111174 N95-17589

31994 P.8

CONFIGURABLE TECHNOLOGY DEVELOPMENT FOR REUSABLE CONTROL AND MONITOR GROUND SYSTEMS

David R. Uhrlaub
McDonnell Douglas Space and Defense Systems
Dept. F166, PO Box 21233
Kennedy Space Center, Florida 32815
Internet: dru@grumpy.ksc.nasa.gov

ABSTRACT

The control monitor unit (CMU) uses configurable software technology for real-time mission command and control, telemetry processing, simulation, data acquisition, data archiving, and ground operations automation. The base technology is currently planned for the following control and monitor systems: portable Space Station checkout systems; ecological life support system; Space Station logistics carrier system; and the ground system of the Delta Clipper (SX-2) in the Single-Stage Rocket Technology program.

The CMU makes extensive use of commercial technology to increase capability and reduce development and life-cycle costs. The concepts and technology are being developed by McDonnell Douglas Space and Defense Systems for the Real-Time Systems Laboratory at NASA's Kennedy Space Center under the Payload Ground Operations Contract. A second function of the Real-Time Systems Laboratory is development and utilization of advanced software development practices.

INTRODUCTION

The control monitor unit (CMU) automates a wide variety of ground operations at moderate cost utilizing standard software components and appropriate hardware. Users can further automate and customize the CMU through programming languages such as C, UNIX shell scripts, defining measurement triggered logic, defining derived measurements, and creating custom graphical displays. The system can be further customized and extended by using the CMU kernel programming interface (KPI) and C.

CMU technology can operate on UNIX-based notebook computers, desktop computers, single-board VME computers, symmetric multiprocessor systems, and distributed systems using real-time shared memory networks, ethernet, or FDDI-based networks. Fault tolerant configurations are possible with minor software enhancements. Configuring CMU system-level software is similar to configuring operating systems by editing text files. Multiple configurations are defined for a multipurpose system or a single configuration for a special purpose system.

CMU software technology is being developed on Digital Equipment's Alpha AXP computers running OSF/1 that conforms to IEEE POSIX standard and real-time application programming interfaces 1003.1 and 1003.4. The OSF/1 user interface supports the X/Motif standard. Application-specific displays are created with SL-GMS, an X windows-based graphical display editor that provides dynamic real-time graphics driven by measurement values. Mission and test data definitions are stored in an Oracle database that supports real-time additions and modifications of measurement definitions.

All data is archived and may be retrieved and analyzed in real-time using DADiSP, a graphical spreadsheet for scientific data analysis. All operation and user guide information is

PRECEDING PAGE BLANK NOT FILMED

1176

maintained in an integrated on-line documentation system with graphics and hypertext facilities provided by the Interleaf Worldview environment.

Performance testing of CMU shows that configurations supporting 10,000 to several million measurements per second (mps) are practical. A two processor DEC Alpha AXP 2100/500 has benchmarked at 170,000 mps. Data acquisition interfaces planned include MIL-STD-1553B, PCM telemetry, IEEE-488, analog, discrete, and serial I/O.

SYSTEM CONFIGURATIONS

A CMU system may be configured in a variety of ways. Two main types of configurations are off-line and real-time. The off-line configurations are standard office-based computers that can define a database, simulate data acquisition, retrieve, display, and analyze data and print it. Simulated data acquisition substitutes for actual hardware data acquisition to provide an environment for developing custom software without utilizing actual end-item hardware. An off-line configuration is a DEC Alpha AXP notebook or desktop workstation, as shown in Figure 1.

