19226

Distr.
LIMITED

IPCT.144(SPEC.)
13 November 1991

UNITED NATIONS
INDUSTRIAL DEVELOPMENT ORGANIZATION

ORIGINAL: ENGLISH

CHANCES AND ISSUES IN SOFTWARE PRODUCTION IN DEVELOPING COUNTRIES*

Prepared by

Hermann Kopetz**
UNIDO Consultant

* The views expressed in this document are those of the author and do not necessarily reflect the views of the Secretariat of UNIDO. This document has not been edited.

** Technical University of Vienna

V.91-30739

TABLE OF CONTENTS

EXECUTIVE SUMMARY

The ongoing technological advances in the field of electronics will open new opportunities and expansive markets for software products in the nineties. The impact of the software industry on the economy of all developing and developed countries will increase significantly.

Highly qualified, well trained and fully motivated personnel form the basis of any software industry. Investment into first rate educational facilities by the government is the most important political action that has to be taken in order to establish the foundation of a software industry.

The two key markets for software which should be selected are intelligent product software and contract programming. Intelligent products offer the addition 1 advantage that more employment opportunities are generated in the developing countries.

Contract programming requires reliable access to international computer networks. This access has to be provided by the authorities in the developing countries.

Any software development requires a sound management structure. There is an opportunity of cooperation between developing and developed countries in the accreditation of software quality management programs. Risk management and productivity management should be addressed specifically.

The market for real-time software is growing rapidly. The development methodology for real-time software is not well established. This offers a challenge and a chance. In this report a rigorous development methodology for real-time software is presented.

1. INTRODUCTION

In the last years, software production has emerged as a major industry of substantial economic significance. It is estimated that over half of the worldwide information processing market in the order of more than 500 billion US $ is, in one way or another, related to software production. This market is still growing by a rate of about 10% per year. Millions of software professionals, system analysts, programmers, managers, etc. are engaged in software production worldwide. It is our opinion that every industrialized or developing country in the world has to face the impact of the software industry. Software can be seen as an industrial product which is imported to serve the local needs or which is locally produced and exported to the world market. In a maturing economy, which is integrated into the economic systems of the world, software should be seen as both: an import and an export product.

Most of the software produced does not appear on the open market in the form of prepackaged software. A formidable amount of software is produced by the computer companies and sold together with their hardware as a computer systems product. A lot of software is integrated in other tangible products to increase their functionality and value. Many of the latest consumer products and industrial products contain integrated microprocessors or computer systems with significant amounts of application software hidden behind a user-friendly man machine interface. Software is also needed for the computer integrated manufacturing (CIM) systems. Computer-based production planning and control systems are installed in most manufacturing plants. It follows that a state of the art software capability is not only needed for the production of software per se, but also in the production of any kind of industrial product.

Software is a know how intensive industry. Highly qualified, well trained and fully motivated personnel form the basis of any software industry. Because productivity ranges are much higher than salary ranges, it wise to attract the best people to software production by offering extraordinary working conditions.

This report surveys chances and issues in software production. In the first section the "world of software" is analyzed to find market opportunities for newcomers from developing countries. Any software production requires the establishment of an appropriate project management structure. In Chapter 3 some topics in software management which have grown in importance over the recent years, i.e., quality management, risk management and productivity management, are discussed in some detail. Chapter 4 deals with the systematic design of real-time software, a discipline which is of great industrial relevance but of comparable recent origin. In the final chapter the summary and concrete recommendations are presented.

2. THE WORLD OF SOFTWARE

In this section we will analyze the "world of software" to look for market opportunities, particularly for newcomers from developing countries. After a short discussion of the operational environment for computer software, which is to be expected in the next few years, we will focus on the emerging software markets. In the final chapter we try to shortlist a number of risks in software production.

2.1 The operational environment

An enduser is interested in the delivery of some specified computational service of an integrated computer system. Only the proper combination of computer software and hardware can provide such a service. From this point of view, software in itself is not a complete product--it requires an operational hardware environment to produce the intended effect. In this section we review the expected trends in the field of computer hardware and operating systems to sketch the operational hardware environment for the future software.

2.1.1 Hardware trends

In the next ten years the technological advances in the field of computer hardware will continue to produce more powerful microelectronic chips at the same rate as we have seen this in the recent past. These advances will effect both the functional capability and the performance of the computer hardware.

By now it is fairly clear that the next generation of memory chips, the 64 Mbyte chip, will be introduced on the market around 1995. Before the end this decade the following generation, the 256 Mbyte chip, should be available. Similar advances can be expected in the capabilities of the Central Processing Units (CPU) and storage media. The interconnection of the computers will be realized by high speed networks in the gigabit range.

A short glimpse backwards should help us to put these extraordinary developments in the proper perspective. When the first personal computers were marketed at the beginning of the eighties, a memory size of 64 kbyte was common. Five years later the typical memory has increased to about 512 kbytes, while today, another five years later, another ten-fold increase to 5 Mbyte can be observed. If this trend continues--and there are convincing indications it will--, a typical personal computer will have a memory of more than 100 Mbytes before the end of this decade.

More importantly, the expected advances in the field of VLSI integration will allow to integrate more than 10 million circuit elements on a single chip before the end of this decade. This level of integration makes it possible to manufacture complex information processing systems on a single chip. Powerful single chip microcontrollers with onboard RAM, ROM and process I/O will be available for all kind of integrated control functions.

Since the design of highly integrated VLSI chips is extremely costly and the marginal manufacturing costs are relatively low, only a few "families" of general purpose processors will dominate the worldwide computer market. This has important implications for the software industry. Only a small number of standardized operating systems will be available.

Over the next decade, the cost to produce computer hardware of a given functionality will decline at a similar speed as in the past decade. An equivalent reduction of the software costs cannot be expected. Therefore, on a system basis the strategic importance of software will increase, i.e. the major fraction of the value of an integrated computer system will be in its software.

2.1.2 Operating systems

The standardization of the operating systems is well under way. Proprietary operating systems from the major computer companies give way to standardized operating systems, such as UNIX and MS/DOS. According to a recent US-statistic concerning the Personal Computer Market, the following operating systems were dominant by the end of 1990:

MS DOS	60760
MS Windows	8860
Apple Macintosh	5051
UNIX	1500
OS/2	700

Major PC Operating Systems shipped through year-end 1990 in thousands of units /Bul91/.

From these numbers it is evident that at the moment the MS/DOS market is the most interesting market for producers of commercial software packages. An established operating system base of more than 50 million installations offers tremendous opportunities for a novel software package of mass appeal.

In the field of real-time software and microcontroller software, no standardization tendencies which are similar to the PC market are observable at the moment. Although the US Department of Defense launched a standard programming language and environment, ADA, for embedded computer application some years ago, this language has not found widespread acceptance outside the military community yet. Many embedded computer applications are based on small proprietary real-time operating systems of focused functionality.

The future trends in the operating system field will be determined by a number of factors. The expected hardware environment, as described above, will open completely new markets, such as multimedia applications integrating text, sound and video. It is not suie whether the present market leader in operating systems, MS/DOS can evolve to handle these new applications in an optimal way.

