COMPSAC '86

The IEEE Computer Society's
10th Annual International Computer Software
and Applications Cnoference

[illegible]

PROCEEDINGS

10th ANNIVERSARY

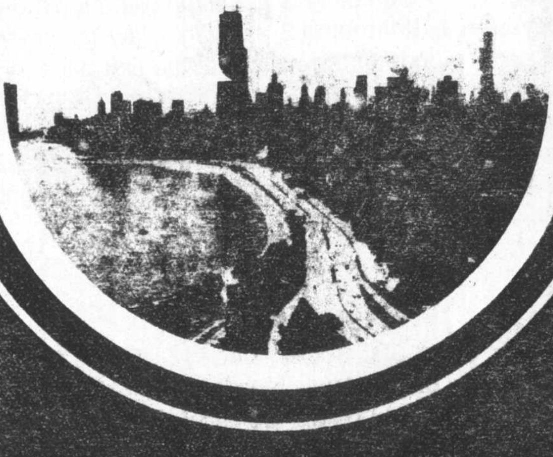
October 8–10, 1986

compsac 86

The IEEE Computer Society's Tenth Annual International

Computer Software & Applications Conference

Americana Congress Hotel, Chicago, Illinois



ISSN 0730-3157
IEEE Computer Society Order Number 727
Library of Congress Number 83-640060
IEEE Catalog Number 86CH2356-4
ISBN 0-8186-0727-0

THE COMPUTER SOCIETY OF THE IEEE

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC
IEEE

IEEE COMPUTER SOCIETY PRESS

Published by IEEE Computer Society Press
1730 Massachusetts Avenue, N.W.
Washington, D.C. 20036-1903

COVER DESIGNED BY JACK I. BALLESTERO

ISSN 0730-3157
IEEE Computer Society Order Number 727
Library of Congress Number 83-640060
IEEE Catalog Number 86CH2356-4
ISBN 0-8186-0727-0 (paper)
ISBN 0-8186-4727-2 (microfiche)
ISBN 0-8186-8727-4 (case)

Order from: IEEE Computer Society
Post Office Box 80452
Worldway Postal Center
Los Angeles, CA 90080

IEEE Service Center
445 Hoes Lane
Piscataway, NJ 08854

IEEE THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

Table of Contents

Opening Session

Welcome—Alan Davis, *General Chairman*

Award Presentation—Roy L. Russo, *President, IEEE Computer Society*

Program Overview—John R. North, *Program Chairman*

The First 10 Years—Stephen S. Yau, *COMPSAC Steering Committee Chairman*

Keynote Address

"Prognostication on Software Productivity"

Winston Royce

Director of Lockheed Software Technology Center, USA

Session GH-1W

Software Productivity Metrics

Minireviewer
Jai K. Navlakha
Florida International University, USA

Minireview of Software Productivity Metrics

Jainendra K. Navlakha
Florida International University
Department of Mathematical Sciences
Miami, Florida 33199

Software productivity is one of the most important attributes of software development process. This minireview introduces the following important concepts about software productivity and its measurement.

1) Why is the measurement of software productivity important.

   a) Rising software costs, particularly as compared to hardware costs.
   b) Management control of projects.
   c) Improving managerial decision making process.

2) Different viewpoints of software productivity.

   a) General viewpoint of productivity - Output/Input.
   b) Economic viewpoint of productivity - Values/Costs, where Value is an increase in revenues attributable to the product or a cost reduction resulting from the use of the product.
   c) Management's viewpoint.
   d) Life-cycle viewpoint.

3) Desirable properties of software productivity metrics.

   a) Ease of application. The measurement process should be straight forward and data collection should not be unnecessarily burdensome.
   b) Cost effectiveness. Data gathering should not be expansive.
   c) Universal applicability to full universe of software including commercial, embedded, diagnostic and AI software.
   d) Dependence on other software measures, e.g., quality, reliability, usability etc.
   e) Independence from development methodology. The methodology may affect other software attributes on which productivity depends.
   f) Other miscellaneous properties.

4) Advantages of measuring software productivity.

   a) Management related advantages like improving decision making capability of top level management and allowing software managers to predict performance of their group.
   b) Motivational. Advantages like increased self-awareness and realization of self-improvement needs among groups and/or individuals.
   c) Better understanding of software attributes on which productivity depends.
   d) Evaluation of software in the form of quantitative comparison of similar products.
   e) Advantages to software buyers and researchers. Measurements allow customers to buy software that provides either absolute highest productivity or maximum productivity per unit of cost. Researchers get to understand software engineering principles better and come to know new areas of investigation.