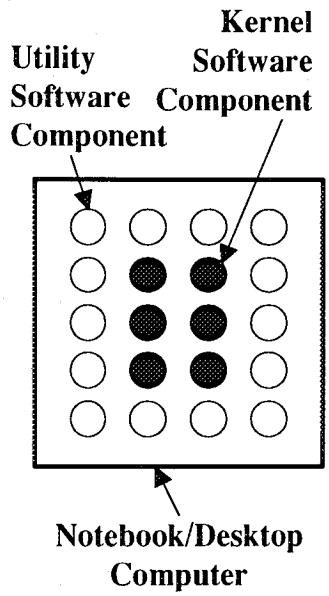

Figure 1. Off-line Configuration

Real-time configurations support data acquisition, end-item commands, and data processing. These configurations may be as small as a single-board computer or portable or desktop computers supporting only a few telemetry or MIL-STD-1553B interfaces. Larger configurations with symmetric multiprocessors and large archive storage devices are configured for handling significant amounts of data for extended periods of time from multiple data acquisition interfaces. Commercial equipment is used to configure fault-tolerant systems. An embedded system using single-board computer technology is another possible configuration, as shown in Figure 2. For large systems where data input and output must be physically distributed, the CMU is configured with ethernet, FDDI, or shared memory networks, as shown in Figure 3. Other configurations currently being developed include portable and mobile weather-proof systems for field use. System configuration is accomplished by modifying one or more ASCII files. Changes to the hardware configuration does not require corresponding software changes.

