RELIABILITY CONSIDERATIONS FOR COMPUTER-ASSISTED CARTOGRAPHIC PRODUCTION SYSTEMS

D. L. Pendleton
National Ocean Survey
National Oceanic And Atmospheric Administration
6001 Executive Blvd.
Rockville, MD 20852

ABSTRACT

The development of computer-based cartographic production systems is examined from the standpoint of hardware and software reliability factors. The inherent complexity in the design of non-trivial systems is examined and specific techniques from the fields of system engineering and computer science are described. The basis for new hardware systems which incorporate multiple procesors, memories, controllers, mirrored disk files, and fail-soft operating systems is explored. Finally, a system development project team concept is summarized, having the objective of ensuring that more reliable software is built by using structured top-down software engineering techniques.

INTRODUCTION

The field of automated cartography is developing and advancing at a rapid rate. Much of this development in the past has been restricted to systems geared to the research environment, as one would expect during a formative period. The field is currently passing through this phase into one that includes both large and small systems planned for volume production work. Production systems, in general, differ from research tools in several major ways. One of the most important of these is the requirement to meet an established schedule on a regular basis. This paper addresses two of the major factors, hardware and software, whose reliability should be considered in the development of such production systems to insure that the negative impacts of system downtime on production schedules are minimized. The goal is to increase the cartographic community's awareness of the state-of-the-art so that users can improve the reliability of their systems by influencing the designs of in-house and commercially available systems.

COMPLEXITY AND LEVELS OF AUTOMATION

The impacts of unscheduled computer system downtime on operations differ depending upon the kind and degree of automation present in the production sequence. For purpose of analysis, an assumption is made that there exists a threshold value for automation support beyond which production operations would be completely dependent upon a properly functioning hardware/software system to meet established production deadlines. That is, in the event of a major system failure, a retreat to previous manual methods would not be possible since the system would be performing complex operations no longer feasible to perform manually. Suffice it to say that the automated system would exist in order to: (1) meet the production requirement with fewer resources, (2) handle a greater workload, (3) produce higher quality products, or (4) meet more stringent deadlines than the manual system it replaced. Given this assumption, the goal of designing significantly automated production systems revolves around questions relating to the major components of hardware, software, the increased complexity of automation required, a myriad of other ancillary factors that comprise a functioning system, and the collective impacts of these elements upon the reliability of the total system.

Figure 1 illustrates the growth of system complexity as the level of automation is increased for a system design. After a level of automation is reached that provides significant computer assistance, the complexity begins to grow exponentially.

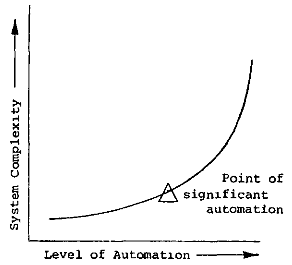


FIGURE 1. Increase in complexity with increase in automation.

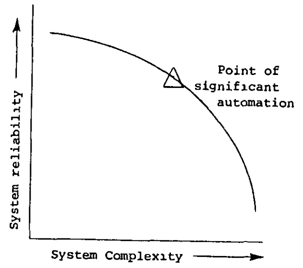


FIGURE 2. Decrease in reliability with increasing complexity.

The dramatic rise in system complexity at the point of exponential growth is due to the fact that the number of required hardware and software system components, and their interfaces, begin to multiply if the system performs any sophisticated functions. As it happens, it is the sophisticated functions in any system that make it really useful and cost effective. It is not surprising, therefore, that when a significantly automated feature is installed in an existing system, a dramatic fall-off in total system reliability is experienced. This effect is shown in Figure 2. It appears that the fundamental problem in implementing and using automated production systems with significantly useful features is one of managing this exponentially growing factor of complexity (Walker, 1978).