On the other side, new strategic alliances are formed in the computer and electronics industry worldwide. The collaboration between IBM and Apple computers can have a considerable effect on the future of the MACINTOSH operating system. The agreement between Digital Equipment Corporation and Philips can have an impact on the multimedia market, which is also at the focus of th. major Japanese electronics companies.

2.2 Software markets

For the purpose of our analysis of the economic opportunities of developing countries in the software world, we will partition the software activities into the following five segments:

(1) Prepackaged software
(2) Key element software

(3) Intelligent software
(4) Contract programming
(5) Software maintenance

Although these segments are partially overlapping, they each have a clear identity, as will be seen in the following discussion.

2.2.1 Prepackaged software

Prepackaged software is defined as that segment of the software industry, which is dedicated to the production, marketing and maintenance of software packages, such as a word processing program.

Although there are many thousands of small software companies trying to establish themselves in this very competitive market, only few companies succeed. The best selling application packages are all based on the MS/DOS operating system. According to a US statistics /Bul91/, the following packages make the list of the top 10 prepackaged software products by the end of 1990:

Rank	Name	Company	Units sold
1	Windows	Microsoft	8.860.000
2	Lotus 1-2-3	Lotus Develp.	8.000.000
3	WordPerfect	WordPerfect	5.500.000
4	WordStar	WordStar Int.	4.200.000
5	dBase	Aston Tate	3.315.000
6	PCPaintbrush	ZSoft	2.050.000
7	Multiplan	Microsoft	1.800.000
8	SuperCalc	Computer Ass.Int.	1.700.000
9	Norton Utilities	Symantec	1.500.000
10	Deskmate	Tandy	1.500.000

Note that none of the packages based on the second most successful operating system, the MACINTOSH, made it to a place in the top 10 best sellers.

These numbers, impressive as they are, should not lead to the false conclusion that prepackaged software is the only major market for the output of the software industry. Although it is the most visible market, it accounts for less than 10 per cent of the total output of the software industry. The world's leading software company, MICROSOFT which produced the leading PC operating system MS/DOS and has two products in the 10 best-sellers list above, recorded sales of about 1200 million US $ last year, less than 2% of the sales volume of the leading integrated computer company, IBM /BUS91/.

Successful products in the "prepackaged software" market address the following issues with great care:

(1) Genuine user need:

A lasting success of any product depends on the relevance and quality of service it can provide to its users. The focus on the true needs of the prospective users is thus the foremost requirement of a software product.

(1) Ease of use:

A product, which is aimed at a mass market, should be usable by an average user without extensive training. "Ease-of-use" is a complex software characteristic which does not only depend on the product per se but also on the background and experience of the prospective use. Many new software companies fail because they underestimate the effort required to make a program easy-to-use.

(2) Documentation:

A mass software product has to be accompanied by a flawless documentation which is organized such that answers to all conceivable questions of its users can be found quickly. Preparing such a documentation is a major effort which is often underestimated.

(3) Quality:

A serious error in a mass software product can nullify an expensive marketing campaign and cause excessive support costs.

(4) Support:

Vendors of successful software packages provide extensive after sales support to their customers. It is common that the average customer will require more than once a telephone based assistance directly from the supplier, since many distributors are not in the position to answer in-depth questions from the customers.

There are many instances of successful software products in small specialized markets. These products are based on comprehensive application know how in a specific area and provide an excellent service for a select customer base. These products are not necessarily linked with all the key attributes of "mass market" software, e.g. a delighted customer might we willing to trade some "ease-of-use" for a profound assistance in his key business needs.

2.2.2 Key element software

Key element software is defined as that segment of the software industry where the software contributes the key element to a complex industrial product. e.g., the software for a telephone switching system or an operating system for a new computer. It is characteristic for this market that an existing industrial organization with an established home market tries to improve or enhance its existing product by new functionality based on software.

The key element software market is dominated by major industrial companies both inside and outside the computer field. The major computer companies have to develop or adapt the system-software and networking-software to their hardware architecture, resulting in an immense software effort.

Our attention will focus on companies outside the established computer field. Many of these industrial companies replace their conventional control systems by software based subsystems and add improved functionality to increase the competitiveness of their products in their respective markets. The cost of the software subsystem can range from a small proportion (a few percent) to a substantial part of the final product price. Characteristic for the key element software is the rapidly rising share of the software subsystem. Let us consider the automobile. A few years ago there was hardly any software based subsystem in an automobile. According to industry estimates, by the end of this decade 10 to 15 % of the cost of an automobile will be in the electronics subsystem, a considerably part thereof in the software. A similar trend can be observed in avionics control systems, telephone switching systems, etc.

The following attributes characterize the key element software:

(1) Dependability:

Since many of the targeted applications, e.g., an avionics system, can potentially fail in catastrophic failure modes, the software must be highly dependable and support fault-tolerant operation. Other applications, e.g., telephone switching, require extreme levels of availability (e.g. less than 2 hours of outage in 20 years of operation).

(2) Real-time response:

Most of the software in this market segment is concerned with real-time applications. i.e., it must be guaranteed that a result will be produced by the computer within the specified time interval.

(3) Complexity:

Many of the applications in this market segment are inherently large and complex. It therefore requires a substantial investment in a sizable software development organization and quality assurance program to meet the demanding requirements of the applications.

2.2.3 Intelligent product software

"Intelligent products" are products which integrate a mechanical subsystem with a computer controlled subsystem intc a compact functional unit fulfilling a specific user need, e.g. an automatic scale with an integrated microcomputer to perform the calibration, the weighing and the recording fun tion. A substantial portion of the cost in the intelligent product is in the application software development. The application software forms an integral part of the intelligent product and is normally stored in a Read Only Memory, i.e., it is integrated in the hardware and cannot be changed easily.

In a recent contribution to the UNIDO Microelectronics Monitor, January 1990,/Kop90/, the characteristics and the design challenges in the production of intelligent products in developing countries have been analyzed.

Although intelligent product software is related to the key element software, there are substantial differences which justify its separate classification. The following attributes characterize the intelligent product software:

(1) Dependability:

The reliability of an "intelligent product" may not be compromised by errors in the software. Since the software is integrated in the intelligent product it is not possible to correct software errors in the field, i.e., in case of a software error the whole product may have to be discarded.

(2) Real-time response:

The software in most intelligent products must respond in real-time.

(3) Complexity:

The inherent software complexity of intelligent product software is normally much lower than the complexity of key element software.

(4) Optimal resource utilization:

Intelligent products are sold in a mass market in high quantities. Therefore is necessary to be aware of the resource requirements of the software.

Because of the lower complexity and stand-alone utility, intelligent products are well suited for the development by innovative small companies.

2.2.4 Contract programming

Contract programming is concerned with the design and implementation of software relative to given functional specifications. In a typical scenario a client, i.e. a industrial company, specifies a software package and formulates a set of acceptance tests. A software service organization, e.g., a small software house, implements this software package and delivers it to the client. Normally the software house will also offer a maintenance contract for the delivered software.

