Software management issues for new system designs

by ROBERT E. LOESH
Jet Propulsion Laboratory
Pasadena, California

and
DONALD J. REIFER
Reifer Consultants, Inc.
Torrance, California

and
STEVEN M. JACOBS
TRW Defense Systems Group
Redondo Beach, California

ABSTRACT

The management of software development for single and dual processor system designs is making progress towards becoming a mature discipline. A good part of the progress can be attributed to the development and use of standard system and software engineering methods and design principles.

However, new computer system designs (networking, distributed systems, embedded systems, multi- and coprocessors, fault tolerant systems, etc.) will create new challenges for managers of software development. The reason for this effect on management is that some of the system and software engineering methods and design principles developed for single and dual processor system designs are not valid for these newer designs.

Some of the issues that software development project managers will need to cope with are:

1. Life cycle model adjustments
2. Rapid prototyping activities
3. Different hardware and software phasing
4. Increased tool development
5. New trade-offs and hybrid developments of off-the-shelf software and newly developed software
6. Development of concurrent design principles
7. New software design principles to support fault tolerance and the use of new memory technologies

The above items are just now being recognized as problems, and solutions for them either do not exist, or are not widely known.

These problems create a series of new challenges that managers must deal with for software development based on the new architectures and requirements. The purposes of this paper are to discuss these issues and to identify some solutions that can serve in the interim as the technology changes to meet these new challenges.
INTRODUCTION

For the past 30 years, there has been one system design approach used to develop computer-based systems; that is, single or dual processors based on a sequential machine, which had limited main memory and similar instruction sets. Processor technology evolved during this time to become faster, bigger, and cheaper. However, there have not been as many improvements to software design and engineering as there have been for hardware design and engineering. Over the past decade, software engineering methods and design principles have been developed that are suitable for the single–dual processor system design approach. As applications became more demanding, single–dual processor system design and special-purpose hardware with only minimal software support were used. The management of software development for single and dual processor system designs has become a mature discipline. A good part of the progress is due to the development and use of standard system and software engineering methods and design principles, as is described in References 1–6. Even so, software engineering project management is far from being currently recognized as a “discipline,” and is looked upon more as art than science.

Shared-resource designs were accomplished through multiuser interfacing and using hardware front-end and rear-end multiplexers and demultiplexers. The same engineering design that was appropriate for multilevel, priority handling of interrupts was used almost exclusively. There has been little or no effect on development methods or design principles. If an application was too large or the time required for processing too long, designers used some variant of the same engineering methods and design principles that were used in the early days of computing. These methods and principles extrapolated concepts taken from automata theory, which preserved the notion of a sequential, centralized database machine.

Whenever an application did not execute as efficiently as desired on a given class of processor, designers would select a very special high frequency function and would then build a special hardware device to perform that function, using, for instance, convolvers, frame synchronization detectors, and character converters. The special-purpose function was removed from software, and mechanized in hardware such as array processors.

Embedded systems were generally built around miniclass, fully integrated computers that interfaced across point-to-point serial communications, or auxiliary storage devices such as tape, removable disk, or in some cases, a disk and drum, through shared controllers. Existing engineering methods and design principles were used with no changes because they could cope with limited distribution of function.

In the not-too-distant past, failure protection and recovery management were handled in similar ways. Fault tolerance was either designed into the hardware at the integrated circuit level, dealt with by checkpointing, or handled by the use of mirror image backup.

Our evolving engineering methods and design principles were applicable. Based on a build-up of experience using these engineering methods and design principles, management of single–dual processor system design developments is becoming more successful. However, in the past five years, progress in the hardware technologies has allowed the development of applications that previously were not technically feasible or were not acceptable from a cost-effectiveness standpoint. Unfortunately, the hardware capabilities and system designs available are radically different from the single–dual processor design that represents today’s state of the art. Following are some new hardware system designs and the attributes they have that affect new software designs:

1. Microprocessor-based embedded systems
2. Multiple processor resource sharing systems
3. Distributed data processing (DDP) systems
4. Coprocessor designs
5. Fault-tolerant systems
6. System architectures
7. New memory and processor designs (e.g., EPROM)
8. VLSI or VHSIC systems

These new design approaches require considerable software support and new software designs. The development of the software for these systems is not supported well by the current software engineering methods and design principles, which exist for single–dual processor system designs. In some cases, the software engineering methods and design principles are not suitable and must be modified. In others, there are voids that require new methods to be developed, understood, and used before a standard evolves.