SYSTEM AVAILABILITY

The reliability of an item is the probability that it will be able to correctly perform a required function over a stated period of time (Arsenault, 1980). The reliability of a system's hardware and software components can be quite good and the system may operate as planned for extensive periods. However, the additional effects of the time required to bring the system back up when it does fail has to be considered. Production systems must perform to a predictable level on a continuing basis with system failures and repairs taken into account. A measure of the total effects of these factors is the system availability, or the probability that the system will be available to perform the required work at the requested time. This can be defined as:

$$A = \frac{\text{UP TIME}}{\text{UP TIME + DOWN TIME}} = \frac{\text{MTBF}}{\text{MTBF + MTTR}} \qquad (1)$$

where MTBF is the "mean time between failures," and MTTR is the "mean time to repair."

The availability of a system with nonredundant components can be computed as the product of the availability of each of the system components A_i:

$$A(\text{System}) = \prod (A_i) \qquad (2)$$

When the availabilities of all components are identical, then

$$A(\text{System}) = (A_i)^n \qquad (3)$$

The important point summarized here is that the total system availability will be less than the availabilities of the individual components, as shown in Figure 3. The problem then, is to find ways of increasing the system availability for hardware and software components in the face of increasingly complex designs.

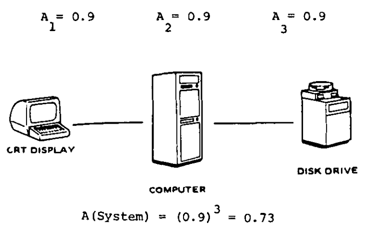


Figure 3. System Availability Less Than That Of Components

A SYSTEMS ENGINEERING APPROACH

Having established the need to take levels of automation, complexity, reliability, and system availability into account in the development of cartographic systems, the next step is to identify existing techniques that have already proven effective elsewhere. The field of systems engineering has a well developed body of knowledge and specific techniques for developing systems throughout a life cycle process (Hall, 1962). All of these techniques are ultimately directed to the specification of hardware, software, facilities, personnel skills, training, and procedures needed to meet customer requirements within predetermined reliability criteria.

One postulate of this paper is the view that these existing techniques should be transferred and applied to the development of cartographic systems in a formal way, instead of reinventing the wheel on a trial-and-error basis. Some of the particular kinds of systems engineering techniques that could contribute to more reliable cartographic systems include those of precise system specification, technical reviews, configuration management, maintainability and reliability, quality assurance, human factors engineering, software engineering, software verification and validation, and production management. The remainder of this paper will focus upon the techniques for systematically increasing a cartographic system's hardware and software availability in the face of increasing design complexity for significant levels of automation.

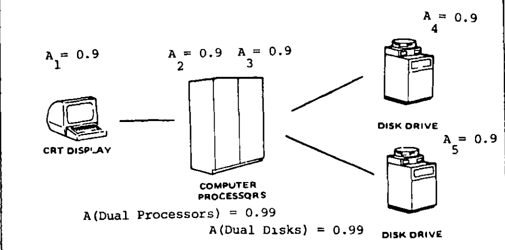
HARDWARE TECHNIQUES

Until around 1975, the generally accepted method of increasing the availability of a computer was to provide a duplicate system (Champine, 1978). This "stand-by" technique was common, especially for batch processing systems, in which punched cards were the primary method of data entry and/or user interface. This method was expensive and cumbersome and not very effective. It required the maintenance of duplicate data bases, duplicate software libraries, and, in some cases, additional staffing. In addition, where the back-up and primary computers differed (such as different manufacturers or different operating systems) the staff was required to be intimately familiar with more than one system. This resulted in a duplication of effort and decreased overall productivity. At about the same time, the methods of using the computer had also begun to change.

During this period applications evolved from the batch (punched card) oriented mode to an on-line environment. System users began to interact directly with hardware and software through terminals rather than key punch machines. Some users began to use timesharing on large mainframe computers, while others acquired dedicated minicomputers and integrated them into their system designs. Powerful microcomputers are now available to perform most if not all of the functions for which minicomputers were recently used at a fraction of the cost. Similarly, the supermini's of today are displacing mainframes for many applications.