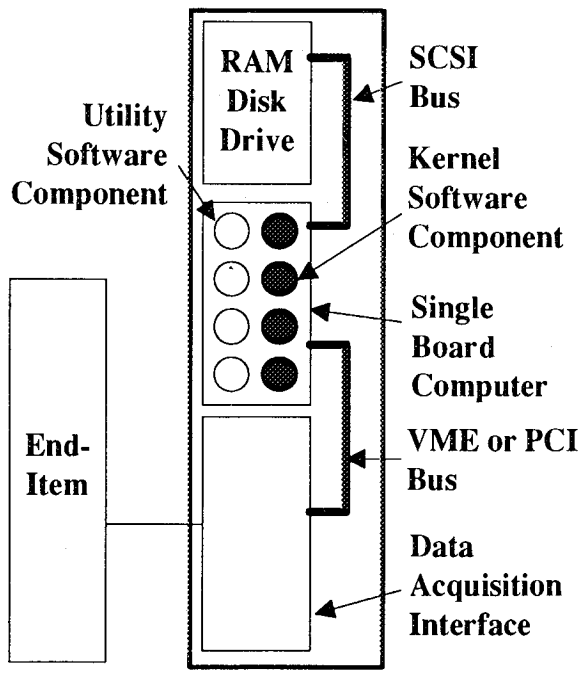

Figure 2. Embedded Configuration

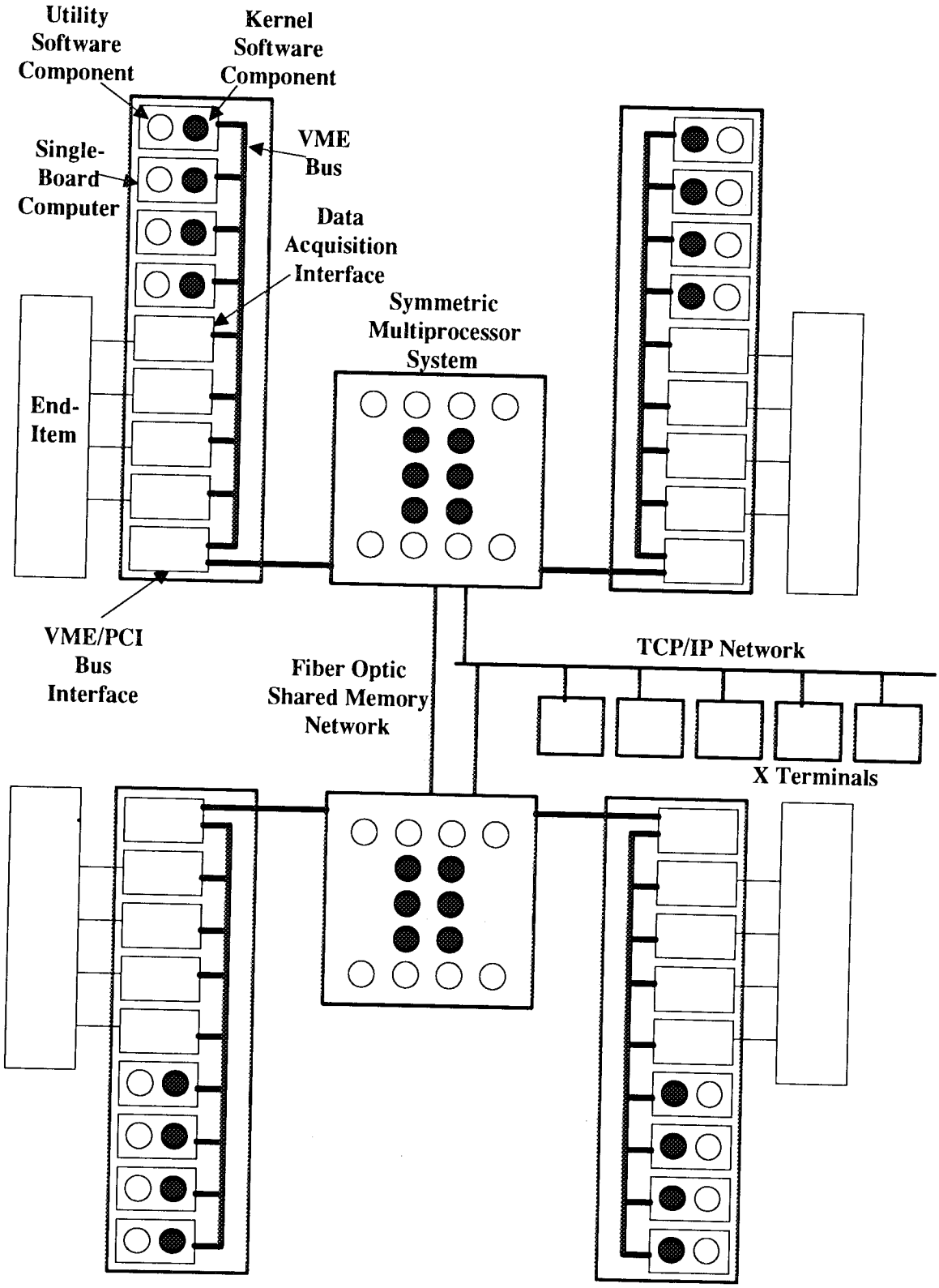

Figure 3. Large Distributed Configuration

SOFTWARE CONFIGURATION

The CMU software architecture has four main elements: the kernel, utilities, kernel programming interface (KPI), and channels as shown in Figure 4. The kernel contains most of the common system-independent functions, while utility software components are more system-specific. The kernel is composed of several UNIX processes. Utilities are generally a single UNIX process. Hardware utilities are unique hardware data acquisition interfaces; user interface utilities provide common and custom graphical displays; and data processing utilities interface to external systems and custom processing functions. The KPI is a high-level interface for developing utilities that communicate with the kernel. All communication between the kernel and utilities

