

https://dspace.org/

Software development for embedded systems

Bapoo, Hansraj

2016-07-14
http://hdl.handle.net/10539/20577

Downloaded from a DSpace repository. https://dspace.org/

courses, access to the local area network as well as access to the internet and e-mail facilities.

The development of the case study has been conducted at the author's employer's premises, i.e. at Meissner, Johannesburg. Resources utilised there, include development tools for the microcontroller used, a three-phase inverter, use of personal computers and the use of laboratory facilities.

■

systems development. This project is limited to small embedded systems development, where the software constitutes a small part of the overall system and is usually executed by small programmable devices, such as microcontrollers. Typical applications that fall under this category are automotive electronics, power electronics and household appliance control. The scope of the project is captured in the three tasks below:

- To define embedded systems and assess the difficulties currently experienced in their development
- To provide and document a practical example of embedded system development
- And finally to formulate a development approach for embedded systems, based on the previous two activities.

The development of a microcontroller system to control a three-phase inverter for an uninterruptible power supply (UPS) is used as a case study for the project. Note that although the case study is performed as far as possible in a real-type environment and conditions, its development is not the main focus of this project. The purpose of the case study is to provide practical experience into the development of embedded systems. It is used as a practical foundation for the formulated development approach.

1.3 Main Deliverables of the Project

The main deliverables for the project are structured around the three tasks identified above. In particular, the following products are provided:

- Documentation in support of the literature survey
- Full documentation of the case study development
- A review of embedded systems development

1.4 Resources and Environment of the Study

The resources for this project are captured in the Project Management Plan, document SDE 005. As a general remark, we note that most of the literature research work has been carried out at the University of the Witwatersrand, Johannesburg. The university provided access to all library services, the facility to attend any relevant conferences and

1 Definition of the Project

1.1 Statement of the Problem

The software development for small embedded systems is often regarded as constituting a very small part of the overall tasks involved in the making of the final product. Traditionally, embedded software has consisted of only a few lines of codes, mostly written in assembler. However, advancement in microcontroller technology is constantly extending the computing power and speed of the devices while reducing their physical size, cost and power consumption. The new developments in hardware technology are extending the use of microcontrollers into more and more complex embedded applications. The popularity of embedded systems in turn, is drawing more players into the market. However, the evolution of the market is not being accompanied with an equal maturation of the organisations. The development of small embedded systems is still mostly being carried out without any defined process, with very little emphasis on quality management practices.

On the other side, there is a tendency in the embedded systems community to acquire the same development tools and facilities that are used in larger data processing software applications. For example, there is an increasing use of high-level languages in embedded systems. It is believed that this is the way to manage the increasing complexity of embedded systems software. The question is: with the investigation of more complex applications and with the introduction of new tools, can embedded systems development still be carried out in the same unstructured approach? And if not, are models currently used for large data processing projects applicable to embedded systems development?

In essence, the problem examined by this study, is the reconciliation of the formal development processes with the technical realities of embedded systems.

1.2 Scope of the Project

The objective of the project is to study the problem exposed above. The project starts with an investigation of current software practices. Based on the practices identified in this exercise, a practical embedded system development is undertaken. The practical development is not strictly structured around any specific model, so as to encourage the formulation of an approach that will take into consideration the need for a disciplined approach as well as the technical realities of embedded

In the second part a complete embedded system development is presented. The case study involves the development of a microcontroller system to control the three-phase inverter of a UPS. This section is supported by documents SDE 104, 105, 106, 107, 108, 110 and 112. SDE 104 is the requirements analysis. SDE 105 some of the inverter control techniques and the phase-locked loop mechanism. It follows directly from the technical nature of the case study and supports the functional specifications, captured in document SDE 106. The analysis and design, both implemented using an object-oriented approach, are documented in SDE 107 and SDE 108 respectively. SDE 110 records the listing of all the major iterations of the design. It also describes the hardware platform. Lastly, SDE 112 documents the results of the final testing exercise.

The final section consists of a single document, SDE 113. This document revises the subject of embedded systems in the light of the case study development. Combining the experience gained in this exercise with the results of the literature surveys performed in the first part, a risk-based, hardware-software co-design approach for embedded systems development is formulated and described. The document ends with a reflection on some of the possible future developments in the domain of embedded systems.

FOREWORD

The format for this Masters Dissertation differs from the conventional format in that it comprises a short body (in the form of a project overview, document number SDE 114, and two papers, document numbers SDE 115 and SDE 116) and a number of appendices. The substance of this project is in documents comprising the appendices. It is therefore considered helpful to provide the reader with guidance regarding the order in which these should be reviewed.

A definition of the project, together with its objectives and scope, are provided in the project overview (document number SDE 114).

For a broad perspective on this project and its major contributions, the reader is directed to the two papers entitled "A Risk-Based Approach to Embedded Systems Development" (SDE 115) and "Space Vector Modulation Inverter Control: An Object-Oriented Application" (SDE 116) respectively. Although the project investigates many aspects of embedded systems, e.g. aspects common to software development in general, development strategies, object-oriented techniques, as well as practical development issues, SDE 115 and SDE 116 attempt to provide the reader with a global view while evaluating the effect of individual factors.

Document SDE 117 reviews the contribution of each of the technical product towards the objectives of the study and then draws concluding remarks about the whole project. It also provides guidance to future work.

A list of the documents (textbooks and articles) referenced in this dissertation, together with some additional reading sources are provided in SDE 118.

The appendices are discussed below:

The Master Document List (MDL), document number SDE 001, is a register of all documents created within this project. The documents are classified as management products, technical products, and quality assurance products. All the documents listed in the MDL were important in defining the project and in supporting its development. However, for the purpose of the dissertation, only a selection of the most pertinent and contributory documents have been included.

The Project Management Plan, document number SDE 005, provides an overview of all the resources managed in the development of this project. It captures the main tasks performed against a time scale.

The remaining documents are technical products and can be divided into three parts. The first part documents the results of the literature survey, firstly on software development in general and then more specifically on embedded systems. This section is supported by documents SDE 102 and SDE 103 respectively.

Software and Hardware Implementation Documentation ..... SDE 110

Testing Documentation ..... SDE 112

Appendix Group IV: Technical Products - Review

Embedded Systems: A Review ..... SDE 113

TABLE OF CONTENTS

Page Number/
Document Number

Declaration ii

Abstract iii

Dedication iv

Acknowledgements v

Table of Contents vi

Foreword viii

Project Overview ..... SDE 114

MSc Paper 1 - H Bapoo ..... SDE 115

MSc Paper 2 - H Bapoo ..... SDE 116

Discussion and Conclusions ..... SDE 117

References\ Bibliography ..... SDE 118

Appendix Group I: Management Products

Master Document List ..... SDE 001

Project Management Plan ..... SDE 005

Appendix Group II: Technical Products - Literature Survey

Software Development: A General Literature Survey ..... SDE 102

Embedded Systems: A Literature Survey ..... SDE 103

Appendix Group III: Technical Products - The Case Study

Requirements Analysis ..... SDE 104

Inverter Control Techniques and Phase-Locked Loop ..... SDE 105

Functional Requirements ..... SDE 106

Software Design Specifications -Part 1 ..... SDE 107

Software Design Specifications -Part 2 ..... SDE 108

ACKNOWLEDGEMENTS

My sincere thanks to:

Dr. Barry Dwolatzky, whose supervision and encouragement are gratefully acknowledged.

Meissner for the facilities provided.

The R&D team at Meissner for the technical support.

My brother, Bhimal, whose constructive remarks were much appreciated.

DEDICATION

To fulfilled dreams, to Marie-Ange, and to all those who believed in me.

ABSTRACT

This report examines the major factors influencing the development of embedded systems. The discussion covers, among other aspects, features of embedded systems that are common to software development in general, the main characteristics of embedded systems, some of the current development strategies, the microcontroller market, language issues and development tools.

Following the assessment study and a practical embedded system development, the report concludes that although embedded systems are typified by relatively few lines of codes, their complex nature necessitates the application of a disciplined development approach. The report presents a risk-based hardware-software co-design development approach in an attempt to reconcile the existing formal development models to the technical realities of embedded systems.

A practical embedded system case study is presented. It involves the development of a microcontroller-based system for the three-phase inverter of an uninterruptible power supply (UPS).

DECLARATION

This project is being submitted for the Degree of Master of Science in Engineering in the University of the Witwatersrand, Johannesburg.

I declare that this dissertation is my own unaided work. It is being submitted for the Degree of Master of Science in Engineering in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other University.

_____

(Signature of candidate)

_____ day of _____ 199 _____

SOFTWARE DEVELOPMENT FOR EMBEDDED SYSTEMS

Hansraj Bapoo

A dissertation submitted to the Faculty of Engineering, University of the Witwatersrand, Johannesburg, in fulfilment of the requirements for the degree of Master of Science in Engineering.

Johannesburg, 1996

subprograms, grouped together because of their common involvement in the start up process. To co-ordinate the activities of those two classes, a third class, systemConfig, is introduced. It uses both classes. All three classes are described in the CRC cards below.

startUp
Responsibilities: Contains subprograms responsible for setting the system at start up and for generating an NMI routine upon an overload condition. Collaborators: - systemConfig

lookUpTable
Responsibilities: Provides a look-up table of switching times values (for usMag = 1) Collaborators: - systemConfig, switchingTimes

systemConfig
Responsibilities: Configures the system, i.e. the software and the micro-controller appropriately. Collaborators: - lookUpTable, startUp

Figure 6 shows the class diagram for those three classes.

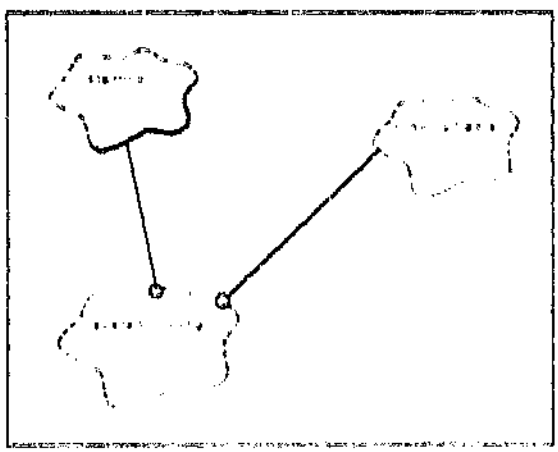


Figure 6: The initialisation procedure classes

The phase-locked loop idea, as we saw it, can be decomposed in two abstractions, phaseDetector and phaseDiffReducer. The first monitors phase difference, the second sets a phase difference reduction mechanism by adjusting the inverter output frequency. The two are described below.

phaseDetector
Responsibilities: keeps track of the difference between reserve and inverter output. Collaborators: - phaseDiffReducer, fSensor, vSensor, vectorMagnitude

phaseDiffReducer
Reponsibilities: Reduces the phase difference between reserve and inverter output. Collaborators: - pllSupervisor, phaseDetector

The co-ordination of the phase difference detection and reduction processes is allocated to a third class, pllSupervisor.

pllSupervisor
Responsibilities Co-ordinates the activities of the phase difference detection and reduction mechanisms. Collaborators: - phaseDiffReducer, angleDisplacement

The phase-locked loop classes are shown together with the classes of the main cycle in figure 8.

As discussed previously, the generation and update of the output port values are managed by a timer interrupt cycle. A single timer could be used for the four periods in a sampling time interval (figure 4 above). However, since the microcontroller contains five timers that are otherwise unused, we decide to use four of them, one four each period. This creates a better separation of concern, and at no cos. This decision gives us four timer classes, timerT3Interrupt, timerT0Interrupt, timerT1Interrupt and timerT2Interrupt. All four classes are related to a parent abstract class, timerInterrupt. The

Design Strategies

Hardware implemented phase-locked loop usually make use of a voltage-controlled oscillator (VCO). In a software scheme, phase difference control can be achieved by monitoring the instantaneous voltage difference between reserve and inverter output, and the direction of the two waveforms.

The switching set in figure 4 shows that the sequence of the output states has to be reversed after each sampling interval, except on a sector change when the sequence is unchanged. The sector location of $\underline{U}s^*$ is known from monitoring the absolute angular displacement, θ, relative to $\underline{U}_1$. θ is reset after every complete revolution of $\underline{U}s^*$.

A quick iteration through the design-coding-testing loop reveals that a purely on-line computation approach is not feasible, because of the stringent real-time constraints. Instead, a combined look-up table and on-line comp-utation method is used. The look-up table stores the values of the parameters ta and tb (equations 4 and 5) over a given number of values of α, at a value of usMag of 1. Only values of α ranging from 0 to 60 degrees need to be stored since all six sectors are symmetrical displaced. The computation loop then adjusts the values of ta and tb by the factor usMag, which is calculated at each sampling interval from a knowledge of the actual inverter output voltage. The computation loop also computes the value of t_o (equation (6)).

Three main activities have been identified so far: (1) the computational activity, (2) the feedback (A/D) activity and (3) the update of the output port value. In order to ensure a faithful PWM representation at the output, the output port should never be idle, even when the other two activities are taking place. We use the double-buffering [7] technique to achieve this concurrent behaviour in a single-microcontroller system. The computational activity is performed in the foreground, whereas the generation and update of the value held by the output port is performed by timer subroutines which can interrupt both the computation loop and the A/D conversion subroutine at any time. The computation loop, which we call the *main cycle*, feeds updated values of the timer and port register variables to the timer subroutines loop, designated as the *interrupt cycle*, at each sampling interval.

The design is implemented on the 16-bit Hitachi H8/3048 microcontroller. The main characteristics of the microcontroller [8] particularly relevant to our design are:

- 128 Kbytes of ROM (internal)
- 4 Kbytes of RAM (internal)
- Maximum clock rate of 16 MHz
- Five 16-bit timer channels
- Eight 10-bit resolution A/D converter channels

Analysis and Design

The abstractions of the system are captured in class-Responsibilities-Collaborators (CRC) cards. Each abstraction or class is kept as simple as possible to encourage future re-use. Classes, in the Booch notation, are represented as clouds with a broken line boundary. Three fundamental relationships are used to relate the classes: 'inherits from', 'has a' and 'uses'. The three relationships are illustrated between the two classes, A and B, in figure 5 below. The arrow indicates that class B 'inherits from' class A. The filled circle indicates that class A 'has' or contains class B. Lastly, the empty circle, indicates that class A 'uses' class B.

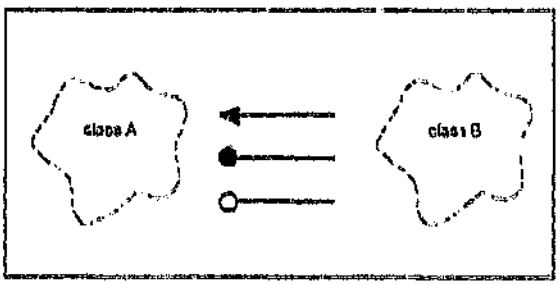


Figure 5: The three main relationships among classes

Four main class categories (or clusters of classes) are identified in our problem:

a. The initialisation procedure

b. The phase-locked loop process

c. The interrupt cycle

d. The main cycle

The initialisation procedure handles all matters relevant to initialising, both the SVM algorithm and the microcontroller register variables. For the initialisation process, two abstractions are recognised, startUp and lookUpTable. startUp is a class utility, i.e. it contains free

SVM is based on the concept that a balanced set of three-phase ac voltages can be represented by a single composite voltage vector, Us of magnitude usMag, where,

$$Us = usMag\ e^{j(wt)} \qquad (1)$$

Its application to three-phase inverter control relies on the fact that there are only eight possible switching states for a three-phase inverter. The eight switching states translate into eight discrete voltages (U_0 - U_7) in the same complex plane as Us. Two of those vectors, U_0 and U_7 do not contribute to the output voltage. They correspond to the state when either all the top switches or all the bottom switches of the inverter are off. The remaining six vectors are displaced by 60 degrees around the complex plane. The situation is depicted in figure 3 below.

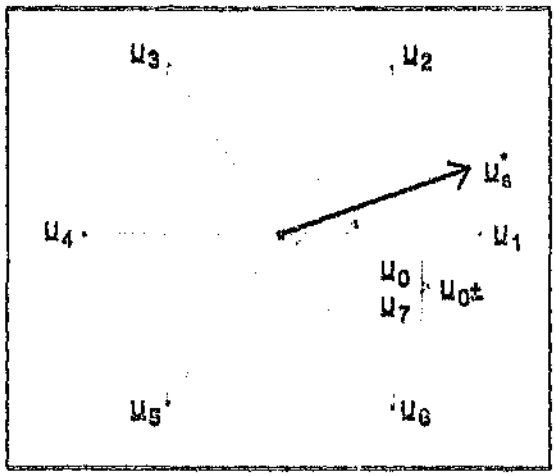


Figure 3: Phasor representation of the switching vectors and the desired vector Us*

From figure 3, it is clear that at any instant of time, a vector in the plane can be synthesised by switching the adjoining voltage vectors of the relevant sector. We call the rotating vector that has to be synthesised, the desired vector, Us*.

For example, to synthesise Us*, when it is in the first sector, as in figure 3, the switching set in figure 4 will be used.

A switching set is comprised of a forward switching sequence (first half -figure 5) and a reverse switching set (second half -figure 5). Δt is known as the sampling interval. The two redundant voltage vectors U_0 and U_7 (corresponding to vectors U_0 and U_7) are introduced to ensure a binary Gray code switching sequence. This has for effect to reduce the harmonics in the inverter output waveforms [1].

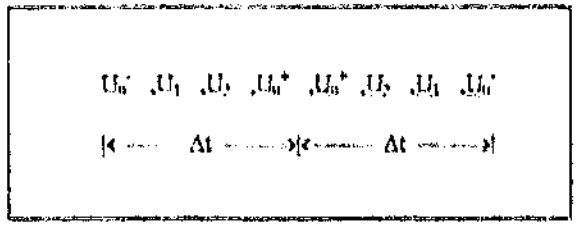


Figure 4: A typical switching set

If Ua and Ub are two adjoining vectors of a general sector, then the desired vector, Us*, will be given by the time average of the two switching voltage vectors and the time spent in the zero switching state:

$$Ua\ ta + Ub\ tb + U_0\ t_0 = Us\ \Delta t \qquad (2)$$

where ta, tb and t_0 are the time duration spent in the voltage vectors states Ua, Ub and U_0 respectively. That is,

$$ta + tb + t_0 = \Delta t \qquad (3)$$

Van der Broek *et al.* [6] provide the solution to equations (2) and (3) as,

$$ta = 0/\pi^2\ usMag\ \Delta t\ (Cos\ \alpha - 1/\sqrt{(3)}\ Sin\ \alpha) \qquad (4)$$

$$tb = 0\sqrt{(3)}/\pi^2\ usMag\ \Delta t\ Sin\ \alpha \qquad (5)$$

$$t_0 = \Delta t - ta - tb \qquad (6)$$

where α is the angle between Us* and Ua (Ua, being taken as the adjoining vector on the left).

Equations (4), (5) and (6) show that if the parameters, usMag and α are known, the time to be spent in the relevant switching states can be calculated. Since each switching state is in turn related to a specific switching configuration of the inverter bridge, the 'on' and 'off' times of each switch of the bridge can be obtained. The end-result of imposing this switching pattern on the inverter, is a PWM output waveform that provides a set of three-phase ac voltages, after that the high-order harmonics content have been filtered out.

In the context of SVM, the main function of the microcontroller system will be to generate the appropriate output pulses (high and low) for the appropriate switching times ta, tb and t_0, in order to construct a PWM output.

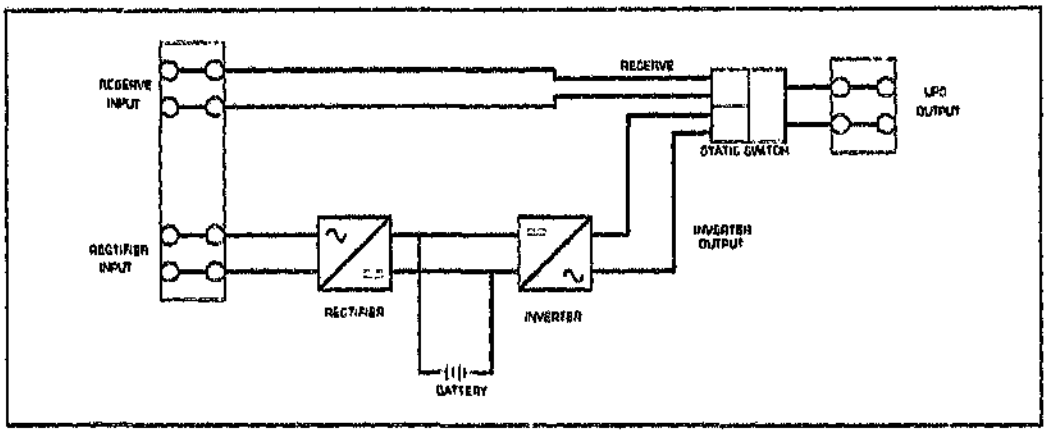


Figure 1: Block diagram of the on-line UPS

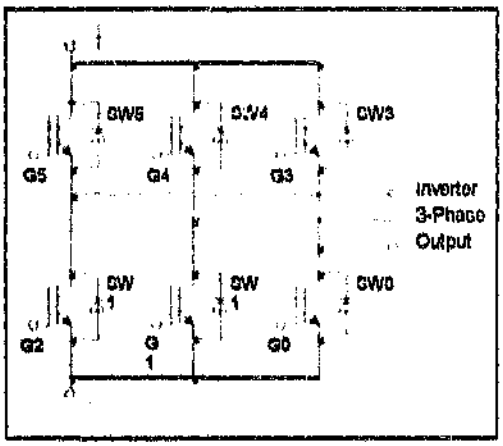


Figure 2: Three-phase inverter bridge

Under a failure of the utility, power to the load is maintained through the energy stored in the batteries. The back-up time depends on the load and the energy-storing capacity. However, irrespective of those two factors, the dc voltage form the batteries will drop after a certain time as the energy level drops. The UPS should ensure that this drop in dc voltage is not reflected at the UPS output, until the shutdown point of the unit. This involves a feedback configuration, whereby fluctuations in the inverter output are monitored and accounted for by varying the pulse-width of the PWM signals.

In order to protect the load against a failure of the inverter bridge itself, there is a second utility source, called the *reserve*. Both the reserve and the inverter output are fed to a static switch. Under normal running conditions the static switch feeds the load with the inverter output. Under an inverter failure, the load is immediately transferred to the alternative source, i.e. reserve. It is important that the running of the load is not affected by that transfer. Therefore, the transfer should occur with the least perceptible break in the output.

Transferring the load suddenly form inverter output to reserve can have adverse consequences to the load if the phase difference between the two sources is too large. Consequently the UPS should constantly monitor and adjust the frequency and phase of the inverter output to match the reserve waveform, within a pre-defined frequency window. This technique is commonly known as phase-locked loop.

To summarise, the role of a microcontroller system for a three-phase UPS inverter-control application are:

a. to generate PWM drive pulses to the switches of the three-phase inverter bridge

b. to provide voltage feedback and regulation

c. to synthesise a phase-locked loop algorithm within a pre-defined frequency window

In this study we investigate the use of SVM in supporting those features.

Principles of SVM

SVM has been well documented in the literature [1][4][6]. Here we will only provide a brief overview of the subject.

Space-Vector Modulation Inverter Control: An Object-Oriented Application

H Bapoo
MIEEE

Abstract

This paper presents an object-oriented approach to designing a Space-Vector Modulation (SVM) control system for a three-phase uninterruptible power supply (UPS) inverter. SVM is used to generate the pulse-width modulation (PWM) control drives for the inverter switches, in a combined look-up table and on-line computation technique. The design is carried out fully in the language C and operates on the Hitachi 16-bit H8 3048 microcontroller. The final system performs well, at a switching frequency of 4 kHz. It provides both voltage and frequency on-line regulation. Difficulties in performing a full object-oriented design implementation on embedded systems in general are also discussed.

Introduction

Space-Vector Modulation is an attractive technique for generating PWM pulses in a software implementation approach. Some of the advantages that this method offers are:

- Its implementation involves few equations
- It contains few variables - angle of displacement and vector magnitude are the two main ones
- The equations to be solved are all linear

In addition, SVM is a low harmonic generation technique that has been successfully implemented both in applications using look-up tables to store the operating conditions [1][2] and in on-line, real-time techniques using transputers [3]. It has become prominent especially in high performance Field Oriented Controlled AC drives [1] and has also been used in fully software-controlled rectifiers [4].

In spite of those advantages, the SVM algorithm in an on-line computation approach can become quite complex. The problem gets even more complicated in the context of the UPS, which requires voltage feedback and adjustments as well as phase-tracking. The hard-real time constraints imposed by the need for a high switching frequency, aggravates the whole issue.

The complexity, reliability and real-time issues faced by this project are common to embedded systems development in general. The whole project was developed using a risk-based, hardware-software co-design development approach. The approach is examined in document SDE 115. The scope of this paper is limited to the analysis and design phases, which are both performed in an object-oriented (OO) method.

Object-oriented techniques can prove to be quite effective in analysing and designing complex systems, even for small embedded software systems. They are particularly good at capturing the main entities (or objects) of the system, at showing their inter-relationships and in displaying the dynamic semantics of the problem. In addition they can easily integrate the hardware elements of the problem into the representations.

This paper presents the object-oriented design of a microcontroller-control system for a three-phase UPS inverter, using SVM. The Booch [5] methodology and notation are used in support of the object-oriented analysis (OOA) and the object-oriented design (OOD).

The UPS Structure

Figure 1 shows the basic structure of an on-line UPS.

UPS's are used in power back-up and protection schemes. An on-line UPS rectifies ac from the utility (mains) to dc, which is stored in batteries. Also feeding from the dc bus, is an inverter bridge that converts the dc into ac. Typically, ac to dc conversion is performed through high PWM switching of the voltage between the two polarities of the dc bus. Control of the three-phase inverter is the subject of this present study. The inverter bridge is pictured in figure 2.

In addition the process ensured that the hardware interface platform was ready before the software was completed.

The final system runs on a single 16-bit Hitachi H8/3048 microcontroller at a PWM switching frequency of 4 kHz. Although this frequency is relatively slow for fast switching devices like insulated-gate bipolar devices (IGBTs), the complete system implements all the additional control features that are usually associated with the inverter of a UPS, like phase-locked loop control, inverter shutdown facility, overload monitoring and static switch control.

Although our approach produced encouraging results on the inverter control project, it remains to be validated on a wider range of embedded systems projects. Further studies on this approach might examine the possibility of incorporating time and cost metrics. Any future work on software development processes will have to be more attentive to the developer and his/her needs.

References

[1] I. Sommerville, "Software Engineering", Fourth Edition, Addison-Wesley Publishing Co. Inc., USA, 1992

[2] B.W. Boehm, "A Spiral Model of Software Development and Enhancement", IEEE Computer, Vol. 21(5), May 1988, pp. 61-72

[3] B.W. Boehm, "Software Engineering", IEEE Transactions on Computers, Vol. C-25, No. 12, December 1976, pp. 1226-1241

[4] M. Chiodo, P. Giusto, A. Jurecska, M. Marelli, H.C. Hsieh, A. Sangiovanni-Vincentelli, L.Lavagno, "Hardware-Software Codesign of Embedded Systems", IEEE Micro, Vol. 14, Part 26-36, August 1994, pp. 26-36

[5] A.D. Stoyenko, L.R. Welch, B. Chao Cheng, "Response Time Prediction in Object-Based, Parallel Embedded Systems", Microprocessor and Microprogramming, Vol. 40, Part 2-3, April 1994, pp. 135-150

[6] R.G. Clark, "The Design and Development of Embedded Ada Systems", Software Engineering Journal, Vol. 5, Part 3, May 1990, pp 175-184

[7] D.J. Smith, K.B. Wood, "Engineering Quality Software", Second Edition, Elsevier Applied Science, UK., 1989

[8] G. Booch, "Object-Oriented Analysis and Design", Second Edition, The Benjamin/ Cummings Publishing Co., Inc., USA, 1994

[9] W.H. Wolf, "Hardware-Software Co-Design of Embedded Systems", Proceedings of the IEEE, Vol. 82, No. 7, July 1994, pp. 967-989

[10] P. Zave, "An Operational Approach to Requirements Specification for Embedded Systems", IEEE Transactions on Software Engineering, May 1982, pp. 250-269

by those activities should not be underestimated. Adapting standards to a new project is certainly not a minor task and should be viewed as a phase of the development itself, instead of being regarded as an extra factor that consumes development time.

Control of changes is also time consuming. It, too should be viewed as an investment in the life-cycle of the software and usually has to be carried out strictly. However this is an area where the development of embedded systems might differ slightly. The development model that we present holds a design-coding-testing iterative loop. To apply strict control over all those minor tentative loops will tend to defeat their very purpose, which is to obtain quick feedback about specific aspects of the design. In that respect, enforcement of a strict QMS might have detrimental effect upon the design and the ardour of the team itself. In addition, the first few outputs of those loops might be of minor importance to the later development of the software. Their inclusions in the documentation will have the further drawback of increasing the volume of documentation and of obscuring the more important features.

A loose form of change control is contrary to good QMS practice but it has the advantage of giving the developer more freedom, with reduced interruptions. In fact, each iteration through the loop can be seen almost as a brainstorming exercise. However to allow complete freedom, without any documentation, throughout the process will, after a number of iterations, start to generate negative returns on this process. This will usually happen as the developer starts to forget some of the factors that evolved during earlier iterations. Therefore it is necessary to discipline the development effort, while still providing it with a freedom margin needed for its success. A balance has to be struck between the two concerns. A solution will be to document only selected revisions at the early stage of the design. Only when a predetermined performance level is achieved or when major characteristics of the software are revealed, is the design process halted and the revision updated.

Contribution from Existing Development Processes and Strategies

The model presented has been influenced by a number of development processes and strategies. In particular:

- It borrows the idea of library of modules and the retry redesign limit test from Stoyenko et al. [5]
- Like the co-design approaches of Chiodo et al. [4] and Wolf [9], it processes the hardware and software elements simultaneously
- It adopts Zave's [10] idea of including the environment into the specifications
- It accepts both uncertainty and change, like the prototyping approach
- Like Boehm's spiral model [2], it is risk-based and iterative in nature

Results

The model was largely developed through a case study on real-time embedded systems development. The project used as a case study aimed at controlling a three-phase inverter (AC/DC converter) for an uninterruptible power supply (UPS) system, using a microcontroller set-up. The system was particularly characterised by stringent time constraints arising from the need to switch the insulated-gate bipolar transistors (power switches) of the inverter bridge at high frequency.

The model was particularly successful in identifying the risk factors early in the process:

- sinusoidal PWM, the preferred inverter control technique on the UPS under consideration was identified as not being suitable for a software approach due to the non-linear equations that have to be solved. Instead SVM was used. Although involving trigonometric functions, the equations associated with SVM are all linear.
- The first few iterations of the design-coding-testing loop quickly revealed the impracticality of a purely sequential approach. The output port cannot remain idle while the values are computed. A double-buffering method was used instead. Computations are now performed in the foreground while the actual management of the output port is carried out by timer subroutines which interrupt the main computation routine regularly.
- Further iterations of the loop showed that even with SVM, a direct on-line computation approach was not feasible, because of the complexity and time limitations involved. Instead a look-up-table method was used. On-line computations were limited to adjusting those stored values to the requirements of the system.
- The process was also successful at pointing out quickly how much the use of floating point variables was slowing down the calculations. This problem was solved by applying scaled arithmetic techniques and by using binary manipulations.

Finally, when all the modules have been tested, both separately and as a system, the software is integrated with the hardware. The whole system is then verified and validated. Success in meeting the overall requirements means the end of the development stage and will carry the system into the maintenance stage of its life-cycle. If however the requirements specifications are not met, then the problem is again referred back to the functional specifications level, where the technical strategies and the hardware/software partitioning can be reviewed. This iterative process is repeated a fixed number of times. After that, if the requirements are still not met, the problem has to be referred to the requirements analysis stage. At this point, the constraints on the design are declared to be too tight and have to be relaxed.

Characteristics of the Approach

The main characteristics of the development approach are summarised below:

a. It follows a natural iterative problem-solving pattern. It recognises both the inclination of the developer to first address those aspects that are unclear or in which he or she is not confident, and the iterative reasoning pattern that we usually apply when trying to solve a problem. Being close to the intrinsic problem-solving process, means that the model could be more easily adopted by the embedded systems developer community at large, without necessitating a sustained training process.

b. It builds the confidence of the developer as the process progresses. As we saw previously, the high risk elements associated with embedded systems projects means that the developer is usually under enormous pressures and has many reservations (even if not openly expressed). By giving the developer the freedom to jump straight to a particular problem area, the model relieves much of the psychological pressures and increases the chances of success. In other words the process is achieving its objective, which is to facilitate the development.

c. With the design-coding-testing strategy, the process can build the final system gradually, i.e. separately developed and tested modules are integrated progressively. On a managerial level, this means that the process naturally accommodates partitioning of tasks to different developers or teams. Separate modules can be developed simultaneously. In addition, the separation of concerns implies that the programmer (or team) who has been allocated a particular task does not have to be conversant with all the details of the problem. For example, someone implementing the A/D module in an inverter control project, does not have to understand the principles of PWM, although both are integral aspects of the control of the inverter. In terms of the object structure, the only information external to the module that the developer needs to know are the preconditions and the postconditions of that module.

d. Testing is performed early in the development process. That means that the project is being constantly verified and validated throughout its development. This can largely contribute to software reliability and quality.

e. Continuous verification and validation can be also beneficial from the customer viewpoint. The customer, whose requirements are often unclear, will become aware of the limitations early in the process. Compromises and alterations in the specifications can be affected at lower expenses.

f. The small intermediate deliverables produced by the process will also be appreciated by the management, who will have a better visibility of the system: hence a better control.

g. As already mentioned, this approach promotes re-use of code by building a library of tested modules.

h. The process encompasses the hardware development while keeping the software development as independent as possible. A parallel hardware/software development (as compared to a sequential approach) has less bottleneck stages.

QMS Considerations

An adequate quality management system (QMS) can considerably contribute to the success of a project. It promotes standard practices which fosters an effective communication. It is especially effective in managing changes brought to the product. In that respect its contribution extends to the whole life-cycle of the product.

The main overhead associated with the application of a quality management system appears in the time it consumes. The areas where time on QMS activities are mostly consumed are:

- in the initial set-up of the standards to the project
- in managing changes

In the early stages of the project, new documents have to be created, templates have to be set up or adjusted, and all the members of the team must become acquainted with the standards. The time and commitment demanded

system; two factors that fits well in the general philosophy of our model. Also at this stage, the time scheduling of the operations should be defined. This is particularly important when several functional partitions

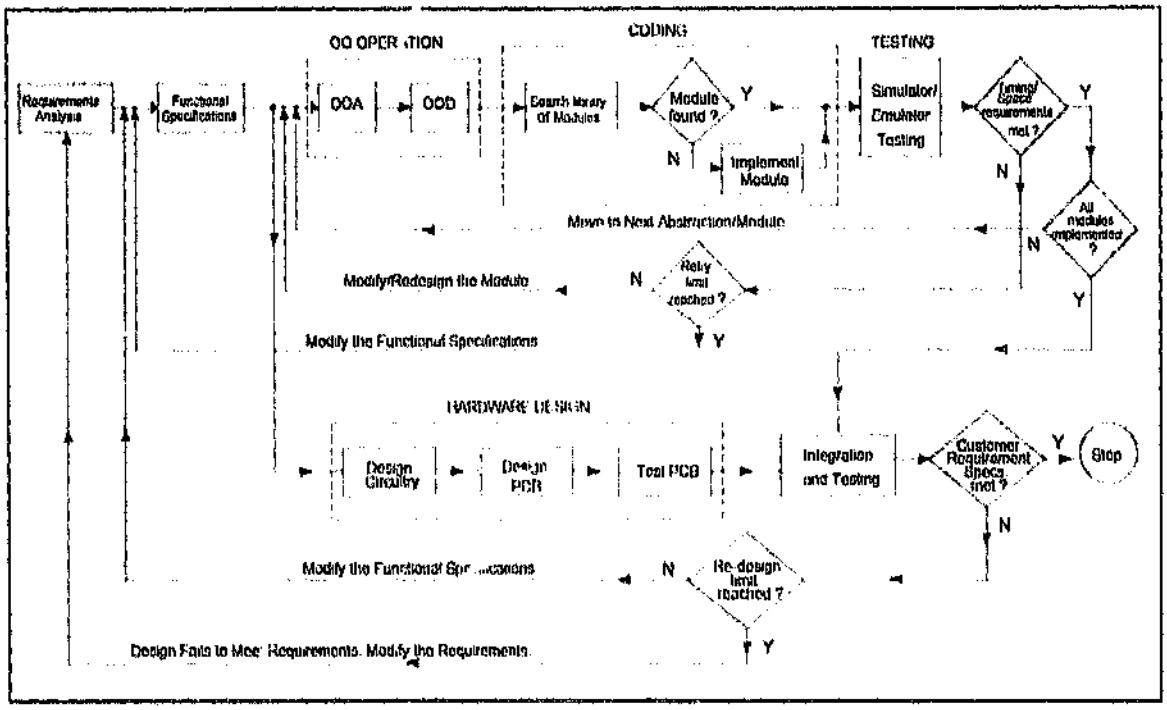


Figure 1: A risk-based, software-hardware co-design approach to developing embedded systems

share one hardware unit. All the information about the abstractions, their inter-relationships, their locations and the dynamic semantics of the problem can be effectively captured in the models of object-oriented development (logical and physical models, static and dynamic models) [8]. Another benefit of an object-oriented approach to embedded systems development is that it can naturally incorporate the hardware elements of the system into the description, thus providing a more accurate representation of the problem.