These changes have had a major impact upon the way users have come to view their work. System failure in a batch operation has little impact on its users if it can be corrected in a few hours; but system failure in an interactive or real-time environment has a catastrophic impact and system failures once taken in stride are now considered intolerable. The upshot of this evolution to on-line operations using mini and microcomputers is to place far greater demands upon system design regarding reliability and the need to exploit the use of new technology to increase overall system availability.

The primary method used to increase system availability through hardware design is by the use of redundant hardware components (Katzman, 1977). Figure 4 shows the effect of using redundant processors and disk subsystems to achieve a system availability greater than those of the individual parallel components and the system availability of Figure 3.



$$A(\text{System}) = (0.9)(1.0 - (1.0 - 0.9)^2)(1.0 - (1.0 - 0.9)^2) = 0.88$$

Figure 4. Parallel Redundancy Increases Availability

This effect is due to the definition of availability for redundant parallel elements:

$$A(\text{System}) = 1 - \prod(1 - A_i) \qquad (4)$$

When the availabilities of all elements are identical, then

$$A(\text{System}) = 1 - (1 - A)^n \qquad (5)$$

This discussion of hardware redundancy is, of course, not new (Arsenault, 1980). The purpose is to highlight the basis for techniques currently being employed by a number of computer hardware manufacturers to build commercially available "fail-safe" systems. By trading off the competing factors of reliability, complexity, performance, maintenance, and cost, designers are able to produce systems with extremely high system availability values (Katzman, 1977). The point to seriously consider, both by in-house system implementors as well as vendors of cartographic systems, is that effective hardware redundancy techniques are available for use in production system designs.

NEW TECHNOLOGY

Special-purpose, one-of-a-kind, and extremely expensive hardware/software systems incorporating ultra high reliability features have been in use for several years in military and space applications. Much of this technology is becoming available to designers of commercial and scientific systems as low cost off-the-shelf items. The hardware includes the use of redundant minicomputers, microprocessors, and integrated operating systems support for automatic reconfiguration of system components in the event of component failure. A major benefit of the architecture of these computing systems is the automatic data file recovery features which protect the integrity of the data base as components fail (Bartlett, 1978).

One manufacturer that has led the field in this area is the Tandem Company of Cupertino, California (Bartlett, 1978). Tandem has had systems in the field for about seven years now and has proven technology. The success of this company appears to have started a trend with several new companies specializing in this type of system. Stratus, Inc. of Natick, Massachusetts, and, August Systems, of Salem, Oregon have announced similar fault-tolerant products (Boggs, 1981). Even the more established firms are beginning to orient their products to fault-tolerant designs. IBM has announced the development of fail-safe features for its processors with similar redundancy characteristics to the Tandem. Briefly, these manufactures use a combination of redundant hardware, with integrated software support, to eliminate system failure due to the failure of any single major component, such as the central processor, central memory, peripheral controllers, and disk drives. Unlike stand-by mode designs, these systems fully utilize all resources, such as the integrated modular processors and shared memory. Therefore, the effective system capacity is increased and available for primary operations. The replacement of failed components can even be performed with the system running. Data base integrity is protected during disk or controller failure by "mirroring," or automatically updating copies of, files. Access latency for data retrievals is reduced by the system use of the mirrored files for data base operations.

It appears that the trend toward commercially available fault-tolerant systems is a major development in the computer industry. It is not too early for designers of digital cartographic systems to make their needs known and incorporate this class of hardware into production system designs.

SOFTWARE TECHNIQUES

One of the major pitfalls inherent in the planning and implementation of computer-based systems is the tendency to overlook or underestimate the software problem. Hardware is tangible and can be seen, touched, moved about and is associated with clear and complete specificiations. If a hardware component malfunctions, it is a relatively straightforward process to trace the problem, isolate the malfunctioning elements, and repair the equipment. As demonstrated above, there exist concrete analytical techniques for predicting the reliability of hardware elements that can be applied to the design process.