Contract programming can be an interesting activity for developing countries. It requires little capital investment, other than excellent training facilities and access to an international electronic network, e.g. Internet. There is a significant potential for cooperation between industrialized and developing countries in the field of contract programming. A major industrial company can start with small contract programming projects and can gradually build up a partnership with an organization in a developing country. There are some developing countries, notably India, who are active in the field of contract programming.

A small software house in a developing country, connected to one of the major international computer networks, e.g., Internet, can receive the specification and deliver products electronically. Also the interactions between the client and the software house can be executed via e-mail such that an interactive dialogue is possible.

2.2.5 Software maintenance

Software maintenance encompasses the elimination of software errors, the adaption of existing software to a new hardware platform, and the enhancement of existing software. It is performed either by EDP departments within organizations or by independent software houses closely linked to an organization . Software maintenance is the largest software activity. Its size is related to the general level of industrialization and automation within a country. Although its volume is still increasing in absolute terms, it is decreasing in relative size. Many proprietary software solutions within companies are replaced by prepackaged softwares. As a consequence, the need for software maintenance is reduced. Software maintenance requires a good understanding of and a smooth cooperation between the organization utilizing the software and the organization providing the service. The maintenance activity is normally not open for outside competition.

2.3 Software distribution

After a new software product has been developed successfully it has to be distributed to an interested clientele. We will distinguish between three different styles of software distribution: commercial software, integrated software and free software.

Since software is an intellectual product and can be copied easily, the intellectual property laws have been expanded to cover software and thus protect the software developer. Still, a sizable fraction of prepackaged software is copied illegally. The present situation concerning the intellectual property rights of the software owner is still a subject of heated debate. On the one side it is felt that the protection of the software rights, e.g., of the "look and feel" of a man machine interface, is not sufficient to protect the ideas of the original inventor. On the other side it is argued that too often copyrights and patents are granted to programs that are obvious. As a consequence programmers must spend more and more of their time on finding ways around existing patents. It is feared that in the future it will be difficult to write useful software because most basic ideas are protected and every new program is likely to infringe patents.

There is some logic in the radical idea, expressed by a minority group, to reduce the scope of the software protection; software companies should make their money by servicing the software products they create. Such a measure would reduce the increasing number of costly litigations and eliminate the legal barriers on creativity in the software field.

2.3.1 Commercial software

The standard way to market prepackaged software is via wholesale and retail distributors, such as a computer shop or a mail-order company selling via a software catalogue. Since there is only a limited amount of shelfspace available in a computer shop, it can be very difficult to convince a computer shop owner to display a new software product if at has not been widely advertised. Expensive marketing campaigns are necessary to bring the new product to the attention of prospective buyers. In many cases, the cost of distribution is significantly higher than the cost of software development. There are only few small software companies who can afford the prohibitive cost of an effective product introduction.

Once a new software product is established in the market it is enhanced and improved, based on the response of the customers. A new version is released about every year and the established customer base can upgrade to the new version for a small fee. As the product gets older it becomes more difficult to make changes. It is tried not to upset old customers by changing operating procedures the users are accustomed to. On the other side, new technology (e.g., the availability of graphical user interfaces) offers improved opportunities which are difficult to reconcile with the traditional product architecture. At this critical phase a competing product which does not have to cope with the past, can take full advantage of the new technology and can triumph. In many cases, the old product runs out and disappears.

Leading software packages are marketed worldwide at comparable prices. For example, we can order a software package directly from an US wholesale distributor at US wholesale prices from Vienna via FAX and get next day delivery via a Courier Service, affecting the payment with a credit card. Such distribution mechanisms force local software companies to sell similar products at matching prices. If a local software company has only a relatively small customer base, it can be difficult to recover the development costs if the product price is determined by "world-market" competitors.

The situation is significantly different if a know how intensive software package is aimed at a specialized market. In such a situation the marketing activities can be more focussed (and low-cost) and the competition is less severe.

2.3.2 Integrated software

"Key-element" software and "intelligent product" software is completely integrated within the product and distributed along the established product marketing channels. This form of software distribution avoids many of the pitfalls of commercial software distribution, such as the problem of illegal duplication, documentation, support, etc./Kop91/. Since the price of the computer hardware drops further and further, the strategic importance of the software in an intelligent product increases. Another advantage of integrated software is the generation of additional revenue by the sale of the associated product hardware.

2.3.3 Free software

In order to avoid the capital expenditures for marketing, advertising and commercial distribution, some authors of software packages decide to distribute their software freely via bulletin boards in computer networks. If a user is satisfied with a program, he is asked to register with the author (or his organization) and pay a small fee for extended documentation, support and future updates. Software which is distributed in this form is sometimes called shareware. The author of the shareware recovers his costs by the fees paid by satisfied users. He retains the copyright and in some cases restricts the use of the shareware, e.g., it may not be repackaged and sold commercially.

Another form of free software is public domain software, which is detached of any copyright restrictions. Public domain software originates often at universities and other research organizations and is freely distributed to the community. The quality and maintenance of public domain software can be a problem.

A thorough report covering many issues relating to free software has been published by UNIDO recently /Bot91/.

2.4 Risk factors

There are a number of risks associated with the production of software and the forming of a software industry. In this section we present a short list of some typical hazards encountered in the software field such that they can be avoided.

2.4.1 Inadequate personnel

There is a worldwide shortage in highly qualified software personnel--this is, at the same time, a chance and risk for developing countries. Starting a major software project with unexperienced and inadequately trained personnel is a sure route to disaster. Technical and management expertise are necessary to direct a successful software project.

2.4.2 Mistaken user needs

Many software projects have failed because the real user needs have not been captured adequately. Establishing a stable and sufficient set of requirements for a software system is extremely difficult, but of paramount importance. Constantly changing requirements can jeopardize any software project. In the field of contract programming the requirements are delivered by the clients, such that this risk factor is avoided.

2.4.3 Unmastered complexity

Software systems are complicated because of the inherent complexity of the problem domain and because of the extrinsic, needless complexity of the software design. This extrinsic complexity must be eliminated through the use of proper design methods (refer to the following chapter on the design of real-time software). Complicated software is expensive to build, difficult to verify and unreliable in its operation.

2.4.4 Poor quality software

A software product should only be released after it has been thoroughly verified. Both, the normal functions and the exception handling sections have to be tested painstakingly. The premature release of a new software version has destroyed well-known software companies.

2.4.5 Inadequate software marketing

The expertise and capital required to market a technically excellent software product is often underestimated. This topic has been discussed at length in the previous section. This risk is avoided in contract programming and, to a certain extent, in intelligent product software, since in this field the marketing is performed via established marketing channels.

2.5 Comparative evaluation

In this section we plan to analyze the different software segments to locate those segments which present good opportunities for software production in developing countries while minimizing the risks involved.