Once the abstractions have been defined, the development process proceeds with the implementation and testing of the modules, starting with those that contain the highest element of risk. Being built from small modules, such an approach will quickly generate a library of tested modules, that can be referred to in future designs.

While the hardware platform is being developed, the modules generated by the software design can be tested on an in-circuit emulator (ICE). The testing phase should reveal whether each designed module meets the space and time constraints, in addition to the functional requirements. If the requirements are not met, the design has to be reviewed; i.e. the design-coding-testing loop is repeated. it is important to relate any change to the design back into the object-oriented development models, so as to keep track of how the changes affect the other elements of the system. If after a pre-determined number of iterations through the design-coding-testing loop, the requirements of the module are still not met, then the problem should be reviewed at the functional specifications level. If it is purely a space or timing problem, the hardware/software partitioning can be revised. Alternatively, the suitability of the technical strategies (e.g. SVM or sinusoidal PWM) has to be reviewed, and if necessary a different technical approach has to be investigated.

As the development of the hardware is completed, some of the modules can be tested directly on the hardware platform (via the ICE at first). This applies especially to those modules that cannot be tested without external physical inputs, like analogue-to-digital (A/D) conversion modules.

needed in embedded systems design, is the freedom to focus on specific risk aspects of the problem in short design-coding-testing loops. This suggestion calls for a modular approach, whereby the developer builds the final system around loosely coupled modules, like those found in object-oriented techniques.

Thus far, three attributes which a development model for embedded systems projects should have, have been identified in this paper:

- Hardware/Software co-design approach
- Modular approach
- Short design-coding-testing iterative loops

In table 1 those three features are used to evaluate the suitability of six development processes for embedded systems. Each process is rated against those features according to how prominent those features are in the process (High, Low or Absent).

Table 1: Evaluating the suitability of other processes for embedded systems development

Development Model/Approach	Feature		
	Short iterative loops	Modular approach	HW/SW Codesign approach
Waterfall model [3]	Low	Absent	Absent
Boehm's Spiral model [2]	High	Low	Absent
Prototyping method	Low	Low	Absent
Chiodo's et al. HW/SW Co-design [4]	Low	Low	High
Stoyenko's et al. object-based parallel approach [5]	Low	High	Absent
Clark's CSP/ *me too* method for real-time embedded systems [6]	Low	High	Low

The Development Approach

Building on the three criteria described above and drawing from existing formal development practices, a risk-based approach for small embedded systems development was shaped. The model is outlined in figure 1. It starts with a requirements analysis, which is typically a document drawn up by the customer describing what the system should do and how it should behave. In our model, it is understood as a pre-contractual document that, although formal, can be subjected to change and negotiation as the project evolves.

In order to relate the customer specifications to the software developers, a functional specifications document should be prepared. This document should include a description of the environment in which the system is to operate. This exercise will create a picture of the system at a higher level of abstraction than what is strictly expressed by the problem itself. At this stage as well, any technical strategy or method relevant to the design should be defined. This task might involve some further investigation into the technical aspects of the problem. For example, if the project is to develop a microcontroller system to control a three-phase inverter, one has to decide, at the functional specifications stage, on the method (Space-Vector Modulation (SVM), Sinusoidal Pulse-Width (PWM) Modulation, Optimal Technique, etc.) of controlling the inverter. One might argue that this would be the customer's responsibility, but to do so would be to assume that the customer knows which technique is best suited for a microcontroller system approach. Techniques used in digital electronics are not necessarily appropriate for software solutions. For instance, in the inverter control example, sinusoidal PWM is a widely used technique. It has been successfully implemented in digital electronics using op-amp circuits. However because of the non-linear equations that have to be solved in the process, (a constraint that is obviously not experienced in a hardware solution) this approach is not appropriate for a software implementation. Such considerations, which it might be argued are outside the scope of the software design, are commonly faced by embedded systems developers and explain why many of them come from an Electrical Engineering background. Always at the functional specifications level, following the decision about the technical strategies, the software/hardware partitioning of the tasks has to be defined. This decision should be based on factors such as cost, component-count, time constraints, reliability issues and space limitations. Sometimes because of reliability and safety considerations a task performed by the software has to be also implemented by the hardware. Several "hardware feedback" and other reliability design philosophies are discussed by Smith and Wood [7].

At this stage the software and hardware designs diverge. For the software, the next step is the analysis and design of the system. Those two phases are combined in figure 1 under the heading "object-orientation operation". Strictly speaking the design can be carried out in any design methodology, structured or object-oriented. However in our case study object-orientation appears to be particularly suitable. Object-orientation naturally provides a modular (object) approach to the design and offers different levels of abstraction perspectives of the

different products like telephones, trains, washing machines, industrial controllers and automobiles)

- The maturity of the organisations developing embedded systems varies vastly, making it difficult to develop a process suitable to all.

- Although embedded systems are generally characterised by the constraints and concerns listed above, the actual degree of importance of each of those factors can vary considerably between projects, making it necessary to approach each project differently.

- The importance of the role of the microcontroller in the overall system can itself vary largely. Consequently the allocated time and budget for embedded systems projects vary accordingly.

Objective of a Development Process

Predictability is generally considered as an important feature of a development process. It should be possible to meet requirements on time, within cost constraints, achieving all required functionality and within a quality framework. Yet the primary objective of a development process, as we see it, should be to facilitate the development of the system and the task of the developer. We see the process as a means to support the developer and not the other way round. Consequently a development model should be built around the needs of the developer and the technical realities he/she faces.

Establishing the Foundation of the Approach

Despite the difficulties mentioned above, we can attempt to sketch a general pattern that embedded systems developments should follow, by drawing on their similarities.

In particular, we believe that a development model for embedded systems HAS to accommodate the following factors:

- Hardware interaction and the environment of the embedded system

- Risk factors

By its very definition, an embedded system consists of an intelligent device buried in a hardware environment. Being often allocated monitoring and control tasks its interface with the environment is critical. The design of this interface, between the software and the rest of the system, is part of the embedded system design. The interface often takes the form of sensing and actuator circuits, together with isolation and power supply circuits. The whole interface design, which we will refer to as the hardware design (in the context of embedded systems) is often built on a printed-circuit board (PCB). Failure to recognise the importance of this interface design early in the process invariably leads to (1) poor hardware-software partitioning of tasks, (2) bottlenecks when the hardware is needed to test the software, and (3) a difficult software-hardware integration. Those considerations indicate strongly that the hardware interface has to be developed along with the software.

Risk factors are any elements that can cause the final system to fail to meet the requirements specifications. Sommerville [1] views risk as simply something which can go wrong and that is often a consequence of inadequate information. Risk elements in embedded systems arise as a result of the complex nature of the tasks they are often required to handle. The hardware interface design itself adds to the risks factors. More important are the increasing number of new application areas in which embedded systems are used. These new areas are often surrounded by unknowns and uncertainties. High degrees of risk and complexity in a system, are often what motivated the decision to employ embedded systems technology in the first place. Therefore, it is essential that the development model for embedded systems be built around the notion of risk.

It is interesting to observe how the developer responds to risks. Risk factors in a design put a lot of pressure on the developer. The embedded systems developer, often an Electrical Engineer by training, is concerned with all aspects of the problem and it is a natural tendency for him/her to isolate major risk issues and tackle these quickly at the outset of a project. The emphasis here is on both the urge to address isolated issues and to explore solutions to them quickly. Few development processes accommodate for this natural tendency. In fact many of them tend to oppose this approach in their effort to "discipline" the process.

Since not all risk elements are known prior to the development and since one risk factor might hide another risk factor, it is important for the process to be iterative. In addition our natural approach to problem-solving is rarely purely sequential. It often follows short and quick conception-implementation-testing iterative loops.

To a large extent the prototyping approach and the spiral model [2] try to confront the problems of risk and uncertainty. However even those processes are concerned with working versions of the complete system or part thereof. The emphasis is more on the overall system than on the separate entities. The result is that again the developer has to address a large number of issues at every iteration through the process. What is

A Risk-Based Approach to Embedded Systems Development

H Bapoo
MIEEE

Abstract

Many of the practices commonly employed in the software development process apply equally well for embedded systems. However, embedded systems have their own characteristics. Failure to recognise these attributes will invariably lead to poor design performances. This paper outlines the distinctive features of embedded systems and proposes modifications to the commonly used software development process which would accommodate these features. More specifically, a risk-based, hardware-software co-design approach is presented. The approach builds on an actual real-time embedded system project.

Introduction

Small embedded systems have traditionally been developed with little reference to formal techniques or around loosely followed general design procedures. There are two main reasons for this. Firstly, because of the relatively small size of the code, developers have been deceived into thinking that small embedded systems can be developed without any formal development process. The tendency is to judge the complexity of a software system only by its size and to fail to recognise such factors as the role the software is to play in the system, complexity of the control function, safety and reliability considerations, and maintainability issues. The second reason is the failure of general software development processes to recognise the characteristics of embedded systems. Consequently, in many cases where a formal development process is in place, developers tend to pay lip-service to the process. Frequent reviews and documentation re-structured at the end of the project, give the impression that the procedures are being followed.

Embedded systems often play a critical role in an application and are subjected to frequent upgrades and modifications. They can thus considerably benefit from a disciplined development approach. This paper identifies some of the difficulties in formulating a development process that is applicable to all embedded systems. Drawing from several paradigms, it presents a model that attempts to match the technical realities of embedded systems development with the intrinsic problem-solving approach of the developer.

Characteristics of Embedded Systems

Many of the established practices commonly used in software development, such as the spiral model the operational model, prototyping and others, are applicable to embedded systems projects. However, the engineering nature of their applications, combined with several other factors, make embedded systems software different from data processing software. Failure to recognise those differences will invariably lead to poor design performances. We start by identifying some of those distinctive features.

Embedded systems projects are often characterised by the following factors:

- High reliability concerns
- Real-time constraints
- Stringent physical resource requirements - often a result of their physical location
- Limited development resources - since they often constitute a small part of the overall system
- Low power consumption constraints
- Large variety of attachments with the environment
- Complex interface with the environment
- Difficulty in testing the end-product - often a result of their physical location and the end-purpose of the overall system (e.g. ballistic-missiles)
- Low cost per part constraint - since embedded systems products are sometimes produced in large numbers

From an assessment of the nature of embedded systems and the characteristics of their development, it becomes apparent that the formulation of a development process that is appropriate for all embedded systems is not feasible. The main reasons are:

- The large diversity in the applications of embedded systems (embedded systems are found in very

A quality management system was seen as an essential element of a design. It was particularly was found to be effective in providing transparency into the development progress and in encouraging a good communication. We believe though, that for embedded systems, a flexible quality management system is desirable, especially in the first few design iterations. This is particular important to promote the creativity of the developer.

Object-oriented proved to be very helpful in analysing and designing the system. It could be a major solution for the complexity and maintainability issues faced in embedded systems development. Coding in object-oriented languages for embedded systems is still a problem, but this limitation will be lifted in time with the introduction of more powerful devices and more efficient compilers.

1.5 Where to From Here

This study has clearly revealed the importance of a change in attitude among the embedded systems development community. Though often electrical engineers by training, embedded systems developers will have to become conversant with the software development strategies and try to adapt those models in the context of their own field of applications. Failure to recognise the need to discipline and document the approach will promote a field that is strongly based on individual efforts. This kind of environment is usually characterised by wasted efforts in re-designing existing software. The integration of new comers in such organisations is also generally a very long and painful process. Organisations should not wait for the need for a disciplined and documented approach to be spurred by the customer. They should view the maturation process as an opportunity of gaining a competitive edge in the industry.

In terms of future works on the findings of this study, firstly the proposed development approach needs to be tested on a wide range of embedded systems applications. Future research work might also investigate the possibility of incorporating software metrics into the model, in an attempt to create a more predictable system. Finally a support environment for the whole development system could be developed. CASE tool support could be especially helpful in automating the coding and testing phases. Any future work on development processes in general will have to be more attentive to the developer and his/her needs. As de Marco[1] explains, software development is fundamentally a research and development

[1] De Marco, T. (March 1995), *Blueprint for Success: Innovate and Invest in People*, IEEE software, vol. 12, no. 2, pp. 108-109.

SDE 113 reviews the main aspects of embedded systems development. In particular it re-examines the issues of development processes and quality management systems. In this document as well, some practical issues about embedded systems development are compiled.

The two technical papers presented in this dissertation, SDE 114 and SDE 115, merely summarise the findings of this study.

1.3 Satisfaction of the Study's Objectives

In terms of the objectives set out, the study has been successful. It has thoroughly described the major aspects of embedded systems. An excellent example of an embedded system development was provided in the case study. Each phase of the development of the case study has been documented. Embedded systems development was critically reviewed, and finally an adaptive development approach for embedded systems was presented.

1.4 Concluding Remarks

This study has highlighted the differences in embedded systems development as compared to data-processing software development. It has shown that embedded systems are not simply cut-down versions of larger software developments but that they have their own characteristics. In particular, we found that embedded systems developments involve difficult compromises among multiple stringent constraints, and a judicial partitioning of hardware-software tasks. The study has identified the need for a risk-based, hardware-software co-design approach to suit the technical realities of embedded systems. However, it was stressed that a universal development process for embedded systems projects was very difficult to achieve because of the large variety of applications of embedded systems.

Through this study we found that the technical complexity of embedded systems often necessitates an in-depth investigation of the technical aspects of the problem prior to initiating the design. This step is particularly important in choosing the right technical strategy for the design. The choice should be based on the application but should also be software approach-based. The technical specifications of the chosen strategy are compiled into the functional specifications document.

It was argued that designing in short design-coding-testing loop provides an efficient development approach that carries low risks and promotes re-use of codes. This approach presents benefits both at the managerial level and at the customer level for the visibility it offers into the product life cycle.

1 Discussion and Conclusions

1.1 Re-examining the Objectives of the Study

So that the conclusions and arguments in this final document can be viewed in the right perspective, the main objective of the study is re-stated: To attempt to reconcile the existing formal development models to the technical realities of embedded systems development.

Three main tasks have been assigned to this project:

- To define embedded systems and assess the difficulties currently experienced in their development
- To provide and document a practical example of embedded system development
- To formulate a development approach for embedded systems, based on the literature survey and on the practical exercise

1.2 Review of the Technical Documents

The essential documentation supporting this study is captured in the appendices.

SDE 114 provides an overview of the project. It defines the project, its objectives and scope.

The two management products, SDE 001 and SDE 005, together provide a description of the tasks and documents attached to the project.

Aspects of embedded systems development that are common to software development in general are documented in SDE 102. This document also describes some of the most prominent software development processes.

SDE 103 focuses on embedded systems. This document reveals that the main characteristics of embedded systems development are the risk elements involved and the strong hardware implications. It provides evidence that the applications of embedded systems and the microcontroller market in general are expanding.

The documents SDE 104, 105, 106, 107, 108, 110 and 112, all support the case study development. The case study provided an excellent example of a real-time embedded system. It was instrumental in drawing the development approach proposed in document SDE 113.

IAS Annual Meeting, 1986, Paper CH 2272-3/86/0000-0244

[7] P. A. Laplante, "Real-Time Systems Design and Analysis, An Engineer's Handbook", IEEE Press, 1993

[8] H8/3048-Series Hardware Manual, Semiconductor and IC Division, Hitachi Ltd., January 1995

[9] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, "Object-Oriented Modelling and Design", Prentice-Hall, 1991

[10] S. Fukuda , Y Iwaji, H. Hasegawa, "PWM Technique for Inverter with Sinusoidal Output Current", IEEE Transactions on Power Electronics, Vol. 5, Number 1, January 1990

[11] SH7700-Series Microcontrollers Overview, Semiconductor and IC Division, Hitachi Ltd., 1995

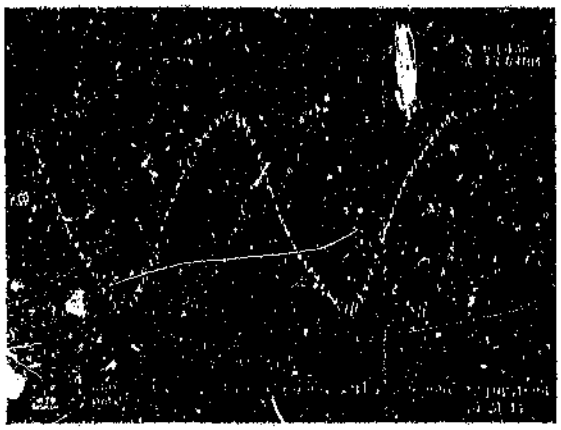


Figure 13 (d): Filtered output of the drive pulses for the bottom half of the inverter bridge

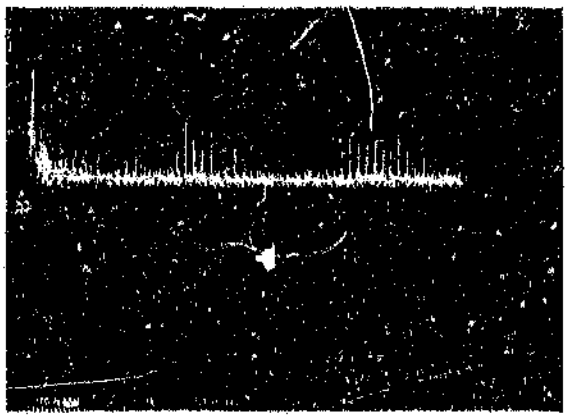


Figure 13 (e): Frequency spectrum (up to 5 kHz) of the filtered drive pulses

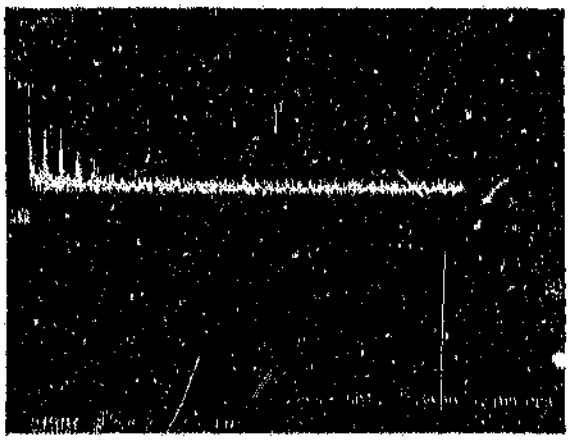


Figure 13 (f): Frequency spectrum (up to 50 kHz) of the filtered drive pulses

Concluding Remark

An SVM three-phase inverter control system for a UPS was designed using object-oriented techniques. The final system runs at a switching frequency of 4 kHz. Although this switching frequency is adequate for low-switching frequency devices like bipolar transistors, it does not take advantage of the capabilities of the high-switching frequency devices (>10 kHz) like IGBTs. However, at the switching frequency of 4 kHz, the software, written fully in a high-level language (C), is also carrying out all the monitoring and control features associated with the inverter of the UPS. Those include, overload monitoring, phase-locked loop control, static switch control and output voltage regulation. Higher switching frequencies could be achieved by using a 32-bit microcontroller running at a higher clock frequency; e.g. the 32-bit SH7700 family of Hitachi microcontrollers [11] run at a clock frequency of 60 MHz (as compared to 16 MHz for the H8/3048). This would be a more expensive alternative but it will certainly lift the switching frequency above the 10 kHz threshold.

References

[1] Granado, R.G. Hartley, G. Diana, "Understanding and Designing a Space Vector Pulse-Width-Modulator to Control a Three-Phase Inverter", The Transactions of SAIEE, September 1989, pp 29-27.

[2] M. P. Kazmierkowski, M.A. Dzieniakowski, W. Sulkowski, "Novel Space Vector Based Current Controllers for PWM-Inverters", IEEE Transactions on Power Electronics, Vol. 6, Number 1, January 1991, pp. 158-165

[3] M. Webster, R.G. Harley, G. Diana, D.C. Levy, "Space Vector Modulation - A Real-Time Application", The Transactions of the SAIEE, September 1989, pp 38-42

[4] B. Kwon, B. Min, "A Fully Software-Controlled PWM Rectifier with Current Link", IEEE Transactions on Industrial Electronics, Vol. 40, Number 3, June 1993, pp 355-363

[5] G. Booch, "Object-Oriented Analysis and Design", Second Edition, The Benjamin /Cummings Publishing Company, Inc., 1994

[6] H. Van der Broek, H. Skudenly, G. Stanke, "Analysis and Realisation of a Pulse Width Modulator Based on Voltage Space Vectors", IEEE

The PWM drive pulses are generated by the microcontroller at port 1, pins 0 to 5. Those pulses are fed to the IGBTs of the inverter bridge (figure 2) via a

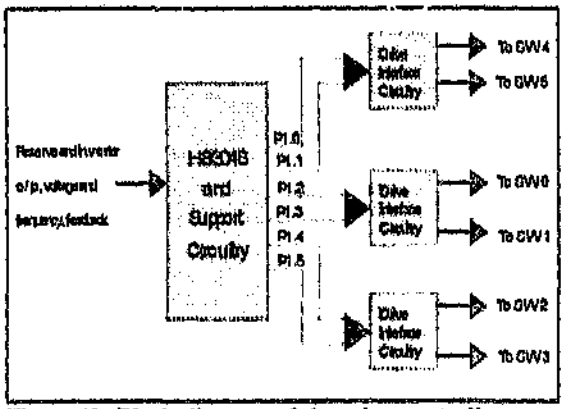


Figure 12: Block diagram of the microcontroller inverter control system

drive circuitry, which provides isolation between the control circuitry and the inverter. The drive circuitry also steps up the microcontroller output pulses from 0-5V to 0-12V.

The system operates at a switching frequency of 4 kHz.

Figure 13 (a) and (b) show a sample PWM drive, as generated by the microcontroller. The frequency spectrum of the signal in figure 13 (c) reveals harmonic contents at frequencies other than at multiples of the switching frequency and its side-bands. This sub-harmonic behaviour is common to SVM and has been observed in other SVM inverter control attempts, such like the one by Fukuda [10].

Applying a simple low-pass filter (R= 4K7, C=0.1μF) to the output drives, projects the fundamental component more clearly. The filtered output of the three drive signals for the bottom half of the inverter bridge is shown in figure 13 (d). It shows all three waveforms as containing a 50 Hz fundamental frequency and being displaced by 120 degrees. Figure 13 (e) and (f) display the frequency spectrum of the filtered output.

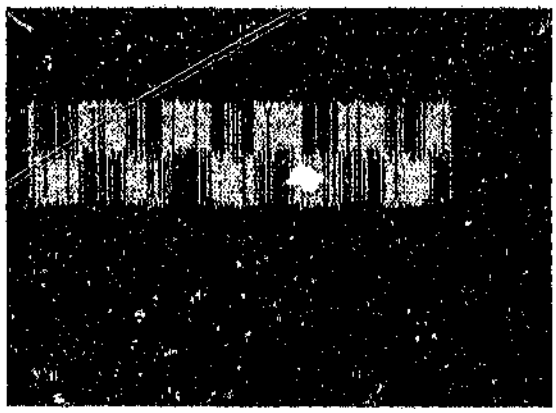


Figure 13 (a): Sample PWM ouput of the micro-controller

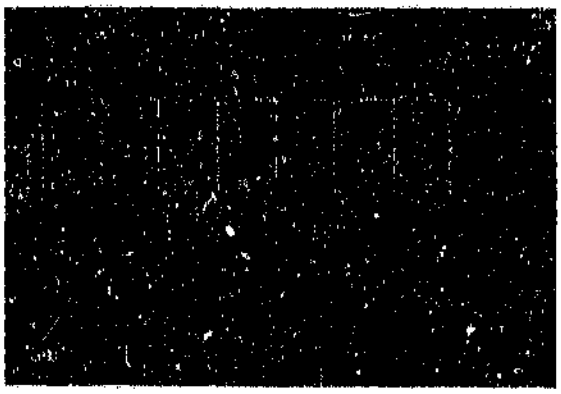


Figure 13 (b): Sample PWM output of the micro-controller - close view

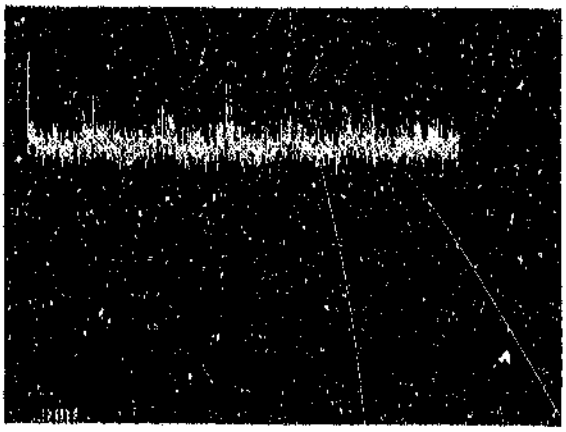


Figure 13 (b): Frequency spectrum of the unfiltered microcontroller output

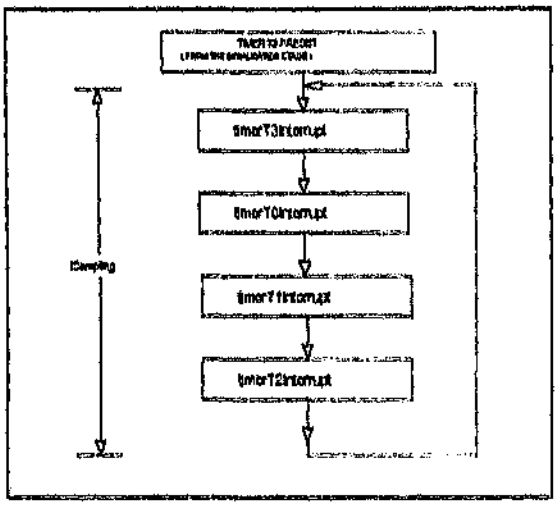


Figure 11: Interrupt cycle flow chart - derived from figure 9 and 7

Software Implementation

Having performed the object-oriented analysis and design, the next logical step is the coding process in an object-oriented language (OOL). However the H8/3048 microcontroller does not support any object-oriented language like C++. This is a problem commonly encountered in embedded systems designs. It can be circumvented by using an "embedded PC" microcontroller such as the Intel 386EX, which is based on the PC platform. Another less common solution is to use software translators, like C++ to C translators. Alternatively the object-oriented design can be directly implemented in C. Rumbaugh et al. [9] outlines an approach to map object-oriented features in C. However this solution is usually achieved at the expense of a more lengthy program, higher complexity in the data structures and in the program in general. A real-time embedded systems project such as ours cannot accommodate those penalties.

In the present design the language restriction was simply circumvented by first deriving flow charts from the state transition diagram of figure 9. The flow chart diagrams are shown in figure 10 and 11. From those two diagrams, the software is implemented in C. The penalty of using this approach instead of coding directly in an OOL, is the failure to provide a direct mapping between the object-oriented design and the program.

In the light of this project, some of the obstacles common to applying an object-oriented approach to embedded systems designs were encountered:

- OOL compilers for microcontrollers are not easily available
- OOLs tend to make a larger use of memory
- As a result of more lines of codes and more complicated data structures, OOLs tend to increase execution time
- Testing the software is more difficult since many of the common embedded systems testing tools, like the in-circuit emulator (ICE), do not support OOLs.
- There is a lack of experience in the programming language. OOLs are generally very complicated and their use are difficult to learn.
- There is a scarcity of documentation and examples on the applications of OO in real-time embedded systems.
- Changing to object-oriented methods is not just moving to a different language. It is a completely new way of looking at the problem domain. The conceptual leap that is demanded certainly requires changes in old practices of the whole design process; a cost that not many organisations are ready to bear at this stage.

Those obstacles do not however lessen the benefits that object-orientation techniques can bring to the design. OOA and OOD can provide considerable insight and clarity to embedded systems designs which often have complex dynamic semantics. A clear and explicit representation can be beneficial to software maintenance and the reliability of the program, irrespective of the implementation language. The object-oriented diagrammatic representations also allows intervention in the software design without an in-depth knowledge of the problem domain. As soon as the problem is captured in the CRC cards, much of the design can be undertaken by someone who only have a basic understanding of the problem domain.

Experimental Results

The basic hardware system consists of the H8/3048 microcontroller and an interface card that both hosts the microcontroller and acts as an interface between the device and the inverter system. Figure 12 outlines the basic set up.

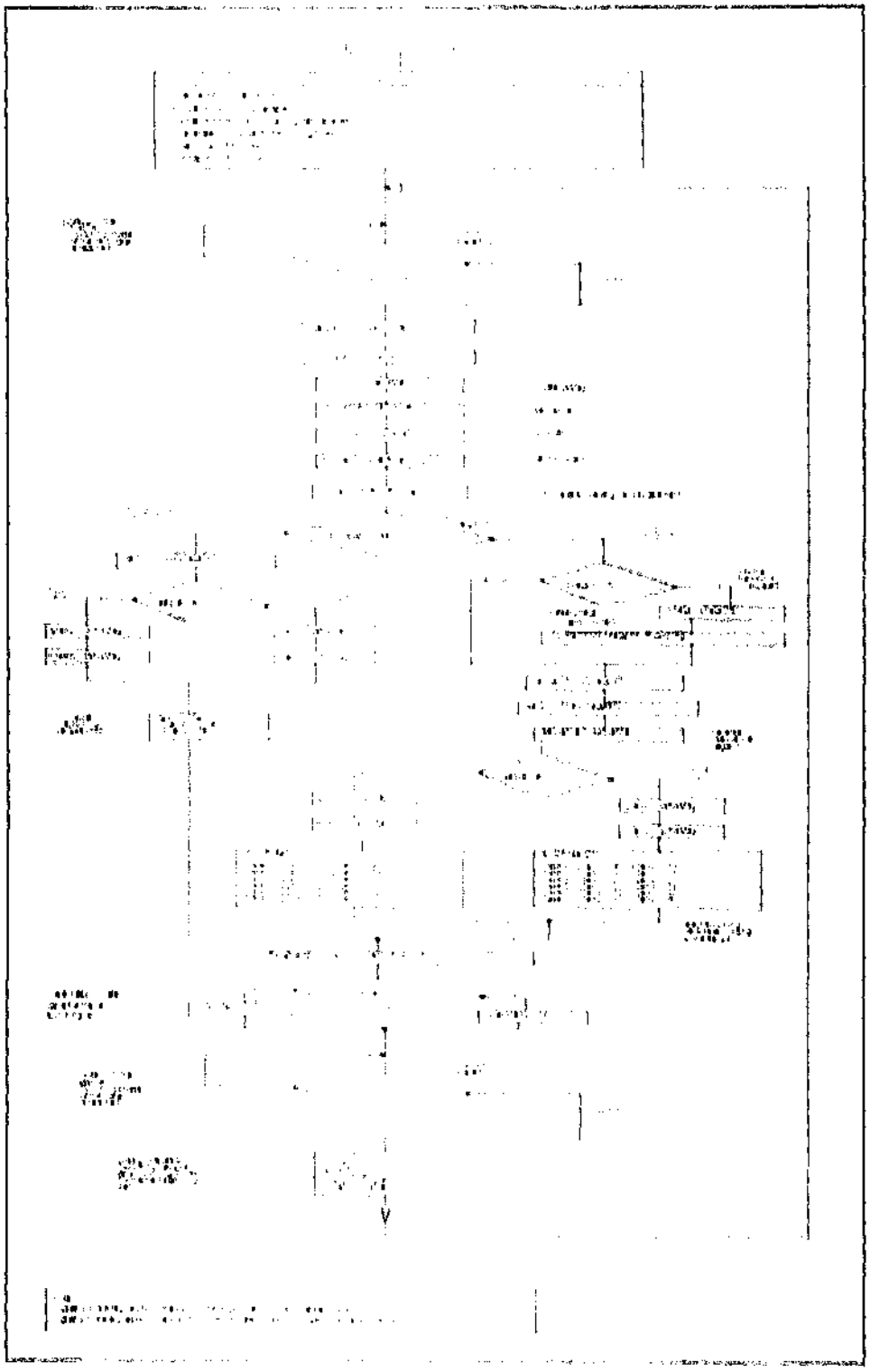


Figure 10: *Main cycle* **flow chart -derived from figure 9 and 8**

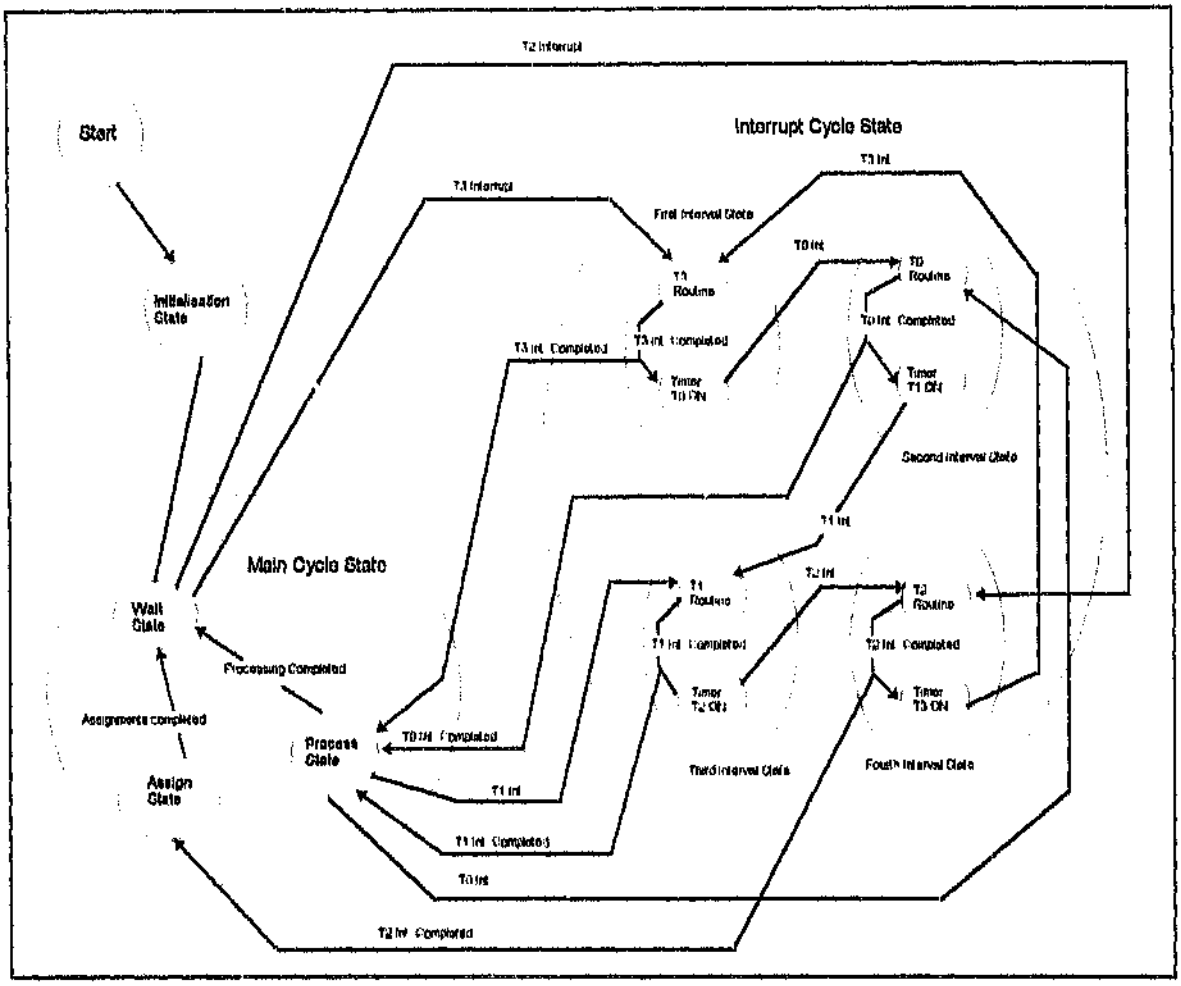


Figure 9: State transition diagram of the overall process

output port. Synchronisation is ensured by including a waiting state into the main cycle state.The beginning of each run through the main cycle loop is synchronised with the end of the first interval stage of the interrupt cycle. That is, the main cycle waits for the first interval stage to complete before calculating a new set of data. Similarly, before making the next set of data (switching times and states) available to the interrupt cycle, the main cycle again goes into a waiting state. This time it waits until the fourth interval of the interrupt cycle is completed. In this way we eliminate the risk of mixing data between sampling intervals. The processing state of the main cycle is where most of the computational activity takes place. This is where the phase-locked loop calculations are performed and where the switching parameters are computed.