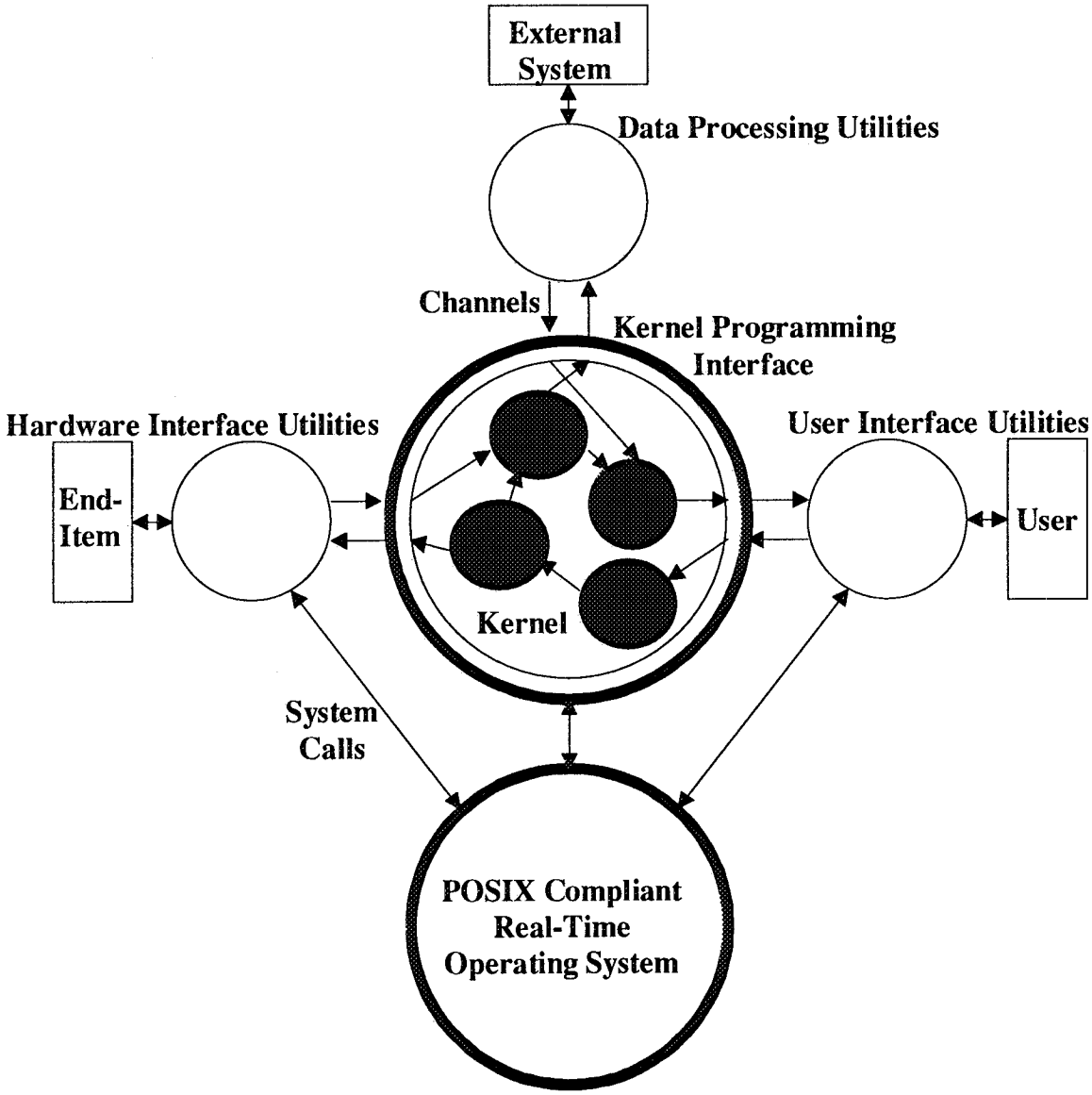

Figure 4. Software Architecture

occur through channels. The four CMU architectural elements combine to provide software functional, architectural, and performance configuration capability.

