Development Firmware, ext. 2590 (Wilsonville).

A HIRING TRAP

Hiring and placing new software people can be a major project in itself unless the hiring manager keeps a few simple guidelines in mind: allow plenty of time to hire the right people, clearly identify the skills and personalities of the people already in the group as well as the skills and personalities required for the new member, use interviews as well as resumes for a complete picture of applicants, use Professional Placement's services, and be prepared to reassign people who don't fit their jobs.

The biggest trap to avoid when building a new staff is thinking that the right people are easy to find. You may need as much as six months to find the right people. If you require several people for a large project, hire a few very early to get a feel for the current employment market.

The same hiring methods and ideas apply to hiring one person or a whole new staff. First, develop a job description for each position. A list of knowledge, skills, and abilities required will pay off many times over as you consider the candidates.

Identify the skills and personalities of existing staff members and determine what kind of new working and interpersonal relationships will benefit your group. You will need different kinds of people for team efforts and for individual assigments. Some people work well in both environments, but they are not easy to find.

THE INTERVIEW

The applicant's experience is important in terms of both amount and quality of experience. Be aware of the difference between five years' experience and one year's experience five times. With careful questioning, determine if the candidate's part in a project was larger or smaller than the candidate claims.

Nonwork-related experience can have value. For someone fresh out of school, find out what kind of projects the candidate was involved in. Someone who makes the most of educational opportunities will probably make the most of a job.

Education is important, but you must still question every candidate carefully. Candidates who have earned degrees have proved an ability to earn degrees; you must find out if candidates also learned something about their fields of study. Someone who has been out of school for several years will have a real-world performance record. Acquiring an education informally may indicate a strong personal motivation that could contribute significantly to job performance.

Personal goals are very important to an employee's long-term effectiveness. A person with goals clearly contradictory to yours and those of the project may be difficult to work with. Most people are quite flexible, however, and are usually willing to work on any project that has clearly defined goals.

GENERAL BACKGROUND

While specific experience is valuable, it can also be to your advantage to consider someone with a broader background. Such a person will be more likely to take an interest in many aspects of a job. It is to your benefit to encourage sharing information and ideas between working groups.

A person with a broad background has another major advantage. When one project is finished and you are ready to start the next, flexible people will make the transition that much simpler.

PROFESSIONAL PLACEMENT HELP

The people in Professional Placement and College Relations are glad to help with staffing problems. Professional Placement people are experienced in evaluating candidates for their personal qualities.

It is best not to decide upon a candidate until the Professional Placement interviewer's meeting. The interviewer will provide additional information and comments that can greatly affect your decision.

UPDATE KNOWLEDGE

An employee who has stopped learning is a big problem. As technology changes, an employee must continue to learn. Without a motivation to learn and grow, an employee becomes useless to you and your group.

Tektronix has an excellent education program, but it is not your only option. There are numerous organizations that offer short seminars on many subjects. They are a good way to give an employee a charge of new thinking.

Continued on page 4

Continued from page 4

UNSUITED PERSONNEL

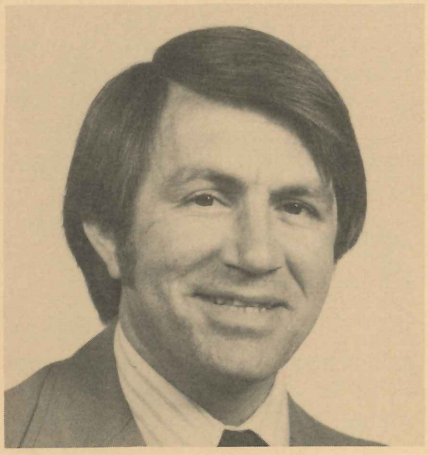
Another serious problem is an employee who does not believe in project goals. There are two things you can do. The first is to convince the person to change direction (or to change yours). This requires a great deal of coaching, but it can be done. The second choice is to find a job that fits the person's goals, perhaps in another group. Solving this problem satisfactorily is one of a manager's most important responsibilities.

KEY PERSON

It can be a serious mistake to give one key person full responsibility for a project. With the life of the project in one person's hands, the project will be seriously threatened if that person leaves. This does not mean that you should avoid strong team members, but you should balance responsibility and expertise as much as possible.

As a manager you may be required to coach team members so that they will understand all aspects of a project. Most people appreciate help with their work. If your group is organized as a team effort, this interaction will seem quite natural.□

IMPLEMENTATION



Rick Potter, Digital Service Instruments Software Engineering, ext. 1933 (Walker Road).