The result of not having appropriate engineering methods or design principles will be missed schedules, poor performance, unachieved capabilities, cost overruns, and in some cases, failure to deliver a responsive system. These deficiencies will manifest as a software design that does not work, (is either poorly mapped onto the hardware or fails to use available capability) or a hardware and software design integration that cannot be tested or modified easily.

Until the proper engineering approach is known and well understood (adopted as a standard), an interim solution is needed. If we are aware of the problems from initiation of these developments, they can effectively be alleviated or averted. The interim solution recommended combines those approaches that limit the application of such new designs...
used, emphasize some special advanced, front-end engineering work to make up for the engineering standards deficiencies, and manage projects with special attention.

EXISTING STATE OF AFFAIRS

Software project management is demonstrating that many developments can be accomplished successfully. Some of the criteria for determining a successful software project are being on schedule and within budget, providing agreed-upon and reliable capability, and that the software fulfills its requirements and works. Although a long way from perfect, the following factors are contributing to increased successes in software development:

1. Projects are estimated and supported more realistically
2. Project management capabilities and techniques are improving and becoming more standard
3. Software engineering standards are evolving, and being more widely accepted and employed

The first two of these items seem to be continuously improving, and are generally unrelated to the system design issues. They are nonetheless affected by the technical adequacy, maturity, and wide use of proven system and software engineering standards.

For single-dual processor designs, a software development life cycle (SDLC) has been defined that incorporates phases and delineated products that support good software management practices. Additionally, standard software engineering methods and design principles are evolving that can be applied to the various SDLC phases. These standards contribute to the development of a good product, and establish an ability for software project managers to estimate and control cost, resources, and time. As these engineering standards receive wider use, they will continue to improve, as well as enhance management's ability to qualify and calibrate their effects. By virtue of a continued and increased use, and an increase in resultant knowledge of effectiveness and cost, software management methodology will improve continually.

Following are some of the software engineering methods and design principles that are becoming industry standards that contribute to improved software project management capability:

1. Problem analysis and requirements generation methodologies
   a. Operational concepts formulation, system interface definition
   b. Man-machine interface definition and prototyping
   c. Data flow diagrams
   d. Structured analysis
2. Requirements generation tools
   a. SREM, PSL/PSA, CADSAT, MEDL-R, DARTS
3. Program design methods and tools
   a. Structured design, HIPO charts, Jackson methodology
   b. PDL, SDDL, MEDL-D, USE.IT, DBMS
4. Program construction methods and tools
   a. Structured code, data structure definition tools (COMPOOLS)
   b. Languages, word processors, SPF, library functions, checkers
5. Program testing methods and tools
   a. White-box testing, black-box testing, module signature
   b. Test coverage analyzers, automatic test program
6. Resource management awareness and control
7. Performance tuning as part of final stages of testing

All of the above contribute to an increased predictability of the technical and software management task, and therefore to increased success.

There is still need for improvement. Additional methods and design principles must be developed to cope with our entry into the age of distributed computer systems. However, there exists a good technology base from which to work, and considerable attention and effort is being directed to its improvement and use. Clearly, over the next three to five years, the situation will continue to improve. Both engineering methods and design principles, and the software project management discipline based on them, will result in a successful, highly consistent level of software development for single-dual processor designs.