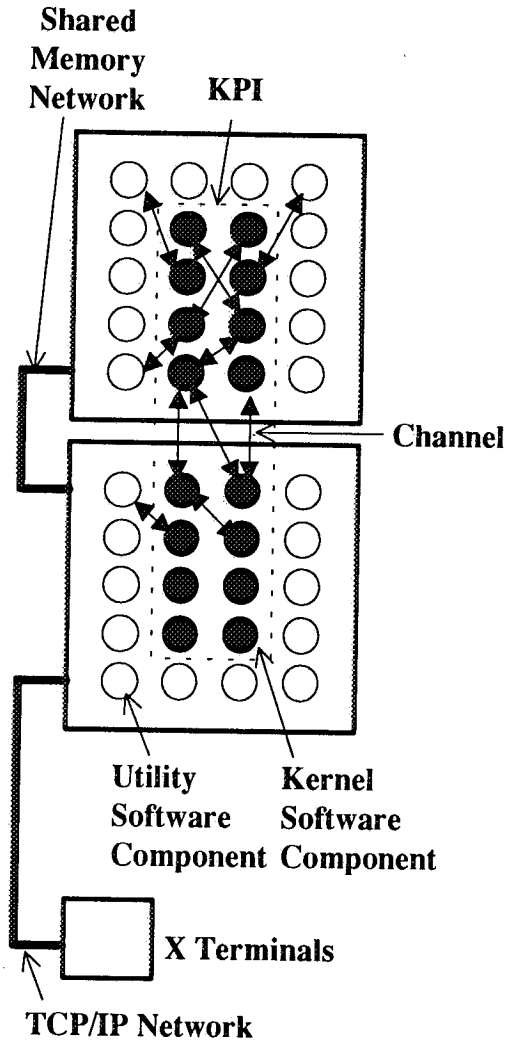

Figure 5. Distributed Kernel

FUNCTIONAL CONFIGURATION

Functional configuration incorporates only the required functionality, conserving system resources and reducing complexity. A system is composed of only the kernel and utility components needed to accomplish a specific task. A system that requires frequent changes in use and measurement definitions is configured to include a commercial relational database system such as Oracle, whereas an embedded system used in static environments where changes in measurements are rare omits the database system and uses a simple ASCII table for storing measurement definitions. An off-line configuration for data analysis such as a notebook computer could omit the commercial database and eliminate the memory and storage requirements. A system used only for monitor and display would not require the CMU archival, retrieval, command, and control components.

ARCHITECTURAL CONFIGURATION

Channels are the primary mechanism for supporting a wide range of hardware and software architectures. Channels provide three key benefits: (1) hardware configurations do not affect the software, (2) a single portable software application programming interface is used for all channel types, and (3) data transfers have very little operating system or application overhead.

Each channel is a connection between two or more software components. A configuration file is used during system startup to configure channels for a specific hardware implementation. Current channel implementations completed or planned are: shared virtual memory, VME bus physical memory, and shared memory networks/reflective memory. Channel support for TCP/IP networks is a simple extension. Kernel components communicate among themselves via channels. Figure 5 illustrates kernel components distributed across two separate computers.

Channels follow the UNIX open/close/read/write/ioctl model. Functions included are: CClose(), CCntl(), CInit(), COpen(), CRead(), CWrapup(), and CWrite(). Benchmarks of a channel configured for shared memory exceeded 2 million mps transfer rates, which would normally consume 120 MB/second, or about 75% of a 160 MB/second system bus bandwidth. Channels have been ported from a 32-bit UNIX System V-based operating system to DEC's 64-bit OSF/1 with minor modifications.

PERFORMANCE CONFIGURATION

A primary design philosophy of CMU software is support for single processor and multiprocessor computers. Multiple computers are combined for higher performance levels. Kernel and utility software components are configured to provide the required performance by taking advantage of additional CPUs. Figure 6 shows actual measured data processing rates for four different CPU configurations and projections for two additional configurations. These configurations use commercial symmetric multiprocessor (SMP) computers. Performance increases of 70% and 100% have been benchmarked for two different CMU software configurations. These increases occur when utilizing a second CPU on a DEC Alpha AXP 2100/500 system. Two 400 MHz 2100/500 computers with four CPUs each connected by shared memory networks alone would support almost 1.4 million mps of data processing. Higher rate systems are configured by using additional SMP and single-board computers linked by shared memory networks and I/O buses.