The four interval states of the interrupt cycle state correspond to the four periods of each sampling interval (figure 4 above). Each timer interrupt routine stops the timer that has called it, updates the output port according to information obtained from the main cycle, and lastly sets the next timer for the next period. Data is fed by the main cycle to the interrupt cycle at the end of each complete sampling interval. As the next timer is set, control passes from the interrupt cycle to the main cycle where the processing activity is resumed. The interrupt routine of the fourth interval sets timer T3, which eventually causes control to pass back again to the first interval state, thus ensuring a infinite loop.

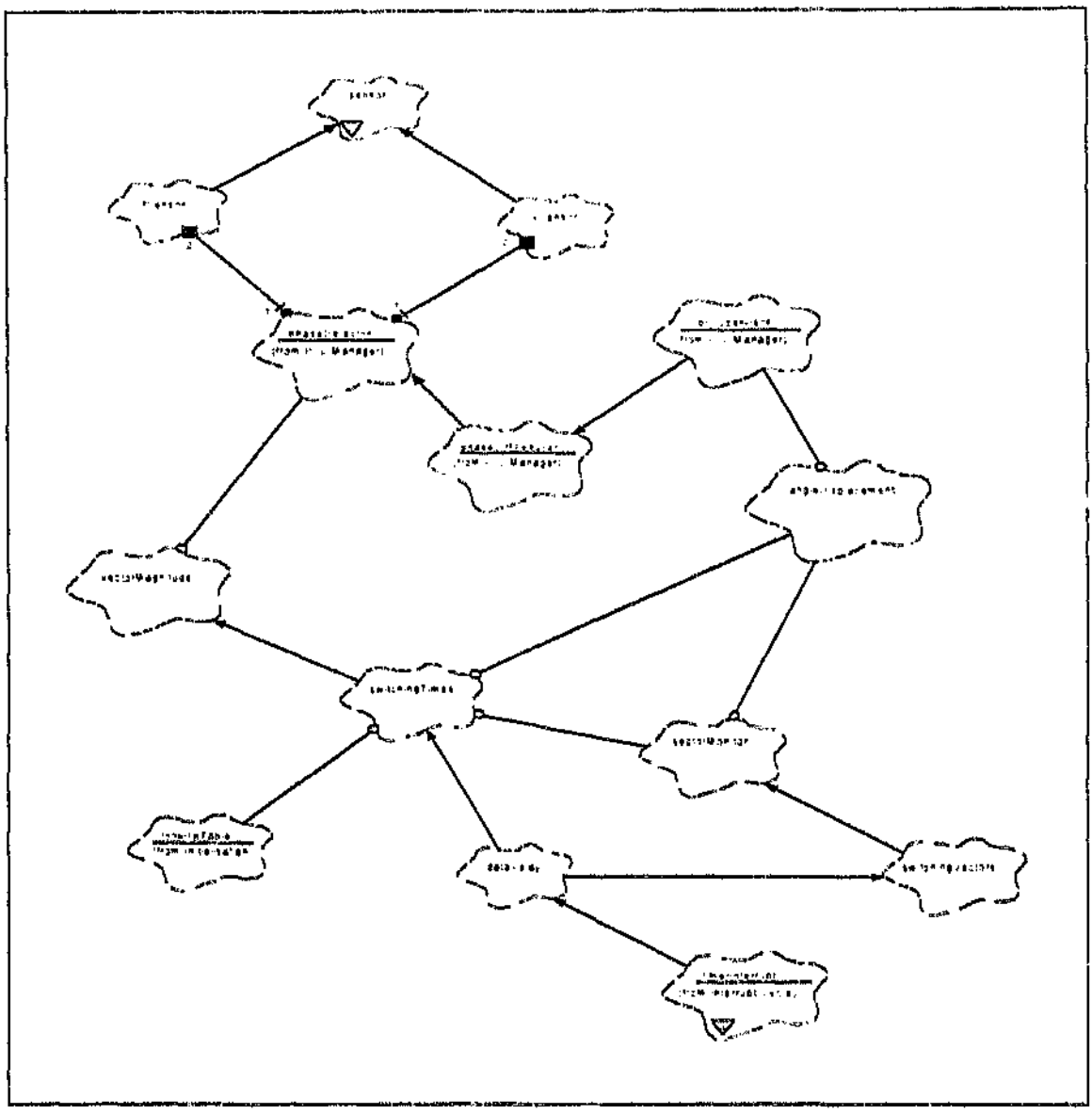


Figure 8: Class diagram for the main cycle

Dynamic Semantics of the Design

The dynamic semantics of the design is represented by a state transition diagram, in figure 9.

The algorithm starts by the initialisation stage where the system is configured properly. At the end of this stage, timer T2 is set to activate the interrupt cycle loop and control passes to the main cycle stage.

Earlier we mentioned that a double-buffering method is used to ensure that the output port is never idle while the switching times and states are being determined. This method requires that the two states, main cycle (responsible for computation) and interrupt cycle (responsible for the output port management) are well synchronised. Otherwise there is the risk of the interrupt cycle obtaining the wrong data, with the result that we do not obtain a PWM waveform at the

vSensor
Respon :bilities: Keeps track of the current voltage Collaborators: - sensor, phaseDetector

fSensor
Reponsibilities: Keeps track of the current frequency Collaborators: - sensor, phaseDetector

As discussed previously, the desired vector Us* has two main parameters, its magnitude and the angle α. The task of evaluating those two parameters can be captured in three classes, angleDisplacement, vectorMagnitude and sectorMonitor. All three are presented below.

angleDisplacement
Responsibilities: keeps track of the angular displacement of the desired vector Us* Collaborators: - sectorMonitor, switchingTimes, pllSupervisor

vectorMagnitude
Reponsibilities: Determines usMag Collaborators: - phaseDetector, switchingTimes

sectorMonitor
Responsibilities: Keeps track of the sector location of the desired vector Us* Collaborators: - angleDisplacement, switchingTimes, switchingVectors

Working out the switching parameters, involves fetching the right timing values from the look-up table and adjusting those values according to the current value of usMag. It also involves choosing the right set of switching vectors from a knowledge of the sector location of the desired vector Us*. Finally the switching parameters have to be relayed to the timer interrupt cycle. Those tasks can be allocated to three classes: switchingTimes, switchingVectors and dataRelay.

switchingTimes
Responsibilities: Calculates the switching times and their order of occurrence. Collaborators: - sectorMonitor, angleDisplacement, vectorMagnitude, lookUpTable, dataRelay

switchingVectors
Reponsibilities: selects the correct switching vectors and their correct order of occurrence. Collaborators: - sectorMonitor, dataRelay

dataRelay
Responsibilities: Assigns correct values to the switching parameters used by the interrupt cycle. Collaborators: - switchingVectors, switchingTimes, timerInterrupt

Figure 8 below shows the class diagram for the main cycle. Notice that a few other classes, from other class ‘es have been included in the diagram, in order he boundaries of the main cycle.

classes are described below. Notice the closed loop behaviour of the timer interrupt classes to ensure a continuous PWM output.

timerInterrupt
Responsibilities: superclass to all 'timer' classes Collaborators: -timerT3Interrupt,timerT0Interrupt,timerT1Interrupt, timerT2Interrupt, dataRelay

timerT3Interrupt	timerT0Interrupt
Responsibilities: Stops timer T3, configures the output port and sets timer T0. Collaborators: - timerInterrupt	Reponsibilities: Stops timer T0, configures the output port and sets timer T1 Collaborators: - timerInterrupt

timerT1Interrupt	timerT2Interrupt
Responsibilities: Stops timer T1, configures the output port and sets timer T2. Collaborators: - timerInterrupt	Reponsibilities: Stops timer T2, configures the output port and sets timer T3 Collaborators: - timerInterrupt

Figure 7 outlines the interrupt cycle class diagram.

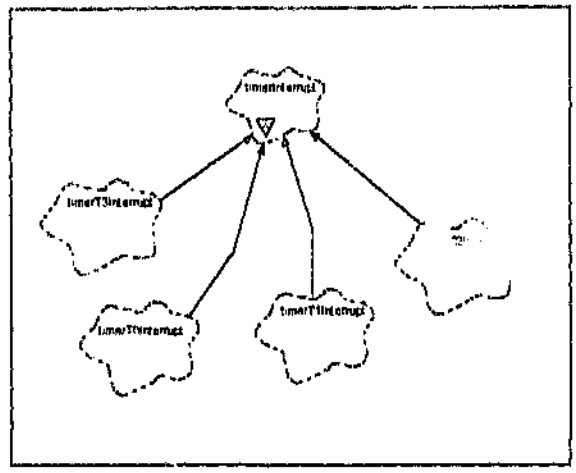


Figure 7: Diagram for the timer interrupt classes

The main areas of concern of the main cycle are:

- sensing activity
- evaluation of the attributes of the desired vector Us* (vector displacement and magnitude)
- Work out of the switching parameters (switching states and switching times)

In the sensing activity, four parameters need to be fed back to the micro-controller: the reserve and inverter output voltages, and reserve and inverter output frequency. Voltage sensing is captured in one class, vSensor while frequency sensing is captured in class fSensor. A third class, sensor, acts as an abstract superclass to the other two classes, vSensor and fSensor.

sensor
Responsibilities: Attaches a sensor to an input port, calibrates the sensor and the calculated value when prompted Collaborators: - vSensor, fSensor

Zhu, J. et al. (May 1995), *Scheduling in Hard Real-Time Applications*, IEEE Software, vol. 12 no. 3., pp. 54-63.

Varnovitsky, M. (1982), *A Microcontroller-Based Control Signal Generator for a Three-Phase Switching Power Inverter*, IEEE IAS Annual Meeting, 1982, Paper CH 1817-6/82/0000-0836, pp. 836-840.

Verner, J. and Tate, G. (April 1992), *A Software Size Model*, IEEE Transactions on Software Engineering, vol. 18. no.4. pp.265-278

Voas. J.M., Miller, K.W. (May 1995), *Software Testability: The New Verification*, IEEE Software, pp. 17-28.

Washabaugh, Douglas, M., Kafura, D. (December 1990), *Incremental Garbage Collection of Concurrent Objects for Real-Time Applications*, Proceedings of the 11th Real-Time Systems Symposium. Lake Buena Vista, Florida, pp. 21-30.

Webster, M. et al. (September 1989), *Space Vector Modulation - A Real-Time Application*, The Transactions of the SAIEE, pp. 38-42.

Weinberg, G. (1971), *The Psychology of Computer Programming*, USA: Van Nostrad Reinhold.

Willlams, T. (July 1992), *Automation Improves Speed and Accuracy of Software Testing*, Computer Design, pp. 57-64.

Williams, T. (September 1992), *Object-Oriented Methods Transform Real-Time Programming* Computer Design, pp. 101-118.

Willlams, T. (November 1992), *Integrated Tool Environment Targets Real-Time Development*, Computer Design, pp. 44-46.

Wolf, W. H. (July 1994), *Hardware-Software Co-Design of Embedded Systems*, Proceedings of the IEEE, vol. 82, no. 7, pp 967-989.

Wulf, W. and Shaw, M. (1973), *Global Variables Considered Harmful*, Sigplan Notices, vol. 8(2).

Yeun, C.K. (1989), *Essential Concepts of Computer Architecture*, Singapore: Addison-Wesley Publishing Co. Ltd.

Zave, P. (May 1982), *An Operational Approach to Requirements Specification for Embedded Systems*, IEEE Transactions on Software Engineering, pp. 250-269.

Zave, P. (February 1984), *The Operational Versus the Conventional Approach to Software Development*, Communications of the ACM, pp. 104-118.

Smith, D.J., Wood, K.B. (1989), *Engineering Quality Software*, Second Edition, UK: Elsevier Applied Science.

Soloway, E., Ehrlich, K. (1984). *Empirical Studies of Programming Knowledge*, IEEE Transactions on Software Engineering, vol. SE-10, no. 5, pp. 595-609.

Sommerville, I. (1992), *Software Engineering*, Fourth Edition, UK: Addison-Wesley Publishing Co.

South African Bureau of Standards (1994), SABS ISO 9001. 1994, *Quality Systems - Model for Quality Assurance in Design, Development, Production, Installation and Servicing*, SA: South African Bureau of Standards, 1994

Stoyenko, A.D., Welch, L.R., Chao Cheng, B. (April 1994), *Response Time Prediction in Object-Based, Parallel Embedded Systems*, Microprocessor and Microprogramming, vol. 40, part 2-3, pp. 135-150.

Strosnider, J.K., Paul, C.J. (Summer 1994), *A Structured View of Real-Time Problem Solving*, vol. 15., part 2, pp. 45-66.

Swartout, W., Balzer, R. (July 1982), *On the Inevitable Intertwining of Specification and Implementation*, Communications of the ACM, pp 438-440.

Taylor, T., Standish, T.A. (December 1982), *Initial Thoughts on Rapid Prototyping Techniques*, Software Engineering Notes, pp. 160-166.

Tzou, S. te al. (November 1993), *A distributed Development Environment for Embedded Software*, Software-Practice and Experience, vol. 23(11), pp. 1235-1248.

Vadivel, S., Bhuvaneswari, G., Sridhara Rao, G. (October 1991), *A Unified Approach to the Real-Time Implementation of Microprocessor-Based PWM Waveform*, IEEE Transactions on Power Electronics, vol. 6, no. 4, pp. 565-575.

Van de Goor, A.J. (1989), *Computer Architecture and Design*, UK: Addison-Wesley Publishing Co. Ltd.

Van der Broek, H., Skudenly, H., Stanke, G. (1986), *Analysis and Realisation of a Pulse Width Modulator Based on voltage Space Vectors*, IEEE IAS Annual Meeting, paper CH 2272-3/86/0000-0244.

Poston, R.M. (March 1995), *Testing Tools Combine Best of New and Old*, IEEE Software, vol. 12, no 2, pp. 122-127.

Rauscher, T.G., Ott, L.M. (1987), *Software Development and Management for Microprocessor-Based Systems*, Englewood Cliffs NJ USA: Prentice-Hall, Inc.

Richardson, J., Kukrer, O. (October 1991), *Implementation of a PWM Regular Sampling Strategy for AC Drives*, IEEE Transactions on Power Electronics, vol. 6, no.4, pp. 645-655.

Rumbaugh, J. et. al. (1991), *Object-Oriented Modelling and Design*, Englewood Cliffs, USA: Prentice-Hall.

Sagoo, J.S., Holding, D.J. (August 1990), *The Specification and Design of Hard Real-Time systems Using Timed and Temporal Petri Nets*, Microprocessor and Microprogramming, vol. 30, part 1-5, pp. 389-396.

Saledian, H. and McClanahan, L.M. (March 1996), *Frameworks for Quality Software Process: SEI Capability Maturity Model Versus ISO 9000*, Software Qality Journal, vol. 5, no. 1, pp. 1-23.

Scharer, L. (1983), *The Prototyping Alternative*, ITT Programming, vol. 1, no. 1, pp. 34-43.

Schneider, G.M., Weigner, S.W., Perlman, D.M. (1982), *An Introduction to Programming and Problem Solving with Pascal*, Second Edition, USA: John Wiley and Sons Inc.

Schneiderman, B., Mayer,R. (1979), *Syntactic/Semantic Interactions in Programming Behaviour: A Model and Experimental Results*, vol. 8, no. 3, pp. 219-238.

Schneidewind, N.F. (May 1992), *Methodology for Validating Software Metrics*, IEEE Transactions on Software Engineering, vol. 18, no. 5, pp. 410-422.

Sen, P. C., Premchandran, G. (1982), *Improved PWM Control Strategy for Inverters and Induction Motor Drives*, IEEE IAS Annual Meeting 1982, Paper CH1817-6/82/0000-0823, pp. 823-830.

Sharp, A. (1993), *Software Quality and Productivity*, USA: Van Nostrad Reinhold.

Shen. C., Ramamritham, K., Stankovic, J.A. (December 1990), *Resource Reclaiming in Real-Time*, Proceedings of the 11th Real-Time Systems Synposium, Lake Buena Vista, Florida, pp. 41-50.

Laffey, T.J. et al., Read J.Y., *Real-Time Knowledge-Based Systems*, AI Magazine, vol. 9(1), pp. 27-45.

Laplante, P.A. (1993), *Real-Time Systems Design and Analysis, An Engineer's Handbook*, USA: IEEE Press.

Le-Huy, H., Dessaint, L. A. (October 1989), *An Adaptive Current Control Scheme for PWM Synchronous Motor Drives: Analysis and Simulation*, IEEE Transactions on Power Electronics, vol. 4, no. 4, October 1989, pp. 486-495.

Levin, R.I., Rubin, D.S. (1991), *Statistics for Management*, Fifth Edition, USA: Prentice-Hall Inc.

Manassewistch, V. (1980), *Frequency Synthesizers Theory and Design*, Second Edition, USA: John Wiley and Sons.

McCraken, D.D., Jackson, M.A. (1981), *A Minority Dissenting Position - Systems Analysis and Design - A Foundation for the 1980's*, Cotterman W.W., et al. (editors), UK: Elsevier Science Publishing Co., Inc., pp. 551-553.

McKeithen, K.B. et. al. (1981), *Knowledge Organisation and Skill Differences in Computer Programmers*, Cognitive Psychology, vol. 13, pp. 307-325.

Mirkazemi-Moud, M., Green, T.C., Williams, B.W. (January 1993), *Analysis and Comparison of Real-Time Sine-Wave Generation for PWM Circuits*, IEEE Transactions on Power Electronics, vol. 8,no. 1, pp. 46-54.

Mohanty, S.N. in Sommerville, I. (1992), *Software Engineering*, Fourth Edition, UK: Addison-Wesley Publishing Co.

Odette, L.L. (1991), *Intelligent Embedded Systems*, USA: Addison -Wesley Publishing Co.

Pacaut, R., Perret, R., Le-Huy, H. (1982), *Microprocessor Control of a Current-Fed Induction Motor*, IEEE IAS Annual Meeting 1982, Paper CH1817-6/82/0000-0457, pp. 457-461.

Partsch, H., Steinbrüggen, R. (September 1983), *Program Transformation Systems*, Computing Surveys, pp. 199-236.

Paulk, M.C. (January 1995), *How ISO 9001 Compares with the CMM*, IEEE Software, vol. 12, no. 1, pp. 75-83.

Hill, I.D., Meek, B.L. (1980), *Programming Language Standardisation*, UK: Ellis Horwood Ltd.

Honka, H., Kattilakoski, M. (1991), *A Simulation-Based System for testing Real-Time Embedded Software in the Host Environment*, Microprocessor and Microprogramming, vol. 32, part 1-5, pp. 127-134.

Humphrey, W.S. (1990), *Managing the Software Process*, USA: Addison-Wesley Publishing Co.

Ince, D. (1994), *An Introduction to Software Quality Assurance and its Implementation*, UK: McGraw-Hill Book Co.

Ince, D. (1994), *ISO 9001 and Software Quality Assurance*, UK: McGraw-Hill Book Co.

Ingalls, D. (1989), *Object-oriented programming. Video tape*, Apple Computer, Inc., Part of the University Video Communications collection, Distinguished Lecture Series, vol. II.

Iwaniw, P. (November 1990), *An Improved Intelligent A.C. Drive for Industrial Applications*, Power Quality Proceeedings, pp.118-124.

Joos, G., Ziogas, P., Vincenti, D. (October 1990), *A Model Reference Adaptive PWM Technique*, IEEE Transactions on Power Electronics, vol. 5, no. 4, pp. 485-494.

Kazmierkowski, M.P., Dzieniakowski, M.A., Sulkowski, W. (January 1991), *Novel Space Vector Based Current Controllers for PWM-Inverters*, IEEE Transactions on Power Electronics, vol. 6, no. 1, pp. 158-165.

Kitchenham, B. and Pfleeger, S.L. (January 1996), *Software Quality: The Elusive Target*, IEEE Software, vol. 13, no. 1, pp. 12-21.

Klapper, J., Frankle, J.T. (1972) *Phase-Locked and Frequency Feedback Systems*, USA: Academic Press, Inc.

Koch, . (May 1996), *Hitachi Microcontroller Seminar Summary -May 1996*, South Africa.

Kwon, B. and Min, B. (June 1993), *A Fully Software-Controlled PWM Rectifier with Current Link*, IEEE Transactions on Industrial Electronics, vol. 40, no. 3, pp. 355-363.

Kusko, A., Galler, D. (1982), *Survey of Microprocessors in Industrial Motor Drive Systems*, IEEE IAS Annual Meeting 1982, Paper CH1817-6/82/0000-0435, pp. 435-438.

Dunham, J.R., Kruesi, E. (November 1983), *The Measurement Task Area*, IEEE Computer, vol. 16, no. 11.

Everett, W.W., Honiden, S. (May 1995), *Reliability and Safety of Real-Time Systems*, IEEE Software, pp. 13-16.

Fenton, N. (January 1996), *Do Standards Improve Quality? - Counterpoint*, IEEE Software, vol. 13, no. 1, pp. 23-24.

Friedman, P. (1981), *Computer Programs in Basic*, USA: Prentice-Hall Inc.

Fukuda, S., Iwaji, Y., Hasegawa, H. (January 1990), *PWM Technique for Inverter with Sinusoidal Output Current*, IEEE Transactions of Power Electronics, vol. 5, no. 1, pp. 54-60.

Fuori, W.M. (1975), *Introduction to American National Standard Cobol*, USA: McGraw-Hill Book Company.

Ghezzi, C., Jazayeri, M. (1982), *Programming Language Concepts*, USA: John Wiley and Sons Inc.

Giddings, R.V. (May 1984), *Accommodating Uncertainty in Software Design*, Communications of the ACM, pp. 428-434.

Goldsack, S.J., Finkelstein, A.C.W. (May 1991), *Requirements Engineering for Real-Time Systems*, Software Engineering Journal, vol. 6, part 3, pp. 101-115.

Gomaa, H., Scott, D.B.H. (1981), *Prototyping as a Tool in the Specification of User Requirements*, The Proceedings of the 5th International Conference on Software Engineering, pp. 333-342.

Gordon, V.S. and Bieman, .M. (January 1995), *Rapid Prototyping: Lessons Learned*, IEEE Software, vol 12, no. 1, pp.86-95

Granado, J., Harley, R.G., Diana, G. (September 1989), *Understanding and Designing a Space Vector Pulse-Width-Modulator to Control a Three -Phase Inverter*, The Transactions of SAIEE, pp. 29-27.

Gupta, R.J., Coelho, C.N. (May 1991), *Program Implementation Schemes for Hardware-Software Systems*, vol. 27, part 1, pp. 48-55.

Harel, D. (1987), *A Visual Approach to Complex Systems*, Science of Computer Programming.

Brodie L. (1981), *Starting FORTH*, USA: Prentice-Hall, Inc.

Brooks, F.P. (April 1987), *No Silver Bullet -Essence and Accidents of Software Engineering*, IEEE Computer, pp. 10-19.

Cameron, J.R. (February 1986), *An Overview of JSD*, IEEE Transactions on Software Engineering, pp. 222-240.

Caroll, J.M., Thomas, J.C., Malhotra, A. (1979), *Clinical-Experimental Analysis of Design Problem Solving*, Design Studies, pp. 84-92.

Chiodo, M. et al. (August 1994), *Hardware-Software Codesign of Embedded Systems*, IEEE Micro, vol. 14, part 26-36, pp. 26-36.

Christian, K. (1986), *A guide to Modula-2*, USA: Springer-Verlag Inc.

Clark, R.G. (May 1990), *The Design and Development of Embedded Ada Systems*, Software Engineering Journal, vol. 5., part 3, pp. 175-184.

Cox, B. (1991), *Object-oriented Programming: An Evolutionary Approach*, New York: Addison Wesley.

Cragon, G.H. (October 1980), *The Elements of Single-Chip Microcomputer Architecture*, IEEE Computer, vol. 13, part 32-41, pp. 27-30.

Crockett, H.D. (March 1995), *What's the Proper Role for CASE Tools? - Counterpoint*, IEEE Software, pp. 19.

Curtis, B (1985), *Human Factors in Software Development*, Second Edition, USA: IEEE Computer Society.

De Carli, (1983), *Direct Computation of Optional PWM voltages*, IPEC Tokyo 1983, pp. 367-383.

De Carlini, U. and Vilano, U. (1991), *Transputers and parallel architectures*, UK: Ellis Horwood Limited.

De Lemos, R., Saeed, A., Anderson, T. (May 1995), *Analyzing Safety Requirements for Process-Control Systems*, IEEE Software, pp. 42-52.

De Marco, T. (March 1995), *Blueprint for Success: Innovate and Invest in People*, IEEE Software, vol. 12, no. 2, pp. 108-109.

Drusinsky, D. (October 1994), *Extended State Diagrams and Reactive Systems*, Dr. Dobb's Journal.

B.odie L. (1981), *Starting FORTH*, USA: Prentice-Hall, Inc.

Brooks, F.P. (April 1987), *No Silver Bullet -Essence and Accidents of Software Engineering*, IEEE Computer, pp. 10-19.

Cameron, J.R. (February 1986), *An Overview of JSD*, IEEE Transactions on Software Engineering, pp. 222-240.

Caroll, J.M., Thomas, J.C., Malhotra, A. (1979), *Clinical-Experimental Analysis of Design Problem Solving*, Design Studies, pp. 84-92.

Chiodo, M. et al. (August 1994), *Hardware-Software Codesign of Embedded Systems*, IEEE Micro, vol. 14, part 26-36, pp. 26-36.

Christian, K. (1986), *A guide to Modula-2*, USA: Springer-Verlag Inc.

Clark, R.G. (May 1990), *The Design and Development of Embedded Ada Systems*, Software Engineering Journal, vol. 5., part 3, pp. 175-184.

Cox, B. (1991), *Object-oriented Programming: An Evolutionary Approach*, New York: Addison Wesley.

Cragon, G.H. (October 1980), *The Elements of Single-Chip Micro-computer Architecture*, IEEE Computer, vol. 13, part 32-41, pp. 27-30.

Crockett, H.D. (March 1995), *What's the Proper Role for CASE Tools? - Counterpoint*, IEEE Software, pp. 19.

Curtis, B (1985), *Human Factors in Software Development*, Second Edition, USA: IEEE Computer Society.

De Carli, (1983), *Direct Computation of Optional PWM voltages*, IPEC Tokyo 1983, pp. 367-383.

De Carlini, U. and Vilano, U. (1991), *Transputers and parallel architectures*, UK: Ellis Horwood Limited.

De Lemos, R., Saeed, A., Anderson, T. (May 1995), *Analyzing Safety Requirements for Process-Control Systems*, IEEE Software, pp. 42-52.

De Marco, T. (March 1995), *Blueprint for Success: Innovate and Invest in People*, IEEE Software, vol. 12, no. 2, pp. 108-109.

Drusinsky, D. (October 1994), *Extended State Diagrams and Reactive Systems*, Dr. Dobb's Journal.

Balzer, R.M., Goldman, N.M., Wile, D.S. (December 1982), *Operational Specification as the Basis for Rapid Prototyping*, ACM SIGSOFT Software Engineering Notes, pp. 3-16.

Barnes, J.G.P (1976), *RTL/2 Design and Philosophy*, UK: Heyden and Sons Ltd.

Barnes, J.G.P. (1982), *Programming in ADA*, UK: Addison-Wesley Publishing Co. Ltd.

Blanchard A. (1976), *Phase-Locked Loops*, USA: John Wiley and Sons.

Blasciak, A.J., Neuder, D.L., Berger, A.S. (April 1993), *Software Performance Analysis of Real-Time Embedded Systems*, Hewlett-Packard Journal, vol. 44, part 2, pp. 107-115.

Blumenthal, M. K. (1982), *Why PWM Inverters Beat Other Drive Actuators when it comes to Flexibility and Distribution System Compatibility*, IEEE IAS Annual Meeting 1982, Paper CH 1817-6/ 82/0000-0536, pp. 536-543.

Boehm, B.W. (December 1976), *Software Engineering*, IEEE Transactions on Computers, vol. C-25, no. 12, pp. 1226-1241.

Boehm, B.W. (1981), *Software Engineering Economics*, Englewood Cliffs NJ, USA: Prentice-Hall.

Boehm, B.W. (May 1988), *A Spiral Model of Software Development and Enhancement*, IEEE Computer, vol. 21 (5), pp 61-72.

Booch, G. (1994), *Object-Oriented Analysis and Design*, Second Edition, USA: The Benjamin/ Cummings Publishing Co., Inc.

Booker, A., McKeeman, J. (August 1993), *Design Considerations for RISC Microprocessor In Realtime Embedded Systems*, Computer Design, pp. 71-72.

Boost, M., Ziogas, P. (March/April 1988), *State-of the-Art Carrier PWM Techniques: A Critical Evaluation*, IEEE Transactions on Industrial Applications, vol. 24, no.2, pp. 271-280.

Bose, B. K., Sutherland, H.A. (March/April 1983), *A high-Performance Pulsewidth Modulator for an Inverter-Fed Drive System Using a Microcomputer*, IEEE Transactions on Industrial Applications, vol. IA-19, no. 2, pp. 235-243.

Bose, B. K. (1992), *Modern Power Electronics*, USA: IEEE Press.

1 References

Abdel-Hamid In Burgess, A. (January 1995), *Mad About or Mad at Measurement?*, IEEE Software, vol. 12, no. 1, pp. 115-116.

Abott, R. (1983), *Program Design by English Descriptions*, Communications of the ACM, vol. 26 (11), pp. 882-894.

Abraxas Software, Inc (1989), *PCYACC, Professional Language Development Toolkit*, USA: Abraxas Software Inc.

Acharya, G.N. et al. (1982), *A Pulse Width Modulated AC Motor Drive for Mining Locomotives*, IEEE IAS Annual Meeting 1982, Paper CH1817-6 ,/82/0000-0500, pp. 500-505.

Adamson, M. (1990), *Small Real-Time Design*, Sigma Press, UK: John Wiley and Sons.

Adelson, B (1984), *When Novices Surpass Experts: The Difficulty of a Task May Increase with Expertise*, Journal of Experimental Psychology,: Learning, Memory and Cognition, vol. 10, no. 3, pp. 483-495.

Agresti, W.W. (1986), *New Paradigms for Software Development*, USA: IEEE Computer Society.

Alavi, M. (June 1984), *An Assessment of the Prototyping Approach to Information Systems Development*, Communucations of the ACM, pp. 556-563.

Allard, J.R., Hawkinson, L.B. (September 1991), *Real-Time Programming in common LISP*, Communications of the ACM, vol. 35(9), pp. 64-69.

Aquilano, N.J., Chase, R.B. (1991), *Fundamentals of Operations Management*, USA: Richard D. Irwin ,Inc.

Asumadu J. A., Hoft R. G. (April 1989), *Microprocessor-Based Sinusoidal Waveform Synthesis Using Walsh and Related Orthogonal Functions*, IEEE Transactions on Power Electronics, vol. 4, no. 2, pp. 234-241.

Auer, A. et al. (1990), *Solution in Software Crisis"*, *Microprocessor and Microprogramming*, vol. 30, part 1-5, pp. 273-280.

Bach, J. (March 1995), *Enough About Processes: What we Need are Heroes*, IEEE Software, vol 12, no. 2, pp. 96-98.

matter and is not a production activity. The most important elements of research and development are the people. Consequently, any future work on development strategies, will have to be weighed against and triggered by consideration of their psychology and their problems.

Another interesting study would be to investigate a formal means of capturing the justification of design decisions in object-oriented applications. Classes specifications and the whole object-oriented structure in general are quite well pictured in the Classes, Responsibilities and Collabotators (CRC) cards, and the four models of object-oriented development (logical and physical, static and dynamic). However during this project it was observed that the reason behind a particular choice of abstraction or the relationship between the objects was left to the developer to be documented in his/her own ways. This situation can lead to two problems. Firstly, since it is not a formal prerequisite of the methodolgy, justification of design decisions are often not even documented. Secondly, whenever it is documented, the lack of standard supporting the exercise, makes it prone to subjective factors such as writing style (too succint or too verbose). The end result is a poor communication of ideas. Definitions and specifications of the elements of a design are important, but often a person external to a project is also concerned about the reason/s the designer made certain decisions. When those questions cannot be answered, a considerable amount of unknown risks can be carried out if modifications are undertaken, or wasted efforts can result if the decision is to avoid making changes and to build a new design. Of course, the independent modules generated by an object-oriented design, can always be re-used, simply with a knowledge of their specifications and definitions, but such information might prove to be insufficient to encourage modifications with confidence on a whole design. Development tools, like Rational Rose /C++, try to circumvent this problem by giving the developer the possibility to add comments to the classes and the relationships, but still, this approach is not formal enough and has the further inconvenience of losing the additional information into the web of classes and objects. The normalisation or standardisation of a method to capture design decisions in object-oriented design, is a potential area for improvement which might lead to an even wider use and acceptance of object-orientation in general.