5) Basili's Goal/Question/Metric paradigm.

Goals at the top level are achievable by answering specific questions or by hypothesizing principles. These in turn depend upon a quantitative metric for each. Thus target goals lead to specific questions or hypotheses and the questions lead to particular metrics.

6) Evaluation of some productivity metrics.

The subjective evaluation of four software productivity metrics - Halstead's effort, McCabe's cyclomatic complexity, Albrecht's function point and lines of code per man-month - is based on desirable properties and derived advantages of any productivity metric.

7) IEEE Working Group on a standard for Productivity Metrics is presently viewing productivity as consisting of a number of classes (e.g., Functionality, Database, Information, Capability and so on), each of which is composed of a number of elements (e.g., Elements of Functionality include reusability, schedule, use of modern practices, complexity of software etc). The effort of the group is directed towards determining a metric for each element of each class and finding a technique to combine these together to derive a standard for software productivity metrics. The latest information about the deliberations of the Working Group will also be included in the tutorial.

REFERENCES:

1. V. Basili, Sample Model for productivity, Invited lecture given to the IEEE Working Group on a standard for software productivity metrics, Sep. 1984.

2. Minutes of the meetings of IEEE Working Group on a standard for software productivity metrics, 1984-1986.

3. J.B. Munsen and R.T. Yeh, Report by the IEEE Software Engineering Productivity Workshop, 1981.

4. J. Navlakha, Software productivity and its management, Proc. of the National Computer Conference, 1985.

5. J. Navlakha, Software productivity metrics: Some candidates and their evaluation, Proceedings of the National Computer Conference, 1986.

Session GO-1W

Software Applications

Chairperson

Frank Houska

AT&T, USA

EVALUATION OF ADA FROM THE VIEWPOINT OF CONTROL ENGINEERING

WOLFGANG A. HALANG

Department of Systems Engineering
University of Petroleum & Minerals
Dhahran 31261, Saudi Arabia

ABSTRACT

The requirements to be fulfilled by languages applied in hard real-time environments, and corresponding language constructs necessitated by them, are summarized. Thereafter, it is discussed how well Ada meets these demands, and which features still need to be incorporated to enable the production of reliable software for time-critical applications. Then, the syntax and semantics of appropriate supplementary real-time language elements for Ada are described, providing mainly the following features: application oriented resource synchronization with time-out and inherent deadlock prevention; support of due dates observing task scheduling algorithms allowing the early detection and handling of overload conditions; accurate timing of operations; and application oriented simulation regarding the operating system overhead for software verification purposes. Finally, the operating system functions required by the proposed language constructs and verification options will be outlined.

1. INTRODUCTION

The new programming language Ada emanated from an effort, by the U.S. Department of Defense, to make available a software production tool suited to program embedded systems. Ada is intended to save considerable software costs by replacing the vast variety of languages presently in use for DoD projects. The evaluation of these gave rise to requirements for a common DoD language that have been refined several times.

Owing to its military background, one expects that Ada should provide remarkable properties with regard to safety, reliability, predictability of program behaviour, and especially with regard to timing features, because military and technical processes are rather time sensitive, and often require the observation of tight timing constraints with only small tolerances. Here, however, Ada falls short of these expectations, since it provides only a relatively poor expressibility of the time behaviour. Tasks are scheduled on the basis of fixed priorities, although recent research [5,7] has revealed that the observation of strict deadlines, typical for hard real-time environments, cannot be guaranteed by the mentioned procedure. We shall see that only a part of the features requisite for efficient problem solving in this area has already been realized in Ada.

NOTE: Ada is a registered trademark of the US-Government-Ada Joint Program Office

Presently it still holds true, that about 80 % of all process control applications - and not only the time-critical ones - are programmed in assembly languages [6]. To increase the efficiency of real-time data processing it is, therefore, necessary to provide all the capabilities required to cope with the demands of hard real-time applications fully within the framework of a high-level language like Ada. This will also enhance the software portability. Ada does not yet adequately meet the corresponding demand pattern. Specifically, a number of problem-oriented language constructions, especially with regard to expressing the programs' exact time behaviour, appear to be missing. The purpose of this paper is, therefore, to take remedial measures, by defining here appropriate extensions of Ada.