Unfortunately, this is not the case for software which is almost never associated with clean and complete specifications. In many ways a system description is the tip of the iceberg, with hardware components being clearly visable and the much larger and more elusive software elements concealed beneath the surface. The issue of software correctness and reliability has emerged as a serious problem only within the last decade. Efforts have been made to develop analytical techniques for application to software reliability without much practical success; and unlike hardware, the use of redundancy is not an effective way to improve software reliability. The two main approaches that have been pursued relate to either formal proofs of program correctness, in the sense of mathematical theorem proving using artificial intelligence techniques, or quality assurance, through the use of rigorous and systematic software verification and validation procedures.

SOFTWARE ENGINEERING

The techniques that are emerging as being of practical importance in attacking the software problem have evolved from the systematic testing approach, but relate to even more fundamental issues than testing alone. It has been estimated that only about one-half of all software errors are due to programming mistakes (Champine, 1978). The remaining errors are due to inadequate specification of the system requirements and noise introduced in technical communications between the individuals performing the steps in the software development process. The entire range of software development activity is revamped and made more precise and systematic under the discipline of software engineering (Mills, 1980). This relatively new approach emphasizes specific structured techniques for controlling the technical communications process and the complexity inherent throughout the software development steps of analysis, specification, design, coding, testing, implementation, and documentation. The buzz words for these tools are composite (modular) design and top-down structured programming. This approach is proving effective because the approach to containing the fundamental software problem consists of the management and control of exponentially increasing complexity.

SOFTWARE DEVELOPMENT METHODOLOGY AND RELIABILITY

The software engineering approach contrasts sharply with the historical method where most software is generated when an enthusiastic programmer or other technical individual quickly dashes off a few lines of code in a flash of inspiration and then expands it into a complete program as more ideas appear. This kind of effort results in a patchwork of code which performs many functions using an unnecessarily complex logic flow. It is nearly impossible to debug the program completely and only the author is able to understand it. For this reason it is very difficult to maintain through normal software changes and the unreliability grows with each modification. The decision is made through default to try to eliminate the ensuing software bugs with a long and difficult period of testing, rather than using a brief but valuable initial period to systematically "design out" most errors before coding is started. Under these circumstances, it is impossible to test for all the errors generated. Even when development is carefully controlled, there is an early point of diminishing return where it is a waste of time to continue further testing due to the large number of parameters involved, the inherent complexity of the code, and the difficulty to simulate the "live data" conditions of an operational environment. For this reason, it has been said that, like wine, software improves with age (Champine, 1978).

BEYOND SOFTWARE ENGINEERING

Even the use of structured software engineering techniques, such as top-down design, stepwise refinement, and structured programming, appear to suffer from certain shortcomings. Claims for their effectiveness in increasing software quality and reliability are sometimes inconsistent among different organizations. It has been suggested that these techniques are good, but are not sufficient and that the organizational structure of the project has at least as great an impact upon the quality of software generated as does the programming techniques employed (Walker, 1978).

This factor of organizational structure for software development (the team approach) has been found to be the single most important new method of reducing the remaining 50% of software errors not due to coding mistakes. The team structure approach is a direct attack on the increasing complexity problem by formalizing precise communication between the major tasks of transforming objectives into system requirements, requirements into a design, and the design into accurate software code. Each successive transformation decomposes subsequent development activities into deeper levels of detail revealing increased complexity. The team organizational structure, if properly established, increases the effectiveness of the necessarily detailed technical communication between team members and reduces the information loss and introduction of extraneous noise due to the transformation process. The number of detailed relationships requiring accurate transformation, and the level of detail in the communications required, grows in proportion to increasing system complexity (Walker, 1978).

Since most efforts are too large for one individual to accomplish, the team organizational structure becomes an operating model of the software development process and is a major tool for controlling communications among members and ensuring a disciplined approach to software development. Even if nothing else is achieved, however, a properly structured team organization will ensure that (1) needed communication will occur, (2) some degree of design activity and problem solution will be used before software coding begins, and (3) basic documentation will be produced. Figure 5 depicts a software development team structure that has been proposed for use in the Office of Aeronautical Charting and Cartography, National Ocean Survey for all system development efforts.