■

Document Name	Document Number	Revision Number	Revision Date	Document Status	Date approved by Board	Minute Reference	File Reference
Minutes: Meeting on 95/02/17 with the Product Manager	sde 505	1.00	95/02/20	Approved	95/02/28	2.1.3	\1995-13\qa\minutes\950217-1.min
Agenda: Meeting on 95/02/28 with the Product Manager	sde 506	1.00	95/02/28	Approved	95/02/28	2.1.2	\1995-13\qa\minutes\950228-1.agd
Minutes: Meeting on 95/02/28 with the Product Manager	sde 507	1.00	95/03/01	Approved	95/03/06	2.1.3	\1995-13\qa\minutes\950228-1.min
Agenda: Meeting on 95/03/06 with the Product Manager	sde 508	1.00	95/03/06	Approved	95/03/06	2.1.2	\1995-13\qa\minutes\950306-1.agd
Minutes: Meeting on 95/03/06 with the Product Manager	sde 509	1.00	95/03/07	Approved	95/04/07	2.1.3	\1995-13\qa\minutes\950306-1.min
Agenda: Meeting on 95/04/07 with the Product Manager	sde 510	1.00	95/04/07	Approved	95/04/07	2.1.2	\1995-13\qa\minutes\950407-1.agd
Minutes: Meeting on 95/04/07 with the Product Manager	sde 511	1.00	95/04/10	Approved	95/05/26	2.1.3	\1995-13\qa\minutes\950407-1.min
Agenda: Meeting on 95/05/26 with the Product Manager	sde 512	1.00	95/05/26	Approved	95/05/26	2.1.2	\1995-13\qa\minutes\950526-1.agd

Document Name	Document Number	Revision Number	Revision Date	Document Status	Date approved by Board	Minute Reference	File Reference
Discussion and Conclusions	sde 117	1.00	96/08/27	Approved	96/08/26	2.4	\1995-13\tp\sde117ww.100
Bibliography /References	sde 118	1.00	96/09/01	Approved	96/08/26	2.4	\1995-13\tp\sde118ww.100
Front pages of Dissertation	sde 119	1.00	96/09/01	Approved	96/08/26	2.4	\1995-13\tp\sde119ww.100
Disposition of Comments Register	sde 120	0.01	96\12\09	Draft			\1995\13\tp\sde120ww.001
QUALITY ASSURANCE PRODUCTS							
Agenda: Meeting on 95/01/31 with the Product Manager	sde 500	1.00	95/01/31	Approved	95/01/31	2.1.2	\1995-13\qa\minutes\950131-1.agd
Minutes: Meeting on 95/01/31 with the Product Manager	sde 501	1.00	95/02/01	Approved	95/02/26	2.1.3	\1995-13\qa\minutes\950131-1.min
Agenda: Meeting on 95/02/06 with the Product Manager	sde 502	1.00	95/02/06	Approved	95/02/06	2.1.2	\1995-13\qa\minutes\950206-1.agd
Minutes: Meeting on 95/02/06 with the Product Manager	sde 503	1.00	95/02/07	Approved	95/02/17	2.1.3	\1995-13\qa\minutes\950206-1.min
Agenda: Meeting on 95/02/17 with the Product Manager	sde 504	1.00	95/02/17	Approved	95/02/17	2.1.2	\1995-13\qa\minutes\950217-1.agd

Document Name	Document Number	Revision Number	Revision Date	Document Status	Date approved by Board	Minute Reference	File Reference
Embedded Systems: A Literature Survey	sde 103	1.00	96/08/25	Approved	96/08/12	2.3	\1995-13\tp\sde103wp.100
Requirements Analysis	sde 104	1.00	96/08/25	Approved	96/08/12	2.3	\1995-13\tp\sde104wp.100
Inverter ControlTechniques and Phase-locked loop	sde 105	1.00	96/08/25	Approved	96/08/12	2.3	\1995-13\tp\sde105wp.100
Functional Specifications	sde 106	1.01	96/07/18	Approved	96/03/12	2.3	\1995-13\tp\sde106wp.100
Software Design Specifications - Part 1	sde 107	1.00	96/07/18	Approved	96/03/25	2.4	\1995-13\tp\sde107wp.100
Software Design Specifications - Part 2	sde 108	1.00	96/07/18	Approved	96/03/25	2.4	\1995-13\tp\sde108wp.100
Progress Report	sde 109	0.01	96/03/30	Approved	96/03/25	2.3	\1995-13\tp\sde109ww.100
Software and Hardware Implementation Documentation	sde 110	1.00	96/08/25	Approved	96/08/12	2.3	\1995-13\tp\sde110ww.100
Project Presentation (SEAL)	sde 111	0.01	96/04/11	Draft		-	\1995-13\tp\sde111ww.001
Testing Documentation	sde 112	1.00	96/08/25	Approved	96/08/12	2.3	\1995-13\tp\sde112ww.100
Embedded Systems: A Review	sde 113	1.00	96/08/25	Approved	96/08/12	2.3	\1995-13\tp\sde113ww.100
Project Overview	sde 114	1.00	96/08/27	Approved	96/08/26	2.4	\1995-13\tp\sde114ww.100
M.Sc. Paper 1	sde 115	1.00	96/09/01	Approved	96/08/19	2.4	\1995-13\tp\sde115ww.100
M.Sc. Paper 2	sde 116	1.00	96/09/01	Approved	96/09/26	2.4	\1995-13\tp\sde116ww.100

2 Master Document List

Document Name	Document Number	Revision Number	Revision Date	Document Status	Date approved by Board	Minute Reference	File Reference
SEAL QMS DOCUMENTS: SYSTEM DEVELOPMENT							
Master Document List	sde 001	1.00	96/09/01	Approved	96/06/03	2.6	\1995-13\mp\sde001ww.100
Document Creation Template	sde 002	1.00	96/09/01	Approved	96/06/03	2.6	\1995-13\mp\sde002ww.100
Quality Plan	sde 003	1.00	96/09/01	Approved	96/06/03	2.6	\1995-13\mp\sde003ww.100
Product Description	sde 004	1.00	96/09/01	Approved	96/06/03	2.6	\1995-13\mp\sde004ww.100
Project Management Plan	sde 005	1.00	96/09/01	Approved	96/06/03	2.6	\1995-13\mp\sde005ww.100
Configuration Management Plan	sde 006	1.00	96/09/01	Approved	96/06/03	2.6	\1995-13\mp\sde006ww.100
Binder Label	sde 008	1.00	96/09/01	Approved	96/06/03	2.6	\1995-13\mp\sde008ww.100
TECHNICAL PRODUCTS							
Research Proposal (WITS)	sde 100	1.00	95/03/06	Approved	95/03/06	2.3	\1995-13\tp\sde100wp.100
Project Proposal (Meissner)	sde 101	1.00	95/03/03	Approved	95/03/03	-	\1995-13\tp\sde101wp.100
Software Development : A General Survey of the Literature	sde 102	1.00	96/08/25	Approved	96/08/12	2.3	\1995-13\tp\sde102wp.100

- The Engineering Manager, Meissner

1.6 Applicable Documents

1.6.1 Specifications

1.6.2 Standards

a. SEAL QMS Document Creation Template, QS 002, Revision 0.01, 21 June 1994

b. SEAL QMS Document Layout, Presentation and Typesetting Guide, QS 003, Revision 0.02, 7 June 1994.

1.7 Assumptions

It is assumed that the reader is familiar with the ISO 9000 series standard for quality systems management.

1.8 Requirements Traceability

This document addresses the following requirements:

a. ISO 9001 (1994) 4.5 Document and Data Control

b. ISO 9001 (1994) 4.16 Quality Records

1 Scope

1.1 Introduction

An essential feature of a quality management system is that it documents the procedures used to implement and maintain the system. This document is the Document Master List which provides a directory of all documents which have the status of Draft, Provisional or Approved.

1.2 Purpose

This Document Master List provides the cross reference to all documents comprising the SDES QMS project.

1.3 Applicability

This document is an essential reference to all documents supported in the SDES QMS projec..

1.4 Definitions

- SI " Software Development for Embedded Systems
- SEAL: Software Engineering Applications Laboratory (University of the Witwatersrand)

1.5 Audience

The audience for this document comprise the various stakeholders of the SEAL, including:

- The product developer
- The product manager
- All full-time and part-time post-graduate students associated with SEAL
- Members of the SEAL Management Board
- Head of the Department, Electrical Engineering
- Individuals who will perform internal and external surveillance audits of the SEAL Quality Management System

Change Forecast

This document will be updated each time a new documented element is added into the SDES QMS project.

Change History

Configuration Control

Project:	SDES
Title:	Master Document List
Doc. Reference:	SDE 001
Created by:	Hansraj Bapoo
Creation Date:	28 August 1995

Document History

Version	Date	Status	Who	Saved as:
0.01	95/08/28	Draft	H.Bapoo	\1995-13\MP\ SDE001WW.001
0.02	96/05/29	Draft	H.Bapoo	\1995-13\MP\ SDE001WW.002
1.00	96/09/01	Approved	H.Bapoo	\1995-13\MP\ SDE001WW.100
1.01	96/12/08	Approved	H.Bapoo	\1995-13\MP\ SDE001WW.101

Revision History

Version	Date	Changes
0.01	95/08/28	New Document - derived from QST001-10
0.02	96/05/29	Updates to Master List
1.00	96/09/01	Updates to Master List and update to revision number and status following Board approval.
1.01	96/12/08	Update to Master List to reflect addition of document SDE120

Management Authorisation

Version	Date	Status	Management Board Minute Reference
1.00	96/09/01	Approved	Section 2.6 of SDE 580
1.01	96/12/08	Approved	-

Table of Contents

Table of Contents ii

Change History iii

Configuration Control iii

Document History iii

Revision History iii

Management Authorisation iii

Change Forecast iv

1 Scope 1

1.1 Introduction 1

1.2 Purpose 1

1.3 Applicability 1

1.4 Definitions 1

1.5 Audience 1

1.6 Applicable Documents 2

1.7 Assumptions 2

1.8 Requirements Traceability 2

2 2

2 Master Document List 3



SDES

Master Document List

Management Product

Version 1.01

Document Status: Approved

Schultz, T.W. (1993), *C and the 8051: Programming and Multitasking*, Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

Swan, T. (1992), *C++ Primer*, First Edition, USA: Sams Publishing.

Yourdon, E. (May 1995), *When Good Enough Software Is Best*, IEEE Software, vol. 12, no. 3, pp. 79-81.

2 Bibliography

Note: This section lists some sources of additional reading on the subjects treated in this study.

Brown, J.F. (1994), *Embedded Systems Programming in C and Assembler*, USA: Van Nostrand Reinhold.

Hintz, K. and Tabak, D. (1992), *Microcontrollers: Architecture, Implementation, and Programming*, USA: McGraw-Hill Inc.

Hovenden, F.M. et al. (March 1996), *Building Quality into Scientific Software*, Software Quality Journal, vol. 5, no. 1, pp. 25-32.

IEEE, Inc. (1985), *Advanced Microprocessors*, New York: IEEE, Inc.

Lehman, M.M. (September 1980), *Programs, Life Cycles, and Laws of Software Evolution*, Proceedings of the IEEE, vol. 68, No. 9, pp. 1060-1076.

Lippman, S. (1991), *C++ Primer*, Second Edition, USA: Addison-Wesley Publishing Co.

MacKenzie, I.S. (1995), *The 8051 Microcontroller*, Second Edition, USA: Prentice Hall Inc.

March, A. (Spring 1996), *A Note on Quality: The Views of Deming, Juran and Crosby*, IEEE Engineering Management Review, vol. 24, no. 1, pp. 6-23.

Micro Cornucopia (March-April 1990), *C++ 2.0: A Stable Standard for the OOP Leader?*, Micro Cornucopia, no. 52, pp. 8-17.

Miller, G.H. (1993), *Microcomputer Engineering*, Englewood Cliffs, New Jersey: Prentice Hall Inc.

Rooijmans, J., Aerts, H. and Van Genuchten, M. (January 1996), *Software Quality in Consumer Electronics Products*, IEEE Software, vol. 13, no. 1, pp. 55-64.

Ross, D.T., Goodenough, J.B. and Irvine, C.A. (May 1975), *Software Engineering: Process, Principles, and Goals*, Computer, pp. 54-64.



SDES

Project Management Plan

Management Product

Version 1.00

Document Status: Approved

Document Name	Document Number	Revision Number	Revision Date	Document Status	Date approved by Board	Minute Reference	File Reference
Minutes: Meeting on 96/08/19 with the Product Manager	sde 596	0.01	96/08/19	Draft	-	-	\1995-13\qa\minutes\960819-1.min

Document Name	Document Number	Revision Number	Revision Date	Document Status	Date approved by Board	Minute Reference	File Reference
Agenda: Meeting on 96/07/22 with the Product Manager	sde 588	1.00	96/07/22	Approved	96/07/22	2.1.2	\1995-13\qa\minutes\960722-1.agd
[illegible] 96/07/22 with the Product Manager							
[illegible] Documents Submitted							
Agenda: Meeting on 96/08/12 with the Product Manager	sde 591	1.00	96/08/12	Approved	96/08/12	2.1.2	\1995-13\qa\minutes\960812-1.agd
Minutes: Meeting on 96/07/22 with the Product Manager	sde 592	1.00	96/08/12	Approved	96/08/19	2.1.3	\1995-13\qa\minutes\960812-1.min
Agenda: Meeting on 96/08/19 with the Product Manager	sde 593	1.00	96/08/19	Approved	96/08/19	2.1.2	\1995-13\qa\minutes\960819-1.agd
Minutes: Meeting on 96/08/19 with the Product Manager	sde 594	1.00	96/08/19	Approved	96/08/26	2.1.3	\1995-13\qa\minutes\960819-1.min
Agenda: Meeting on 96/08/26 with the Product Manager	sde 595	1.00	96/08/26	Approved	96/08/25	2.1.2	\1995-13\qa\minutes\960819-1.agd

Document Name	Document Number	Revision Number	Revision Date	Document Status	Date approved by Board	Minute Reference	File Reference
Message from Prof. Walker Re: Reply to sde 577	sde 578	0.01	96/06/03	Draft	-	-	\1995-13\qa\960603-1.txt
Message to Prof. Walker Re: [illegible]	sde 579	0.01	96/06/03	Draft			\1995-13\qa\960603-2.txt
Minutes: Meeting on 96/06/03 with the Product Manager	sde 580	0.01	96/06/03	Approved	96/06/27	2.1.5	\1995-13\qa\minutes\960603-1.min
Message to Prof Landy Re: Borrowing of Oscilloscope	sde 581	0.01	96/06/03	Draft			\1995-13\qa\960603-3.txt
Message from Prof Landy Re: Reply to sde 581	sde 582	0.01	3/06/03	Draft	-	-	\1995-13\qa\960603-4.txt
Message to Prof Landy. Re: Reply to sde 582	sde 583	0.01	96/06/03	Draft			\1995-13\qa\960603-5.txt
Message from Russel Hersch. Re: Query on FAQ	sde 584	0.01	96/06/24	Draft	-	-	\1995-13\qa\960624-1.txt
Message to G. Diana. Re: Query on SVM	sde 585	0.01	96/06/28	Draft	-	-	\1995-13\qa\960628-1.txt
Message from G.Diana. Re: Reply to sde 585	sde 586	0.01	96/07/01	Draft	-	-	\1995-13\qa\960701-1.txt
Message from P. Ryalls. Re: OO Applications	sde 587	0.01	96/07/05	Draft	-	-	\1995-13\qa\960705-1.txt

Document Name	Document Number	Revision Number	Revision Date	Document Status	Date approved by Board	Minute Reference	File Reference
Agenda: Meeting on 96/04/29 with the Product Manager	sde 570	1.00	96/04/27	Approved	96/04/29	2.1.2	\1995-13\qa\minutes\960429-1.agd
[illegible] 96/04/29 with the Product Manager	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible] of Cygnus. Re: Software Support for Embedded Products	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Message to Cygnus Re: Reply to SDE 572	sde 573	0.01	96/05/17	Draft	-	-	\1995-13\qa\960517-1.txt
Message from Warren Ellis of Intel. Re: Embedded Products	sde 574	0.01	96/05/17	Draft	-	-	\1995-13\qa\960517-2.txt
Message from Kathy Powers of Cygnus. Re: Reply to SDE 573	sde 575	0.01	96/05/21	Draft	-	-	\1995-13\qa\960521-1.txt
Agenda: Meeting on 96/06/03 with the Product Manager	sde 576	1.00	96/05/31	Approved	96/06/03	2.1.2	\1995-13\qa\minutes\960603-1.agd
Message to Prof. Walker Re: Next meeting and SEAL courses	sde 577	0.01	96/05/31	Draft	-	-	\1995-13\qa\960531-1.txt

Document Name	Document Number	Revision Number	Revision Date	Document Status	Date approved by Board	Minute Reference	File Reference
Correspondence from C. du Plessis Re: Reply to SDE 560	sde 562	0.01	96/03/08	Draft	-	-	\1995-13\qa\960308-2.txt
[illegible] Walker Re: Query on Progress Reports	[illegible]	[illegible]	[illegible]	[illegible]			[illegible]
[illegible] 96/03/25 with the Product Manager	[illegible]						[illegible]
Note from Prof.Walker Re: Reply to SDE 563	sde 565	0.01	96/03/25	Draft			\1995-13\qa\960325-1.txt
Minutes: Meeting on 96/03/25 with the Product Manager	sde 566	1.00	96/03/25	Draft	-	2.1.3	\1995-13\qa\minutes\960325-1.min
Note from Dr. Dwolatzky Re: Postponnement of meeting date to 29 April 1996	sde 567	0.01	96/04/18	Draft	-	-	\1995-13\qa\960418-1.txt
Note to Dr. Dwolatzky Re: Confirmation of date of meeting for the 29 April 1996	sde 568	0.01	96/04/18	Draft	-	-	\1995-13\qa\960418-1.txt
Note to Dr. Dwolatzky Re: Allocation of PC in the SEAL Lab	sde 569	0.01	96/04/20	Draft	-	-	\1995-13\qa\960420-1.txt

Document Name	Document Number	Revision Number	Revision Date	Document Status	Date approved by Board	Minute Reference	File Reference
Agenda: Meeting on 95/12/08 with the Product Manager	sde 554	1.00	95/12/11	Approved	95/12/08	2.1.2	\1995-13\qa\minutes\951208-1.agd
Minutes: Meeting on 95/12/08 with the Product Manager	sde 555	1.00	95/12/11	Draft	95/12/08	2.1.3	\1995-13\qa\minutes\951208-1.min
Agenda: Meeting on 96/01/30 with the Product Manager	sde 556	1.00	96/01/22	Approved	96/01/30	2.1.2	\1995-13\qa\minutes\960130-1.agd
Minutes: Meeting on 96/01/30 with the Product Manager	sde 557	1.00	96/01/30	Draft	96/02/08	2.1.3	\1995-13\qa\minutes\960130-1.min
Agenda: Meeting on 96/02/08 with the Product Manager	sde 558	1.00	96/02/08	Approved	96/02/08	2.1.2	\1995-13\qa\minutes\960208-1.agd
Minutes: Meeting on 96/02/08 with the Product Manager	sde 559	1.00	96/02/16	Draft	-	2.1.3	\1995-13\qa\minutes\960208-1.min
Note from N. Wilkon. Re: H8/3048 ZTAT Programming	sde 560	0.01	96/02/26	Draft	-	-	\1995-13\qa\960226-1.txt
Correspondence to C. du Plessis Re: C++ to C Translator	sde 561	0.01	96/03/08	Draft	-	-	\1995-13\qa\960308-1.txt

Document Name	Document Number	Revision Number	Revision Date	Document Status	Date approved by Board	Minute Reference	File Reference
Note to Prof. Walker on 95/09/04. re: SPM agendas	sde 546	0.01	95/09/04	Draft	-	-	\1995-13\qa\950904-2.txt
Correspondence to Prof. Walker re: query on SEAL documents	sde 547	0.01	95/09/04	Draft	-	-	\1995-13\qa\950904-3.txt
Correspondence to Prof Walker. re: additional query on SEAL documents	sde 548	0.01	95/09/04	Draft			\1995-13\qa\950904-4.txt
Correspondence form Prof.Walker re: reply to sde 548	sde 549	0.01	95/09/05	Draft			\1995-13\qa\950905-1.txt
Correspondence from Prof Walker. re: reply to sde-546 and sde-547	sde 550	0.01	95/09/05	Draft	-		\1995-13\qa\950905-2.txt
Agenda: Meeting on 95/09/11 with the Product Manager	sde 551	1.00	95/09/11	Approved	95/09/11	2.1.2	\1995-13\qa\minutes\950911-1.agd
Minutes: Meeting on 95/09/11 with the Product Manager	sde 552	1.00	95/09/30	Approved	95/12/08	2.1.3	\1995-13\qa\minutes\950911-1.min
Note to the Product Manager on 95/10/06. re: Contribution to Open-Day on 95/10/20	sde 553	0.01	95/10/06	Draft	-	-	\1995-13\qa\951006-1.txt

Document Name	Document Number	Revision Number	Revision Date	Document Status	Date approved by Board	Minute Reference	File Reference
Correspondence to Neville Wilken of T.E.C. re: support from Hitachi	sde 537	0.01	95/08/25	Draft	-	-	\1995-13\qa\950825-1.txt
Correspondence from Prof Walker re: minutes of SPM meeting on 95/08/28	sde 538	0.01	95/08/29	Draft			\1995-13\qa\950829-1.txt
Audit Report. SPM meeting on 95/08/28	sde 539	1.00	95/08/28	Approved	95/08/29	2.1	\1995-13\qa\audit\qa9513pl.par
Correspondence to G.Diana of the Un.Of Natal on 95/08/29. re: query on S.V.M	sde 540	0.01	95/08/29	Draft		-	\1995-13\qa\950829-1.txt
Note from Prof. Walker on 95/08/31. re: SEAL server	sde 541	0.01	95/08/31	Draft	-	-	\1995-13\qa\950831-1.txt
Note from Prof. Walker 95/08/31. re: archiving of files on SEAL server	sde 542	0.01	95/08/31	Draft	-	-	\1995-13\qa\950831-2.txt
Correpondence to Prof. Walker on 95/09/01. re: copy of 9513pl.par	sde 543	0.01	95/09/01	Draft	-	-	\1995-13\qa\950901-1.txt
Note from Prof. Walker. re: reply to sde-543	sde 544	0.01	95/09/01	Draft	-	-	\1995-13\qa\950901-2.txt
Note to Prof. Walker on 95/09/04 re: reply to sde-542	sde 545	0.01	95/09/04	Draft	-	-	\1995-13\qa\950904-1.txt

Document Name	Document Number	Revision Number	Revision Date	Document Status	Date approved by Board	Minute Reference	File Reference
Correspondence from Neville Wilken of T.E.C on 95/08/14 re: Hitachi contact persons	sde 529	0.01	95/08/14	Draft	-	-	\1995-13\qa\950814-2.txt
Correspondence to Mario Klein of Hitachi on 95/08/14 re: General query on the H3048	sde 530	0.01	95/08/14	Draft	-	-	\1995-13\qa\950814-3.txt
Correspondence to Neville Wilken of T.E.C on 95/08/14 re reply to sde-529	sde 531	0.01	95/08/14	Draft			\1995-13\qa\950814-4.txt
Correspondence from Neville Wilken of T.E.C on 95/08/16 re: Hitachi contact persons	sde 532	0.01	95/08/16	Draft	-	-	\1995-13\qa\950816-1.txt
Correspondence to Neville Wilken of T E.C on 95/08/18 re: reply to sde-532	sde 533	0.01	95/08/18	Draft	-	-	\1995-13\qa\950818-1.txt
Correspondence from R. Nolf of Hitachi on 95/08/21. re. reply to sde-527	sde 534	0.01	95/08/21	Draft	-	-	\1995-13\qa\950821-1.txt
Correspondence to Robert Nolf of Hitachi on 95/08/21. re : reply to sde-534	sde 535	0.01	95/08/21	Draft	-	-	\1995-13\qa\950821-2.txt
Correspondence from Neville Wilken of T.E.C on 95/08/16 re: Support from Hitachi	sde 536	0.01	95/08/21	Draft	-	-	\1995-13\qa\950821-3.txt

Document Name	Document Number	Revision Number	Revision Date	Document Status	Date approved by Board	Minute Reference	File Reference
Correspondence from Prof. Walker on SEAL server on 95/08/02	sde 521	0.01	95/08/02	Draft	-	-	\1995-13\qa\950802-1.txt
Progress Report Document on the 95/08/04- for faculty purposes	sde 522	1.00	95/08/04	Approved	95/08/04	2.3	\1995-13\qa\950804-1.txt
Agenda: Meeting on 95/08/04 with the Product Manager	sde 523	1.00	95/08/04	Approved	95/08/04	2.1.2	\1995-13\qa\minutes\950804-1.agd
Minutes: Meeting on 95/08/04with the Product Manager	sde 524	1.00	95/08/07	-	-	2.1.3	\1995-13\qa\minutes\950804-1.min
Correspondence from Prof. Walker . re: minutes of SPM meeting on 95/08/07	sde 525	0.01	95/08/07	Draft	-	-	\1995-13\qa\950807-1.txt
Correpondence from Neville Wilken of T.E.C on 95/08/10 re: Hitachi contact persons	sde 526	0.01	95/08/10	Draft	-	-	\1995-13\qa\950810-1.txt
Correpondence to Robert Nolf of Hitachi on 95/08/11 re: General query on the H3048	sde 527	0.01	95/08/11	Draft	-	-	\1995-13\qa\950811-1.txt
Correpondence to Robert Nolf of Hitachi on 95/08/14	sde 528	0.01	95/08/14	Draft	-	-	\1995-13\qa\950814-1.txt

Document Name	Document Number	Revision Number	Revision Date	Document Status	Date approved by Board	Minute Reference	File Reference
Minutes: Meeting on 95/05/26 with the Product Manager	sde 513	1.00	95/05/29	Approved	95/06/26	2.1.3	\1995-13\qa\minutes\950526-1.min
Agenda: Meeting on 95/06/26 with the Product Manager	sde 514	1.00	95/06/26	Approved	95/06/26	2.1.2	\1995-13\qa\minutes\950626-[illegible]gd
Minutes: Meeting on 95/06/26 with the Product Manager	sde 515	1.00	95/06/27	Approved	95/08/04	2.1.3	\1995-13\qa\minutes\950626-1.min
Correspondence to Rational Software on 95/07/21, re: C++ Translator	sde 516	0.01	95/07/21	Draft	-	-	\1995-13\qa\950721-1.txt
Correspondence from Cindy Wilson of Rational Software re: reply to sde-516	sde 517	0.01	95/07/21	Draft	-	-	\1995-13\qa\950721-2.txt
Correspondence to Rational Software on 95/07/28, re: reply to sde-517	sde 518	0.01	95/07/28	Draft	-	-	\1995-13\qa\950728-1.txt
Correspondence from T. Baxter of Rational Software, re: reply to sde-518	sde 519	0.01	95/07/28	Draft	-	-	\1995-13\qa\950728-2.txt
Correspondence from Prof. Walker on SEAL server on 95/07/28	sde 520	0.01	95/07/28	Draft	-	-	\1995-13\qa\950728-3.txt

Project Member	Course Title	Offered by	Course date	Attended/ Completed?
	OOD techniques	individual	95/06/27	
H.Bapoo	Learning how to use the Rational Rose software	Training will be left to the individual	95/06/14 - 95/06/27	yes
H.Bapoo	Software Project Management -ELEN 534	Elec. Eng. Dept.WITS	95\06\28 - 95\06\30	yes
H.Bapoo	Learning of C and C++	Training will be left to the individual	95/04/25 -	yes
H.Bapoo	Study of the H3048 architecture	Training will be left to the individual	95/08/04 -	yes
H.Bapoo	Study of the software aspects specific to the H3048 (Assemly language, C specifics, C-compiler)	Training will be left to the individual	95/08/10 -	yes
H.Bapoo	Learning how to use the H3048 emulator and its accompanying software	Training will be left to the individual	95/08/22	yes
H.Bapoo	Realtime Computer Control - ELEN 541	Elec. Eng. Dept. WITS	96\02\20 - 96\03\07	No - Course cancelled
H.Bapoo	Intelligent Control Techniques - ELEN 542	Elec. Eng. Dept. WITS	96\05\01 - 96\05\19	No - Course cancelled

Who	Contact Details(postal address, phone, fax, e-mail addresses)	SDES role
	e-mail Bapoo@odie.ee.wits.ac.za	

4.5 Human Resource allocation

The allocation of individuals to specific tasks is given in WBS and Schedule.

4.6 Hardware and Software Resources.

The following hardware and software resources are required:

Hardware/software items	Quanity	Date required	Date supplied
PC at workplace	1	95/02/01	95\02\01
Office space and PC in the SEAL Lab	1	95/02/01	Not available
Access to the Wits Electrical Engineering computer network	1	95/03/01	95\03\01
H3048 C-compiler	1	95/08/10	95\08\10
H3048 emulator	1	95/08/22	95/08/22
Borland C++ package	1	95/08/29	95/08/29
Development tools to download the compiled program onto the microcontroller	1	96/06/15	96/06/15
IThree-phaseUPS	1	96/05/01	96/05/01
Access to laboratory facilities, including use of oscilloscope, digital multimeters and other standard laboratory equipments	1	95/12/01	95/12/01

4.7 Training Needs Identification.

The following training is required for this project:

Project Member	Course Title	Offered by	Course date	Attended/ Completed?
H.Bapoo	Advanced Engineering Software Design -ELEN533	Elec. Eng Dept., WITS	95\05\01 - 95\06\19	yes
H.Bapoo	Further study of OOA and	Training will be left to the	95/05/20 -	yes

4 Product Development Plan

4.1 Work Breakdown Structure

4.2 Dependencies

4.2.1 Internal dependencies

This is a one-man (one developer) project and has no internal dependencies.

4.2.2 External dependencies

Will require the use of diverse hardware and software tools, on specific dates, as detailed in section 4.6.

4.3 Required Human Resources

The following resources are required to develop the product:

Number	Resource	Commitment
1	Product Manager	100% throughout the project
1	Product Developer	100% throughout the project

4.4 Available Human Resources

The following human resources are available as on 95/03/01:

Who	Contact Details(postal address, phone, fax, e-mail addresses)	SDES role
Dr B. Dwolatzky	Electrical Engineering Department, University of the Witwatersrand tel e-mail Dwolatzk@odie ee wits .ac za	Product Manager
H. Bapoo	P.O. Box 802 Isando 1600 tel +27 11 392-1642 fax +27 11 392-5343	Product Developer

- Keep track of all documents associated with the project

3.3.2 Product Manager

- Monitor progress and resource utilisation of the Product Team, and initiate corrective action where necessary.
- Identify required resources and make these available to the team.
- Act as the focal reporting point for the product
- Ensure that Product Reviews are held as planned.
- Interface to the SEAL Management board.
- Be responsible for the timely delivery of the product.
- Give direction to the Product development team.

3 Project Management Plan

3.1 Overview

The aim of this Project Management Plan is to define the structure of this project and to create the technical products.

This resource plan must be formally approved by the Product Manager before work on the Technical products is undertaken.

3.2 Product Development Team

The development team comprises:

- The Product Manager of the Software Development for Embedded Systems project.
- The developer of this product\service.

3.3 Roles and Responsibilities

3.3.1 Product Developer

- Establish and maintain a product plan for developing this product\service.
- Develop the software and hardware elements associated with this project.
- Ensure all Technical Exceptions are properly reported.
- Prepare Detailed Work Plans as necessary.
- Prepare and present regular Checkpoint Reports to the Project Manager.
- Take direction from the Project Manager for matters related to the project.
- Carry out literature surveys as required by the project.
- Act as the Project Editor for the product.
- Prepare the agenda and minutes for each meeting held

2 Product Quality Plan

The team associated with this project is totally committed to quality and all system documentation produced as part of the product will be reviewed internally by the Review team and produced in accordance with the standards and formats associated with the SEAL QMS.

The Quality Plan associated with this project is identified in SDE-003, as defined in QS 195 SEAL QMS Project Documentation and Support Standard.

1.6 Assumptions

It is assumed that the reader is familiar with the ISO 9000 series standard for quality systems management.

1.7 ISO 9001 Requirements Traceability

a. ISO 9001 (1987) 4.1 Management Responsibility

b. ISO 9001 (1987) 4.2 Quality System

c. ISO 9001 (1987) 4.3 Contract Review

d. ISO 9001 (1987) 4.4 Design Control

e. ISO 9001 (1987) 4.5 Document Control

f. ISO 9001 (1987) 4.8 Product Identification and Traceability

g. ISO 9001 (1987) 4.9 Process Control

h. ISO 9001 (1987) 4.10 Inspection and Testing

i. ISO 9001 (1987) 4.13 Control of Non-conforming Product

j. ISO 9001 (1987) 4.14 Corrective Action

k. ISO 9001 (1987) 4.16 Quality Records

l. ISO 9001 (1987) 4.18 Training

- full-time staff members of the SEAL
- All full-time and part-time post-graduate students associated with the SEAL
- Members of the SEAL Management Board
- Head of the Department, Electrical Engineering
- Individuals who will perform internal and external surveillance audits of the SEAL Quality Management System.
- Engineering Manager, Meissner
- Product developer
- Product Manager

1.5 Applicable Documents

1.5.1 Specifications

None

1.5.2 Standards

a. SEAL QMS Document Creation Template, QS 002, Revision 0.01, 5 April 1994.

b. SEAL QMS Document Layout, Presentation and Typesetting Guide, QS 003, Revision 0.01, 1 April 1994.

1.5.3 Procedures

None

1.5.4 Guidelines

None

1.5.5 Other Documents

a. SDE 100, Research Proposal (WITS)

b. SDE 101, Project Proposal (Meissner)

1 Scope

1.1 Introduction

This Plan provides an overview of the required resources to complete the Software Development for Embedded Systems project. It will form the terms of reference for all contributing parties during the project.

- This Plan outlines the human resources requirements.
- It provides a list of tasks to be performed and a time scale.
- It provides a means for capturing important project metrics regarding estimated and actual effort to perform allocated tasks.

The costs associated with the development of the product are outside the scope of this Plan.

1.2 Purpose

The purpose of this Plan is to:

- Present a clear statement of all deliverables.
- Present a clear statement of work allocation and responsibilities to be undertaken by all parties.

Present details of resources associated with this product.

1.3 Definitions

SEAL: Software Engineering Applications Laboratory - Faculty of Engineering (University of the Witwatersrand)

UPS: Uninterruptible Power Supply, used as a case study for this project

OOA: Object-Oriented Analysis

OOD: Object-Oriented Design

1.4 Audience

The audience for this document comprise the various stakeholders of the SEAL, including:

Change History

Configuration Control

Project:	SDES
Title:	Project Management Plan
Doc. Reference:	\1995-13\MP\SDE006WW.100
Created by:	H Bapoo
Creation Date:	28 August 1996

Document History

Version	Date	Status	Who	Saved as:
0.01	95/08/28	Draft	H.Bapoo	\1995-13\MP\ SDE006WW.001
0.02	96/05/29	Draft	H.Bapoo	\1995-13\MP\ SDE006WW.002
1.00	96/09/01	Approved	H.Bapoo	\1995-13\MP\SDE006WW.100

Revision History

Version	Date	Changes
0.01	95/08/28	New Document based on QST 124-10
0.01	95/08/28	Editorial developemnt of the text and addition of section 7
1.00	96/09/01	Updates to section 7 and update to revision number and status following Board approval

Management Authorisation

Version	Date	Status	Management Board Minute Reference
1.00	96/09/01	Approved	Section 2.6 of SDE 580

Change Forecast

This document will be updated each time there is a change to one of the tables in this Plan i.e. resources, task element, or schedule.