A software project's implementation phase can be straight-forward and a small part of the entire software job, *if* the project manager has a thorough understanding of four major aspects of software implementation: (1) using a methodology, (2) defining a realistic schedule, (3) developing code, and (4) monitoring progress.

METHODOLOGIES

Because software methodologies have recently received much publicity, many people incorrectly view methodologies as cure-alls for software problems. **Webster's Dictionary** defines **methodology** as "a body of methods, rules, and postulates employed by a discipline." More informally, software methodology is a recipe for developing software.

A methodology must be task-specific. A different methodology is required for developing scientific programs than for developing business programs. Just as either a house or a boat could be built from wood depending on the method used, different software methodologies produce different types of programs.

Software designers should select a methodology at the beginning of the project. Early selection and adoption of a methodology encourages designers to begin planning, documenting, and defining specifications early in the project cycle.

Currently Tek Labs, Information Display Division, and Digital Service Instrument's Software Engineering Group are investigating software methodologies. From these efforts, Tektronix may be able to develop a stereotyped methodology that will fit into the new product introduction process.

SCHEDULES

Several factors make developing realistic schedules difficult.

Programmer optimism: most programmers underrate the difficulty of new projects.

Management optimism: most managers overrate their software people.

Management restrictions: at times, managers set unrealistic goals (for example, "We must finish by WESCON").

Not allowing designers time to respond to criticisms presented in design reviews: most schedules include design reviews, but not time for redesign.

Creeping feature creature: adding features part way through development requires an adjustment to the schedule.

Not revising schedules: a scheduled 12-week project that slips one week after only three weeks will probably be a 16-week project, not a 13-week project.

Over Emphasis on coding: contrary to what many designers think, coding requires only about 20% of project labor.

Not identifying programmer productivity: figure 1 shows the wide range of productivity from programmer to programmer and language to language.

To overcome these problems, software project managers can make schedules more accurate by (1) collecting data about projects completed (how far off were the results from the first estimates? what factors caused the delays?); (2) defining project specifications before developing a project schedule; (3) studying books and articles that describe software design, workload estimating, and scheduling (example books are: Frederick Brooks, **Mythical Man Month**, Richard C. Gunther's **Management Methodology for Software Product Engineering**, and Philip W. Metzger's **Managing a Programming Project**); and (4)

developing schedules based on data and phase ratios derived from published information and from the manager's own experience (an example of a phase ratio is 20:10:5:3 for hours spent in analysis - and - design: implementation:testing:documentation).

CODE GENERATION

Once designers select a methodology and define a schedule, they can turn their attention to code generation. Though many designers consider code generation to be the toughest part of a software project, code generation shouldn't take more than 20% of project labor ... *if* the software design is good and *if* the project specifications are firm and clearly defined. The rest of the project should be devoted to designing, planning, debugging, and integrating the software.

TOOLS

Having appropriate tools (hardware, task-specific software, and development methodology tools) makes designers' efforts much more productive. An ideal situation at Tek would be for the programmers to use the Scientific Computer Center's Cyber system for number crunching and for high throughput tasks and smaller computers for emulation and text editing. Task-specific tools include high-level languages, linkers, and loaders. Designers should tailor their development methodology to their task.

MONITORING PROGRESS

A software manager is responsible for monitoring the progress of the project. Like the gentlemen in figure 2, managers should not rely on miracles to complete projects on time, but should instead use the tools available to them.

PERT (network analysis) charts show the interactions between various parts of a task. **GANTT** charts depict individual assignments throughout the project. Along with GANTT charts, managers can set development milestones at two to four-week intervals and ask designers to write milestone reports.