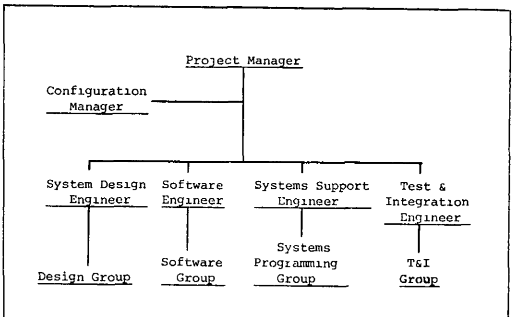


Figure 5. Software Development Team Structure

THE SOFTWARE DEVELOPMENT TEAM

The structure depicted in Figure 5 is a variation on the chief programmer team concept introduced by Mills and modified by Tausworthe (Tausworthe, 1979).

The project manager functions as the lead technical authority, or "chief programmer," in addition to required project management duties. Not shown on the chart is a higher level manager who performs most of the traditional project management functions concurrently for several of these projects.

The configuration manager assists the project manager to insure that all activities are performed under the disciplined software engineering approach outlined in the official standards and procedures handbook. This handbook is an integral component of the project team concept and serves as a basic standards reference for all team members and all projects. The configuration manager also performs the traditional configuration control function for the team.

The system design engineer assists the project manager in all phases of the requirements definition, analysis, and system design activities. The system design is produced using top-down structured techniques and design walkthroughs. Specific deliverables are produced out of these tasks and are placed under immediate configuration control after acceptance by management.

The software engineer is responsible for designing and implementing the software modules using structured programming techniques. As modules are added to the program library, they are placed under configuration control. Subsequent changes to these modules can then only be made through the configuration management process which requires project manager approval.

The test and integration engineer is responsible for the development and execution of a comprehensive system test plan in parallel with the design and programming efforts. As modules become available for testing, they are exercised against the plan and deficiencies noted. The test engineer does not debug the module, but returns it to the software development library under configuration control so that the programming group can correct the identified deficiencies.

The systems support engineer performs the typical systems programming function and insures that the operating system, data base management system, compilers, graphics packages, etc. are properly installed and operating. Assistance is provided to other team members for problems experienced with the system hardware and vendor-supplied software.

CONCLUSION

Rapid advances are being made in the area of systems reliability that designers of cartographic systems should exploit. These developments are occuring in two primary areas: (1) new ultra-reliable hardware systems incorporating built-in component redundancy using microprocessors and minicomputers, and (2) advances in the way reliable software can be developed using top-down structured techniques and software development team concepts. The field of cartographic automation has reached the point of development that requires the serious consideration of these new techniques. Without the use of these or similar methods, the design and development of truely effective cartographic production systems could be significantly retarded.

REFERENCES

Arsenault, J. E. 1980, ed. Reliability and Maintainability of Electronic Systems, Computer Science Press, Rockville, Maryland.

Bartlett, J. F. 1978, A "Nonstop" Operating System:Hawaii International Conference of System Sciences.

Boggs, D. W. 1981, Fault Tolerant Computer Enhances Control System Reliability: Control Engineering, Vol. 28, NO. 10, pp 129-132.

Champine, G. A. 1978, What Makes A System Reliable: Datamation, Vol. 24, NO. 9, pp 194-206.

Hall, A. D. 1962, A Methodology For Systems Engineering, Van Nostrand Reinhold, New York.

Katzman, J. A. 1977, System Architecture for Nonstop Computing:Compcon, Feb. pp. 77-80.

Mills, H. D. 1980, Principles of Software Engineering: IBM Systems Journal, Vol. 19, NO. 4, pp 414-420.

Tausworthe, R. C. 1979, Standardized Development of Computer Software, Prentice Hall, Englewood Cliffs, New Jersey.

Walker, M. G. 1978, A Theory For Software Reliability: Datamation, Vol. 24, NO. 9, pp 211-214.