In the previous sections we have identified five important software segments:

(1) Prepackaged software
(2) Key element software
(3) Intelligent product software
(4) Contract Programming
(5) Software Maintenance

The fifth segment, software maintenance, is not open for competition and will not be considered any further in this analysis.

We have also recognized five important risk factors in software production:

(1) Inadequate personnel
(2) Mistaken user needs
(3) Unmastered complexity
(4) Premature release of the software
(5) Inadequate software marketing

Out of these five factors, two--inadequate personnel and premature release of the software--are critical for all types of software projects.

We will therefore compare the sensitivity of the software segments in relation to the three remaining risk factors, mistaken user needs, unmastered complexity and inadequate software marketing. The following matrix displays the level of risk according to our assessment:

	User needs	Complexity	Marketing
Prepackaged software	C	C	C
Key element software	M	C	M
Intelligent products	M	M	M
Contract programming	U	C	M

(C: Critical, M: Medium, U: Uncritical)

According to this comparison, contract programming and intelligent products software are the software production alternative with the lowest risk. It is followed by key-element software. According to this comparison, the prepackaged software market is most risky.

Contract programming is definitely an interesting software production alternative for developing countries. It requires a well educated software workforce, access to state of the art software development workstations and access to an international computer network. Since the prices of workstations have dropped drastically over the last five years, the capital investment for contract programming is relatively small. The major investment is in the educational field.

There is, however, a major disadvantages in contract programming. The well educated software workforce, which is the most precious resource, will not generate any secondary business activity. This is different in intelligent product software production, where in addition to the software a tangible product has to be manufactured, generating work for a less qualified workforce. The payoff of the investment in the educational field in relation to the amount of employment generated is thus substantially higher.

Intelligent product software is real-time software. Key element software is real-time software. It is likely that the software which has to be produced in contract programming is also real-time, since this is the growing software category in the nineties. In the third part of this report we will therefore investigate the present state of the art in the design of real-time software.

The most important government actions which have to be taken in order to establish the foundation of a software industry are investment in first-rate educational institutions and the provision of reliable access to international computer networks.

Building first-rate educational establishments in the fields of computer science can be a cooperative effort between developed and developing countries. Technical exchange programs in both directions can bring the required know how base into the developing countries. If such programs are supported by focussed financial incentives and investment funds then the basis for small software companies can be formed.

3. SOFTWARE MANAGEMENT ISSUES

In the last few years the techniques for managing software have been further refined. Professional software management is now established in most software companies and quality books about software management are available, such as the practical book by DeMarco "Controlling software projects" /DeM82/. In this section we focus on three software management issues which require special management attention and are sometimes overlooked when setting up a new software production organisation: quality management, risk management and productivity management.

3.1 Quality management

Since many purchasers of software are very concerned about the software quality, they start to require that a accredited quality management system is used in the development of the software they are buying.

The International Standard Organization has standardized such a quality management system which is applied to software and other service industries. In its ISO standard 8402 quality is defined as the "Totality of features and characteristics of a product, process or service that bear on its ability to satisfy stated or implied needs".

Quality attributes

We call those characteristics of the software which are relevant for the software quality **attributes**.

We distinguish between **functional** and **nonfunctional** quality attributes. Functional attributes are concerned with the mapping from the input domain to the output domain of a software system, i.e., the description of the system functions. Nonfunctional attributes refer to all other requirements not directly related to the systems functions, such as reliability, performance, adherence to development standards, development cost, etc.. In practice, the complete set of requirements is normally in partial conflict. For example, faced with a tradeoff between cost and reliability, the designer is forced to make a decision affecting quality attributes.

Quality can only be managed if the quality attributes are specified precisely in such a form that they can be measured and tested. General quality attributes, such as "the software system must be well-structured", are meaningless. Quality control refers to the operational techniques and activities to ascertain that the stated quality attributes are accomplished.

It has been recognized that quality has to be built in at the point of production and not after the production process at the point of inspection. Therefore modern software development processes integrate quality management with the software production process. In the development of safety critical software, the quality control agencies require the certification of the software production process as well as the certification of the software product.

Quality management system

Quality management starts with the introduction of a quality management system as part of the software production process. The quality management system provides the framework for a coordinated quality policy in an organization. It specifies the strategy and tactics which have to be followed to achieve the intended level of software quality.

In the following section we present an overview of the ISO standardized quality management system (ISO 9001) which is applied to software and other service industries:

(1) Management responsibility:

The organization must 'define and document management policy and objectives for and commitment to quality'. In particular, the responsibilities of all staff who perform and verify work affecting quality have to be defined. The documentation of all quality related activities has to be recorded in a corporate quality control manual.

(2) Contract review:

Any contract to produce software has to be reviewed from the point of view of quality management. In particular, the quantified quality attributes must be agreed between purchaser and supplier and must be documented in the contract. Tests how to measure the level of quality required must be contained in all contracts. The same applies to software acquired for the project.

(3) Design control:

The developer has to establish procedures in order to demonstrate the quality of the design at each design step.

(4) Inspection and testing:

Inspection and testing must take place during the development and the product status must be recorded at all times. Where appropriate, statistical techniques required for verifying the acceptability of product characteristics have to be established.

(5) Quality records:

The developer must ensure that sufficient records are maintained to demonstrate that the required quality has been achieved. The quality record keeping must make sure that no unrecorded quality actions took place. Change control must be an integrated activity of the quality management system.

(6) Internal quality audits:

The quality control system itself must be subject to periodic reviews to maintain its effectiveness.

(7) Training:

Training needs and training levels for all staff involved in the software production process must be specified and recorded.

In some countries, there are organizations to accredit quality control systems. For example, such an organization grants its stamp of approval that the objectives of the ISO 9001 standard is met by a particular quality management system installed in a given organization. In this area of quality management system approval a cooperation between industrialized countries and developing countries could be of benefit to both partners.

Quality control

Whereas the quality management system is concerned with setting up the framework for quality management, quality control is concerned with the operational techniques and activities executed in order to achieve the intended level of quality. i.e., that the quantified quality attributes of the software are met. Quality control is carried out on intermediate and final software products with the intent to uncover weakness in the preceding development process. Quality control can be decomposed in five activities /OUL91/:

(1) Define the software quality attribute and its measure.
(2) Define the attribute check procedure.
(3) Carry out the check procedure.
(4) Record the result
(5) Take and record any corrective action taken.

It is important that step 1 and 2 are carried out before the product is ready, i.e., in the requirements analysis phase or in the contract specification phase in contract programming.

If the specification of an intermediate software product is available in a formal notation with formally defined semantics, some properties of the product can be checked mechanically. However, if the intermediate product is not amenable to such automated checks, one has to rely on less formal checking techniques, such as checklists designed to help check for completeness. Birrel and Ould /Bir88/ contain extensive check lists for most of the major items produced during software development.

3.2 Risk management

There are many risks involved in software development which can manifest themselves as technical failures (impaired functionality, poor reliability) and management failures (schedule and cost overruns). Risk management is concerned with the identification, analysis and elimination of these risks before they effect the software project. Risk management involves two steps: risk assessment and risk control /Boe89/.