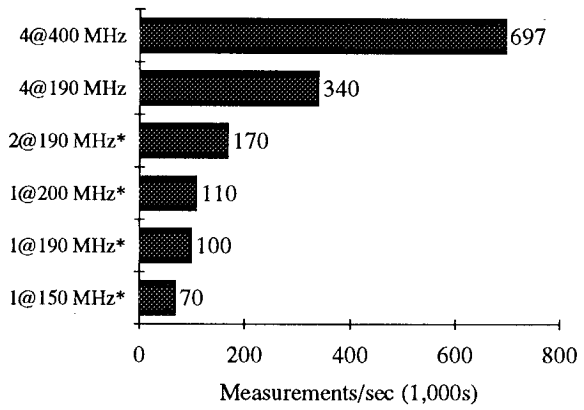

Figure 6. Data Processing Rates
*** Actual Measured Rates**

Archival and retrieval rates are scaled to match system requirements by combinations of magnetic, optical, and tape devices. RAID arrays of storage devices are used to achieve the required storage capacity and data rates. All data are time stamped to the nearest microsecond. Nominal automated control delays are on the order of a millisecond. Specialized configurations using multiple single-board computers in addition to the host computer can support automated control delays in the tens and hundreds of microseconds.

REUSABILITY

The CMU promotes reuse in several ways. Portable software based on POSIX application programming interfaces supports code reuse. The kernel/utility/channel software architecture provides the ability to configure widely varying systems from a single set of software. Reconfigurability allows reuse of CMU software at the component level.

In addition to software reuse, CMU is defining reuse techniques for all products created during development. This would allow custom high-performance real-time systems to be developed quickly and reliably at low cost. Reusability has been extended to modular testing software developed and maintained with the production software for all levels of testing. Methods for reuse of requirements, requirements traceability, and on-line user documentation are also being developed and implemented. Analysis and design models from Cadre Teamwork/Ensemble CASE tools are structured to promote maximum reusability across systems. Hardware design and documentation follow this same reuse philosophy.

SOFTWARE DEVELOPMENT

An evolutionary life-cycle is being used to rapidly improve products and processes in support of McDonnell Douglas' and NASA's TQM initiatives. Process-based methods such as the Software Engineering Institute's (SEI) Capability Maturity Model have been used over the past two years for software process improvement. With development cycles being reduced to four months between requirements and final verification testing, the opportunity for

process analysis and improvement is much greater than that of a conventional two or three-year development cycle of a large real-time system. As with financial investments, the compounding interest effect is already producing significant benefits in cost and quality in CMU software development. Early evaluation by users has already resulted in several improvements over the original requirements. These improvements can be incorporated with little or no impact on cost and schedule since they were identified early in development.

Development techniques incorporated include formal inspections of requirements, system designs, detail designs, code, test plans, test code, and user documentation. Other practices implemented are process and product metrics, nearly 100% path coverage during software unit/component testing, 100% automated testing from unit through system verification, and extensive use of CASE tools. The first two alpha releases of CMU software, A0.1 and A0.3, have had a delivered defect density of 0.06 defects per 1,000 source lines of code (KSLOC). This is compared to industry results [SEI, 1993] in Figure 7.

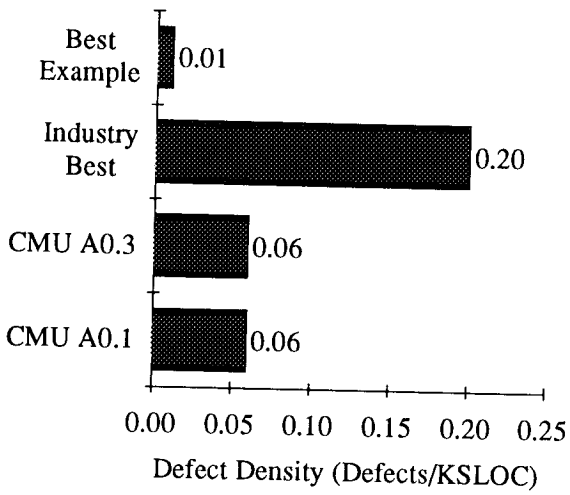

Figure 7. Delivered Defect Density

Only light operational usage via demonstrations and evaluation by the developers and users has been experienced since this is pre-delivery software. The current software is operational two years before its first delivery date, so significant random use and testing naturally occurs during development. This provides further opportunity for defect removal before final delivery to the customer.

Testing was found to remove 3% of all defects detected while inspections removed 97%. On a per defect basis, inspections are 40 times more effective than testing although the testing effort is significantly higher than the inspection effort. Inspections and testing combined for a total defect removal efficiency of 99.87%.