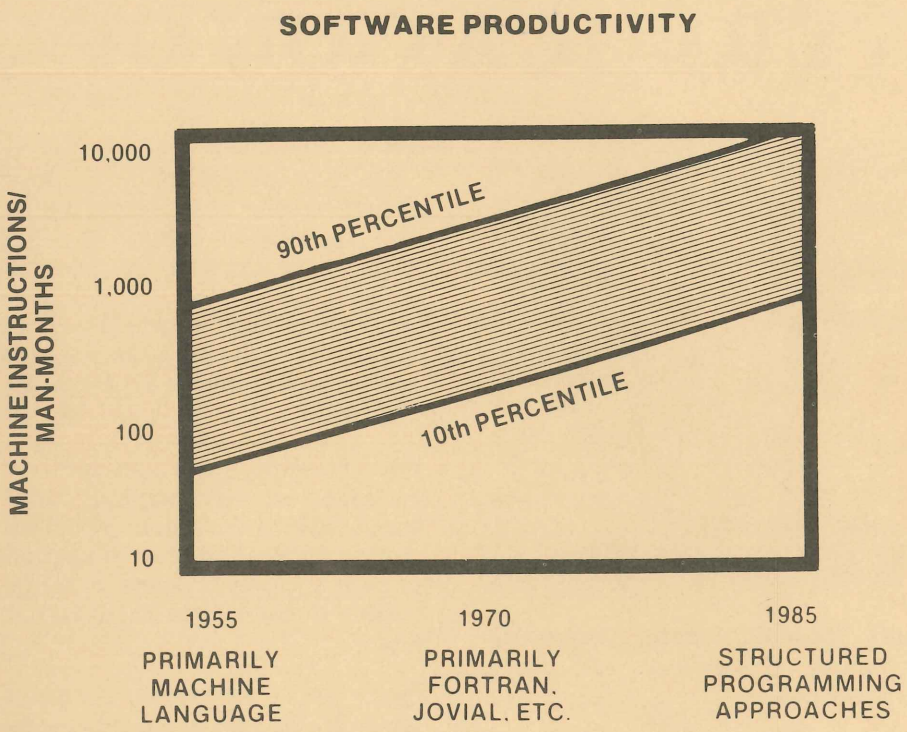


Figure 1. Although the average productivity of programmers has increased over the last twenty-five years, the continuing wide range of efficiency makes accurate project scheduling difficult without concrete knowledge of how productive a particular group's programmers are.

"I THINK YOU SHOULD BE MORE EXPLICIT HERE IN STEP TWO."

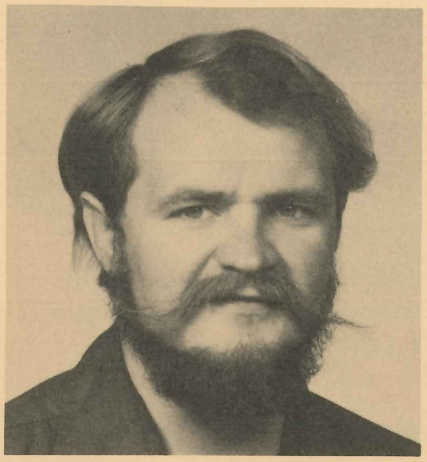
Figure 2. Managers shouldn't rely on miracles to complete projects on time, but should use monitoring tools such as PERT and GANTT charts and development milestones.

RIO (Responsibilities, Indicators, and Objectives) documents and individual work plans are management-by-objective tools that establish measurable criteria for evaluating each assigned task.

SUMMARY

To keep software implementation straight-forward and a small part of the overall job, managers must develop firm specifications, task-specific methodologies, realistic schedules, and methods for monitoring progress. □

VERIFICATION OF SOFTWARE



Rik Smoody, Graphic Computing Systems Evaluation, ext. 2422 (Wilsonville).

We all want to create a quality product. Evaluation is a vital part of producing effective, reliable software; evaluation is *not* an adversary to design. The evaluator's job is helping the designer find bugs, but that shouldn't be threatening to the designer. Look at it this way: each of us makes mistakes, each of us knows it, and everyone knows everyone else does too. The question to ask is: do you want an evaluator to find mistakes now, or have customers find them later?

Errors are less expensive to correct in early phases of product development: the software is fresh in the designer's mind, and only a few people need worry about an error. In later design stages, attempts to retrofit a software system to correct a bug may introduce more serious bugs.

A mistake designed into a product, but not corrected, must be lived with. People use it as a feature and you must maintain upward compatibility. At Tektronix, our software should reflect our commitment to excellence.

BLACK-BOX TESTING

Black-box testing of software by an independent verification team is often intuitively appealing to marketing people. Snappy demo programs show what the box can do, but meaningful testing is made difficult by the lack of access to internal operations. A simple real-number sorter or a comparator, for example, cannot be tested exhaustively with a black-box approach.

For a customer using a product in a configuration not anticipated by the tester, there is no way to guarantee quality. Since the internal operation of such a system is unknown to the tester, the customer pushes unknown limits.