Risk assessment

Risk assessment is concerned with the identification and the analysis of the risks which are associated with a software project. We distinguish between generic risks, i.e., those which are common to all software development projects and specific risks which are those that apply only to a particular project.

Risk identification

Risk identification can start with the examination of general checklists for the most common generic and project specific risks. In section 2.4 we already have identified some of the most important generic risks such as

(1) Inadequate personnel
(2) Imprecise requirements
(3) Unmastered complexity
(4) Poor quality

In addition we have to consider project specific risks as for example:

(1) Imprecise description of the expected work
(2) Unrealistic project schedules and budgets
(3) Inappropriate staffing with lacking know how base
(4) Deficient communication among project members
(5) Poor project control
(6) Application of unproven technologies
(7) Insufficient hardware resources
(8) Ill-suited system software
(9) Unrealistic performance requirements
(10) Unstated or illusory assumptions

These checklists can only serve as a first starting point for risk identification. They have to be complemented by checklist based on local experience.

Risk analysis

Risk analysis starts with the rating of the identified risks in relation to the particular project. These rating has to reckon the criticality of the identified risk and guess the probability of its manifestation. In some situations it will be necessary to develop an analytical dependability model of the given system in order to assess the significance of a given risk factor. The result of the risk analysis is a weighted list of risk factors relevant for the particular project.

Risk control

Risk control is concerned with the determination of management actions to eliminate, or at least reduce, the identified risks. One first action will be the improvement of the management visibility of those aspects of the project which are prone to the risk. This can be done by requiring up to da documentation of the achieved progress and by the installation of project management baselines.

If a particular failure mode of a computer system is recognized as very critical, the risk reduction strategy can consist of increasing the resources for verification or the provision of software fault tolerance for a particular function. Furthermore, operational loss limiting techniques may be installed.

If a high probability of a schedule overrun is suspected, renegotiations with the client are sought in order to modify the completion date or to reduce the functionality which has to be delivered on time.

3.3 <u>Productivity management</u>

In a competitive environment those organizations will thrive which can deliver a given software product of high quality at the lowest cost. **Software productivity**, defined as the relation of software output to the cost of producing this software, is an important parameter of an organization and has to be managed explicitly.

Software productivity can be increased by either reducing the amount of work required to produce a given product or by increasing the effectiveness of the development staff. The analysis of software cost models gives valuable insights into the key factors determining software productivity. Boehm /Boe87/ has identified a number of such key factors:

(1) Staff effectiveness:

All cost models indicate that the selection, motivation and management of the people involved in a software project is a key productivity factor. Employing the best people is a good strategy, because the productivity ranges of people is normally much wider than their salary ranges. Continuous training of the personal in technical and managerial matters has a high productivity payoff.

(2) Simple products:

During the architecture design phase, every effort has to be made to decompose the system into components which can be designed, implemented and tested independently. The interfaces between components have to be clearly specified and should be free of side effects. If a number of design alternatives to implement a given requirement are available, understandability should be a prime selection criterion. If there are open questions about the implementation of a key software component, rapid prototyping should be considered to learn about the difficulties of the solution.

(3) Modern development techniques:

The application of modern software development techniques, such as object oriented design techniques combined with an integrated tool support which covers technical as well as managerial aspects in an integrated fashion can increase the software productivity and avoid unnecessary clerical rework to accommod te software changes. Automated support tools, such the provision of an integrated documentation system, can eliminate costly development steps. A good description of the present state of the art in modern development techniques and tool support is contained in /Ald91/, where more than ten support environments are discussed. The selection of the most appropriate support environment for a particular organization depends on the type of software produced and the programming methodology/languages chosen. It is a difficult task which requires careful analysis.

(4) Reuse components:

The Lines of Code (LOC) which are produced is still considered a reasonable metric for software size. If this size can be reduced by the reuse of software components or the employment of application generators the software productivity is increased. Some software organization monitor regularly the software reuse factor, i.e., the LOC which have been taken from software libraries and reused related to the total LOC delivered. Software reuse requires careful management planning. In a "building up" phase standards for software reuse are established and reusable software components are identified and classified in a reuse database. In the "design" phase software is selected from this reuse database on the basis of the given requirements profile. There are a number of support systems for software reuse available, which are described in some detail in/Hal91/. Software reuse can be organized on a wider scale than just within a company. If a number of organizations--national or international--agree on a standard reuse database, they can all benefit from such a joint effort.

The level of achieved software productivity is an important measure of success of any commercial software organization. Productivity and quality issues have to be key items on the priority list of top management.

4. THE DESIGN OF REAL-TIME SYSTEMS

A real-time system has to meet the deadlines dictated by its environment. If a real-time system misses a deadline, it has failed. Designing real-time systems is challenging: the functional specifications must be met within the specified time constraints.

Temporal properties are system properties. They depend on the entire system architecture, i.e., the structure of the application software, the scheduling decisions within the operating system, the delays of the communication protocols, and most important, on the performance characteristics of the underlying hardware.

In the following section we discuss the two contrary design methodologies, best effort design and guaranteed timeliness design. Subsequently we give an overview of a rigorous methodology for the design of real-time systems and introduce the RT-transaction as the unifying concept between specification and implementation. After a short discussion of the architecture design the issue of scheduling is treated in some detail. Finally, the topic of testing is examined.

4.1 Real-time system design methodologies

There are two approaches to the design of real-time systems, the best effort approach and the guaranteed timeliness approach.

The best effort approach assumes that a definitive specification of the load- and fault-hypotheses is not available and the analysis of all worst case situations is not workable. It therefore endorses a flexible architecture adaptive to the evolving execution situations. The design process is guided by heuristic rules taking into account the average execution times and input rates. Not before the final integration phase it can be established by testing whether the required temporal properties are satisfied. The verification of the temporal properties can only be performed probabilistically.

Many of the known methodologies for the design of real-time systems are based along the best-effort paradigm. Some approaches have tried to enrich existing conventional methodologies with mechanisms for the specification of time-related properties, e.g., SDRTS (Structured Design of Real-Time Systems) /WAR86/, and DARTS (Design Approach for Real-Time Systems) /GOM86/. Other real-time system design methodologies have been built from scratch, such as SDL (Specification and Description Language) for telecommunication systems /BEL89/, MASCOT (Modular Approach to Software Construction, Operation and Test) /SIM86/ and SREM (Software Requirements Engineering Methodology) /ALF85/.

In comparison, the methodology for the guaranteed timeliness design is of later origin and not covered as well in the literature as the best effort methodology. The guaranteed timeliness methodology is more systematic and rigorous than the best effort methodology. Given the same requirements specification, it will lead to a more intelligible design which is easier to verify.

The guaranteed timeliness approach /Kop91b/ is based on the assumption that the peak load scenario (the load hypothesis) and the worst case fault scenario (the fault hypothesis) are contained in the requirements specification. It advocates a static architecture which must be capable to handle all conceivable input cases within the specified time-constraints, provided the specified load and the fault-hypothesis are not violated. During the design the worst case situations concerning program execution times and task activation rates must be analyzed exhaustively.