Figure 8 shows the detected and remaining defect densities during each phase of software release A0.3. At the conclusion of code inspections, the remaining defect density was 1.53. The remaining defects were removed during component and integration testing, resulting in a delivered defect density of 0.06. The total potential defect density was 49.9. The detected defect severity during inspection phases averaged 4, or trivial. Average defect severity during testing phases was 3, or minor. Only 2% of all defects detected were of severity 1 or 2, critical or major defects.

Net project software development productivity improved 60% compared to initial releases of earlier projects and is much greater than published for similar projects [Jones, 1991]. Software size is estimated using feature points and bottom-up detailed estimates. Software product quantity average within 10% of the estimates. Software development schedule compliance has improved in the first two releases from 14% to 6% without using overtime. Release A0.3 slipped one week during a six month development schedule.

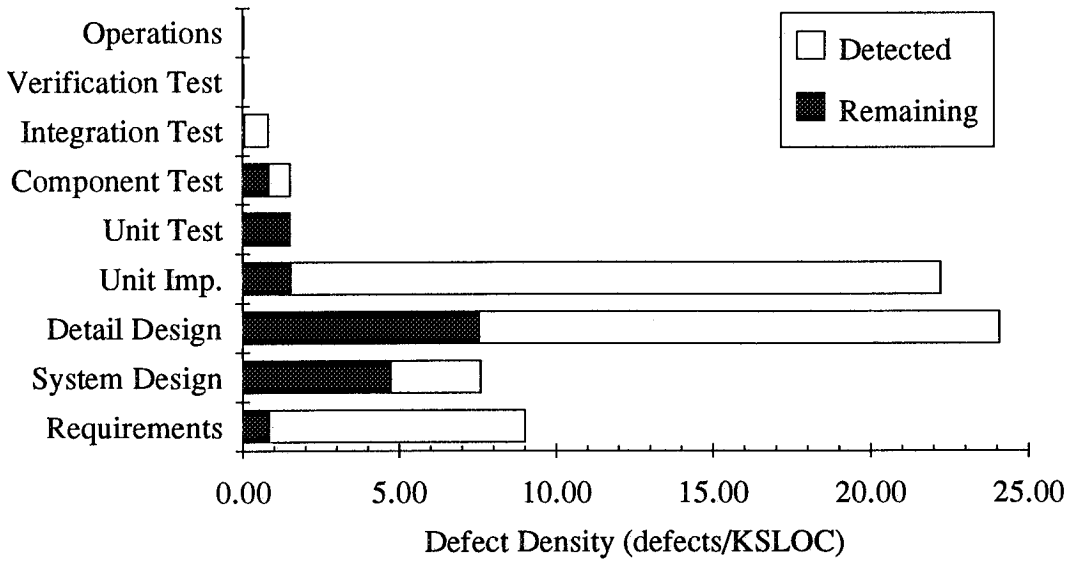

Figure 8. Development Defect Densities

CONCLUSIONS

Configurable software and hardware technology for demanding ground control and monitor systems has been demonstrated. The technology is reusable across small and large systems. Evolutionary development combined with continuous process improvement is an effective approach for developing real-time systems within budget and schedule constraints. Comprehensive inspection and testing contribute to world-class quality levels for software development.

REFERENCES

Jones, Capers (1991). Applied Software Measurement. New York, NY: McGraw Hill, Inc.

Gilb,Thomas (1988). Principles of Software Engineering Management. New York, NY: Addison-Wesley.

Software Engineering Institute (SEI) - Carnegie Mellon University (1993). 1993 Software Engineering Symposium Conference Notes. Pittsburgh, PA: Carnegie Mellon University.

Acknowledgments

The author would like to thank the CMU team members for their invaluable contributions: Jeannine Baker, Rick Bard, Sam Coniglio, Jim Gaines, Ray Ho, Ho Pham, Lawrence Robinson, Bill Snoddy, and Maria Thomas.

Systems Engineering

3. Standards

* *Presented in Poster Session*