SOFTWARE ENGINEERING STATUS FOR NEW DESIGNS

The prognosis for new system designs, however, is not in the same healthy state. Early experience with distributed systems indicates that the management of software designs is facing considerable difficulty. It is plagued with, for example, poor system performance, unreliable system operations, high cost overruns, missed schedule commitments, and disappointing capability. While the total problem does not lie with the software engineering and associated management technology base, a significant part is due to the following issues:

- Some of the existing methods and design principles do not apply or do not work
- There are real voids where no methods or design principles exist

Following are a number of areas that have significant deficiencies:

1. Existing engineering principles, methods, and tools for concurrent technology are only partially applicable
2. Engineering methodologies are deficient
3. Requirements tools are deficient
4. Program design methods and tools are deficient
5. Program construction methods and tools lag but will mature in the next five years
6. Testing methods and tools are deficient
7. Performance tuning still must be learned
8. Resource management will require additional development of methods and tools
9. System architecture tools are deficient (1) for specifying system configurations for applications, (2) for performing hardware-software tradeoffs, and (3) for comparing candidate system architectures.

Finding solutions to the above deficiencies, for most software development, tends to rely on trial and error. Attempts to use methods and best-guess designs proliferate. Some consequences of this approach to engineering are increased costs, wasted time (delayed schedules), poor performance, and a lack of reliability. In some cases the deficiencies have resulted in total project failure, that is, in nonexecutable systems; of course, the software project manager is held accountable for this.

The situation will improve as problem areas are defined and publicized. Energy and funding will be devoted to finding solutions. Additionally, the random approaches will crystallize methodologies and design techniques that work successfully. From these principles will emerge a new generation of defined, tried and proven, publicized, and readily usable engineering tools and standards.

It has taken nearly 30 years to achieve a significant threshold of software engineering standards applicable to single- or dual processor system design. The difficulty of the engineering problem for new system designs is mitigated by new methods using prototyping, simulation, requirements tools, etc. Yet there is a greater focus of interest by many organizations willing to support the research of viable solutions. Industrial interest, communication, and concern about system and software engineering have increased considerably.

The result should be that we will make rapid progress in developing and identifying appropriate engineering standards within these areas in the next seven to ten years. By 1990, there should exist a reasonably capable system for engineering such new designs, with a high probability for success in the management of such software developments.

WHAT TO DO NOW

It is clear that implementation of these new system designs is an urgent matter. Managers of current systems do not have the luxury of waiting a decade to find solutions. Also, if the engineering methods and design principles derived by trial and error, or discovered through research, are not used for actual software developments, they cannot be qualified or accepted as standards. Without application, newly developed methods and design principles will never fully emerge. Therefore, interim solutions are needed now.

Experimentation and published results about what works and what does not are required. There currently exists a set of management and engineering techniques that can be employed to help prevent software development failures and contribute to the emergence of new engineering standards. The following suggestions, based on the authors' experiences, may prevent major problems in software developments resulting from these new system designs:

1. Unless an argument is good, stay with the current design technology (single- or dual processor system designs)

2. Go to only a part of new system designs; do not try too much at once
3. Allow for more budget and schedule
4. Do more prototyping
5. Do more in-house research and tool development
6. Watch others
7. Document your experiences
8. As the engineering methods and design principles emerge and become standard, incorporate them

The bottom line is use caution and preserve resources. The key is to control the risk. If indeed we move into these new system designs carefully and meet our professional responsibilities to share our methods, design principles, experiences, and results (both good and bad), we can take advantage of the new technology and still not suffer major setbacks in development and system execution and support. The management of software development for such new system designs can be applied successfully, both during the interim while standards are being developed, and of course after such standards exist.

ACKNOWLEDGEMENT

The authors wish to acknowledge the following people for their review of and contributions to this paper: Herman A. Regusters, Senior Consultant, Ground Data Systems Group, Jet Propulsion Laboratory, as well as William B. Allen-dorfer, Imad Bitar, William B. Howard, Odette Knedr, L. David Lutton, Steven P. Munt, and Maria H. Penedo of TRW.

REFERENCES