When developing process control programs, not only the software correctness, in the sense of mathematical mappings as in batch processing environments, has to be proved; also, its intended behaviour in the time dimension, and the interaction of concurrently active tasks, need verification. To this end, language features for the specification of maximum task run-times, of updated residual run-times required by tasks before their completion, and of timing constraints for certain operations, as well as for enabling the determination of module run-times by the compiler, are necessary. Furthermore, resource-claim capabilities have to be provided which allow the deadlock-preventing scheduling of tasks. The latter should be based on algorithms guaranteeing the observation of the tasks' due dates, provided that this is possible. If, finally, the time frame for the operating system overhead is known, then the program behaviour, as independent from external events, becomes foreseeable. When the above mentioned features are given, an off-line simulation of the software can be carried through, not only simplifying the verification process, but also allowing the inclusion of randomly occurring external events into the consideration. This test procedure, oriented at hard real-time environments, would yield exact information if time constraints can be observed and whether due dates can be granted.

The leading idea behind all the proposals outlined in this paper, is to facilitate reliable and predictable program execution, this being a prerequisite for the safety approval of software in hard real-time environments.

2. CRITIQUE OF ADA'S REAL-TIME FEATURES

A series of shortcomings will now be mentioned making it questionable whether Ada in its present form is apt for time-critical process control applications.

As in other real-time languages, parallel processing is organized on the basis of tasks which can be hierarchically ordered. Initiation and termination are provided as sole tasking functions. Ada is lacking in important timing features, and its capabilities for the time scheduling of task executions and operations are very limited. This is because the only way for expressing time dependencies, is to delay the executions of tasks by given periods and absolute time specifications appear to be impossible. Hence, it is necessary to schedule, explicitly within its bodies, the execution of tasks according to external demands, whose arrivals are actively, and if need be, also selectively awaited. Ada's task state model is very simple and allows only a determination of whether a task is active or already terminated. With an interrupt, only a single task can be reactivated. The operations of enabling or disabling interrupts are not contained in the language. Furthermore, the prevention of deadlocks is not supported. With regard to conventional elements, bit handling is only indirectly possible.

When comparing the version [2] of Ada with its preliminary specification [1], one finds that its feasibility for real-time programming has even been impaired by abolishing several significant features. Thus, the initiate statement was deleted. Tasks are immediately activated when their declarations have been elaborated. The once-assigned priorities may not be dynamically changed. Semaphores and signals, as predefined synchronization mechanisms, have been relinquished; and finally, the cumulative processing time of tasks is no longer available.

Remarkable, in Ada, are its elegant synchronization concept, and the possibility to control whether rendezvous and synchronization operations take place within given time frames. But, on the other hand, the non-deterministic selection of an entry call by select statements, in case several rendezvous are simultaneously possible, introduces unpredictability into the language. The semantics of Ada imply that processors and other resources are dispatched according to the First-in-First-out strategy with regard to priorities. Thence, when also considering the above mentioned shortcomings, and the absence of most of the features discussed in the next section, which ought to be present in a contemporary real-time language, it must be concluded that Ada is less of a progress as far as its suitability for real-time applications is concerned.

A language must reflect the users' way of thinking, and the users generally are engineers and technicians - but not computer scientists. So, the language features should be easily conceivable, safe to handle, and application oriented. Experience reveals that Ada also has disadvantages in this respect, because some of its constructs - although being very elegant - are hard to understand.

3. PROPOSAL OF ADDITIONAL LANGUAGE ELEMENTS

When comparing the requirements for real-time languages and systems, that were outlined in section 1, with the capabilities of Ada, it becomes obvious that various elements especially important for the production of reliable software are still missing, or are only rudimentarily present, and that some changes of Ada's semantics are necessary.

First of all, statements allowing the complete controllability of tasks from the outside in dependence on complex schedules involving interrupts, time specifications, and further conditions should be introduced into Ada. Since these, and the other missing features, which were already mentioned, are present in other languages, such as Pearl [4], we do not need to discuss them here in more detail. Instead, we shall propose some further language elements required in real-time applications.

For the purpose of securing data integrity, when several tasks access the same resource, Ada only provides the means of performing a rendezvous. This feature, however, appears too expensive for the purpose of synchronizing the access to certain resources; it veils the kind of operation being carried out, and does not support deadlock prevention. Ada already provides a time-out feature for synchronization operations. Moreover, the process of claiming resources, and assigning them to tasks, must be supervised in a real-time environment. In this respect, in [9], the statement is found that Ada multitasking seems to assume that a calling task, requesting a resource, can wait until that resource is available.

For encapsulating the access to protected resources, and to enforce their later release, we therefore introduce a lock statement, similar to the one described in [3]. We describe it here for shared variables only, but its extension, to other protected resources, is straightforward. Its syntax reads as:

```
lock synchronization_clause_list [nonpreemptively]

     [timeout_clause]
```