Only an experienced computer scientist can understand a complex system. A computer scientist quickly tires of black-box test programs, because they are trivial and boring, and soon seeks other duties. This migration of computer scientists has been a major problem at Wilsonville. Even junior programmers know there has to be a better way to test, but they often don't know how, or lack the authority, to change things.

Testing is especially difficult when testers aren't sure which parts of the system they are testing. Testers may evaluate some parts but not even touch other parts. This approach can amount to nothing more than monkey-testing. Many monkeys and many typewriters still take a long time to produce the complete works of William Shakespeare.

EVALUATION AS PART OF THE DESIGN EFFORT

Communication between designers and evaluators is essential if an evaluation team is to avoid playing a guessing game. An evaluation team provides expertise in testing methodology; evaluators are the testing gurus of the project and can develop new tools.

As evaluation becomes part of the design process, reliability and testability become design factors. Evaluators can thoroughly test modules as they are built, while functions and design criteria are still fresh in the designer's mind. Thus, designers can fix errors more easily. The evaluator can generate test data while the designer can still remember possible trouble spots and can recognize correct answers. The evaluator can run each module through its paces to see if it performs as expected and to make sure all code is tested.

Evaluators may find their jobs more enjoyable and certainly more educational if they work closely with designers. For many people, evaluation is an entry-level position; they would like to be designers' apprentices. Evaluation groups can provide a training ground, but should never be holding-pens.

AUTOMATED TEST TOOLS

Commercially-available testing tools are usually not adequate for Tektronix software evaluation. They are usually limited to one language, such as FORTRAN or COBOL, or one assembler. What if we don't happen to be programming in that language?

Some testing tools are trivial. Other tools are limited by lack of structure, or they do a lot of work to compensate for an inadequate compiler. For example, a tool may test for definition of variables. Better languages, such as PASCAL, do not allow undeclared variables.

Some tools may be so general that specific constraints are lost. For example, if a program never uses Labelled Common, it is wasteful to use a test system which requires a lot of cumbersome code to see that there are no Labelled Common errors.

Commercially-available tools are often cumbersome for the small amount of useful work they do. They are not worth what they cost.

In the literature and at conferences, we discover tools used in other groups. They are often tailored to

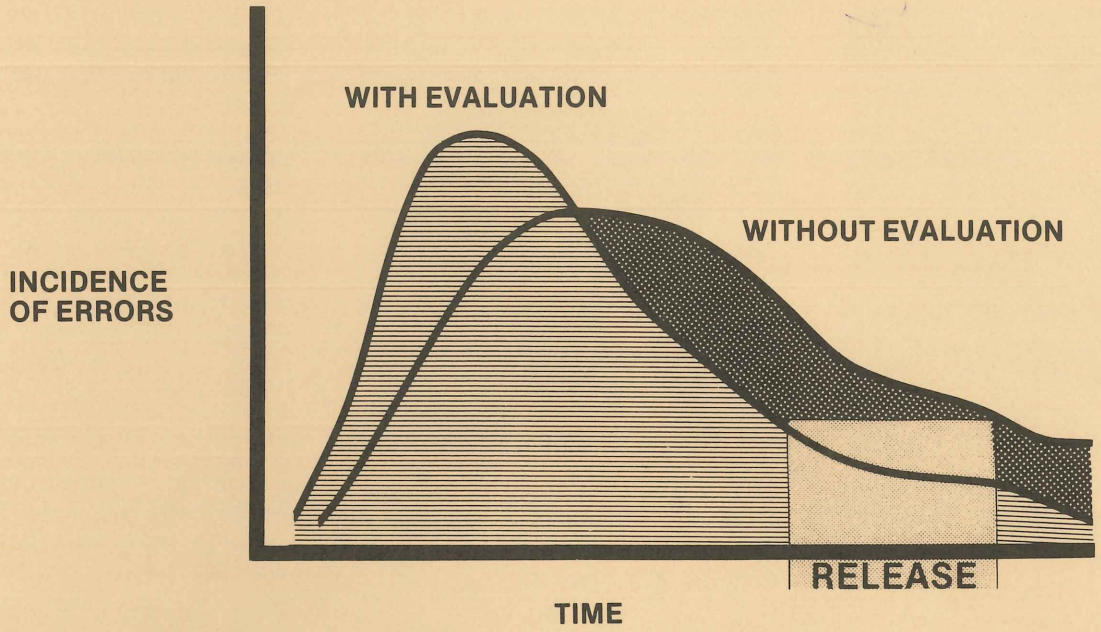


Figure 1. Software evaluation enables project designers to release a product earlier, or release a product having fewer errors, or both.

those groups' applications and don't fit our needs. Even if competitors do have tools which we could use, they are unlikely to sell them. Since efficient testing tools are not readily available outside Tektronix, we may have to develop our own tools. This can be done most effectively by maintaining a staff of tool-building experts. TSL, the Tina System Language, and RAID, a debugging aid used with TSL, are two examples of Tek-built tools. They are a significant step towards reducing bugs in software designs. Tektronix needs to develop more tools for quality assurance.