5.2 Reviewing....10
5.3 Reporting....11
5.4 Document control....11
5.5 Change Control....11
6 Work Breakdown Structure, Obligations and Schedule....12
7 Time Allocation Chart....17

Table of Contents

1 Scope 1
1.1 Introduction 1
1.2 Purpose 1
1.3 Definitions 1
1.4 Audience 1
1.5 Applicable Documents 2
1.6 Assumptions 3
1.7 ISO 9001 Requirements Traceability 3
2 Product Quality Plan 4
3 Project Management Plan 5
3.1 Overview 5
3.2 Product Development Team 5
3.3 Roles and Responsibilities 5
4 Product Development Plan 7
4.1 Work Breakdown Structure 7
4.2 Dependencies 7
4.3 Required Human Resources 7
4.4 Available Human Resources 7
4.5 Human Resource allocation 8
4.6 Hardware and Software Resources. 8
4.7 Training Needs Identification. 8
5 Project Control 10
5.1 Planning 10

4 Managing Quality in Software Development 22
4.1 Definitions 22
4.2 Achieving Quality Software 22
4.4 The Capability Maturity Model 26
4.3 Standards and Software Management 28
4.5 Audits 31

5 Software Metrics 33
5.1 Limitations 34
5.2 Categorising Metrics 34
5.3 Time and Costs Metrics 37
5.3.1 Boehm's Constructive Cost Model 39
5.4 Software Packages for Software Metrics 44

6 Software Design 45
6.1 Introduction 45
6.2 Function-Oriented Design 48
6.3 Object Oriented Design 49

7 Psychological Factors in the Development of Software 53
7.1 Problem Solving 53
7.2 Appraising Programming Intelligence 55

Table of Contents

Table of Contents ii

Change History iv
Configuration Control iv
Document History iv
Revision History iv
Management Authorisation iv
Change Forecast iv

1 Scope 1
1.1 Introduction 1
1.2 Purpose 1
1.3 Definitions 1
1.4 Audience 1
1.5 Applicable Documents 2
1.6 Assumptions 6
1.7 ISO 9001 Requirements Tracoability 6

2 The Development Processes 7
2.1 The Conventional Life-Cycle Model 7
2.2 The Spiral Model 9
2.3 Prototyping 10
2.4 Operational Specification 13
2.5 Transformational Implementation 16

3 Software Development and Documentation 19
3.1 User Requirements Specification 19
3.2 Functional Specifications 19
3.3 Software Design Specification 20
3.4 Subsystems specifications 20
3.5 Module Specifications 20
3.6 Module Definitions 21
3.7 Utility Requirements Specifications 21
3.8 Development Notebooks 21
3.9 Requirements Matrices 21



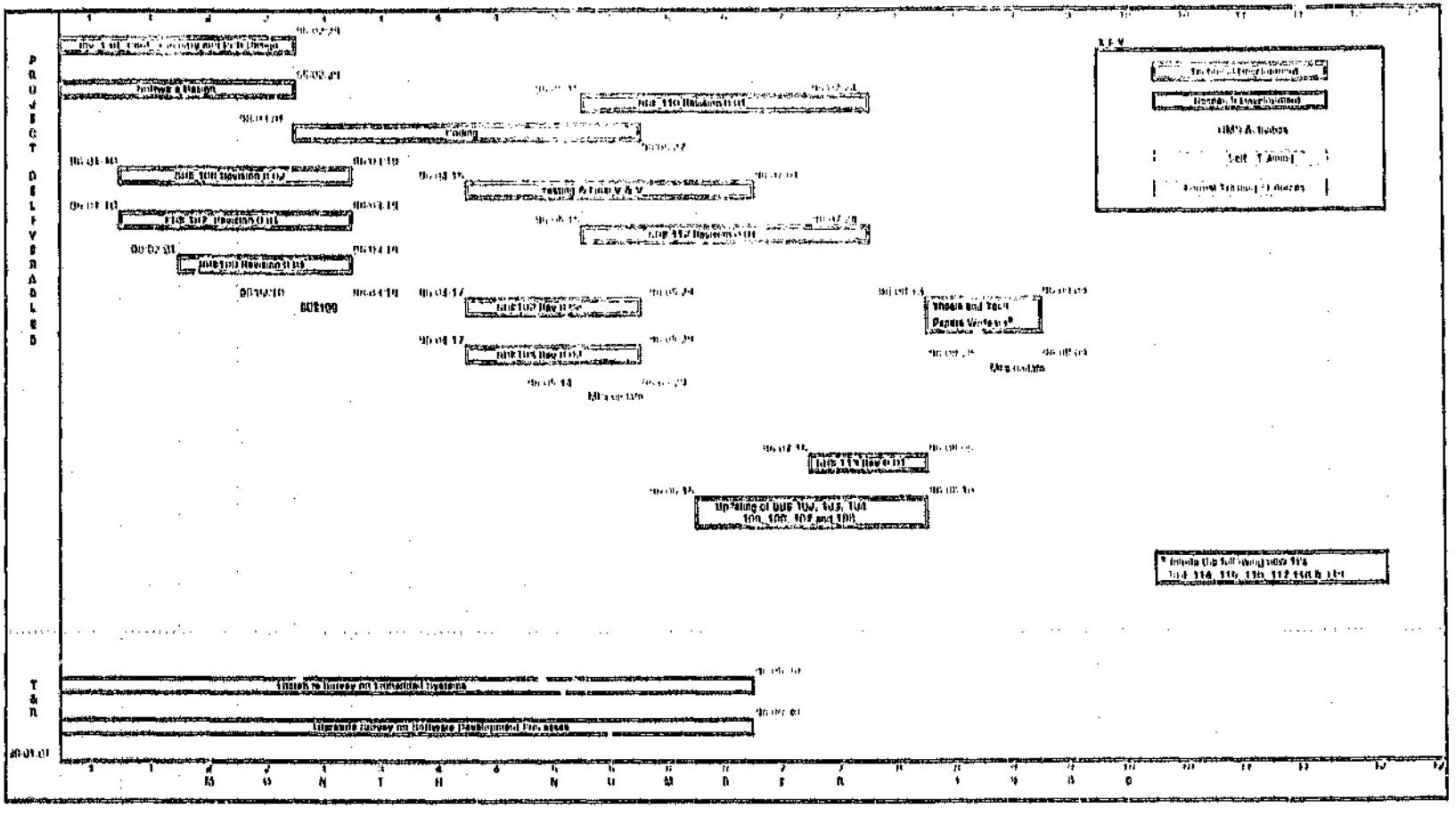
SDES QMS

Software Development: A General Literature Survey

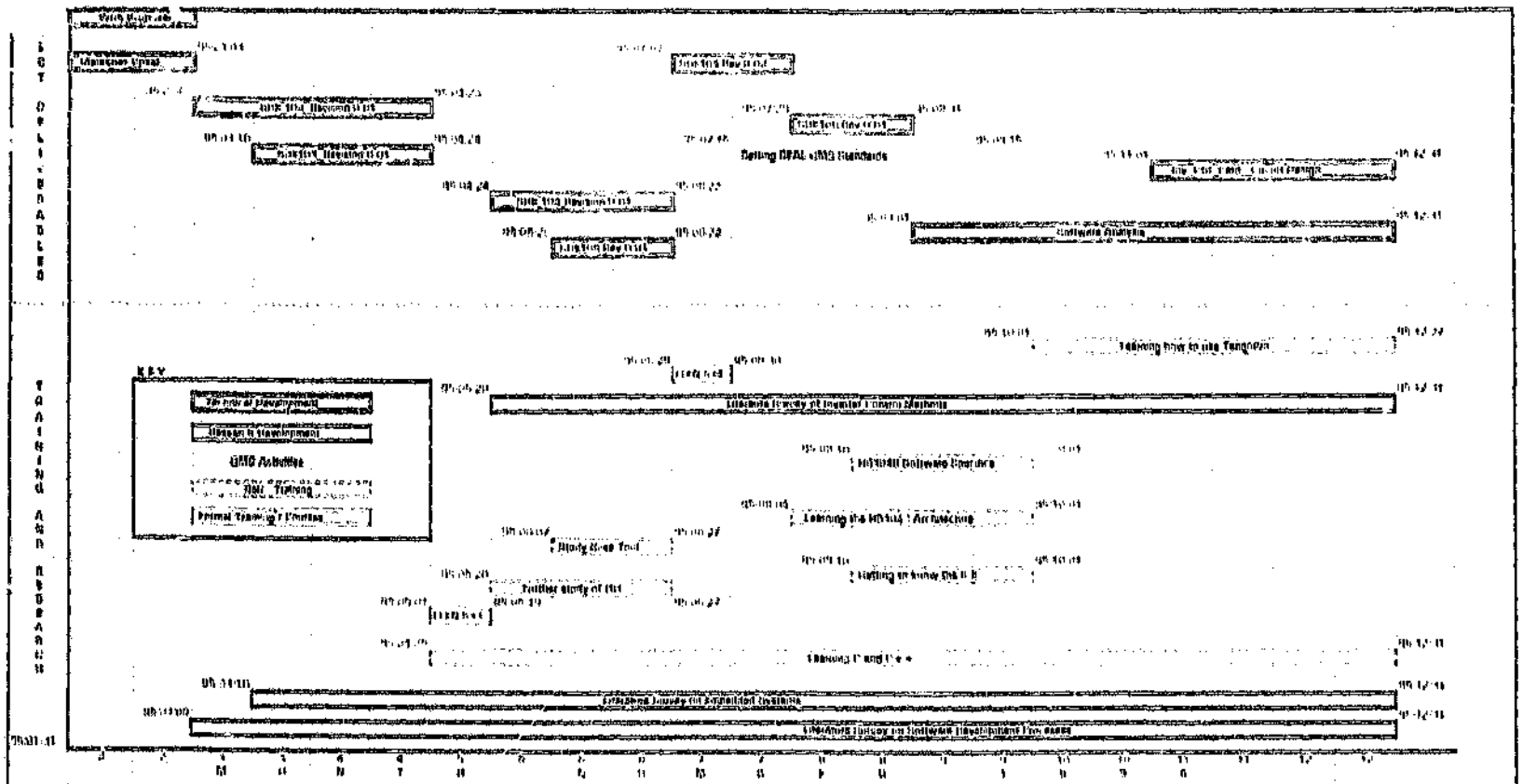
Technical Product

Version 1.00

Document Status: Approved


PROJECT DELIVERABLES
TASK
SDE100

Time Allocation Chart



M Sc. Paper 2	SDE116	H B	Produce the document	Review the document and specify its status	7	7	06/08 /16	06/08 /16	06/08 /23	06/08 /23
Discussion and Conclusions	SDE117	H B	Produce the document	Review the document and specify its status	11	11	06/08 /12	06/08 /12	06/08 /23	06/08 /23
References/ Bibliography	SDE118	H B	Produce the document	Review the document and specify its status	11	11	06/08 /12	06/08 /12	06/08 /23	06/08 /23
Front Pages of Dissertation	SDE119	H B	Produce the document	Review the document and specify its status	11	11	06/08 /12	06/08 /12	06/08 /23	06/08 /23

Software Design Specifications - Part1	SDE107	H.B	Produce the document	Review the document and specify its status	70	70	96/01 /10	96/01 /10	96/03 /19	96/03/19
Software Design Specifications - Part 2	SDE108	H.B	Produce the document	Review the document and specify its status	68	68	96/02 /01	96/02 /01	96/03 /19	96/03 /19
Progess Report	SDE109	H.B	Produce the document	Review the document	1	1	96/03 /29	96/03 /29	96/03 /30	96/03 /30
Software and Hardware Documentation	SDE110	H.B	Produce the codes and the document	Review the codes and the document and specify the status	68	68	96/05 /15	96/05 /15	96/07 /24	96/07 /24
Project Presentation (SEAL)	SDE111	H.B	Produce the document	Review the document	10	10	96/04 /01	96/04 /01	96/04 /11	96/04 /11
Testing - Specifications and Results	SDE112	H.B	Perform the Final V&V on the whole system and produce the test documentation	Verify the test results, review the document and specify its status	68	68	96/05 /15	96/05 /15	96/07 /24	96/07/24
Embedded Systems: A Review	SDE113	H.B	Produce the document	Review the document and specify its status	25	25	96/07 /15	96/07 /15	96/08 /10	96/08 /10
Project Review	SDE114	H.B	Produce the document	Review the document and specify its status	11	11	96/08 /12	96/08 /12	96/08 /23	96/08 /23
M.Sc. Paper 1	SDE115	H.B	Produce the document	Review the document and specify its status	8	8	96/08 /08	96/08 /08	96/08 /16	96/08 /16

Project Proposal (Meissner) Version 0.02	SDE101	H.B	Produce the document	Review the document and specify its status	7	7	95/02 /20	95/02 /20	95/02 /27	95/02/27
Project Proposal (Meissner) Version 1.00	SDE101	H.B	Produce the document	Review the document and specify its status	4	4	95/02 /27	95/02 /27	95/03 /03	95/03/03
Lit. Survey of Software Development Processes in General - Version 0.01	SDE102	H.B	Produce the document	Review the document and specify its status	49	49	95/03 /06	95/03 /06	95/04 /24	95/04/24
Embedded Systems - A Literature Survey - Version 0.01	SDE103	H.B	Produce the document	Review the document and specify its status	39	39	95/03 /10	95/03 /18	96/04 /24	95/04/24
Requirements Analysis Version 0.01	SDE104	H.B	Produce the document	Review the document and specify its status	33	33	95/05 /20	95/05 /20	95/06 /22	95/06/22
Requirements Analysis Version 0.02	SDE104	H.B	Produce the document	Review the document and specify its status	14	14	95/07 /07	95/07 /07	95/07 /28	95/07/28
Proposed Technique for the Control of the Three-Phase Inverter - Version 0.01	SDE105	H.B	Produce the document	Review the document and specify its status	33	33	95/05 /20	95/05 /20	96/02 /06	95/06/22
Functional Specifications Version 0.01	SDE106	H.B	Produce the document	Review the document and specify its status	32	32	95/07 /29	95/07 /28	95/08 /31	95/08/31
Functional Specifications Version 0.02	SDE106	H.B	Produce the document	Review the document and specify its status	70	70	96/01 /10	96/01 /10	96/03 /19	96/03/19

Project Artifact (WBS Item)	Doc. No	Who	Obligations		Effort (days)		Start Date		End Date	
			Developer	Manager	Est.	Act	Sched-uled	Actual	Sched-uled	Actual
Contract between Supplier and Customer	SDE007	H.B	Produce the document	Review the document and specify its status	60	60	95/07 /15	95/07 /15	95/09 /15	95/09 /15
Binder Label	SDE008	H.B	Produce the document	Review the document and specify its status	60	60	95/07 /15	95/07 /15	95/09 /15	95/09 /15
Research Proposal (WITS) Version 0.01	SDE100	H.B	Produce the document	Review the document and specify its status	7	7	95/01 /31	95/01 /31	95/02 /06	95/02/06
Research Proposal (WITS) Version 0.03	SDE100	H.B	Produce the document	Review the document and specify its status	11	11	95/02 /06	95/02 /06	95/02 /17	95/02/17
Research Proposal (WITS) Version 0.04	SDE100	H.B	Produce the document	Review the document and specify its status	8	8	95/02 /20	95/02 /20	95/02 /28	95/02/28
Research Proposal (WITS) Version 1.00	SDE100	H.B	Produce the document	Review the document and specify its status	6	6	95/02 /28	95/02 /28	95/03 /06	95/03/06
Project Proposal (Meissner) Version 0.01	SDE101	H.B	Produce the document	Review the document and specify its status	14	14	95/01 /31	95/01 /31	95/02 /14	95/02/14

6 Work Breakdown Structure, Obligations and Schedule

Project Artifact (WBS Item)	Doc. No	Who	Obligations		Effort (days)		Start Date		End Date	
			Developer	Manager	Est.	Act	Sched-uled	Actual	Sched-uled	Actual
Master Document List	SDE001	H.B	Produce the document	Review the document and specify its status	60	60	95/07 /15	95/07 /15	95/09 /15	95/09 /15
Document Creation Template	SDE002	H.B	Produce the document	Review the document and specify its status	60	60	95/07 /15	95/07 /15	95/09 /15	95/09 /15
Quality Plan	SDE003	H.B	Produce the document	Review the document and specify its status	60	60	95/07 /15	95/07 /15	95/09 /15	95/09 /15
Product Description	SDE004	H.B	Produce the document	Review the document and specify its status	60	60	95/07 /15	95/07 /15	95/09 /15	95/09 /15
Project Management Plan	SDE005	H B	Produce the document	Review the document and specify its status	60	60	95/07 /15	95/07 /15	95/09 /15	95/09 /15
Configuration Management Plan	SDE006	H.B	Produce the document	Review the document and specify its status	60	60	95/07 /15	95/07 /15	95/09 /15	95/09 /15

5.3 Reporting

Control is affected by both monitoring and reporting upon progress to management and members of the team.

5.3.1 Responsibilities of the Product Developer

The product developer will report to the Product Manager at intervals defined in the Minutes of Product Review Meetings detailing:

- Progress to date
- Effort required to complete
- Problems encountered and action taken

5.3.2 Responsibilities of the Product Manager

- The Product Manager will initiate corrective action as necessary to keep the project on schedule.
- The Product Manager will report the progress of the project to the SEAL Management Board.

5.4 Document control

Document control will be performed as per SEAL QMS policy and procedures:

- QS125 SEAL QMS Document and Data Control
- QS 195 SEAL QMS Project Documentation Standard
- In accordance the requirements of QS 195, this document will be numbered as SDE-005.
- The Management and Technical products identified in the WBS in Section 7 will be reflected in the the Master Document List for this project (SDE-001).

5.5 Change Control

Any work outside of the Product Description (SDE-004) will be formally approved by the Project Manager.

5 Project Control

5.1 Planning

The planning of different tasks of this project will be performed by the Product developer under the approval of the Product Manager.

Any detailed plan will not be executed without the approval of the Product Manager.

5.2 Reviewing

- All draft versions of the product will be reviewed by the Product Manager.
- In addition, certain versions which are so specified in the Work Breakdown Structure will be reviewed by the members of other product groups.

5.2.1 Planned Arrangements for Reviews

All project artifacts (i.e. Management, Technical, and Quality Assurance Products) are subject to review. Management and Technical Products may be subject to a number of revisions before a product is approved by the Product Manager. QA Products are not normally changed after being produced and are simply approved by the Project Manager. (Changes are only made if there are factual errors in the first revision.)

The obligations of Project Manager and Product Develop(s) associated with each review activity are listed in the Work Breakdown Structure, Obligations and Schedule (Section 7).

5.2.2 Scheduling of Reviews

Each Management and Technical Product identified in the Work Breakdown Structure (Section 7) will be reviewed by the Product Manager. This will normally take place in a review meeting. These meetings will be listed in the Project Schedule, and typically identified with the product(s) (or WBS item(s)) under review.

5.2.3 Records of Reviews

Records of review meetings are maintained as minutes taken at the meetings. These review meeting minutes will be listed as entries in the QA Products in the Project Master Document List (SDE-001).

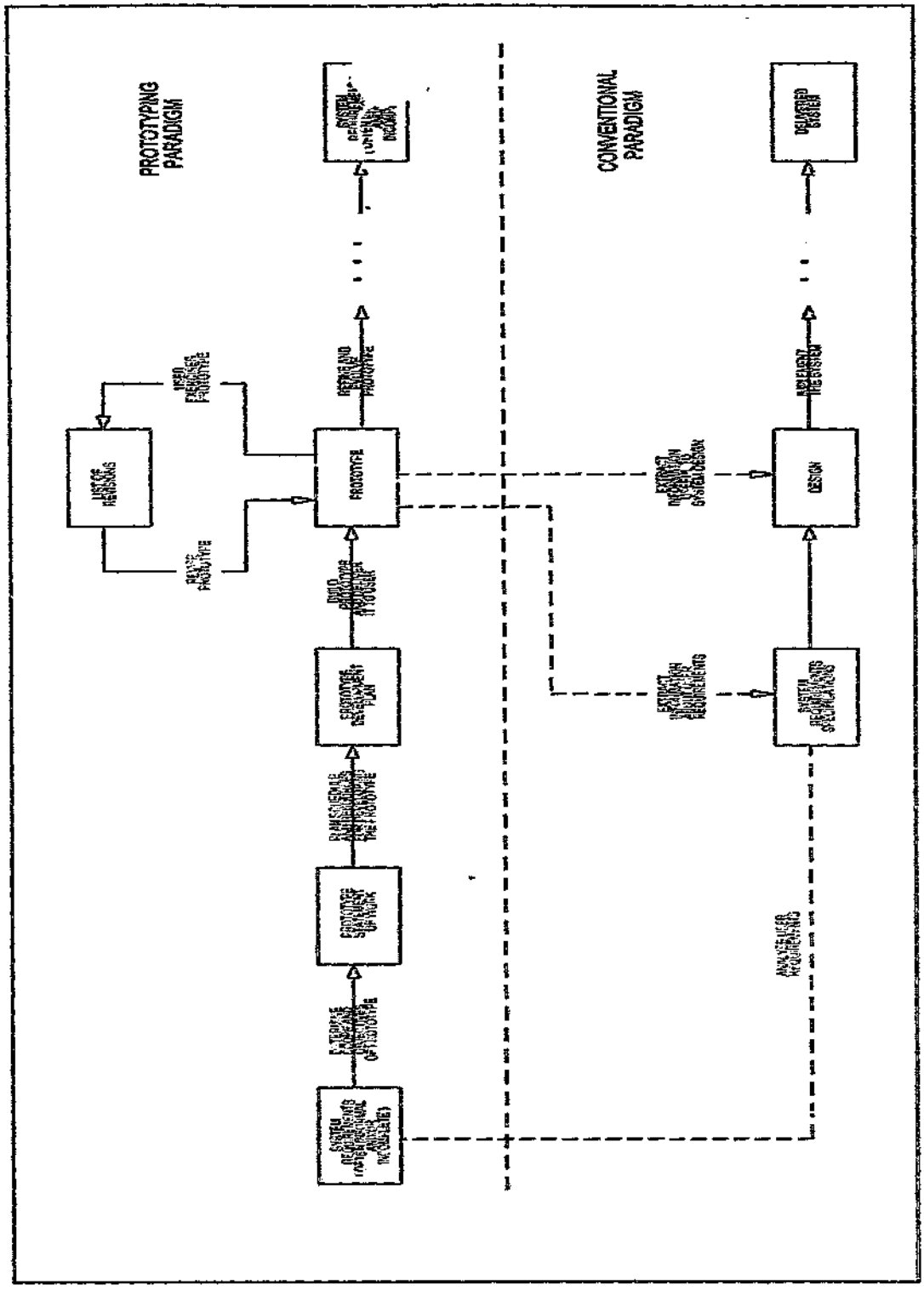


Figure 3: The Prototyping Paradigm And Its Relationship To The Conventional Paradigm

An outline of the spiral model is shown in figure 2.

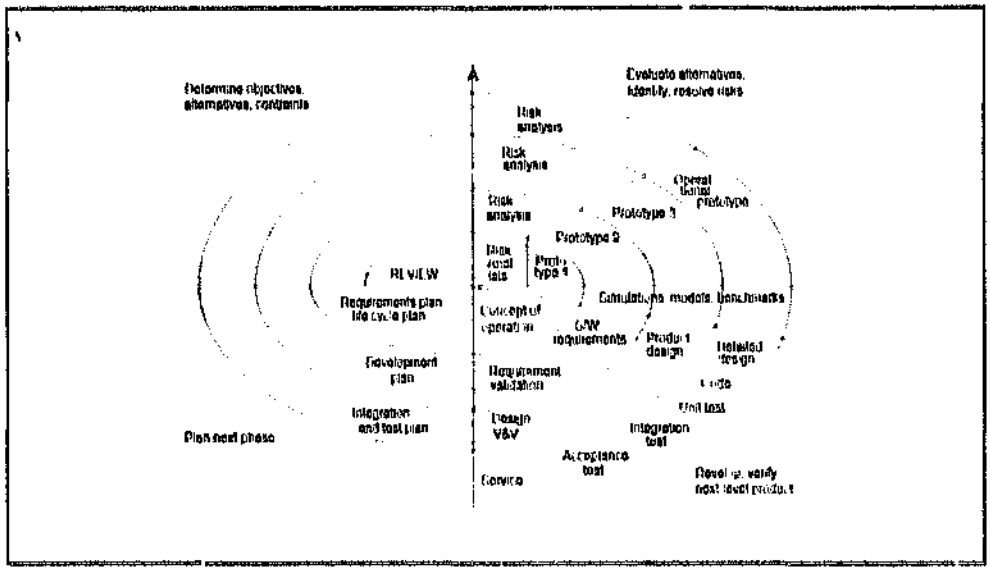


Figure 2: Boehm's spiral model [6]

In contrast to the waterfall model which is a document driven approach, the spiral model is risk-driven. An assessment of management risk items is performed at regular stages in each cycle. At the end of each cycle, following a risk analysis, a review procedure assesses whether to move on to the next cycle of the spiral. Risk, here is understood as simply something which can go wrong [8]. For example, if the intention is to use a new programming language, a risk is that no compiler is available. Risks are a consequence of inadequate information and are resolved by initiating some action to discover information which reduces uncertainty.

2.3 Prototyping

Prototyping is the process of building a working model of a system or part of a system. It is a familiar practice in engineering, to obtain an early version of a product or system. Engineering prototypes can vary in size (scaled down or full size) and functionality (limited capability or a complete set of functions). In any case, the objective of prototyping is to clarify the characteristics and operation of a product or system by constructing a version that can be exercised. The aim of (rapid) prototyping is to accelerate the learning process about whether a system design meets the user needs, and to do so as cheaply and rapidly as possible [9].

language.

- Test: Verification that the code executes without failure, and validation that the completed software is acceptable to the users.

To this list may be added the phase of operations and maintenance (or evolution), to represent the working life of the software.

Tracing the evolution of the life-cycle model reveals that it has been in place since 1970 [2], with its basic structure established even earlier.

A principal virtue of the waterfall model has been to recognise the design as a critical activity. Many examples have shown the expensive consequences of premature coding without adequate analysis and design [2].

However, new evolutions in the software development environment have challenged the adequacy of the conventional method. Currently, there are four times as many computers as there are programmers, and access to computers and software is no longer limited to people with specialised skills, as it was when the waterfall model was originally developed [2]. A wide array of software development support tools are now being used for every phase in the conventional life-cycle model. Of greater significance, some current tools and environments logically span several conventional phases, thereby challenging the usefulness of the life-cycle's partitioning into phases. The theme of these observations is that the life-cycle model reflects the time period in which it evolved. Dramatic changes since then are prompting a reassessment of the model to determine if it is the appropriate paradigm. Some of the criticisms of the waterfall model are: its separation of specification from implementation is unrealistic [3]; it is inadequate for domain-dependent software [4]; it fails to accommodate end-user development and fails to recognise that users generally do not know the system requirements in advance [5].

2.2 Spiral Model

Another approach to the conventional waterfall model is a *spiral* model described by Boehm [6]. The spiral model reflects the underlying concept that each cycle involves a progression that addresses the same sequence of steps, for each portion of the product and for each of its level of elaboration, from an overall concept of operation document down to the coding of each individual program [7].

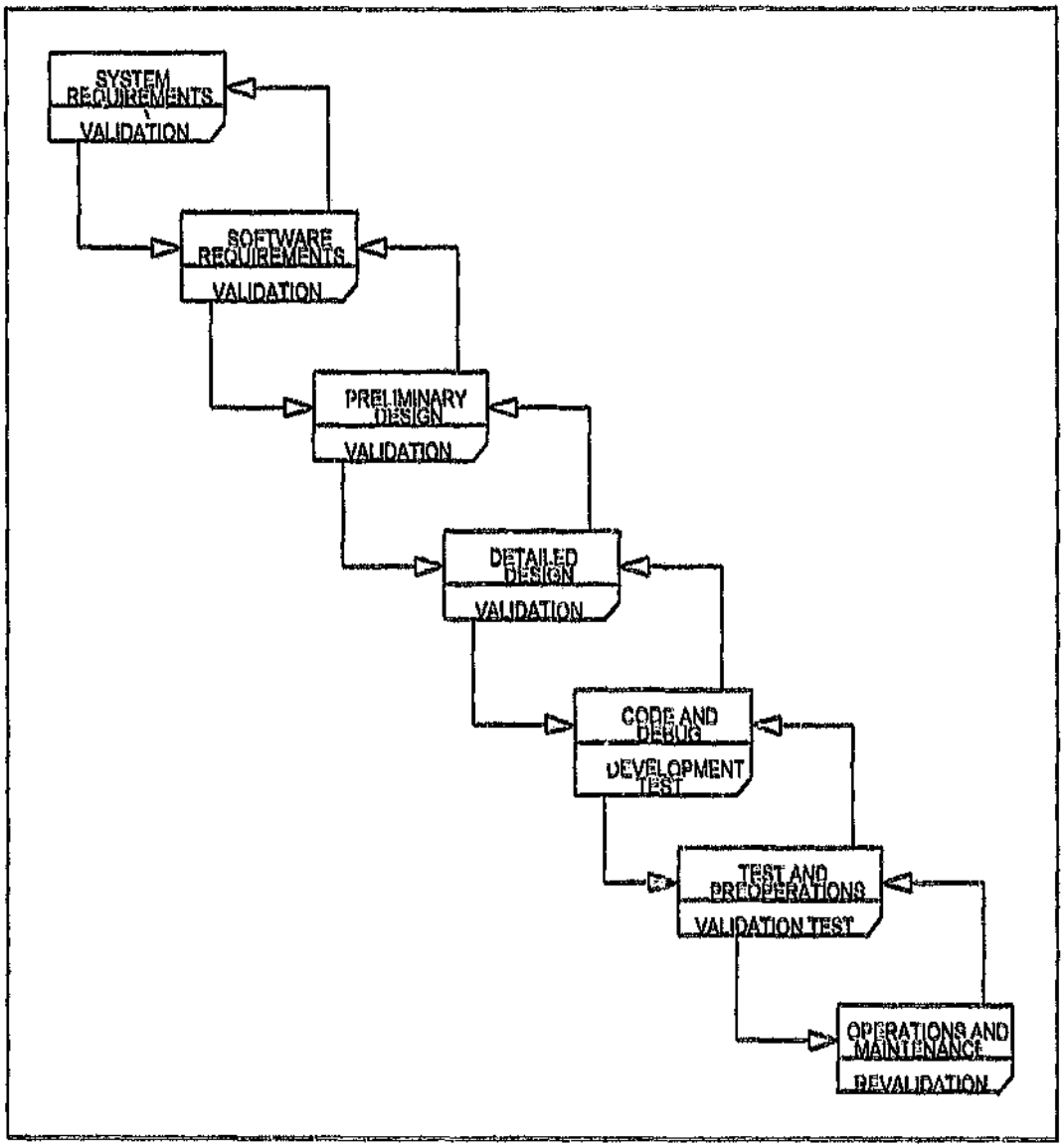


Figure 1: The Conventional "Waterfall" Software Development Life-Cycle Model

- Specification: A statement of "what" the software will do, following a detailed analysis of the requirements.

- Design: "How" the software will meet the requirements; the structure of software units that perform specific functions

- Code: Implementation of the design in a programming

2 The Development Processes

Embedded systems has been traditionally developed by electrical engineers. The software was previously limited to bit manipulation routines. However over the last decade the software written by electrical engineers has grown considerably in complexity. More sophisticated hardware and development tools, widespread use of higher level languages - at levels way above the machine code assemblers - and the increasing sophistication of the embedded system design and development process have all had an impact on the requirements for embedded system software. These factors have also clearly had an impact on the embedded system software developer. Though primarily electrical engineers by training, they are becoming software engineers by necessity, as a result of the growing awareness of the need to discipline the development process irrespective of the size of the software. The present document provides an overview of the existing software development methodologies as well as other elements of the development which are becoming growing concerns for the embedded systems developer. The subjects treated in this document are common and applicable to software development in general, but until recently they were not considered to be of major interest to the small embedded systems realm.

We distinguish between five main software development approaches:

- Waterfall Model
- Spiral Model
- Prototyping
- Operational Specification
- Transformational Implementation

They are discussed briefly below.

2.1 The Conventional Life-Cycle Model

The conventional life-cycle (waterfall) model as presented by Boehm [1] is shown in figure 1 as a series of phases with validation and feedback at each phase. The waterfall model has been stretched, shrunk, and otherwise modified since its inception. When we speak of the conventional life-cycle, we include all such variations that have preserved the essence of the waterfall model, i.e. that software is developed by the following sequence of general activities:

Software, March 1995, pp. 108-109

[45] Verner J., Tate G., "Software Size Model", IEEE Transactions on Software Engineering, Vol. 18, No. 4, April 1992, pp. 265-278

[46] Abdel-Hamid in Burgess A., "Mad About or Mad at Measurement?", IEEE Software, January 1995, pp. 115-116

[47] Bach J., "Enough About Processes: What we Need are Heroes", IEEE Software, March 1995, pp. 96-98

[48] Kitchenham B., Pfleeger S.L., "Software Quality: The Elusive Target", IEEE Software, January 1996, pp. 12-21

[49] Fenton N., "Do Standards Improve Quality - Counterpoint", IEEE Software, January 1996, pp. 23-24

[50] Saiedian H., McClanahan L.M., "Frameworks for Quality Software Process: SEI Capability Maturity Models versus ISO 9000", Software Quality Journal, vol. 5, no. 1, March 1996, pp. 1-23

[51] ISO9126 Information Technology - Software Product Evaluation - Quality Characteristics and Guidelines for their USE, International Organisation for Standardisation, Geneva, 1992

1.5.5 Other Documents
None.

1.6 Assumptions
None.

1.7 ISO 9001 Requirements Traceability

- ISO 9001 Clause 4.4.4 - Design Input

ACM, Vol 26 (11), pp 882-894

[33] Williams T., "Object-Oriented Methods Transform Real-Time Programming", Computer Design, September 1992, pp 101-118

[34] Weinberg G., "The Psychology of Computer Programming", Van Nostrad Reinhold, USA, 1971

[35] Caroll J.M., Thomas J.C., Malhotra A., "Clinical-Experimental Analysis of Design Problem Solving", Design Studies, 1979, pp 84-92

[36] Odette L.L., "Intelligent Embedded Systems", Addison-Wesley Publishing Co., USA, 1991

[37] Curtis B. "Human Factors in Software Development", Second Edition, IEEE Computer Society, USA, 1985

[38] Schneiderman B., Mayer R., "Syntactic/Semantic Interactions in Programming Behaviour: A Model and Experimental Results", Vol 8, Number 3, 1979, pp 219-238

[39] McKeithen K.B., Reitman J.S., Rueter H.H., Hirtle S.C., "Knowledge Organisation and Skill Differences in Computer Programmers", Cognitive Psychology, Volume 13, 1981, pp 307-325

[40] Soloway E., Ehrlich K., "Empirical Studies of Programming Knowledge", IEEE Transactions on Software Engineering, Volume SE-10, Number 5, 1984, pp 595-609

[41] Adelson B., "When Novices Surpass Experts: The Difficulty of a Task May Increase with Expertise", Journal of Experimental Psychology,: Learning, Memory and Cognition, Volume 10, number 3, 1984, pp 483-495

[42] Gordon V.S., Bieman J.M., "Rapid Prototyping: Lessons Learned", IEEE Software, January 1995, pp. 85-95

[43] Paulk M.C., "How ISO 9001 Compares with the CMM", IEEE Software, January 1995, pp. 75-83

[44] De Marco T., " Blueprint for Success: Innovate and Invest in People", IEEE

[20] Humphrey W.S., "Managing the Software Process", Addison-Wesley Publishing Co., USA, 1990

[21] Ince D., "An Introduction to Software Quality Assurance and its Implementation", McGraw-Hill Book Co., UK, 1994

[22] Ince D., "ISO 9001 and Software Quality Assurance", McGraw-Hill Book Co., UK, 1994

[23] SABS ISO 9001: 1994, "Quality Systems - Model for Quality Assurance in Design, Development, Production, Installation and Servicing", South African Bureau of Standards, 1994

[24] Dunham J.R., Kruesi E., "The Measurement Task Area", IEEE Computer, Vol 16, Number 11, November 1983

[25] Levin R.I., Rubin D.S., "Statistics for Management", Fifth Edition, Prentice-Hall, USA, 1991

[26] Rauscher T.G., Ott L.M., "Software Development and Management for Microprocessor-Based Systems", Prentice-Hall, Inc., Englewood Cliffs NJ, USA, 1987

[27] Mohanty S.N., "Software Cost Estimation: Present and Future", Software Practice and Experience, Vol 11 (2), pp 103-121

[28] Boehm B.W., " Software Engineering Economics", Prentice-Hall, Englewood Cliffs NJ, USA, 1981

[29] Booch G., "Object-Oriented Analysis and Design", Second Edition, The Benjamin/ Cummings Publishing Co., Inc., USA, 1994

[30] Sharp A., "Software Quality and Productivity", Van Nostrad Reinhold, USA, 1993

[31] Shlaer S., Mellor S.J., "Migration from Structured to OO Methodologies", Project Technology, http://www.projtech.com

[32] Abott R., "Program Design by English Descriptions", Communications of the

Microprocessor and Microprogramming, Vol 30, Part 1-5, pp 273-280

[8] Sommerville I., "Software Engineering", Fourth Edition, Addison-Wesley Publishing Co., U.K, 1992

[9] Taylor T., Standish T.A., "Initial Thoughts on Rapid Prototyping Techniques", Software Engineering Notes, December 1982, pp 160-166

[10] Gomaa H., Scott D.B.H., "Prototyping as a Tool in the Specification of User Requirements", The Proceedings of the 5th International Conference on Softwar Engineering, 1981, pp 333-342

[11] Scharer L., "The Prototyping Alternative", IT I Programming, Vol 1, Number 1, 1983, pp 34-43

[12] Alavi M., "An Assessment of the Prototyping Approach to Information Systems Development", Communucations of the ACM, June 1984, pp 556-563

[13] Balzer R.M., Goldman N.M., Wile D.S., "Operational Specification as the Basis for Rapid Prototyping", ACM SIGSOFT Software Engineering Notes, December 1982, pp 3-16

[14] Zave P., "The Operational Versus the Conventional Approach to Software Development", Communications of the ACM, February 1984, pp 104-118

[15] Cameron J.R., "An Overview of JSD", IEEE Transactions on Software Engineering, February 1986, pp 222-240

[16] Clark R.G., "The Design and Development of Embedded Ada Systems", Software Engineering Journal, Vol. 5, Part 3, pp 175-184

[17] Partsch H., Steinbrüggen R., "Program Transformation Systems", Computing Surveys, September 1983, pp 199-236

[18] Smith D.J., Wood K.B., "Engineering Quality Software", Second Edition, Elsevier Applied Science, U.K, 1989

[19] Aquilano N.J., Chase R.B., "Fundamentals of Operations Management", Richard D. Irwin ,Inc., 1991

- Product Developer
- Product Manager

1.5 Applicable Documents

1.5.1 Specifications
None

1.5.2 Standards
a. SEAL QMS Document Creation Template, QS 002, Revision 0.01, 5 April 1994
b. SEAL QMS Document Layout, Presentation and Typesetting Guide, QS 003, Revision 0.01, 1 April 1994

1.5.3 Procedures
None

1.5.4 Guidelines

[1] Boehm B.W., " Software Engineering", IEEE Transactions on Computers, Vol. C-25, Number 12, December 1976, pp 1226-1241

[2] Agresti W.W., "New Paradigms for Software Development", IEEE Computer Society, USA, 1986

[3] Swartout W., Balzer R., "On the Inevitable Intertwining of Specification and Implementation", Communications of the ACM, July 1982, pp 438-440

[4] Giddings R.V., "Accommodating Uncertainty in Software Design", Communications of the ACM, May 1984, pp 428-434

[5] McCraken D.D., Jackson M.A., "A Minority Dissenting Position - Systems Analysis and Design - A Foundation for the 1980's", Cotterman W.W., et al. (editors), Elsevier Science Publishing Co., Inc., 1981. pp 551-553

[6] Boehm B.W., "A Spiral Model of Software Development and Enhancement", IEEE Computer, Vol 21 (5), pp 61-72

[7] Auer A., Levanto M., Okkonen A., Okkonen J., "Solution in Software Crisis",

1 Scope

1.1 Introduction

This survey provides an overview of the current methods and practices in developing software in general. It also covers the management of the software development process as a whole and the quality standards associated with the process

1.2 Purpose

The objective of this exercise is to provide a sound background of those important elements of the development of embedded systems that are common to software development in general.