In practical applications there is a need for both design approaches. There are complex high level requirements--often ill specified--where the average performance is important and an occasional failure to meet a deadline can be tolerated. In such an application a best effort design is viable. However, for other requirements where a failure is expensive to fix or can have catastrophic consequences, the guaranteed timeliness approach is better suited.

4.2 Real-time transactions

Let us now introduce the concept of a real-time transaction. A RT-transaction refers to the execution of a sequence of processing and communication actions between a stimulus from the environment and the corresponding time-constrained response(s) to the environment. The stimulus of a RT-transaction is the activation of a distinguished state or state-change of specified state variables. The following are examples for simple RT transactions: "If the temperature and the pressure in a vessel are above a designated limit for more than 50 milliseconds, a control valve has to be opened within a 200 millisecond period" or "If motor 1 is running and push-button A is pressed for three seconds than start up motor 2 within 1 second.

A RT-transaction expresses all concerns about functionality and timeliness of a sequence of computational and communication steps in a single concept. The simplest RT-transaction consists of two communication steps and one computational step: In the first communication step a sensor value is accessed, then, in the computational step a new setpoint is calculated, and in the final communication step an output value is delivered to a transducer. Complex RT-transactions can have mane interleaved communication and computational steps.

At the requirements level, two time parameters are associated with every RT-transaction. The maximum response time, MART, denotes the maximum real-time which may elapse between the stimulus and the corresponding response. The minimum interval time, MINT, indicates the minimum time which may pass between two RT-transactions instances of the same type.

RT-transactions are not only utilized at the requirements level. The are also used during the architecture design to decompose the requirements and at the runtime to describe the operational behavior of the RT-system. Thus the RT-transaction is a unifying concept covering all design phases.

4.3 Architecture design

During the architecture design the RT-transactions specified at the requirements level have to be decomposed into subtransactions for the computational steps and messages for the communication steps until finally every RT-transaction is reduced to a sequence of task executions and message exchanges. The tasks have to be allocated to processors and the messages have to be allocated to communication channels.

In order to improve the regularity and predictability of the RT-system it is advantageous to introduce a synchronized time grid of appropriate granularity system-wide. The granularity of the time grid has to be chosen such that it is in agreement with the maximum response times and minimum interval times of all RT-transactions and the performance parameters (e.g. delay characteristics) of the communication protocols. Significant system actions, such as scheduling a task or sending a message are only started at the grid points of the time grid. External state changes, which can initiate a new RT-transaction, are also recognized at these grid points by polling the external state variables. This mechanism of polling implements an implicit flow control schema such that the RT-system cannot be flooded with external requests. It is up to the instrumentation to guarantee that significant state changes are stored until the next polling cycle.

The decomposition of RT-transactions into tasks and messages has to proceed until all synchronization requirements can be resolved at the message level. There are no remaining synchronization points within a task, i.e., whenever a task is started it can terminate without waiting for synchronization conditions dependent on the execution of other tasks. The computational model for a task is thus straightforward. Every tasks needs a defined set of input messages and produces a defined set of output messages after a real-time interval, called the task execution time. This task execution time depends only on the task code and the capability of the hardware. In the design phase the worst case task execution times have to be estimated. During the implementation and testing phase they have to be calculated analytically and measured experimentally.

4.4. Scheduling

Given a set of RT-transactions and set of hardware resources it must be decided by the scheduling subsystem which RT-transactions or parts thereof--the tasks and messages--should be executed on a particular processor or communication channel at any given point in time. The proper design of the scheduling subsystem is therefore of critical importance for the timeliness of the real-time system.

There are basically two different approaches to the solution of the scheduling problem: the dynamic approach allied with the best effort design methodology and the static approach allied with the guaranteed timeliness design methodology.

Dynamic scheduling

In the dynamic approach the scheduling decisions are made at run time on the basis of the momentary system load and given system parameters. It is up to the run-time scheduler, which has to operate under real-time constraints, to produce schedules which on the one side will meet all synchronization and mutual exclusion requirements of the transactions and on the other side will meet the given deadlines.

Most practical real-time systems rely on a simple priority controlled dynamic scheduler for CPU allocation. Communication channel access is controlled dynamically either by polling, a token protocol or by a carrier sense multiple access (CSMA) protocol. In many cases the task priorities are static, i.e., they are determined at compile time. Whenever a set of tasks is ready at run time, the run-time scheduler selects the task with the highest static priority. Mutual exclusion and task synchronization are achieved by explicit synchronization between the tasks, e.g., by the use of semaphores or signals. For such a system it is very difficult to guarantee that all hard deadlines will be met, since undesirable phenomena, such as priority inversion, can occur. It is even difficult to guarantee the deadline for the highest priority task if task preemption is constrained due to mutual exclusion requirements.

Liu and Layland /Liu73/ have shown that static priority dynamic scheduling is optimal if the following assumptions are satisfied:

(1) All tasks are independent
(2) All tasks are periodic and the deadline equals the period.
(3) The sum of all CPU utilizations is less than ln 2.

Since the independence assumption (1) is unrealistic for practical systems--tasks have to cooperate to achieve a common objective--recent research has focused on scenarios where task executions are constrained by mutual exclusion requirements. It has been shown /Mok83/ that in the general case it is impossible for a dynamic scheduler to find optimal schedules. Restricted solutions have been reported by /Sha90/. It can be difficult to enforce all the required restrictions at run time.

Static scheduling

In the static approach all scheduling decisions are made at compile time and handed over to the run-time dispatcher in the form of a scheduler database, parameterized with the global grid points of time. To keep the size of this scheduler database manageable, it is wise to introduce some basic system cycle, after which the schedule is repeated. This requires that a period is assigned to every RT-transaction such that the peak load (minimum MINT) can be serviced without violating the MART requirement. At any given grid point in time the run-time dispatcher has to look up the scheduler database to find out which RT-transaction or task has to be selected for service next. If a transaction, planned for a given point in time, does not require any service, then the reserved time slot on the processor or communication medium will be allocated to a ready background task. Since all requirements concerning task synchronization and mutual exclusion are already considered implicitly in the scheduler database we call this form of task synchronization implicit synchronization. Phenomena like priority inversion are avoided.

The scheduler database which has to be interrogated at run time, has to be generated at compile time by an off line scheduling algorithm. This algorithm has to find a feasible schedule for allocating the CPUs and the communication medium for the specified set of RT-transactions. For this purpose, the scheduling problem is represented as a guided search through a search tree. Heuristic search strategies with well studied properties can be used to direct the search by omitting useless subtrees of the search tree or by retracting the search at 'promising nodes'. For a detailed description of such an off-line scheduler see /Foh90/.

Schedule switch

The problem of giving fast service to aperiodic emergency transactions whenever they occur while still maintaining a reasonable resource utilization in normal situations has led to the concept of a schedule switch.