Effective tool use requires planning. Test tools deserve the same kind of consideration that other development tools, such as compilers and simulators, receive. Test tools should be defined at the early stages of development, when everyone has planning fever. At the same time, the evaluation strategy must be developed. Parallel to other design efforts, evaluators should define what tests a program must pass to be considered "fit".

No tool can solve all problems, nor can it be state-of-the-art forever. Consider the advancements in programming languages in the last few years. Test tools are only a little way behind, and are constantly advancing.

SOME TOOLS

Well-designed **high-level system languages** help make high-powered tools practical and help designers write programs more quickly and more legibly. In that sense, the language itself is a very important tool.

A **structured walkthrough** can help spot many errors in early design stages. In walkthroughs, a few people, other than the designer, read the code and documentation and try to understand how the program works. They also point out parts of a design they believe may not work. High-level languages and effective coding standards help make this a more productive practice. Evaluators on a new computer terminal project in our area are performing walkthroughs as a service to the designers.

A **control flow analyzer** helps determine if there are any funny program jumps. A **data flow analyzer** points out if a routine is a Peeping Tom, a burglar, or an exhibitionist. Such program behavior may be intentional (if so, it should be documented in the source code), but it should be watched as a potential trouble spot.

Coverage testing is testing *all* code in a program. This alone will not ensure quality, but it is a necessary step in that direction. Tek-built RAID can keep track of whether there are untested pieces.

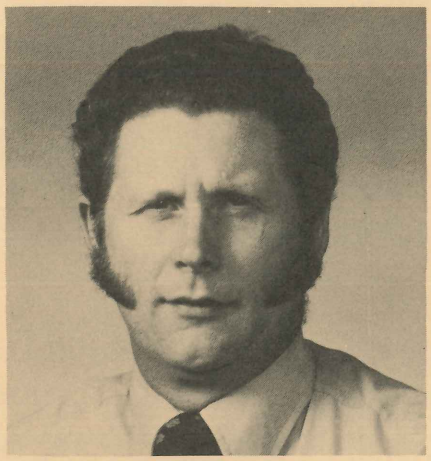
Test set generators may either: (1) look directly at the source code and define a set of data which will test the module, or (2) be given a shorthand description of the test data and generate the entire set. The first kind is very hard to build. The second kind is common.

Test fixtures are driver routines which feed data to the module in question. Results are then formatted or saved.

Automatic log book keepers and **report generators** help manage software design projects.

CONCLUSION

Evaluators must work closely with designers from the very beginning of a project. Testing should be well planned and methodical so that testing can be automated wherever possible. □



Doug Bingham (Copier/Plotter Product Development) discussed firmware production and maintenance.

Jack Grimes (Graphic Computing Systems Engineering) discussed firmware in the design stage of firmware development.

To add your name to the **Forum Report** mailing list, or change your delivery station, fill out the coupon inside.

***Forum Reports** can not be WOW'd. However, unused **Forum Reports** may be sent along with other non-WOWable paper, in large quantity, to Material Evaluation at D.S. 71-474. Non-WOWable paper should be marked "SALVAGE."*

Managing editor: Burgess Laughlin, ext. 6795, d.s. 19-313. Compiled and edited by the Technical Publications Department for the benefit of the Tektronix engineering, software, and scientific community in Beaverton, Grass Valley, Walker Road, and Wilsonville.

Cover and Graphic Design: Joan Metcalf. Graphic Assistance: Jackie Miner. Typesetting: Jean Bunker.

MAILING LIST COUPON

- ☐ Forum Reports
- ☐ ADD
- ☐ REMOVE
- ☐ CHANGE

Name: _____

_____

Old Delivery Station: _____

New Delivery Station: _____

Payroll Code: _____

(Required for the data processing computer that maintains the mailing list)

Not available to field offices. Allow four weeks for change.

MAIL COUPON TO: 19-313

COMPANY CONFIDENTIAL

Forum Reports Not Available To Field Offices