1.3 Definitions

- COCOMO : Constructive Cost Model
- OOA : Object-Oriented Analysis
- OOD : Object-Oriented Design

1.4 Audience

The audience for this document comprise the various stakeholders of the SEAL, including:

- Full-time staff members of SEAL
- All full-time and part-time post-graduate students associated with SEAL
- Members of the SEAL Management Board
- Head of the Department, Electrical Engineering
- Individuals who will perform internal and external surveillance audits of the SEAL Quality Management System
- Engineering Manager, Meissner

Change History

Configuration Control

Project:	SDES QMS
Title:	Software Development: A General Survey of the Literature
Doc. Reference:	SDE 102
Created by:	H. Bapoo
Creation Date:	24 April 1995

Document History

Version	Date	Status	Who	Saved as:
0.01	95/04/24	Draft	H.Bapoo	:\1995-13\TP\SDE102WP.001
0.02	96/05/29	Draft	H.Bapoo	:\1995-13\TP\SDE102WP.002
0.03	96/08/05	Draft	H.Bapoo	:\1995-13\TP\SDE102WP.003
1.00	96/08/25	Approved	H.Bapoo	:\1995-13\TP\SDE102WP.100

Revision History

Version	Date	Changes
0.01	95/04/24	New Document
0.02	96/04/02	Change of document format to comply with SEAL Document Presentation Guide QS 003
0.03	96/08/05	Document restructured and additional information added
1.00	96/08/25	Update to revision number and status following Board Approval

Management Authorisation

Version	Date	Status	Management Board Minute Reference
1.00	96/08/25	Approved	Section 2.3 of SDE 592

Change Forecast

This document will be updated whenever the literature survey provides new materials that are pertinent to the development of software in general.

- Fault tolerant design techniques.
- Ability to operate in degraded mode under fault conditions.

The traditional approach to organising software quality takes on one of two structures. The first is that of an organisational umbrella which provides a corporate quality service to the company. The second is to attach to a project or development a separate dedicated quality function purely involved in that project activity and nothing else. The result here, is a view of quality as part of the 'production process' whereby it is bolted on the same way as other components. It applies equally well to hardware and software that a typical project organisation separates the quality function and leads to the idea that responsibility for quality matters lies elsewhere.

One of the main problems of establishing a quality system is that the traditional approach tends to erect barriers between the development team and the quality function. As Smith and Wood [18] put it, quality must be accepted as part of the working practice and only then will its level be raised. This is largely a matter of motivation in that the average programmer or designer only sees the task in narrow terms. He will view the task of programming and designing within a localised framework rather than in terms of the whole project. Only when the team members look at problems, and their solutions in overall terms, will their perception of quality and the means of incorporating it into the product be achieved.

One possible approach is to introduce into the project the concept of overall quality improvement. This involves the idea of quality circles, now popular in manufacturing industries [19]. The idea is to make all team members responsible for quality and thus distribute the function throughout the project rather than identify a single responsibility.

By extending the principle to all activities (including testing), greater emphasis can be focused on quality-related aspects. The general level of motivation towards achieving better quality is improved since team members then have a far greater understanding of the reasons for it and the benefits which will result.

Currently quality systems concentrate on establishing the existence of standards and controls and seek, by means of audit, to verify that they are applied. They operate by specifying various areas for control and requiring that suitable audits take place to verify that they are being applied. The role of quality management, in these systems, is to monitor conformance with procedures and standards by

4 Managing Quality in Software Development

4.1 Definitions

Quality is an elusive feature of a product. The main reason, stems from the inability to specify a requirement in its entirety. The perception of quality in software is seen principally in terms of the time a software system operates 'correctly'.

The term *quality* refers to the whole concept of specifying, designing and implementing software and hardware which meets the requirements of the user. It involves all stages in the 'life-cycle' and thus addresses the various methods and tools which can be used to achieve it.

Quality control is the activity of measuring achievements and performance against some preceding specification or standard.

Quality assurance embraces the activities which check that the quality control process is taking place and that the standards, methods and tools are adequate.

The organisational split between these activities varies between companies and there is no correct way of apportioning tasks and responsibilities. What matters is that the tasks are carried out.

The *quality tasks* facing the Software Engineer are:

a. Verify the requirement
b. Validate the design
c. Perform adequate tests.

4.2 Achieving Quality Software

Various features of software have a direct bearing on its quality. These include:

- Usability and ease of diagnosis.
- Traceability of requirements through code.
- Visibility of functions in the code.
- Documentation clarity and consistency.
- Spare capacity in timing and memory resources.

specification.

3.6 Module Definitions

This is a more detailed level of design than the module specifications. It involves the actual coding of the module specification requirements and addresses the testing and performance of the modules. It includes:

- Diagrams
- Source code listing
- Test specification
- Test procedure
- Test results

This package then constitutes a total description of the module and its design and test history.

3.7 Utility Requirements Specifications

This should contain a description of the hardware requirements including the operator interface, hardware memory requirements, processor hardware, data communications hardware, software support packages, etc.

3.8 Development Notebooks

It is an excellent thing for each designer to have a development notebook - a looseleaf file containing his written notes, listings, changes, correspondence, etc. This can be an invaluable aid to diagnosis during testing when the reasons for certain lines of code or some changes, have become blurred in one's memory.

3.9 Requirements Matrices

These provide a graphical system of cross-referencing between specifications as well as a method of checking off each requirement against the test specifications.

The functional specifications document takes the requirements of the user specification and describes the actual processing functions which are to be carried out in order to achieve those requirements. It addresses language, memory requirements, data bases, the partitioning of the system into subsystems, inputs, outputs, interface communications, data flow, etc.

The important difference between the functional specifications document and the user requirements specification is that whereas the latter states what is required, the former describes how it will be achieved in terms of functions It is usually prepared by the supplier in response to the user requirements specification and, once agreed, becomes a part of the contractual documentation against which acceptance is ultimately reviewed.

During the detailed process of generating the functional specifications, deficiencies and ambiguities in the user requirements specification will arise. These must be negotiated and documented as mentioned above.

3.3 Software Design Specification

Once the software and hardware functionality has been separated, one can proceed to consider the software aspects separately. This proceeds in a top-down manner until subsystems or module specifications are derived, according to size and complexity. Whereas the functional specifications document addresses the externally apparent functions of the system, the software design specifications document addresses the way in which these functions will be implemented in software.

3.4 Subsystems specifications

If the system is small enough it may be possible to do without subsystems specifications and to move directly to module specifications. In large systems, however, subsystems ultimately consist of modules and the subsystem specification describes the functions of the subsystem, the data flow between modules and the interfaces to the other subsystems.

3.5 Module Specifications

This is the 'foundation' of the hierarchy since it contains the basis of the system code. The module specifications document describes the inputs and outputs of the modules and the logic to be performed by them. The flowchart will be part of the

3 Documentation Support for Software Development

The document structure supporting a development varies according to the process in place. This section presents some of the most common documents attached with software development [18].

3.1 User Requirements Specification

This document describes the functions to be performed by the system. it should be complete and unambiguous and should be quantitative wherever possible in order to facilitate the assessment of whether the requirements have been met. Since the user requirements specification is a formal contractual document it is important that its preparation is a thorough and painstaking process involving all interested parties. A good structure is described in the IEE document [18] and includes:

a. An overview providing a perspective of the system within the plant or total environment in which it is to be employed. It should indicate the overall objectives of the system.
b. System objectives setting out in full detail the objective as they relate to the operating requirements.
c. System interfaces outlining how it is required to communicate with the world. This includes both human and electrical/data interfaces.
d. System environment which covers all the features which would affect the hardware and software design.
e. System attributes describing and listing the parameters which constrain the design.
f. Test considerations including diagnostic requirements which will affect operation and maintenance.
g. A commercial section addressing licensing, etc.

Ideally the user requirements specification should be a pre-contractual document since it specifies what the contract will provide. In practice, however, it may well be subject to negotiation and change after the contract has been established, in which case both parties must carefully evaluate the performance, cost and schedule implications.

3.2 Functional Specifications

not to the code.

The altered code is produced by a two-step process:

- First, appropriate changes are made to the "formal development", which is the record of the sequence of transformations and decisions that were used to convert the specification into the original code.

- Second, the modified formal development is replayed to produce the new version of the system.

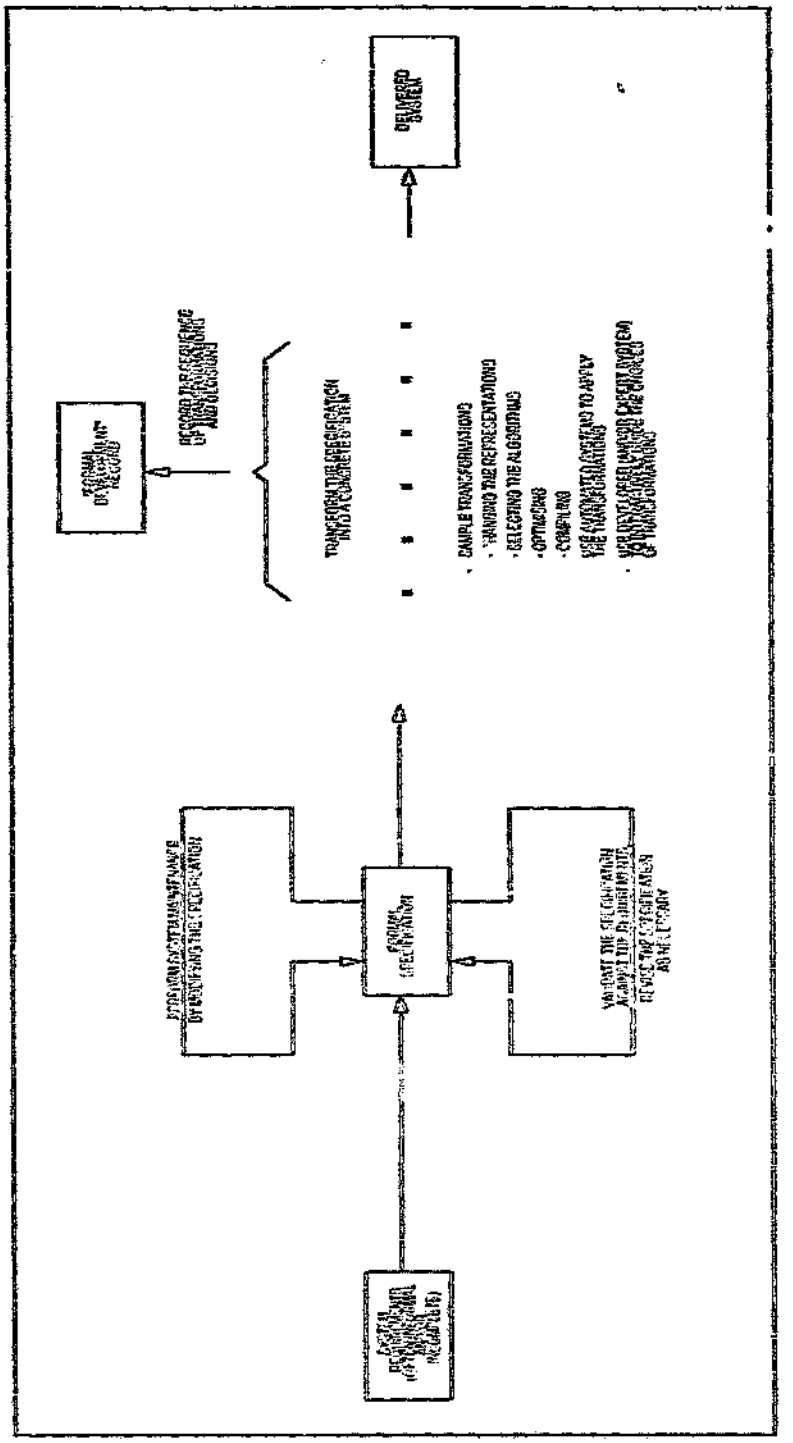


Figure 5: The transformational Paradigm

specification has many of the characteristics of a high-level design, and premature design decisions may be taken. However it is a fact that specification and design are intertwined, and one cannot be considered in the absence of the other [3].

The steps involved in transforming the operational specification into a delivered system will, ideally, be assisted by automation in the future. However, even without automated support, the operational specification has value by:

- Separating the development process into problem-oriented and implementation-oriented phases.

- Providing users with an executable system model early in the process that is useful for clarifying requirements and providing some early validation.

2.6 Transformational Implementation

Transformational implementation is an approach to software development that uses automated support to apply a series of transformation that change a specification into a concrete software system. Transformational programming, as defined by Partsch [17], is a methodology of program construction by successive applications of transformation rules (partial mappings from one program scheme to another such that an element of the domain and its image under the mapping constitute a correct transformation). The transformational approach addresses the labour intensiveness of software development by using specialised computer software to transform successive versions of the developing system mechanically. It is not now a generally available alternative for the production of large software systems.

Figure 5 shows that the formal specification is the system baseline. It is :

- The object that is validated against user requirements; under an ideal realisation of this paradigm - that is, all transformation automated and correctness-preserving - the specification is the only object that needs to be validated.

- The starting point for the transformation phase.

- The object on which maintenance is performed; any change to the functioning of the system is expressed as a change to the specification,

feasibility of the design to be implemented in the target hardware-software environment.

Preparing an operational specification requires using a different basis for separating early development activities. The goal is to produce an operational specification that deals only with problem-oriented issues and to express it in some language or form that enables it to be executed, evaluated or interpreted to reveal the behaviour of the system. The resulting operational specification is a kind of prototype: one that generally produces all of the functional behaviour of the system but does so by using different media that will be used in the delivered system.

Some media for expressing operational specifications are: languages such as GIST, PAISley, semantic models such as virtual processors, state machines and Jackson System Development [15].

Figure 4 shows the new paradigm built around the preparation of an operational specification. From the system requirements, however informal or incomplete, the developers construct the operational specification. Because it is capable of exhibiting behaviour, the operational specification can be exercised by both the developers and the users to begin validating that requirements are begin met.

The operational specification provides an opportunity to understand and clarify user needs early. With the conventional life-cycle model, the opportunity to experience system behaviour is not available until much later in the process - after the implementation phase.

When the validation/revision loop in figure 4 has resulted in a satisfactory operational specification, developers can begin to consider the implementation-oriented issues. One possibility consistent with perceiving the operational specification as a prototype, mentioned by Agresti [2], is to revert to the conventional life-cycle model and begin the design phase. However, this option usually fails to capitalise on the merits of the operational specification as already offering the ability to produce system behaviour. A more common strategy is for developers now to transform the behaviour-producing mechanisms of the operational specification into structures that respond to the realities of the target environment.

The main criticism of the operational approach [16] is that an executable

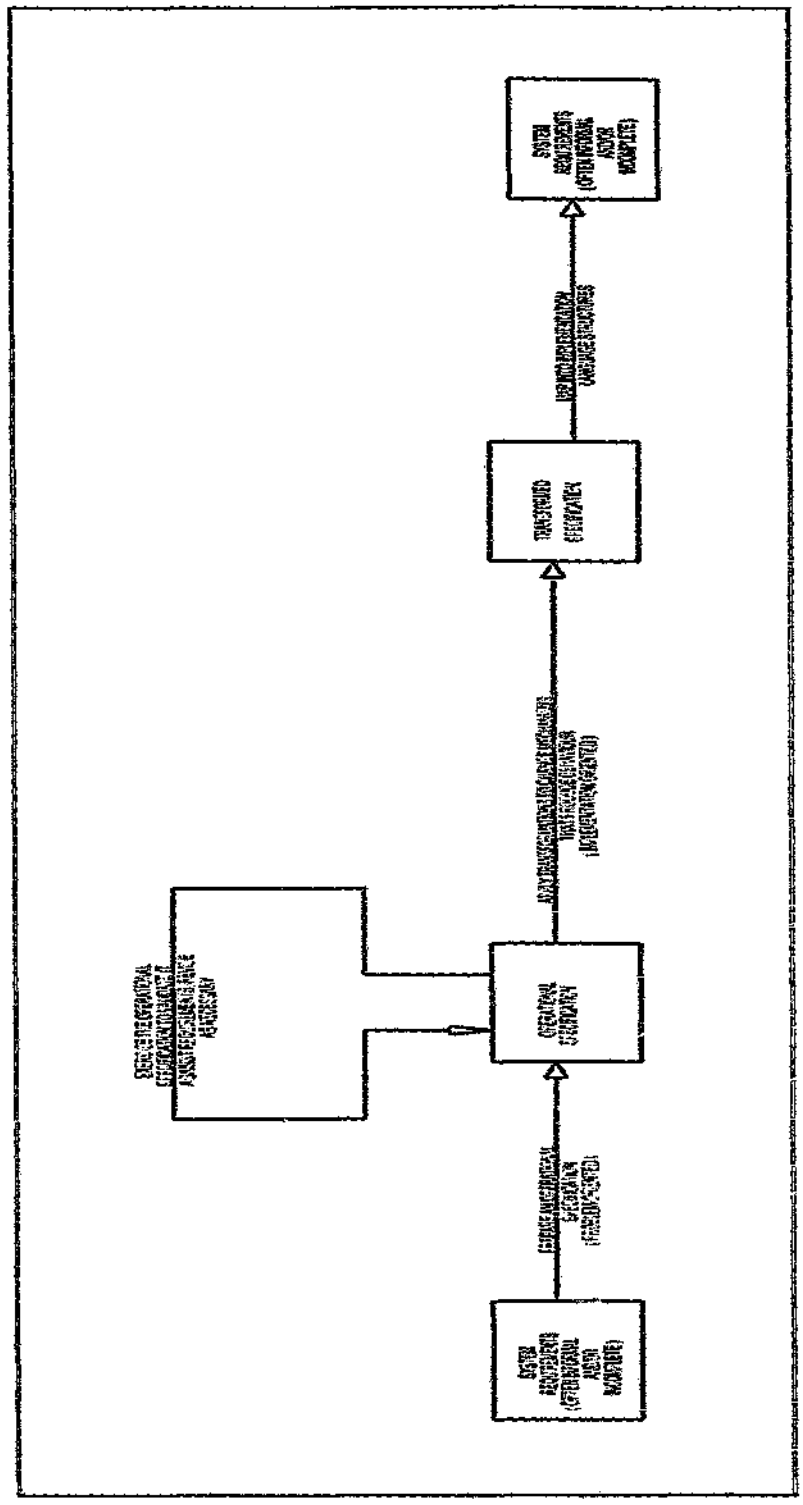


Figure 4: The Operational Paradigm

investment of more effort?)

Figure 3 shows both alternatives. In the case of a throw-away prototype, the developers must still extract useful information from it to contribute to the requirements specification and design of the complete system.

Studies by Gordon and Bieman [42] tend to show that the evolutionary approach is more popular than the throw-away approach. Furthermore, they describe throw-away prototyping as being economically feasible for small contracts.

The prototyping paradigm illustrates that prototyping can serve as a separate route to software development through the refinement process at the top of figure 3. Alternatively, prototyping can be joined to the conventional life-cycle model in a hybrid prototyping/waterfall paradigm.

2.4 Operational Specification

An operational specification is a system model that can be evaluated or executed to generate the behaviour of the system. It can be viewed as a prototype [13]. An operational specification of a system is developed early in the development process, during what is typically the requirements analysis phase. During the specification phase, the developers formulate a system to solve the problem and specify this system in terms of implementation-independent structures that generate the behaviour of the specified system [14]. The operational specification is executable by a suitable interpreter.

A key feature of the conventional life-cycle model is its separation of external system behaviour (the "what" of requirements specification) from internal system structure (the "how" of design). Separation of activities on this basis introduces many problems, to which the approach of operational specification directly responds [2]. A second major problem with the rationale for separation in the conventional model is that it leaves too many issues to impinge on design decisions. The designer needs to consider:

- Problem-oriented issues of decomposing high-level functions into successively lower levels.
- Purely design issues such as information-hiding and abstraction.
- Implementation issues such as system performance constraints and the

The software development paradigm that incorporates prototyping, as described by Agresti [2] is shown in figure 3. The starting point, at the left side of the figure, is some notion of the user's needs. This may vary widely in formality and concreteness, ranging from a detailed requirements document to some vague ideas about a proposed system, obtained by interviewing prospective users. From this expression of needs, the prototype developer determines the objective and scope of the prototype. Often the prototype addresses aspects of the system - such as the user interface - that are difficult to specify through conventional means.

The methodology of prototyping emphasises creativity, and accepts both uncertainty and change. It can bridge the communication gap between the system developer and the user [10]. However the benefits of prototyping will only be observed if the process is supported by effective management [11]. Some key management requirements of prototyping are:

- Decide what is intended to be learnt from building the prototype [?]
- Decide when should the modelling process end [11]
- Securing the proper time and staff commitment [11]

In addition, the prototyping philosophy and plans should be understood by both designers and users [12]. Without proper planning and depending on the scope of the prototype, its development can be an expensive and time-consuming process. Effective planning will help ensure that the prototype is delivered in a timely manner. There is nothing worse than - rapid prototyping that is not [2].

The development plan should maximise the use of very high level language or screen management utilities to accelerate prototype development. When the objectives of the prototyping effort have been attained, a basic decision must be made: either refine the prototype into the complete system or begin a conventional development process, having benefited from building the prototype. The decision should be based on the costs and benefits of each alternative. Two key factors are:

- How much functionality is already present in the prototype.
- Will the design support a maintainable system (Is the prototype worth the

perspectives. In this section we look at some of those perspectives.

Relating the topic to the CM model (section 4.3), the core set of measures recommended for initial measurement focus on factors of size, effort, schedule and quality [50]. Saiedian and McClanahan [50] lists four basic software measures for the CM model and relate them to the above factors. The list is reproduced in table 1 below.

Table 1: CMM Levels and Software Characteristics

Units of Measure	Characteristics Addressed
Counts of source statements	Size, progress, reuse
Counts of staff hours	Effort, cost, resource allocation
Calendar dates	Schedule
Counts of software problems, failures and faults	Quality, readiness for delivery, improvement trends

Smith and Wood [18] divide software metrics in four broad groups:

a. *Code metrics*. These address the more easily qualified measurements at the code level. They are however, not available until well into the design-cycle and hence do not provide the complete answer. They include:

- Lines of code: the number of lines.
- Cyclometric complexity: a measure of path and execution complexity.

b. *Structure metrics*. These addresses the higher levels of design and include:

- Information complexity: a measure involving data flow and data relationships between components.
- Invocation complexity
- Review complexity
- Stability

c. *Hybrids metrics*. Hybrid metrics are modifications of structure metrics weighted according to various code-related measurements. Three hybrid metrics are:

- Weighted information flow
- Weighted review complexity

5.1 Limitations

In practice software metrics often fail to provide the expected predictability. Several factors account for this:

a. The use of only single independent variables. A simple model might address the relationship between, say, error rate and a particular complexity measure. This approach fails to recognise the interrelationships and trade-offs between parameters. One designer might achieve error-free code by patient documentation and review of his code, whereas another might prepare code faster and expend effort on debug and test. The resulting error rate might well be the same although the metrics are not.

b. The use of only a single independent variable. In other words only one metric is chosen to model the software performance. Multiple regression techniques [25] might prove to be helpful in this instance. Multiple regression analysis can derive a mathematical model that will take all influential factors into considerations and at the same time provide a measure of the weight or relative importance of each contributing factor.

c. The assumption of linearity. This assumption has often led to the rejection of a metric as a valid indicator of software quality. More sophisticated models may be necessary.

d. The assumption that the initial estimates do not affect the project. Most estimation and analysis tools assume that a project's historical data will not change irrespective of the estimate managers make [46]. Abdel-Hamid [46] has found that systems react to initial estimates -different estimates create different projects. So fitting raw historical results to a new project may be a bad idea if the historical data comes from poorly managed, inefficient projects.

e. Misuse. Each metric describes a particular feature of the software. Judging only on the number of lines of source code would lead to the erroneous conclusion that the use of assembly language is more productive than high level language.

5.2 Categorising Metrics

The various software measures can be grouped and examined in different

5 Software Metrics

We have seen that the process of developing software is not just about coding - it involves documentation, design, programming, file building, testing, training, quality assurance and, above all, management. The processes that govern those phases have been exposed earlier. A key ingredient to the improvement of those processes is feedback. Only through feedback measurements can some of the management questions be answered. The importance of software metrics was already underlined in section 4.3 above, where the ability to capture relevant data was identified as the main characteristic of level 3 of the CM model. As it is, the development of software has been characterised by cost overruns and schedule slippage [18]. The main objective of software metrics and the main reason why so much research is done on the subject, irrespective of whether it still appears as an inexact science, is to obtain the ultimate control over the software process. The ability to provide accurate measurements is what differs empirical practices to firmly founded sciences. The best argument on the subject comes from Lord Kelvin [24]:

> When you can measure what you are speaking about and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the stage of science.

The need for a predictable process, as expressed in those famous words of Lord Kelvin, is what is driving the software industry to strive to reach a stage of maturity where statistical process control can be effectively exercised. A predictable system will be able to answer questions like 'How much will it cost to develop?', 'How long will it take and with how many staff?', 'What levels of productivity can be expected?' The answers to these questions become more and more crucial to managers as the level of investment in software projects increases. All of these questions, in principle, can be answered with a single quantity. The question is - can we supply such a number, or metric, for software?

Several measurement systems have been developed in an attempt to answer the above question with some degree of accuracy. In general we find that a reasonable approach to the subject is to build one's own database of results and to proceed progressively. This approach starts with studying the many software projects in the past, gathering data and setting up models to try to predict the parameters required. The historical data provides a means of validating the metrics against actual working systems.

reviewed.

e. To establish that there are adequate controls over test and integration.

f. To establish that there is a real capability, on the part of the team, to implement the requirements into a system.

g. To establish that strong integration controls exist.

h. To establish that there is control over bought-in software.

Audits often relies on checklists. As with software metrics, it is important to view checklists in the right perspective and to be aware of their limitations. Smith and Wood [18] point out the advantages and disadvantages of the use of checklists. On one hand they provide a means of ensuring that each question is remembered and thus enable the auditor to select the most appropriate questions. Furthermore the checklists can be revised and updated as additional lessons are learned and, thus, one is always presented with the total of previous experience. On the negative side, however, there is a temptation to expect yes/no answers and this can lead to a false view of the situation. In most cases it is the reason for the answer rather than the answer itself which provides the real information. For example, "Is a high or low level language being used?" has no right or wrong answer. The reasons given for the answer are, however of great importance.

Fenton [49] mentions five other shortcomings of standards:

- Software standards tend to overemphasise processes. He [49] argues that software standards almost entirely neglects the product and concentrate on the development process.

- When precision - and hence mandatory enforcement - is impossible, traditional standards bodies use the terms "codes of practice" or "guidelines". This lack of precision and support often frustrate the software developer.

- A large number of requirements presented in software standards are presented in such a way that it is impossible to determine conformance in an objective sense.

- Some software standards prescribe, recommend, or mandate the use of technology that has not been validated objectively.

- Many software standards attempt to address a too large portion of the system-development life cycle. Such standards are very difficult to apply and often are left on the shelf.

4.5 Audits

Audit involves an assessment of the controls used in the design and management process and an evaluation of their effectiveness.

Audit is a worthwhile activity and should be used both within an organisation and as a method for controlling vendors. The objectives of the audit are:

a. To establish that there is adequate control over the software design process.

b. To establish that standards are being used for the documentation and production of the software.

c. To assess, aided by checklists, the comprehensiveness of the controls and the integrity of the product.

d. To seek the evidence that the standards are being applied and periodically

quality measurement factors for software. It is discussed further in section 5.2.

Each country has a specific instantiation of the ISO 9000 series of standards, for example in South Africa the standard is known as SABS 9000 while in the United Kingdom it is known as BS5750. The requirements of the standard is partitioned into 20 headings [23]:

a. Management Responsibility
b. Quality System
c. Contract Review
d. Design Control
e. Document and Data Control
f. Purchasing
g. Control of Customer-Supplied Product
h. Product Identification and Traceability
i. Process Control
j. Inspection and Testing
k. Control of Inspection, Measuring and Test Equipment
l. Inspection and Test Status
m. Control of nonconforming product
n. Corrective and Preventive Action
o. Handling, Storage, Packaging, Preservation and Delivery
p. Control of Quality Records
q. Internal Quality Audits
r. Training
s. Servicing
t. Statistical Techniques

One of the complaints about the ISO 9000 series standard is that it is not industry-specific [21]. It is expressed in general terms, and can be interpreted by the developers of diverse products such as ball-bearings, hair dryers, automobiles, sports equipment and televisions as well as software. However, in a sense, those are the same factors that make the strength of the ISO 9000 series standard. By focusing on the factors upon which a quality system should be based irrespective of the organisation type and by not imposing a specific implementation method, it allows freedom of implementation techniques (according to the industry type, geographical location and other factors) while providing a "common language" for organisations of different industries.

make it easier to communicate with future and present developers. In addition they reduce the likelihood of the developer to make errors. Software Quality Standards is a subject on its own. While it is the subject of a lot of debate, it is taking a growing importance in the development of software. It is widely discussed in the literature. Treatment of the subject, for the present study, will be limited to the introduction of the major terminologies and the ISO 9000 series of standards.

Quality manuals are usually divided into standards, procedures and guidelines. Ince [21] defines those terms as follows:

A *standard*, in the context of software development, is a directive which describes how a particular document should be structured, and what data should be contained in the document.

In contrast a *procedure* is a text which describes how a particular software task is to be carried out.

A *guideline* is a text which provides advice on an activity. It is not prescriptive as a standard or a procedure.

The ISO 9000 is becoming the main way customers are judging the competence of a software developer. It has been adopted for use by over 130 countries [22]. A number of documents have been produced which relate the standard to the software industry. Some of the relevant standards for the software industry are:

- ISO 9001 Quality Systems - Model for Quality Assurance in Design, Development, Production, Installation and Servicing. This is a standard which describes the quality system used to support the development of a product which involves design.

- ISO 9000-3 Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software. This is a specific document which interprets ISO 0001 for the software developer.

- ISO 9004-2 Quality Management and Quality System Elements - Part 2. This document provides guidelines for the servicing of software and facilities such as user support.

- ISO 9126 Information Technology - Software Product Evaluation - Quality Characteristics and Guidelines for Their Use. This document provides six

- well-defined training programmes

An organisation operating at Level 3 will probably abide to its well-defined development mechanism even when in a crisis situation; i.e. this stage is marked by strong commitment and faith of all concerned parties into the process. At such a maturity level, advanced technology can usefully be introduced.

d. Level 4 - Managed
While the factors encountered at the previous levels should continuously be refined, at Level 4, the organisation concentrates its effort at building a system driven by comprehensive process measurements and analysis. This level is typically characterised by data gathering and analysis, quality measures, quality estimates and quality plans. It is at this level that most significant quality improvements begin.

e. Level 5 - Optimising
Once an organisation has established process measurements and quality plans, it can use these tools to efficiently assess the tasks involved, learn from the gathered experience and use this knowledge to make improvement at the process level itself. This is the main difference between Level 5 and the other levels. The previous levels are also concerned with optimisation, at various degrees, but they all focus on the product whereas Level 5 focuses on the process. It is important to realise that level 5 is not an optimised state. There is no such entity as an optimised organisation, for process improvement is a way of working rather than a destination, and as such it is a continuous optimising process.

Notice that many similarities can be drawn between ISO 9001 and the CMM. In fact there has been studies to map the twenty clauses of ISO 9001 to the CMM key practices [43].

A possible pitfall when using the CM model is to regard software development as being a production activity, whereas, as De Marco [44] observes, it is more of a research-and-development activity.

4.4 Standards and Software Management

Software Quality Standards in general provide a framework for doing things that

- Even if the above plans exist their is no strict management of their applications

- Tools are neither well integrated with the process nor uniformly applied

- Success depends essentially on individual efforts

The litmus test here is to observe how the organisation behaves in a crisis situation. If it abandons established procedures and essentially reverts to coding and testing, it is likely to be at the Initial Process Level.

b. Level 2 - Repeatable
At Level 2 a stable process of statistical control has been reached. Commitments, costs and schedules estimates and plans are met. Changes are done effectively. This level is rather characterised by a routine control and implementation of the development process. The weaknesses of this level can be summarised as follows:

- At this level the organisation is not mature enough to be able to accommodate for new tools without major disruption.

- When faced with the development of a new product, e.g. a larger-scale project, the team will probably be disoriented.

- Changes in the team or the management can cause severe disruption

All the three weaknesses above arise because of factors that tend to disrupt the routine trend upon which Level 2 is based.

c. Level 3 - Defined
At this level, the organisation has a deeper insight into the process, i.e. the mechanism of each stage of the development process is understood and is no longer merely a routine exercise driven by project milestones. This stage is characterised by:

- Documentation of each stage of the process, i.e. greater visibility and control

4.3 The Capability Maturity Model

Often the need for improvement and the achievement of a more mature organisation are recognised while the goal itself or the meaning of improvement is not clearly defined. Furthermore when involved in process improvement of software development one problem often encountered is that of not being able to measure progress. In those respects, the capability maturity model (CMM) [20] provides an excellent framework. It draws a clear picture of the ultimate goal for the organisation and provides means of gauging progress along the way. The CM model characterises the software process into five maturity levels. By establishing their organisation's position into this maturity structure, software professionals and their managers can more readily identify areas where improvement action will be most fruitful. The capability maturity framework is shown in figure 6.