Let us assume there exists a set P of operational phases of a system and that at any time the system can only be in any one of these phases, e.g., the startup phase, the normal processing phase, the termination phase, an emergency phase, etc.. For every phase Pi a corresponding schedule Si is developed. All these schedules are developed off-line and will guarantee that all tasks meet their deadlines. Whenever an aperiodic event occurs which requires immediate response, ie. the transition to another phase or an emergency service request, the scheduler will switch to the new schedule which has been designed for this new scenario.

4.5 Fault-tolerance

In real-time applications the correct results must be delivered in time, even after a fault has occurred. This can be realized by the implementation of active redundancy, i.e. two fail-silent computers operate in state synchronism in parallel and produce the same results. If software faults have to be tolerated, the parallel execution of diverse software versions with acceptance tests can be implemented.

Active redundancy requires replica determinism, i.e., all decision in the two node have to be identical. Replica determinism can be destroyed by uncoordinated access to the local time base, by dynamic scheduling decisions and by algorithms based on random number generators, such as ETHERNET. Guaranteed timeliness architectures, which are driven by a synchronized global time, are better suited for the implementation of active redundancy than best effort architectures which cannot maintain replica determinism without application specific overhead code.

4.6 Testing

More than 50% of the resources required for the development of a real-time system is spent on testing. Testability is thus an important criterion for the selection of an architecture.

The confidence in the timeliness of a RT-system based on dynamic scheduling can only be established by extensive system tests on simulated loads. Testing on real loads is not sufficient, because rare events, which the system has to handle (e.g., the occurrence of a serious fault in the controlled object), will not occur frequently enough in an operational environment to gain confidence in the peak load performance of the system. The predictable behavior of the system in rare-event situations is of paramount utility in many real-time applications.

Since no detailed plans for the intended temporal behavior of the tasks of a dynamic system exist, it is not possible to perform "constructive" performance testing at the task level. In a system where all scheduling decisions concerning the task execution and the access to the communication system are dynamic, no temporal firewalls exist, i.e., a variation in the timing of any task can have consequences on the timing of many other tasks in different nodes. The critical issue during the evaluation of a dynamic system is thus reduced to the question, whether the simulated load patterns used in the system test are representative of the load patterns that will develop in the real application context. This question is very difficult to answer with confidence. Field maintenance data indicate that some application scenarios cannot be adequately reproduced in the simulated system test [Geb88].

In a system based on static scheduling, the results of the performance test of every system task can be compared with the established detailed plans. Since the time-base is discrete and determined by the granularity of the time grid, every input case can be observed and reproduced in the domains of time and value. Therefore testing of static systems is more systematic and constructive. To achieve the same test coverage, the effort to test a dynamic system is much greater than that required for the testing of the corresponding static system. The difference is also caused by the smaller number of possible execution scenarios that have to be considered in a static system, since in such a system the order of state changes within a granule of the observation grid is not relevant [Sc90a].

4.7 Conclusion

RT-systems of medium complexity--most intelligent product software falls into this category--should be designed according to the guaranteed timeliness methodology. This methodology, based on static mechanisms, produces simpler structures than the best effort methodology based on dynamic mechanisms. Static systems are easier to understand and test and therefore will support a higher level of dependability.

5. SUMMARY AND CONCLUSIONS

(1) Already today, software production is a major industry worldwide. Its impact on the economy of all developing and developed countries will increase significantly over the next ten years.

(2) The likely technological advances in the field of computer hardware will open new opportunities and expansive markets for software products in new fields like multimedia system, intelligent products and others.

(3) Highly qualified, well trained and fully motivated personnel form the basis of any software industry. Because productivity ranges are much higher than salary ranges, it wise to attract the best people by offering extraordinary working conditions.

(4) The market of prepacked software, which accounts for less than 10% of the software activities, is very competitive and risky.

(5) Intelligent products which integrate mechanical subsystems and software controlled subsystems into a compact functional unit, offer an interesting new mechanism for the distribution of "integrated" application software.

(6) The well educated software workforce which develops the integrated products software generates additional employment opportunities for the less educated workforce producing the intelligent product hardware.

(7) Contract programming can be an interesting software activity in developing countries. It requires access to an international computer network and a good software development infrastructure.

(8) Software maintenance, which forms the major segment of all software activities, is not open for outside competition.

(9) Software distribution into the open market is capital intensive and difficult. Special markets which rely on a solid know-how base of the supplier are easier to tackle.

(10) Developing countries should consider and take advantage of the free software market.

(11) The key risk factors in the production of software are inadequate personnel, mistaken user needs, unmastered complexity, poor software quality and inadequate marketing.

(12) Software project planning and control is a mature discipline which has been established in most software production organizations.

(13) Explicit quality management is becoming an important asset of a software production organization. Many purchasers of software require and accredited quality management system at the site of the software developer.

(14) Software quality management system accrediting can be a topic of cooperative effort from developing and developed countries.

(15) Formal risk management should be established in order to reduce the organizational risks in software development.

(16) An systemized procedure for software reuse can increase the software productivity of an organization significantly. Software reuse can be a topic of cooperation among a set of related software suppliers.

(17) Although real-time software is a key software market of the future, the software development methodology and the software tools for real time software are not as well developed as those for commercial software. This is a challenge and opportunity at the same time.

(18) There are two competing design methodologies for the design of real-time software, best effort design and guaranteed timeliness design.

(19) Best effort design is more flexible but cannot guarantee timeliness properties under peak load.

(20) Guaranteed timeliness design is more rigorous and systematic and supports testability in a superior way.

(21) The implementation of active redundancy in order to provide fault-tolerant operation is straightforward in systems based on guaranteed timeliness design. It is very difficult in best effort systems.

(22) In applications of medium complexity, such as intelligent product software, static time-triggered software architectures are optimal from the point of view of understandability, testability and dependability.

(23) The most important political actions that have to be taken in developing countries in order to establish the foundation of a software industry are investment in first rate educational establishments and the provision of reliable access to international computer networks, such as Internet.

(24) The market for contract programming can be developed by proposing cooperative agreements for contract programming with companies from developed countries whenever they invest in developing countries.

6. CONCRETE MEASURES

In this section some concrete measures to promote the creation of a software industry in developing countries will be presented. The focus will be on contract programming and intelligent product software.

The idea behind this blueprint is the creation of a central technology group which will assist the developing countries in the buildup of the infrastructure and know-how of a local software industry, similar to the International Center for Microelectronic Applications and software proposed by /Schware/. This central technology group, we call it a central technology clearinghouse, will enter into appropriate agreements with a local technology nucleus, situated in a cooperating developing country. After a successful buildup phase, the local technology nucleus should be in position to handle its own affairs, but still maintain a link to this clearinghouse for technology updates and quality control. The clearinghouse can then redirect its resources to assist another country in the development of its software industry.

In order to accomplish this goal, the following concrete steps are necessary:

6.1 Establish a central software technology clearinghouse

This central software technology clearinghouse could be situated at the UNIDO in Vienna. It should be a small group of unbureacratic professionals who act as coordinators and take advantage of external experts and services whenever needed. This software technology clearinghouse should perform the following major functions:

(1) Advise the participating country in the setup of a technology nucleus.