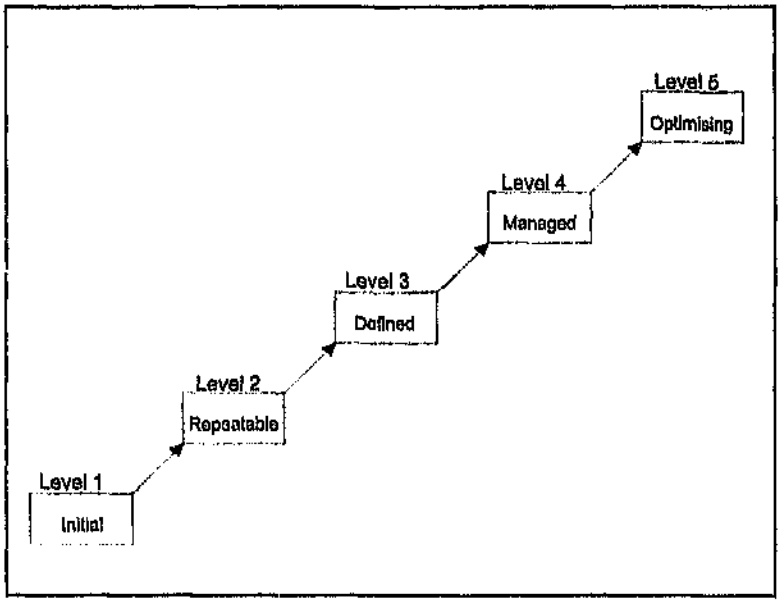


Figure 6: The Five Levels of the CM Model

The five process maturity levels are:

a. Level 1 - Initial
The first level is also known as the chaotic stage. It is characterised by the following factors:

- The organisation operates without formalised procedures, cost estimates and project plans

code in order to assess its capability of carrying out the requirements and its conformance to standards.

Test and integration. Testing involves a number of methods. It is important to plan a structured set of tests which build on each other so that confidence in modules leads to testing of subsystems (groups of modules) until the entire system is integrated and ready for functional and environmental test. Types of test include:

- Formal proof of correctness
- Validation of code by inspection or walkthrough
- Static tests
- Dynamic tests involving test data

Failure feedback. Repetition of a proven piece of code with the same inputs will only continue to generate the previous result. Hence, software faults will only be revealed by exercising the program in different ways. This arises in field use, as a result of which reliability growth will be observed following modifications which arise from field failures. The more structured and formal the system of failure feedback, the faster will be the reliability growth. There are three distinct areas of failure data collection:

a. Informal reporting prior to an item becoming subject to configuration control.
b. Failure reporting during test.
c. Field failure reporting from the user.

A large number of designers ignore the above approaches. It is tempting to proceed with coding when only a part of the total design is conceived. Indeed, commercial pressures make this almost inevitable.

Many designers pay lip-service to the techniques of quality and go so far as to install quality manuals and procedures. Frequent audits against these standards and procedures give the impression of achieved quality [18].

Finally, it is important to point out that quality is essentially a complex concept. As Kitchenham and Pfleeger [48] observe, because it means different things to different people, quality is highly context-dependent. Just as there is no one automobile to satisfy everyone's needs, so too there is no universal definition of quality.

means of:

- Establishing quality plans
- Quality management reporting
- Review meetings
- Quality audits

There are a number of activities and methodologies which contribute to software quality. These fall into two categories [18]:

a. Quality which attempt to identify and remove errors. These disciplines also minimise the probability of errors being committed.

b. Specification and programming methods and their associated automatic tools which make it highly unlikely that certain types of software error will be committed by removing many of the traditional steps in software design.

The former group, commonly referred to as software quality, includes the following areas:

Configuration management. This is the essential quality activity of keeping control of the issue and status of all hardware, documents, firmware and software. Control of changes is additionally important since there is no visibility to the software on discs and PROMs other than via labelling and documentation.

Documentation standards. Since the only visibility to software is via documentation, a formal structure of specifications, design documents (e.g. diagrams) and code listings is essential. Naturally these should be appropriate to the size of the software package.

Programming and design standards. The objective is to write clear structured software using well-defined modules whose functions are easily understood. There is no prize for complexity and there are various methods for developing structure, including flow, hierarchical and structured diagrams as well as pseudo code.

Design reviews. These are assessments of the design against the requirements. Since the early design reviews are carried out before the product is ready for demonstration or test they take the form of an assessment of the functions provided by the design specifications and flow charts. Code inspection and walkthrough are a part of later reviews and involve detailed examination of the

- Object-Oriented Analysis (Sally Shlaer and Stephen Mellor)
- Object-Oriented Design (Grady Booch)
- Object-Oriented Requirements Specification (Baillin)
- Object-Oriented Systems analysis (Hewlett-Packard)
- Object-Oriented Structured Design
- ObjectOry (Ivar Jacobson)
- Synthesis/Analysis (Meilir Page-Jones and Steven Weiss)

Two major design strategies stand out, and are discussed separately in the later sections:

- *Functional design*: The systems is designed from a functional view-point, starting with a high-level view and progressively refining this into a more detailed design.

- *Object-oriented design*: Objects have a set of attributes defining their state and operations which act on these attributes.

The design process involves describing the system at a number of different levels of abstraction. In reality design process are often not sequential, but go in parallel with different design products developed to different levels of detail in the course of the design process. It is common to begin the next stage before a stage is finished simply to get feedback from the refinement process. Project planning may require difficult parts of the design to be tackled first so that management can make more informed estimates of the system development time. This can be reconciled with the concept of prototyping introduced earlier.

Sommerville [8] maintains that there is no 'best' strategy for large projects and that often functional and object-oriented approaches are complementary rather than opposing techniques and each may be applicable at different stages of the design process. An object-oriented approach to software design seems to be the most natural at the highest and lowest levels of system design. At these levels, an object-oriented approach leads to the production of independent components which are usually more maintainable than functional components.

It is important to be aware of the limitations of those design methods. In practice, the guidance given by the methods is informal so that conformity of solutions is unlikely. These "methods" are really standard notations and embodiments of good practice. By following these methods and applying the guidelines, a reasonable design should emerge but designer creativity is still required to decide on the

methodologies can help to create a good product if used correctly, but they will not necessarily prevent the developer from producing a poor product.

There is a wide range of methodologies available (although some are just refinements on others). No single methodology can be claimed to be perfect to all problems. The solution, it appears, is to be familiar with a range of them in order to increase the chances of selecting the right one for the particular task. In his treatment of methodologies, Sharp [30] lists some of the methodologies and diagramming techniques. The list is reproduced below:

- Action Diagrams
- Control Flow Diagrams
- Data Flow Diagrams
- Decision Tables
- Entity Relationship Diagrams
- HIPO Charts
- Jackson Structured Design
- Modern Structured Analysis
- Petri Nets
- Program Design Language
- Problem Statement Language
- Real-time Structured Analysis (Hatley and Pirbhai)
- Real-time Structured Analysis (Ward and Mellor)
- Statecharts
- State Transition Diagrams
- Structured Analysis (DeMarco)
- Structured Analysis (Gane and Sarson)
- Structured Analysis (Yourdon)
- Structured Analysis and Design Techniques (SADT)
- Structured Analysis and System Specification (SASS)
- Structured Requirements Definitions
- Warnier-Orr Diagrams

In addition there are the object-oriented analysis and design techniques. Some of the common ones are listed below:

- CRC Cards (Ward Cunningham)
- Hierarchical Object-Oriented Design
- Object Modeling Technique (James Rumbaugh)
- Object-Oriented Analysis (Peter Coad)

6 Software Design

6.1 Introduction

The design stage of the development process is itself the subject of a lot of study. Its main features are exposed in this section.

Software design involves a number of different stages:

a. Study and understand the problem.

b. Identify gross features of at least one possible solution (if possible consider more than one solution).

c. Describe each abstraction used in the solution.

In order to assist the design process, formal methodologies have been developed. Languages like English are too verbose and imprecise to be good at communicating the way systems work, and they do not provide a framework for doing things consistently. In addition the very act of writing down formally the mechanism of the design, helps in finding the shortcomings and weaknesses of the design. Methodologies aim at solving the problem of writing down the way that systems work or should work.

A formal definition of a methodology [29] is a collection of methods applied across the software development life cycle and unified by some general, philosophical approach. A method [29] is a disciplined process for generating a set of models that describe various aspects of a software system under development, using some well defined notation (e.g. object-oriented design method). Alternatively, a methodology, can be viewed as a formal structure or language for capturing some aspect of the system being described or the solution being proposed [30]. For example, data flow diagrams describe the way that data flows around the system; state transition diagrams describe the possible states and changes of state in a software system.

A design methodology fits within the broader framework of a development process. Consequently a design methodology depends much on the other aspects of the development of the software. For instance, on its own, a methodology will not compensate for poor requirements. Like any tool,

5.4 Software Packages for Software Metrics

Several software packages which attempt scheduling models to provide software metrics are currently available. SLIM, GECOMO, PRICE S, ARTEMIS are a few of the prominent names in this field [18].

Those packages seem to be more useful for large projects (e.g. over 100 000 source code statements). Some benefits is likely from the discipline needed to collect and input the necessary data.

However, a number of useful studies and user surveys have revealed quite a large variation in response. Whilst some projects have certainly benefitted from the use of these tools, some users have reported as much as a 50% discrepancy in the values produced [18]. It seems that in spite of quite large data bases and complicated parametric models it is still possible to produce projects which are able to lie outside the boundaries of the models being used.

Certainly on larger projects (i.e. over 100 000 source code statements) some benefit is likely from the tools and from the disciplines needed to collect and input the necessary data.

The nominal effort equations for the intermediate model are:

Organic	$MM_{nom} = 3.2 \times KDSI^{1.05}$	(7)
Semidetached	$MM_{nom} = 3.0 \times KDSI^{1.12}$	(8)
Embedded	$MM_{nom} = 2.8 \times KDSI^{1.20}$	(9)

The total effort is then computed from

$$MM = MM_{nom} \times P \left(\prod_{i=1}^{15} \text{effort multiplier}\right) \quad (10)$$

where the effort multipliers are obtained from table 3.

Carefully examining table 3 reveals some reasons why the development time of embedded systems projects can so easily be underestimated Suppose one estimates the development time for the software in a new product based on previous experience. If the previous experience were not with embedded software, one may not consider the added difficulty that will be encountered because of the high reliability, very high execution time constraints, and very high main storage constraints that are normally a part of embedded environment. The multipliers for these factors are 1.15, 1.30, and 1.21, which implies an 81 percent increase in effort over a nominal project for just these three factors.

Two additional points noted from examining table 3 are:

- The use of modern programming practices and the use of software tools can improved one's situation from the usual very low rating to a very high rating. This result in an effort reduction of more than 50 percent.

- The use of good people can reap significant benefits. For example, very highly rated programmers can cause the effort to be reduced by one-half compared to using very poorly rated programmers. Thus, when hiring it is important to perform a cost/benefit analysis in making the tradeoff between the quality and expense of applicants. Note that this does not imply assigning only the "best" people to a project.

This model as any other empirical model, should be used only as a guideline. If common sense dictates that certain other factors are important in a particular environment, they should be considered. After some experience using the model, it is appropriate to calibrate it to a particular environment.

caution.

The intermediate COCOMO model [26] takes into account an additional fifteen factors that have been found to affect the cost of a software product. These factors are listed in table 3.

Table 3: Software Development Effort Multipliers

Cost Drivers	Ratings					
	Very Low	Low	Nomi-nal	High	Very High	Extra High
Product Attributes						
Required Software Reliability	0.75	0.88	1.00	1.15	1.40	
Data Base Size		0.94	1.00	1.08	1.16	
Product Complexity	0.70	0.85	1.00	1.15	1.30	1.65
Computer Attributes						
Execution Time Constraint			1.00	1.11	1.30	1.66
Main Storage Constraint			1.00	1.06	1.21	1.56
Virtual Machine Volatility		0.87	1.00	1.15	1.30	
Computer Turnaround Time		0.87	1.00	1.07	1.15	
Personnel Attributes						
Analyst Capability	1.46	1.19	1.00	0.86	0.71	
Applications Experience	1.29	1.13	1.00	0.91	0.82	
Programmer Capability	1.42	1.17	1.00	0.86	0.70	
Virtual Machine Experience	1.21	1.10	1.00	0.90		
Programming Language Experience	1.14	1.07	1.00	0.95		
Project Attributes						
Use of Modern Programming Practices	1.24	1.10	1.00	0.91	0.82	
Use of Software Tools	1.24	1.10	1.00	0.91	0.83	
Required Development Schedule	1.23	1.08	1.00	1.04	1.10	

In this model, the basic computations for man-months are modified by an effort adjustment factor. This effort adjustment factor is the product of values for each of the fifteen factors accounted for by the model.

program or reduced experience of the programmers increases the expected estimated effort significantly.

In the embedded COCOMO model, which is of special relevance for this present study:

$$MM = 3.6 \times KDSI^{1.20} \quad (5)$$
$$TDEV = 2.5 \times MM^{0.32} \quad (6)$$

This model rightly recognises and accounts for the additional effort and time required due to the additional constraints imposed on an embedded system.

Key assumptions included in the COCOMO model are [26]:

- The delivered source instructions are the program instructions including formatting statements, data declarations, and control language statements. They do not include comments or utility software. Utility code should however, if it is developed using the same process as delivered program software.

- The development period covered by the model begins at the design phase after the requirements have been specified and ends at the end of the integration and test phase.

- The requirements specification is not substantially changed after the requirements phase.

This last assumption can frequently be the cause for the large discrepancy between estimated completion time and actual completion time, particularly in embedded systems typified by many microcontroller applications. Frequently, as other components of a microcontroller-based system are developed, problems emerge and the solution selected is to change the software requirements to solve the problem. A change in the software requirements after the design has begun is the most common cause for software projects to be late. Unfortunately, many managers do not recognise and understand this problem [26].

The equations presented previously should be used only for rough estimations. In a study of sixty-three projects [26], the actual results showed that the estimates were within a factor of 1.3 only 29 percent of the time and within a factor of 2 only 60 percent of the time. These models should, therefore, be used with

better estimates. Each model has three separate set of equations to account for three types of project development modes: organic, semidetached and embedded. In an organic development mode, the software is developed by a small group of programmers who are experienced both in the particular type of application to be developed and in the development environment. A semidetached mode of development is an intermediate stage between organic and embedded. In a semidetached mode either the project members are less experienced than in an organic mode or the project has a mixture of organic and embedded mode characteristics. Embedded mode software is usually part of a large complex project and must operate within tight constraints imposed by the other components of the system. Since much microcontroller software is part of an embedded system this model is often most appropriate to adapt to the microcontroller environment. Many of the other models have been developed using data from large-scale systems on mainframe computers and do not account for the extra difficulty encountered in an embedded environment.

The top level organic COCOMO model provides the following estimates for the effort and development time required for software projects:

$$MM = 2.4 \times KDSI^{1.05} \quad (1)$$
$$TDEV = 2.5 \times MM^{0.38} \quad (2)$$

In these equations:

> KDSI is thousands of delivered source instructions in the software system.
> MM is the required effort (in man-months) for the software development project.
> TDEV is the required development time (in months) for the software development project.

This model is appropriate for small- to medium-size programs (64 KDSI or less) in an environment where programmers have experience with the kind of project to be developed.

In the semidetached COCOMO model:

$$MM = 3.0 \times KDSI^{1.12} \quad (3)$$
$$TDEV = 2.5 \times MM^{0.35} \quad (4)$$

As expected those equations reveal that the additional complexity and size of the

- *complexity*, indicated by the number and size of the procedural fragments in Fourth Generation Languages (4GLs) and the number of conditionals or branches in more conventional program units

- *calls*, to lower level functions form high-level modules, including menus

- *files*, relations, reads and writes, indicating file functionality

Notice how those size drivers can change our perspective of "small" embedded systems projects.

An illustration of the difficulty of estimating software costs can be found in a report by Mohanty [27]. Mohanty presented nineteen different published models for estimating software costs and then used twelve of the models to compute the cost of a hypothetical system. The estimated costs range from $362,500 to $2,766,667 for the same software project. As Mohanty summarises, "it is highly unlikely that any two models will estimate the same cost for a given project. We also observe that, even today, almost no model can estimate the true cost of software with any degree of certainty." The high scope for error associated with algorithmic models for absolute project cost estimation is also emphasised by Sommerville [8]. He explains that one of the cause of such diverse results of cost estimation is the difficulty to estimate product size in the first place. However algorithmic models can still provide a guideline, or some quantitative basis for management on how to tackle a particular project. This aspect of the models is valuable for reducing risk, even if the actual cost estimates produced by the model are inaccurate. The most reliable way to develop software cost estimates currently, according to Sommerville [8] is to determine which factors influence and how much these factors influence the cost of software development at a given facility. By measuring the factors which have been incorporated into published models and building a data base of these project characteristics and cost, one may improve cost estimating at a particular installation.

5.3.1 Boehm's Constructive Cost Model

As an example of a cost estimating model, Boehm's Constructive Cost Model [28], COCOMO, which is actually a hierarchy of models, will be introduced. This model has for advantage of taking into consideration the special characteristics of embedded systems, instead of merely offering a single set of metrics for all software projects. The top level model provides quick, rough estimates. The second level model takes into account more factors in an attempt to provide

From a management viewpoint, the most common cause of problems in software projects is the lack of a consistent, complete, and reasonable plan [26]. This is true for a wide variety of problems ranging from schedule slippage to failure to satisfy the customer's requirements. It is therefore essential that time and cost be built into the planning process. There are two principal objectives to the planning process [26]:

- First, an overall project plan, which includes estimates of the effort, cost, and schedule required to develop and maintain the desired system. This is used to determine the feasibility and desirability of the project.

- Second, after a project's feasibility has been determined, a detailed plan is developed. This plan contains a set of measurable milestones which will be used to determine if the project is staying on schedule. To be useful a milestone must be a demonstrable event. For example, "publication of functional requirements" or "successful compilation of a module" are appropriate milestones. "Completion of 50 percent of the coding" is not a useful milestone because it is not verifiable.

Unfortunately, there is no foolproof estimating techniques for planning software development time and costs [26]. Most estimates currently used are based on experience and historical data. Many large companies that have been developing software for years, build a data base as projects are developed to aid in later estimates. Since many developers of embedded systems software are just beginning to develop large amounts of software, they have no previous history to fall back on. Even if some members of the staff have had previous experience in estimating software costs in other environments, care must be taken when translating that experience into estimating the development of microcontroller software in a new environment.

The main obstacle at obtaining accurate costs and time estimates seems to be the difficulty in accurately judging the size of the software in the first place [28]. Verner and Tate [45] identify size as the most important single cost driver in effort/costing models. They explain that size should not only be evaluated on number of lines of code (LOC) criterium. Some of the size drivers they propose are:

- *data elements*, implying that the size of a module depends heavily on the amount of data it processes

to do it. Rather, the standard suggests that if the characteristic cannot be measured directly (particularly during development), some other related attribute should be measured as a surrogate to predict the required characteristic. However, no guidelines for establishing a good prediction system are provided.

No matter which approach to software measurements is used, once correlations have been established between the variables of interest and the metrics, then the problem lies in the repeatability of the model. It is tempting to assume, having observed a historical connection between error and coding time, that a repeatable model has been established. This is potentially dangerous, as has been found in the case of the somewhat similar hardware rate regression models [18].

As already mentioned, in order to construct meaningful metrics it is important to have sufficient failure data to enable the models to be derived. The manner of data collection is important but unfortunately there is still very little interest shown in collecting detailed information on the performance and failures of systems. Furthermore, the metrics and modelling methods cannot be validated, let alone generated, without adequate field failure data. Ideally the following data is required [18]:

- A description of the running conditions of the program
- Data and time of the start of the run
- Data and time of the incident/failure
- Date and time of restart
- Date and time of the normal termination had there been no failure.
- Effect of failure on system performance
- Data and I/O load and any environmental factors
- Detailed narrative of the incident

Clearly this quality of data is only available from a highly formalised and controlled maintenance reporting system. It costs money to collect that level of information and, therefore, is only likely to be generated in a project where management are suitably enlightened as to its benefits in terms of reliability improvement in future maintenance planning. Furthermore, if we agree that quality is a complex concept (see section 4.2), it becomes evident that finding a single, simple measurement of software quality is just not going to be possible. Software metrics will have to be context dependent in order to suit the existing concept of quality in any particular environment.

5.3 Time and Costs Metrics

- Weighted stability measure

d. *Programmer-related metrics*. These are concerned with the capability of the development team. It is measured both in relation with the complexity of the application and the nature of the team and include the following considerations:

- Number of changes
- Years of programming experience
- Mix of programming experience
- Number of pages of documents

Earlier (section 4.4) we mentioned the ISO 9126 standard. The standard recommends six software quality measurement characteristics that are outlined in table 2.

Table 2: ISO 9126 Quality Characteristics

Quality Characteristic	Definition
Functionality	A set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs.
Reliability	A set of attribute that bear on the capability of software to maintain its performance level under stated conditions for a stated period of time.
Usability	A set of attributes that bear on the effort needed for use and on the individual assessment of such use by a stated or implied set of users.
Efficiency	A set of attributes that bear on the relationship between the software's performance and the amount of resources used under the stated conditions.
Maintainability	A set of attributes that bear on the effort needed to make specified modifications (which may include corrections, improvements, or adaptations of software to environmental changes in the requirements and the functional specifications).
Portability	A set of attributes that bear on the ability of the software to be transferred form one environment to another (this includes the organisational, hardware or software environment).

As observed by Kitchenham and Pfleeger [48], the ISO 9126 standard recommends measuring characteristics directly, but does not indicate clearly how

Table of Contents

Table of Contents ii

Change History iv
Configuration Control iv
Document History iv
Revision History iv
Management Authorisation iv
Change Forecast iv

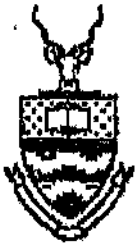
1 Scope 1
1.1 Introduction 1
1.2 Purpose 1
1.3 Definitions 1
1.4 Audience 2
1.5 Applicable Documents 2
1.6 Assumptions 7
1.7 ISO 9001 Requirements Traceability 7

2 Defining the Embedded Systems Concept 8
2.1 Definition 8
2.2 Peculiarities of Embedded systems 10
2.3 Advantages and Disadvantages of Using Microcontrollers . 6

3 Real-Time Issues 16

4 Reliability 19
4.1 Reliability Design Philosophies 19
4.2 Error Management 23

5 Software Testing 25
5.1 Faults 25
5.1.1 Causes of Faults 25
5.2 Test Management 27
5.3 Limitations of testing 28
5.4 Test Strategy 28



SDES QMS

Embedded Systems - A Literature Survey

Technical Product

Version 1.00

Document Status: Approved

scale performs poorly in practice because he is missing some "minor" ability, such as the ability to get along with the people he has to work with, a factor that is always difficult to determine beforehand even when it is included in the test.

These are theoretical conjectures. However, as pointed out by Weinberg [34], nobody has put together a test with any measurable validity. The closest thing we have to a validation of a programmer's aptitude test are studies which show that people who score high on the tests are better students in the ensuing programmer's training. When it gets to the specificities of the actual programming, however, there is often no correlation - or even a slight negative correlation - between test scores and rated performance.

This sorry picture is not unique to programming. Intelligence tests generally - such as the famous IQ tests - are able to predict success in school-type situations - and in nothing else. In fact, as one wit puts it, intelligence tests measure the ability to take tests. There is reason to believe that this appraisal is not far from the truth. For example, in IQ tests, speed is very much a factor, as it is in school situations generally. But, in the office, the difference between one hour and one hour and ten minutes is only ten minutes - not the difference between an A and a C.

Another typical distortion of intelligence tests is in the emphasis they place on short-term - rather than long-term - memory. They could hardly be expected to do otherwise, given the constraints under which they must be administered. In "real life" it is selective memory which pays - the ability to forget the unimportant and retain the important over long periods.

Lastly, one observes that the very existence of aptitude tests can distort the results of the tests. In big cities, an eager parent can take his child to a special tutor who guarantees to raise his IQ by so many points for so many dollars. The same techniques are used by certain programming schools to help their graduates get jobs with those companies that rely heavily on aptitude testing. It would appear that in such cases the existence of aptitude tests can restrain the formation of the developers and can be detrimental to the quality of the developers. A shift in values is observed, as emphasis is changed to passing the test and not in attaining the objectives that the test is aiming at. So, no matter how much we would like to have a magic wand which would point out the best programmer prospects, it seems we are just going to have to learn to do without [34].

banking or avionics, while experts analyse the deep structure of the program, that is the algorithmic structure providing its solution [41][39][40] according to plans and rules. However, Soloway and Ehrlich [40] have demonstrated that more experienced programmers are no better than novices at comprehending a program whose algorithmic structure violates these plans.

On the subject matter, Weinberg [34] observes that just like there is not a best method of solving problem there is also not a best set of characteristics of a software developer. Instead there are different attributes that would appear to be of higher importance at different stages of the software development. For example, when making the overall design of a program, what we most need is the ability to create new programming ideas and to screen them on the basis of broad principles. If the design is to be developed by object-oriented techniques the ability to capture key abstractions and to reject others less useful is essential. Thus the programmer lacking in either ability - creativity and selectivity - will be handicapped in attempting to design programs. When coding, however, different abilities come to the fore. Instead of the broad, sweeping mind, the mind which is clever at small things now excels. Then, when testing, the programmer must switch to yet another group of gifts - particularly the eye for wholeness. Weinberg calls this *gestalt*, which is a general feeling that something is out of place without any particular localisation. Documentation as well requires some special skills. In fact good documenters tend to be rare. In the first place it should be clear that if the program is not well written, there is not much that documentation can do to resuscitate it. Since programmers usually document their own productions - especially for small embedded systems developments - the good documenter has to be a good programmer to begin with - and then he must add the capacity to express himself verbally and graphically.

The above discussion stresses the need for different skills and aptitudes in developing software. This naturally tends to call for measurement or testing of those skills in the selection process of software developers. A note of caution is necessary when aptitude tests are concerned. In the first place, programming is a diverse activity, and any test that gives a single "grade" is hardly adequate to measure the aptitudes required [34]. Some tests have been designed to give multiple scores, but even if these had individually valid measures. They would be difficult to apply sensibly, and the chances are that the average personnel manager would simply avoid such difficulty. Moreover, just because of the multidimensionality, someone who is sorely deficient in one area may turn out to be outstanding in another - provided that the opportunity to use his strong points exists. On the other hand, sometimes a person who scores high on every possible

leading to the answer should be correct as well. The goal of formal logic is to represent problem solving as a formal relationship between beliefs and conclusions - to find procedures that can guarantee correct solutions; in other words, to find the legal mode of reasoning. One of the simplest forms of formal logic is the propositional logic invented by Boole (Boolean logic).

Always on the subject of problem-solving, one is often confronted with the difficult decision of making assumptions. Making assumptions is often regarded as not being wise as it can often lead to wrong conclusions. Overlooking a factor in a problem is just one special case of assumptions leading us astray. We assume that a certain factor is not important - probably without even thinking about it in any conscious manner. We are led similarly astray by assuming that a certain factor is important, when it has no significance for the problem at hand. People who spend much time debugging other people's programs soon learn not to listen to explanations before tackling the problem, for these explanations tend to put the listener on the same false track of assumptions that prevented the speaker from finding the bug for himself.

Could we say, then, that the first rule of problem solving is "don't make assumptions"? We could say that, but we would be precisely wrong. If we are to be successful at solving problems, we must make assumptions. If we really face each problem as entirely new, it would be impossible to improve out problem-solving performance. Intelligent behaviour, then, does not consist in eschewing assumptions, but in being sufficiently flexible to manipulate assumptions as the occasions demands. In other words, being intelligent, in the context of problem solving, is not having some magic formula which one can apply to every problem. It is, rather, having a number of "formulas" and not being so much attached with one that it cannot be dropped for another [34].

7.2 Appraising Programming Intelligence

Differences among programmers in skill and performance remain one of the most important determinants of software productivity [37]. Therefore, it comes to no surprise that there is a considerable amount of research [38][39][40] to determine what are the cognitive abilities that make a good programmer.

One aspect of software development that cognitive science studies has brought up is the difference in approaches of novices and experienced programmers towards programming. Novices comprehend a program through its surface structure [41], that is, the particular application area of the program such as

measure, we can easily fall into believing that the worst programmers are the best - because they work so hard at it.

Problem-avoiding behaviour, then, is intelligent behaviour at its highest, although not very intelligent if one is trying to attract the eye of a poorly trained manager. It will always be difficult to appreciate how much trouble we are not having, just as it will always be difficult to appreciate a really good job for problem solving. Once the problem solution has been shown, it is easy to forge. the puzzlement that existed before it was solved. For one thing, one of the most common reasons for problem difficulty is the overlooking of some factor. Once we have discovered or been told that this factor is significant, working out the solution is trivial. If we present the problem to someone else, we will usually present him with that factor, which immediately solves nine-tenths of the problem for him. He cannot imagine why we had such trouble, and soon we begin to wonder ourselves.

Caroll, Thomas and Malhotra [35] describe the problem in moving from the idea for a system to its final implementation as one that involves the transformation of a linearly derived sequence of desired components into a hierarchical arrangement of functions or data transformations. Once the requirements have been delineated, they must be organised so that the inherent structure of the problem becomes visible. The next step is to construct a solution structure that matches the problem structure. To the extent that these structures are logically organised and matched, the system will possess a structural integrity that will expedite the implementation.

Two schools of reasoning approaches are generally recognised. Cognitive science and logic [36].

Cognitive science is the experimental science whose subject is human problem solving. The focus of the cognitive science is on the human mechanisms for problem solving. The ultimate goal of this field of study is an understanding of the biological basis of cognition. The basic hypothesis is that human cognition is information processing; according to this hypothesis, a cognitive process can be seen as a sequence of internal states successively transformed by a series of information processes. An associated hypothesis is that information is stored in several memories having different capacities and accessing characteristics.

Logic is the science of human problem solving "as it should be". In other words, logicians have always been concerned more with how people should reason than with how they do reason. Getting the right answer is not enough - the argument

7 Psychological Factors in the Development of Software

The fact that the software development activity involves decision making, creativity and heuristics at the human level gives rise to strong psychological influences in the development process itself. When in addition one considers that software is often developed by large teams, the issue of human psychology becomes even more apparent.

The subject of psychological factors in this context, directly addresses the issue of human interaction with software development. The issue of "people", though often given less attention (as compared to the other two P's of software development: Process and Problem), could be a lot more crucial to the development. In fact, if we recognise that processes surrounds and emanates from the people, it is logical to think that be focusing on the people, we can derive answers for the many questions we have about the other two P's. Bach [47] identifies the human processor, rather that the process, as being the central issue. He claims that the process is useful but is not central to successful software projects. De Marco [44] brings this concept further, suggesting that people should be managed as a capital asset.

Two major spheres of influence of psychology in the development of software are recognised. The first is concerned with the way the human mind reaches for solutions when confronted with problems and choices. It affects directly the development processes and methodologies. The second area where psychology has a strong bearing is in determining and testing the cognitive abilities that are required to perform different programming tasks. Those two areas are briefly discussed below.

7.1 Problem-Solving

As a first reasoning step to problem-solving or alternatively a step prior to problem-solving, should be problem-avoiding. As Weinberg [34] observes, a programmer who avoids a problem altogether is more effective than one who brings it upon himself, whether or not he ultimately solves it - Weinberg is generally recognised as the one that formally started the psychology of computer programming. This statement from Weinberg should seem quite obvious, but however, abstract ideas about intelligence rarely fall into accords with our beliefs about concrete situations. Lacking any objective measure, we often judge how difficult a program is by how hard a programmer works on it. Using this sort of

outstanding order and pattern strategies (like the periodic table or the classification of living species) in that it is intrinsic. That is its main advantage over functional strategies which tend to force a set approach to design, an approach that is often in conflict with the natural perception of the human mind.

entities (such as hardware components) and their controlling objects in the system. Besides, hardware factors in a design can be represented by hardware objects. When each instrument is considered as an object which conceal hardware details, the instrument design and the hardware design can go in parallel.

A note of caution about reusable components is necessary. To be completely self-contained, a component should not use other components which are externally defined. However, this is contrary to good practice which suggests that existing components should be reused. Thus, some balance must be struck between the advantages of reusing components and the loss of component adaptability that this entails.

The disadvantage of object-oriented design is that the identification of appropriate system objects is difficult. Techniques for object-identification are available but none of them lead to a conform solution. Instead different solutions can be obtained each time the techniques are applied. One of those techniques is suggested by Abott [32] and constitutes in describing the system in a natural language and then in selecting the objects from the nouns and the operations from the verbs. However, the inherent ambiguity of natural language descriptions means that this should only be considered as a guide to the designer and other domain information must be used for object identification. In addition, object-oriented methods are relatively immature and are changing rapidly and so are not nearly as widely used as methods based on functional decomposition.

As far as language considerations are concerned, Sommerville [8] claims that object-oriented design is a design strategy and is not, strictly speaking, dependent on any specific implementation language. As such, it can be implemented in plain high-level languages like Pascal or C. Examples of object-oriented languages (OOLs) are C++, ADA, Smalltalk and Eiffel.

Finally we note that object-orientation, in the broader meaning of the term, is more than a new design technique, and is certainly much more than a new software language. It is a natural way of perceiving the elements around us, both material elements and conceptual elements. This aspect of object-orientation is well expressed by Williams [33] who says that object-oriented software development is a response to a need - a need for less confusion, for less repetitious work - and is a natural outgrowth of the human's mind tendency to think in visual metaphors. That is why there is such emphasis on graphical representation in the field. Object-orientation has one thing in common to all

Object-oriented design is based on information hiding. It differs from the functional approach to design in that it views a software system as a set of interacting objects (entities), with their own private state, rather than a set of functions.

In the design process, the lowest system objects and the high-level objects are often identified first. This situation is normal in object-oriented design which is rarely a top-down process. Other objects are often defined to fill the gap in between, out of more information about the data and process.

An object-oriented design is based on entities (objects) which have a hidden state and operations on that state. The design is expressed in terms of services requested and provided by interacting objects.

The characteristics of an object-oriented design are:

- shared data areas are eliminated. Objects communicate by exchanging messages rather than by sharing variables. This reduces overall system coupling and there is no possibility of unexpected modifications to shared information.

- Objects are independent entities that may readily be changed because all state and representation is held within the object itself.

- Objects may be distributed and may execute either sequentially or in parallel.

Perhaps the principal advantage of object-oriented design is that the nature of objects leads to the creation of loosely coupled systems. It is fundamental to object-oriented design that the representation of an object is concealed within that object and is not visible to external components. The system does not have a shared state and any object can be replaced by another object with the same interface. To summarise, an object-oriented approach to design has the following advantages:

- The developed system should be easier to maintain as the objects are independent.

- Objects are appropriate reusable components.

- For more classes of system, there is a clear mapping between real-world

The first stage of function-oriented design should be to develop a system data flow diagram. These diagrams should not normally include control information but should document data transformations.

Structure charts are a graphical means of showing the hierarchical component structure of a system. They show how elements of a data flow diagram can be realized as a hierarchy of program units.

For loosely coupled and highly cohesive units, each unit should be dealing with one of the following four types of data flow:

- Input. The program unit is responsible for accepting data from a unit at a lower level in the structure chart and passing that data on to a higher level unit in some modified form.

- Output. The program unit is responsible for accepting data from a higher level unit and passing it to a lower level.

- Transform. A program unit accepts data from a higher level unit, transforms that data and passes it back to that unit.

- Coordinate. A unit is responsible for controlling and managing other units.

The first step in converting a data flow diagram to a structure chart is to identify the highest level input and output units. These units are those concerned with passing data up and down the hierarchy but are furthest removed from physical input and output. Identifying the highest level input and output transforms depends on the skill and experience of the system designer.

Data dictionaries are useful in maintaining system specifications, but are also equally useful in the design process. Each identified entity in the diagram should have a data dictionary entry giving information about its type, its function and perhaps, a rationale for its inclusion.

By comparison with object-oriented design, the design components in this approach are cohesive around a function whereas object oriented cohesion is around some abstract data entity.

6.3 Object-Oriented Design

system decomposition and to ensure that the design adequately captures the system specification. Design is a creative process which requires experience and some flair on the part of the designer.

Finally the design process should not be viewed as a separate entity. One should always keep in perspective the interaction of the design process with the rest of the development phases. One way to ensure this interaction is through proper documentation. Accordingly, the configuration management must be such that it should be easy to incorporate changes made to the design in all design documents. If this is not the case, changes made to a design description may not be included in all related specifications. The design documentation may become inconsistent. Later changes are more difficult to make (the component is less adaptable) because the modifier cannot rely on the consistency of the design documentation

6.2 Function-Oriented Design

A function-oriented design strategy relies on decomposing the system into a set of interacting function with a centralised system state shared by those functions [8]. Functions may also maintain local state information but only for the duration of their execution.