(2) Provide technical and contractual assistance in hardware and software acquisition, particularly to small and medium sized enterprises.

(3) Organize the local software training and arrange qualified instructors who will conduct the training at the local site.

(4) Act as a central clearinghouse in respect to public domain software, standards, new products etc. Arrange access to appropriate data-banks which can be queried by electronic mail from the developing countries.

(5) Establish contacts with computer companies and other clients interested in cooperating with the developing countries. Investigate opportunities for contract programming.

6) Assist the local technology nucleus in the selection of appropriate pilot projects.

(7) Assist the local technology nucleus in the setup of a software

quality management program and act as a software quality accreditation body.

(8) Monitor the software productivity of the local software development organization.

(9) Assist the local software organisation in the international product marketing.

(10) Act as a "service hotline" to resolve upcoming problems and disseminate relevant technology and marketing information.

6.2 Establish a local software technology nucleus

A cooperating developing country must organize a local technology nucleus which will be the partner of the central technology clearinghouse. After a successful buildup phase, financed by public funds, this technology nucleus should form the core of a new software company which will operate under its own profit/loss responsibility.

Immediately after the decision to proceed with the startup of a software industry has been made, two highly qualified local people should be selected as future mission leaders. They have to be sent to the software technology clearinghouse for a training period of at least one year. After returning to their home country, they have to perform the following tasks:

(1) Draft a detailed business plan with many milestones and quantified goals. Monitor the achievements in relation to the business plan.

(2) Select and recruit the proper highly qualified staff. This should be done in cooperation with the leading educational institutions.

(3) Select and purchase the hardware/software platform in close cooperation with the software technology clearinghouse.

(4) Organize the theoretical and practical training in cooperation with the software technology clearinghouse.

(5) Select the first pilot application in close cooperation with a local client, possibly some public institution. Utmost care must be taken when selecting the first pilot application. This pilot application should address a genuine user need. It should be developed under the guidance of the technology clearinghouse to guarantee highest quality and ease of use.

(6) Invest significant effort in marketing activities and prepare to operate autonomously in the local and international market.

6.3 Establish a reliable electronic-mail service

This whole schema will only work, if a reliable electronic mail service can be established between the central technology clearinghouse and the local nucleus. This task must me performed by the PTT in the developing country. This e-mail link is used to transfer specifications and programs, to provide immediate assistance in case of questions, to support access to the international data bases, and to disseminate relevant information to the local organisations.

Given the present work patterns in the international software community, such an e-mail service is absolutely essential. In case this service cannot be provided, it is better not to start the whole effort!

6.4 Monitor software quality and productivity

It has been stated repeatedly that only highest quality computer products will succeed in the competitive world market. It is therefore essential that a state-of-the art quality control program, which is monitored by external experts, is put into place from the very beginning. This requires a close cooperation between the central technology clearinghouse and the local software organization.

7. REFERENCES

[Alf85]
Alford, M., SREM at the age of eight, IEEE Computer, Vol 18, Nr. 4, 1985, pp. 36-46

[Ald91]
Alderson, A., Configuration Management, in: Software Engineering Reference Handbook, ed. by J.A.McDermid, London 1991, pp.34.1-34.17

[Bel89]
Belina F., Hogrefe, D., The CCITT-specification and description language SDL, in Computer Networks and ISDN systems 16, Elsevier Science Publishers, 1988/89, pp.311-341

[Bir88]
Birrel, N.D., Ould, M.A., A Practical Handbook for Software Development, Cambridge University Press, 1988

[Boe87]
Boehm, B.W., Improving Software Productivity, IEEE Computer,Sept. 1987, pp.43-57

[Boe89]
Software Risk Management, IEEE Tutorial, IEEE Press, 1989

[Bot91]
Bothelho, A.J.J., Emerging issues in the selection and distribution of public domain software for developing countries, Report prepared for UNIDO, Vienna, Austria, April 30, 1991

[Bul91]
Bulekley, W.M., Technology, economics and ego conspire to make software difficult to use, The Wall Street Journal, Technology section on software, R8, May 20, 1991

[Bus91]
Business Week, The World most valuable companies, July 15, 1991, pp. 43-80

[DeM82]
DeMarco, T., Controlling Software Projects, Yourdon Press, N.Y, 1982

[Foh90]
Fohler, G., Koza, C., Heuristic scheduling for distributed real-time systems, Research Report 6/89, Institut für Technische Informatik, TU Wien, Austria

[Geb88]
Gebman, J., Mciver, D., Shulman, H., Maintenance data on the fire control radar, Proceedings of the 8th AIAA Avionics Conference, San Jose, Cal. Oct. 1988

[Gom86]
Gomaa, H., Software development of real-time systems, Comm. ACM, 1986, Vol.

29, Nr. 7., pp. 657-668

[Hal91]
Hall, P., Boldyreff, C., Software reuse, in: Software Engineering Reference Handbook, ed. by J.A.McDermid, London 1991, pp.41.1-41.12

[Kop87]
Kopetz, H., Ochsenreiter, W., Clock Synchronization in Distributed Realtime Systems, IEEE Transactions on Computers, August 1987, pp.933-940

[Kop89]
Kopetz, H., Damm, A., Koza, C., Mulazzani, M., Schwabl, W., Senft, C., Zainlinger, R., Distributed Fault-Tolerant Realtime Systems: The MARS Approach, IEEE Micro, Vol. 9, Nr. 1, pp., 25-40, Febr. 1989

[Kop90]
Kopetz, H., The production of intelligent products in developing countries, Microelectronics Monitor, UNIDO Vienna, Issue Nr. 29, January 1990, pp.63 - 71

[Kop91]
Kopetz, H., Zainlinger, R., Fohler, G., Kantz, H., Puschner P., Schütz, W., The design of real-time systems: from specificationto implementation and verification, Software Engineering Journal, May 1971, pp. 72 - 82

[Liu73]
Liu, C.L., Layland, J.W., Scheduling Algorithms for Multiprogramming in a Hard Realtime Environment, Journal of the ACM, Februay 1973, pp.46-61

[Oul91]
Ould, M.A., Quality Control and Assurance, in: Software Engineering Reference Handbook, ed. by J.A.McDermid, London 1991, pp.29.1-29.12

[Roo91]
Rook, P., Project Planning and Control, in: Software Engineering Reference Handbook, ed. by J.A.McDermid, London 1991, pp. 27.1-27.36

[Schware]
Proposal for an international center for microelectronic applications and software (MAS),UNIDO-Report,Vienna

[Sha90]
Sha, L., Rajkumar, R., Lehoczky, J.P., Priority Inheritence Protocols: An Approach to Real-Time Synchronization, IEEE Transactions on Computers, Vol. 39, No. 9, Sept. 1990, pp. 1175-1185

[War86]
The transformation schema: An extension of the data flow diagram to represent control and timing, IEEE Trans. on Software Engineering, Vol 12, Nr. 2, 1986, pp.198-210