Function-oriented design conceals the details of an algorithm in a function but system state information is not hidden. This can cause problem because a function can change the state in a way which other functions do not expect. Changes to a function and the way in which it uses the system state may cause unanticipated interactions with other functions.

In structured analysis and design, there is no explicit way of showing the subject matters [31]. Processes are processes and modules are modules. However, the layering concept is so powerful that many project teams simply show the various subject matters as processes at the highest level: a copier process, a user interface process and a device control interface. It can be difficult, though, to define precisely the flows between these processes because the point of view changes. From the point of view of the copier, a flow may correspond to the number of copies, but from the point of view of the user interface, we have button pushes from the number pad on the copier.

A functional approach to design is therefore most successful when the amount of system state information is minimised and information sharing is explicit.

Table 2 shows an informal classification of systems, based on properties that show up at the requirements level. It pictures the embedded system in the context of software development in general and provides an easy means of comparison. Requirements for "support systems" are generally much less definite than requirements for applications systems. And while the performance requirements for embedded systems may be couched in absolutes, the performance requirements for support systems will be relative to resources and resource utilisation, and the performance requirements for data-processing systems will be relative to load, resources, and psychological factors. The most complex systems, such as nationwide airline-reservation systems, should probably be viewed as having subsystems of all three types.

Table 2: A requirements-level system classification [1]

TYPE	CHARACTERISTICS	EXAMPLES
embedded system	special-purpose (application) absolute performance requirements	industrial process-control system flight-guidance system
data-processing system	special-purpose (application) relative performance requirements	batch business program on-line database system
support system	general-purpose relative performance requirements	operating system software development tool

2.2 Peculiarities of Embedded Systems

Although much of the development techniques used in traditional software development hold true for embedded systems projects, microcontroller software often exhibits differences from the typical data processing programs, which constitute a significant portion of traditional software systems. Those aspects particular to embedded systems are treated below.

Microcontroller software frequently represents only a small part of a system composed of several electrical, mechanical, and other components. The end user need not (and often is not) aware that a microcontroller is integrated in the system.

industrial controller versus a train".

Among the most prominent fields of application of embedded systems technology are :

- the automotive industry
- the defense industry
- household products
- Electronic displays

The automotive market is the most important single driving force in the microcontroller market [3], especially at it's high end. Several microcontroller families were developed specifically for automotive applications and were subsequently modified to serve other embedded applications [4]. Table 1 below demonstrates the growing importance of embedded systems technology in the automotive industry. Table 1 also shows the predicted data for the years to come.

Table 1: Average Semiconductor Content per Passenger Automobile (in Dollars) [5]

Year	'90	'91	'92	'93	'94	'95	'96	'97	'98	'99	'00
Semi-conductor Content ($)	595	634	712	905	1066	1237	1339	1410	1574	1852	2126

Automotive is not the only market that is growing. Consumer electronics is also a booming business. In the average North American's home there are 35 microcontrollers. By the year 2000 - that number is expected to grow to 240 [4].

Although the present study focuses on the wider applications of embedded systems which make use of small programmable devices such as microcontrollers, embedded systems in the broader meaning of the term can also include larger intelligent systems such as a personal computer [6]. When the personal computer is used by itself as a word processor, it would probably not be considered an embedded system. If however it was dedicated as the user interface, data processor and controller for a weather station, lighting, heating, ventilation, air condition, fire suppression and burglar alarm, then it could be considered an embedded system. It would be an integral part of a larger home automation system.

2 Defining the Embedded Systems Concept

2.1 Definition

First of all let us start by answering the question: what makes a system "embedded" ?. The term "embedded" was popularized by the U.S. Department of Defense [1] in conjunction with its common languages (ADA) development project. "Embedded" refers to the fact that these systems are embedded in larger systems whose primary purposes are not computation, but this is actually true of any computer system. In contrast, the common concept that unites the systems we choose to call "embedded" is *process control*: providing continual feedback to an unintelligent environment.

Often real-time embedded systems are controllers, being part of an electro-mechanical application. Such systems are reactive; they are simulated by the environment with signals, and return -as their reaction- signals to the environment. In the past these systems were usually built from analogue and digital hardware components. Today software has become dominant; analogue filters are replaced by digital ones, the anti-shuttering of a switch is not done by a capacitor, but with an interrupt-routine and a timer-subsystem. The goals of these replacements are usually to:

- minimise the number of components
- make the design more reliable
- make the system more maintainable
- reduce the production cost of the whole system

An "embedded system", then, is an electronic product that incorporates an intelligent programmable device, like a microcontroller. Most of these products do not look like computers. Often there is neither a display nor a keyboard. Virtually all the electronic products that now surround us is based on a tiny computer hidden inside the works. That covers products ranging from the cellular phone, the television remote control, the Automatic Teller Machine (ATM) machine to even innocuous products like the electronic greeting card. Even the mouse connected to the PC has a tiny embedded processor inside.

As the general manager of Intel's Embedded processor Operation Tom Franz [2], puts it, "unlike the PC industry, it's (embedded systems) a very broad industry. It's so diverse - it's an industry where you have applications such as terminals versus

[49] Poston R.M., "Testing Tools Combine Best of New and Old", IEEE Software, March 1995, pp. 122-127

1.5.5 Other Documents

a. SDE 102, A Literature Survey on Software Development

1.6 Assumptions

It is assumed that the reader is familiar with the major software development processes and associated issues as exposed in the Literature Survey on Software Development Processes, SDE 102.

1.7 ISO 9001 Requirements Traceability

- ISO 9001 Clause 4.4.4 - Design Input

[35] Hill I.D., Meek B.L., "Programming Language Standardisation", Ellis Horwood Ltd., U.K, 1980

[36] Friedman P., "Computer Programs in Basic", Prentice-Hall Inc., U.S.A, 1981

[37] Schneider G.M., Weigner S.W., Perlman D.M., "An Introduction to Programming and Problem Solving with Pascal", Second Edition, John Wiley and Sons Inc., 1982

[38] Ghezzi C., Jazayeri M., "Programming Language Concepts". John Wiley and Sons Inc., 1982

[39] Barnes J.G.P., "Programming in ADA", Addison-Wesley Publishing Co. Ltd., U.K., 1982

[40] Brodie L., "Starting FORTH", Prentice-Hall, Inc., U.S.A, 1981

[41] Fuori W.M., "Introduction to American National Standard Cobol", McGraw-Hill Book Company, 1975

[42] Barnes J.G.P, "RTL/2 Design and Philosophy", Heyden and Sons Ltd., U.K, 1976

[43] Christian K., "A guide to Modula-2", Springer-Verlag Inc., USA, 1986

[44] Blasciak A.J., Neuder D.L., Berger A.S., "Software Performance Analysis of Real-Time Embedded Systems", Hewlett-Packard Journal, Vol. 44, Part 2, April 1993, pp 107-115

[45] Honka H., Kattilakoski M., "A Simulation-Based System for Testing Real-Time Embedded Software in the Host Environment". Microprocessor and Microprogramming, Vol. 32. Part 1-5, pp 127-134

[46] Williams T., "Integrated Tool Environment Targets Real-Time Development", Computer Design, November 1992, pp 44-46

[47] Williams T., "Automation Improves Speed and Accuracy of Software Testing", Computer Design, July 1992, pp 57-64

[48] Koch C., "Hitachi Microcontroller Seminar", South Africa, May 1996

Journal, Ocober 1994

[23] Harel D., "A Visual Approach to Complex Systems", Science of Computer Programming, 1987

[24] Booch G., "Object-Oriented Analysis and Design", Second Edition, The Benjamin/Cummings Publishing Company, Inc., USA, 1994

[25] Wolf W. H., "Hardware-Software Co-Design of Embedded Systems", Proceedings of the IEEE, Vol. 82, No. 7, July 1994, pp 967-989

[26] Gupta R.J., Coelho C.N., "Program Implementation Schemes for Hardware-Software Systems", Vol 27, Part 1, May 1991, pp 48-55

[27] Chiodo M., Giusto P., Jurecska A., Marelli M., Hsieh H.C., Sangiovanni-Vincentelli A., Lavagno L., "Hardware-Software Codesign of Embedded Systems", IEEE Micro, Vol 14, Part 26-36, August 1994, pp 26-36

[28] Clark R.G., "The Design and Development of Embedded Ada Systems", Software Engineering Journal, Vol 5., Part 3, May 1990, pp 175-184

[29] Stoyenko A.D., Welch L.R., Chao Cheng B., "Response Time Prediction in Object-Based, Parallel Embedded Systems", Microprocessor and Microprogramming, Vol 40, Part 2-3, April 1994, pp 135-150

[30] Boehm B. W., "Software Engineering Economics", Englewood Cliffs NJ: Prentice-Hall, USA, 1981

[31] Yeun C.K., "Essential Concepts of Computer Architecture", Addison-Wesley Publishing Co. Ltd., Singapore, 1989

[32] Van de Goor A.J., "Computer Architecture and Design", Addison-Wesley Publishing Co. Ltd., U.K, 1989

[33] De Carlini U. and Vilano U., Transputers and parallel architectures, Ellis Horwood Limited, 1991, UK

[34] Cragon G. H., "The Elements of Single-Chip Microcomputer Architecture", IEEE Computer, Vol. 13, part 32-41, October 1980, pp 27-30

[10] Smith D., Wood K., "Engineering Quality Software", Second Edition, Elsevier Applied Science Publishers Ltd., UK, 1989

[11] Sommerville I., "Software Engineering", Fourth Edition, Adison-Wesley, USA,

[12] Real-time FAQ, "What Exactly is Meant by Real-Time", real-time?-state.edu/hypertext/faq/usenet/realtime-computing/faq/faq-doc-4.html

[13] Laplante P.A., "Real-Time Systems Design and Analysis, An Engineer's Handbook", IEEE Press, USA, 1993

[14] Voas J.M., Miller K.W., "Software Testability: The New Verification", IEEE Software, May 1995, pp 17-28

[15] Everett W.W., Honiden S., "Reliability and Safety of Real-Time Systems", IEEE Software, May 1995, pp 13-16

[16] De Lemos R., Saeed A., Anderson T., "Analyzing Safety Requirements for Process-Control Systems", IEEE Software, May 1995, pp 42-52

[17] Goldsack S.J., Finkelstein A.C.W., "Requirements Engineering for Real-Time Systems", Software Engineering Journal, Vol 6, Part 3, May 1991. pp 101-115

[18] Sagoo J.S., Holding D.J., "The Specification and Design of Hard Real-Time systems Using Timed and Temporal Petri Nets", Microprocessor and Microprogramming, Vol 30, Part 1-5, August 1990, pp 389-396

[19] Laffey T.J., Cox P.A., Schmidt J.L., Kao S.M., Read J.Y., "Real-Time Knowledge-Based Systems", AI Magazine, Vol 9(1), pp 27-45

[20] Strosnider J.K., Paul C.J., "A Structured View of Real-Time Probklem Solving", Vol 15., Part 2, Summe 1994, pp 45-66

[21] Zhu J., Lewis T.G., Jackson W., Wilson R.L., "Scheduling in Hard Real-Time Applications", IEEE Software, May 1995

[22] Drusinsky D., "Extended State Diagrams and Reactive Systems", Dr. Dobb's

1.5.2 Standards

a. SEAL QMS Document Creation Template, QS 002, Revision 0.01, 5 April 1994

b. SEAL QMS Document Layout, Presentation and Typesetting Guide, QS 003, Revision 0.01, 1 April 1994

1.5.3 Procedures

None

1.5.4 Guidelines

[1] Zave P., "An Operational Approach to Requirements Specification for Embedded Systems", IEEE Transactions on Software Engineering, May 1982, pp 250-269

[2] Franz T. In "Intel Will Follow AMD With the 486 Embedded Market Entries", http://www.elibrary.com/

[3] Adamson M., "Small Real-Time Design", Sigma Press, John Wiley and Sons, U.K, 1990

[4] Hersch R., "Embedded Systems Development Information", Orion Instruments, http://www.oritools.com/info/

[5] WSTS & ICE -1994 In Hersch R., "Embedded Systems Development Information", Orion Instruments, http://www.oritools.com/info/

[6] Nelson M., Real-Time Controls, http://wco.com/~mnelson/a2.html

[7] Rausscher T.G., Ott L.M., "Software Development and Management for Microprocessor-based Systems", Prentice-Hall, Inc., USA, 1987

[8] Booker A., McKeeman J., "Design Considerations for RISC Microprocessor In Realtime Embedded Systems", Computer Design, August 1993, pp 71-72

[9] Tzou S., Lim J., Meno J., Palmer D., "A distributed Development Environment for Embedded Software", Software-Practice and Experience, Vol 23(11), November 1993, pp 1235-1248

- MALPAS : MALvern Programming Analysis
- OTP : One-Time Programmable
- PAISLey : Process-Oriented Applicative and Interpretable Specification Language
- PE : Processing Element
- PES : Programmable Electronic System
- RAM : (Read/write) Random Access Memory
- RISC : Reduced Instruction Set Computer
- ROM : Read Only Memory
- SISC : Specific Instruction Set Computer
- UART : Universal Asynchronous Receiver/Transmitter
- VDU : Video Display Unit
- VSLI : Very Large Scale Integration

1.4 Audience

The audience for this document comprise the various stakeholders of the SEAL, including:

- Full-time staff members of SEAL
- All full-time and part-time post-graduate students associated with SEAL
- Members of the SEAL Management Board
- Head of the Department, Electrical Engineering
- Individuals who will perform internal and external surveillance audits of the SEAL Quality Management System
- Engineering Manager, Meissner
- Product Developer
- Product Manager

1.5 Applicable Documents

1.5.1 Specifications

None

1 Scope

1.1 Introduction

Embedded systems are growing in popularity. New applications of embedded systems are constantly being investigated. Accordingly they constitute a large part of software development projects. Many of the practices uncovered in the Literature on Software Development, SDE 102, are applicable to the development of embedded systems. However, the engineering nature of their applications, combined with several other factors, make embedded systems projects different from data processing applications software projects. Failure to recognise those differences will invariably lead to poor design performances.

1.2 Purpose

This document captures the main features of embedded systems and answers the frequent questions on the subject. It underlines the peculiarities of embedded systems and probes the new development areas in that field.

1.3 Definitions

- A/D : Analogue-to-Digital
- ADO : Abstract Data Object
- ADT : Abstract Data Type
- AI : Artificial Intelligence
- ATM : Automatic Teller Machine
- ASIP : Application Specific Processor
- CFSM : Codesign Finite State Machine
- CISC : Complex Instruction Set Computer
- CSP : Communicating Sequential Process
- DSP : Digital Signal Processing
- EMI : Electromagnetic Interference
- EPROM : Electrical Programmable ROM
- EEPROM : Electrically Erasable and Programmable ROM
- FSM : Finite State Machine
- ICE : In-Circuit Emmulator
- LCD : Liquid Crystal Device

Change History

Configuration Control

Project:	SDES QMS
Title:	Embedded Systems - A Literature Survey
Doc. Reference:	SDE103WP.100
Created by:	H. Bapoo
Creation Date:	29 May 1996

Document History

Version	Date	Status	Who	Saved as:
0.01	96/04/24	Draft	H.Bapoo	:\1995-13\TP\SDE03WP.001
0.02	96/05/29	Draft	H.Bapoo	:\1995-13\TP\SDE03WP.002
0.03	96/08/04	Draft	H.Bapoo	:\1995-13\TP\SDE03WP.002
1.00	96/08/25	Approved	H.Bapoo	:\1995-13\TP\SDE03WP.002

Revision History

Version	Date	Changes
0.01	95/04/24	New Document
0.02	96/05/29	Change of document format to comply with SEAL Document Presentation Guide QS 003
0.03	96/08/04	Document restructured and additional information added
1.00	96/08/25	Update to revision number and status following Board Approval

Management Authorisation

Version	Date	Status	Management Board Minute Reference
1.00	93/08/25	Approved	Section 2.3 of SDE 592

Change Forecast

This document will be updated each time the literature survey generates new materials that are pertinent to the development of embedded systems products.

6 Planning Embedded Systems Projects 32
6.1 Development Strategies 32
6.1.1 Formal Requirements Specifications Methods 32
6.1.2 Dealing with Timing Issues 33
6.1.3 Hardware-Software Co-Design Techniques 36
6.1.4 Object-Oriented Approach 40
6.2 Applying Software Metrics - A Case Study 43
6.3 Pitfalls to Avoid When Planning 44

7 About Programmable Devices 46
7.1 The Devices 46
7.2 The Microcontroller Market 50
7.2.1 The Market Size and Product Range Distribution 50
7.2.1 The Players 52

8 Language Issues 31
8.1 Factors to Consider When Choosing a Language 54
8.2 High Level Languages v/s Assembly Languages 55
8.1 Overview of some High-Level Languages In the Context of Embedded Systems 56

9 The Development Tools 60

Appendix
Appendix A:Example of Applying COCOMO to an Embedded System Project 63

b. *Two channels with self-test.* Each channel is subjected to a periodic self-test initiated with the software. The most realistic type of test involves temporary disabling of a channel's output and then injecting a simulated signal to the input. In this way the function can be verified. If a single channel fails the self-test it can be declared unhealthy. Depending on the safety philosophy, the same can either revert to single-channel operation and inform the operator that there is reduced integrity or suspend processing. The self-test has to be thorough since error states which escapes detection will cause system failures.

c. *Three channels with two-out-of-three voting.* An expensive alternative which compares three solutions and can permit one out of the three to become unhealthy. This solution can increase both safety and reliability due to the two-out-of-three redundancy which applies to both hazardous and spurious failure modes.

One approach to the common cause failure problem involves software diversity, which is a particular form of redundancy involving the design and coding of separate software for each of the replicated channels. This is sometimes called *N version programming* [13] where two or even three separate designs are implemented. These processors are coded to the same specifications, but by different teams. It is therefore highly unlikely that more than one of the systems can lock up under the same circumstances. Where there are two or more channels, circuitry and/or software has to be designed in order to decide which is the healthy channel in the event of an error. With three channels, a two-out-of-three voting arrangement can be employed.

Since diverse software is obtained by carrying out separate design and coding activities for the replicated channels, this will involve one or more of the following features:

- Different algorithms
- Different storage media
- Different CPU architectures
- Different supply voltages
- Different CPU clock rates
- Different languages

Software diversity is extremely expensive and is unlikely to be applied except in highly hazardous applications such as nuclear or aerospace equipment. It is not

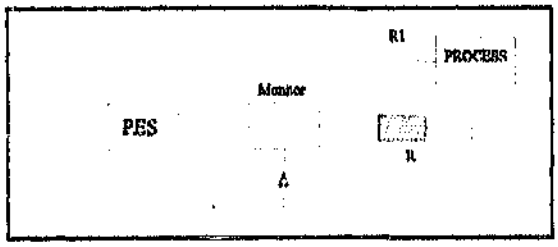


Figure 1(c): Stand-by hard-wired control

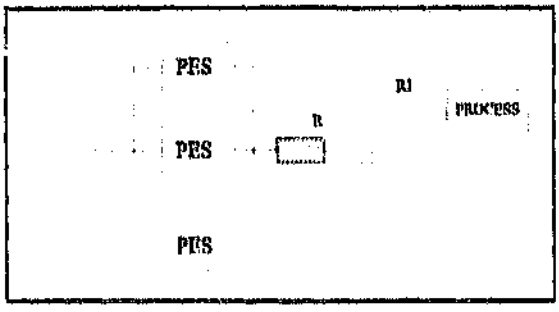


Figure 1(d) : Software Diversion method

An approach to modelling software reliability involves an attempt to identify measurable complexity and size parameters (known as metrics) which can be correlated with failure rates. The problem in this case is that the number of complex interacting variables which govern software quality is probably greater than can be realistically modelled [10]. Unlike hardware there is no simple repeatable model involving the failure rate of specific elements.

Using redundancy is a common way of improving reliability, as we have already shown. When using redundancy the following additional factors should be considered [10]:

a. *Two channels with a comparator.* If the two channels do not agree the processing cannot proceed since there is no determination of which channel is healthy. An alarm or shut-down condition should then be initiated. In this case the overall reliability may be less than for a single channel since twice the number of failures will be likely. On the other hand, safety is enhanced since, for an hazardous failure mode, both channels need to fail.

solenoid whose contact causes some safety action. It is possible to imagine the PES causing an unwanted operation or even failing to operate when required. Due to uncertainty associated with software failures this solution is seldom favoured.

b. The system retains hard-wired control. Figure 1(b) also shows this arrangement whereby the PES carries out functions whilst the solenoid is still operated from the input. Currently this solution is usually favoured.

c. Figure 1(c) shows an alternative application of the override principle. In this case a hard wired monitor examines the outputs and disconnects the PES from the solenoid in the event of predetermined undesirable combinations of output.

d. Diverse software is employed whereby duplicated or triplicated systems are provided such that each system is separately designed and programmed. Some measure of protection against software failure is thus obtained and the outputs from the diverse systems can be compared and voted. This is shown in figure 1(d).



Figure 1(a): Software controlling the safety-related function directly

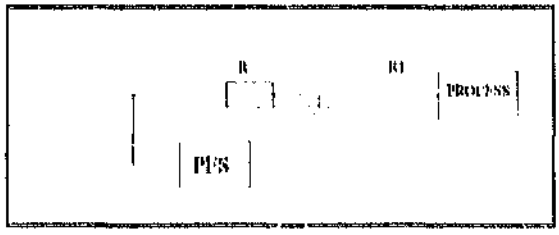


Figure 1(b): The system retains hard-wired control

4 Reliability

In section 2.2 above, the importance to build reliability in domestic products and embedded systems in general was raised. Capital at stake through large volumes of the product and safety considerations make reliability an even more important issue in embedded systems than in data processing applications [3]. It may be annoying to receive an electricity bill for ten million rands, but it is more annoying if the engineers lose control of the power station, or if they lose control of their cars on their way to work and cause a collision. In that respect, small systems can be used in applications where safety is critical, just as with large systems. It is the software designer's responsibility to ensure that the software copes safely with all that may be thrown at it.

Using software in potentially hazardous situations leads to two main difficulties [10]:

a. Due to the complexity of software failure modes the possibility of hazardous failures is greater.
b. Since the use of software makes failures difficult to predict, it is difficult to perceive if the integrity of a system is adequate.

A frequent misconception is that the elimination of errors during design is the sole factor in achieving quality software. This is too simplistic an assumption. In practice all software is likely to contain residual faults, albeit at very low levels, after extensive debug and quality activities.

A number of design features involving both hardware circuitry and software techniques can contribute substantially to the system reliability. Some of these features prevent faults from occurring in the first place and others improve the fault tolerance of the system so as to render them non-critical.

4.1 Reliability Design Philosophies

Smith and Wood identify [10] four basic design philosophies which should be considered when incorporating software elements into a safety system.

a. The software directly controls the safety-related function. Figure 1(a) shows a contact providing an input to a programmable electronic system (PES) which, in turn, provides an output. The PES output operates a

exceptional sensor value is detected; Control systems are systems which continuously control hardware actuators depending on the value of associated sensors. These system obviously have a great deal in common and differ only in the way in which systems actuators are initiated. Often, in embedded systems, they will be found on the same platform.

The need to respond to stimuli which occur at different times means that the architecture of a real-time system must be organised so that control can be transferred to the appropriate program component as soon as a stimulus is received. This is normally achieved by designing the system as a set of concurrent cooperating process. Part of the real-time system (sometimes called the real-time executive) [11] is dedicated to managing these processes.

Because the processes in a real-time system must cooperate, the real-time system designer must coordinate the communications between processes. Process coordination mechanisms ensure mutual exclusion to shared resources. In real-time embedded systems which involve data acquisition and processing, the execution speeds and periods of the acquisition process and the processing process may be out of step. Sometimes, when significant processing is required, the data acquisition will go faster than the data processing; at other times, when only simple computations need be carried out, the processing will be faster than the data acquisition. To smooth out the speed differences, most data acquisition systems synchronise the two events using some kind of buffer mechanism or algorithm.

to synchronise a television picture waveform with a tolerance of less than 1 microsecond, is operating in real ... In terms of embedded systems, operating in real time can mean response times ranging from fractions of a second to several minutes, depending on the applications. A good example is a robot that has to pick up something from a conveyor belt. The piece is moving, and the robot has a small window to pick up the object. If the robot is late, the piece will not be there anymore, and thus the job will have been done incorrectly, even though the robot went to the right place. If the robot is early, the piece will not be there yet and the robot might block it.

Occasionally one sees reference to 'real-time' systems when what is really meant is just 'fast' [12]. As pointed out above, the response times of real-time systems can vary widely. Real-time is not necessarily synonymous with 'fast'; that is, it is not the latency of the response per se that is the issue, but the fact that a bounded latency sufficient to solve the problem at hand is guaranteed by the system. Similarly 'real-time' systems are referred to when what is meant is 'on-line' or "an interactive system with better response time than we used to have". Often this is just marketing tactics.

One will also occasionally see discussions of 'soft' versus 'hard' real-time systems. In many of these discussions, 'hard' real-time means the type of real-time system discussed above, and 'soft' real-time means systems which have reduced constraints on 'lateness' but still must operate very quickly and repeatably. However, the definition is controversial, as some mean by 'hard' and 'soft' the degree of time constraints [12].

In a real-time system, events (or stimuli) often cause the system to move to a different state. For this reason state machine modelling has been widely used as a way of expressing a real-time systems design. The state machine approach is appropriate for the modelling of real-time systems where a particular input causes a thread of actions to be initiated. State machine models are good, language-independent way of representing the design of a real-time system [11]. However the problem with the state machine approach is that the number of possible states increases rapidly. It is therefore necessary to adapt the approach for larger systems so that all states need not be included in a single diagram. One solution is to use thread diagrams.

Embedded monitoring and control systems are an important class of real-time systems which continuously check sensors and take actions depending on the sensor reading. Monitoring systems are systems which take action when some

3 Real-Time Issues

As noted in section 2.2 above, embedded systems are often associated with real-time applications; in fact often those two terms "real-time systems" and "embedded systems" are interchanged. This section looks at characteristics of *real-time* embedded systems.

The expression 'real-time' is widely used, without any clear definition of its meaning. In fact there are several definitions of real-time, most of them contradictory. There does not seem to be full agreement over the terminology and this tends to make the topic controversial. The term 'real-time' is originated in the field of analogue computing, where electronic representations of systems could operate faster or slower than their real counterparts, or in 'real time' [3]. For example, the equations describing the growth of a population of animals could be run many times faster than the animals' breeding cycle, while the dynamics of a chemical reaction could be studied at a much slower speed than the natural events. An electronic analogue which ran at the same speed as the natural event it modelled was described as operating in real time.

Modern real-time digital systems may not be modelling any natural event, so the link between processing speed and the outside world does not exist. Nowadays the term is used to describe systems which deal with data input from the external world as it arrives, and send the appropriate responses to the output at the correct time. This is distinct from batch processing, where input data are saved up for later processing as a group.

For the purpose of this study the term 'real-time' will be used when referring to systems where the correctness of the computations not only depends upon the logical correctness of the computation but also upon the time at which the result is produced. A formal definition is given by Sommerville [11]:

"A real-time system is a software system where the correct functioning of the system depends on the result produced by the systems and the time at which these results are produced."

If the timing constraints of the system are not met, system failure is said to have occurred. This definition covers a wide range of applications. A database management system which allows an operator to modify data directly from a terminal could be called 'real-time'. The systems used for booking airline tickets would fall into this category. Equally, a digital video processing unit, which has

- Impossible to predict software failure modes and rates
- Exhaustive testing is impossible because of time constraints since the number of permutations in software is extremely high
- More susceptible to common-mode failures
- Easier to corrupt data and programs

The "unpredictability" of software has been leading to the introduction of "hardware feedback" [10], that is to say diversity, in order to prevent software failures from leading to hazardous situations. This is in no small measure a result of poor record of software development. This can be largely attributed to the immaturity of software engineering and to the fact that embedded systems has long been viewed (and still is by many organisations) as being too small to warrant the implementation of software engineering practices.

- Difficulty in testing the end-product

- Low cost per part constraint (since they are usually produced in large numbers)

2.3 Advantages and Disadvantages of Using Microcontrollers

From both reliability and operating standpoints there are advantages and disadvantages arising from programmable devices. Smith and Wood [10] summarise those features:

Reliability advantages:

- Less hardware (fewer devices) per circuit due to high levels of integration
- Fewer device types
- Consistent architecture (configuration) leading to a common approach to hardware design
- Easier to support several models in the field with spares
- Simpler to modify or reconfigure
- Provides a running log for investigation
- Allows self-test

Safety advantages:

- Removes human operators from hazardous areas
- Provides sophisticated process interlocks
- Interprets rates of change of process parameters and gives timely warning of potential hazardous conditions
- Provides early warning diagnostics
- Provides centralised displays and graphics on video display units (VDUs)

Reliability and safety disadvantages:

- Difficult to 'inspect' software for inherent faults
- Difficult to impose standard approaches to software design
- Difficult to control software changes
- Testing and validation of high-scale integration devices and their associated software is difficult owing to the high package density and consequent lack of visibility and interface to the functions

cooling strategy. A part that requires forced air cooling - and the bulky, expensive, noisy and unreliable accompanying fan - will be unsuitable for most embedded applications. Often, embedded systems are powered from batteries. To provide for maximum battery life in these applications, the embedded systems architecture should permit speed/power trade-offs and other design techniques to manage power. In the same vein, the designer must provide support for dual power supply system peripherals (3.3V and 5V). Designs that can tolerate some voltage fluctuations are also an advantage.

The principal memory for program storage in many microcontroller systems is read-only memory (ROM), due to its cost savings over other memory types, such as electrically programmable ROM (EPROM) and read/write random access memory (RAM). The cost per system aspect is especially important when considering that microcontroller products often have sales or usage volumes in the tens or hundreds of thousands [7] (such as calculators, automobiles, cash registers, and games). Section 7.2 looks at the microcontroller market.

Finally, embedded systems can be extraordinarily hard to test [1]. The complexity of the system/environment interface is one obstacle, and the fact that these programs often cannot be tested in their operational environments is another. It is not feasible to test flight-guidance software by flying with it, nor to test ballistic-missile-defense software under battle conditions.

To summarise, embedded systems are often characterised by the following factors:

- High reliability concerns
- Real-time constraints
- Stringent resource requirements
- Limited development resources
- Low power consumption
- Large variety of attachments with the environment
- Complex interface with the environment

electrical noise. In spite of this, they are expected to continue to function without operator intervention, whatever mistake the user can make. The stringent resources factor is not limited to the operation of embedded systems. It also extends to the development itself. Unlike other software development, embedded software is cross developed on a separate host rather than on the target hardware on which it must ultimately run [9].This is because most embedded systems do not support the peripheral devices and software tools needed for software development. In addition, the hardware platform that forms the interface between the software and the rest of the system is often not yet developed when the software development starts. Developing software for yet to be developed hardware, makes it even more challenging [9].

The continual demands of an unintelligent environment cause these systems to have relatively rigid and urgent performance requirements, such as real-time response requirements and "fail-safe" reliability requirements. Immediate response to asynchronous external event typifies normal requirements. It seems that this emphasis on performance requirements is what really characterises embedded systems, and causes us to be more aware of their roles in their environments than we are for other types of system.

For example, users of domestic appliances incorporating embedded microcontrollers are generally less tolerant of system bugs than computer professionals, to whom they are all in a day's work. People expect their cars to transport them, their televisions to entertain them and their washing machine to clean their clothes. They will probably only think of the controllers when they go wrong, and then their displeasure will not be limited to the program designer. They might decide never to purchase that manufacturer's product again, taking away thousands of rands worth of business.

The automotive market, which we find in section 2.1 as being the largest consumer of microcontrollers, is demanding. Electronics must operate under extreme temperatures and be able to withstand vibration, shock, and electromagnetic interference (EMI). The electronics must be reliable, because a failure that causes an accident can (and does) result in multi-million dollar lawsuits [3]. Reliability standards are high - but because these electronics also compete in the consumer market -they have a low price tag. Real-time issues and reliability issues are discussed in section 3 and 4 respectively.

Low power is another important requirement for most real-time embedded applications [8]. Power dissipation affects board density, power supply needs, and

The input to most embedded systems comes from some kind of transducer monitoring a process, rather than entered at a keyboard or from punched cards etc. Of course there may still be switch closures to be monitored, magnetic cards to be read, touch screens to be interrogated and a whole range of other devices to be serviced. The variety of attachments is one of the characteristics of these systems. A large mainframe may operate with only serial RS232 serial input/output lines, parallel printer ports and perhaps a network gateway, but a small controller can be expected to take data from half a dozen different kinds of source, each with its own characteristics and speed of operation, and send output in a variety of formats to a range of peripheral devices. Some of the inputs can be digital (e.g. switch closures), others can be analogue, in which case it needs to be converted to digital form using an analogue-to-digital (A/D) converter.

This leads to another factor common to embedded systems which is the complexity of its interface with the environment. The interface between an embedded system and its environment tends to be complex, asynchronous, highly parallel, and distributed. This is another direct result of the "process control" concept, because the environment is likely to consist of a number of objects which interacts with the system and each other in asynchronous parallel [1]. Furthermore, it is probably the complexity of the environment that necessitates computer support in the first place. This characteristic often makes the requirements difficult to specify in a way that is both precise and comprehensible.

A software product such as a word processor or payroll package will usually be used by someone deliberately using a computer to carry out a particular task. They may have attended a training course and be familiar with the software's operation, and they can respond to queries and error messages displayed on the screen in front of them by typing data on the keyboard. If the program enters an unwanted state due to a mistake in entering a command they can recover from the situation, if necessary by resetting the computer and starting again. If all else fails they may be persuaded to read the instruction manual.

Embedded systems do not often enjoy these luxuries. They tend to be subjected to stringent resource requirements. These requirements, are mainly physical, and depend upon the nature on the system being constructed. The stringent requirements arise because embedded systems are often installed in places where their weight, volume or power consumption must be limited (such as satellites), or where temperature, humidity, pressure and other factors cannot be as carefully controlled as in the traditional machine room. They may be located under a car bonnet or close to large electric motors where they will be subject to

systems, Jiang Zhu et al. [21] proposed a new graphical language for the design and scheduling in hard real-time applications. Their graphical language has for advantage to include deadline scheduling algorithms into the same representation. It also integrate both data flow and control flow into the same diagram. Figure 4 shows the basic component of the design diagram.

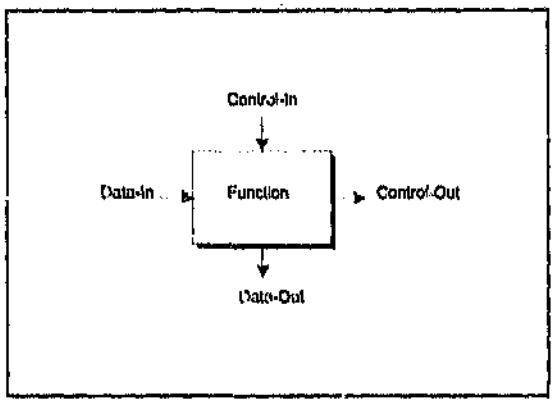


Figure 4: Basic Component of the Hard Real-Time Scheduling Design Diagram

This approach tends to be more function-oriented. Furthermore, converting the design diagram into a task set that will ensure that the design objectives are met is a complex process. As Zhu et al. [21] themselves put it, decomposition of the design into a task set is still fuzzy and is based more on heuristic methods and rules of thumb.

Another approach at capturing the flow of events in embedded systems is documented by Drusinsky [22]. Drusinsky uses Harel diagrams [23] (or extended state diagrams). The need for more elaborate methods of capturing the flow of events in embedded systems arise mainly from the change in complexity of the interface of such systems with the environment. The movement has been from transformational systems to reactive systems. Transformational systems are those invoked when the inputs are ready and the outputs are produced after a computation period [22]; see figure 5(a). Examples are voice compression systems. Reactive systems on the other hand have inputs that are not ready at any given point in time [22]; see figure 5(b). A typical reactive system is an intelligent traffic-light controller which never has all its inputs ready -the inputs arrive in endless and perhaps unexpected sequences.

system. However unlike the object-orientation approach Zave's model is still restricted to the requirements specifications and does not offer a direct path to the design and implementation process.

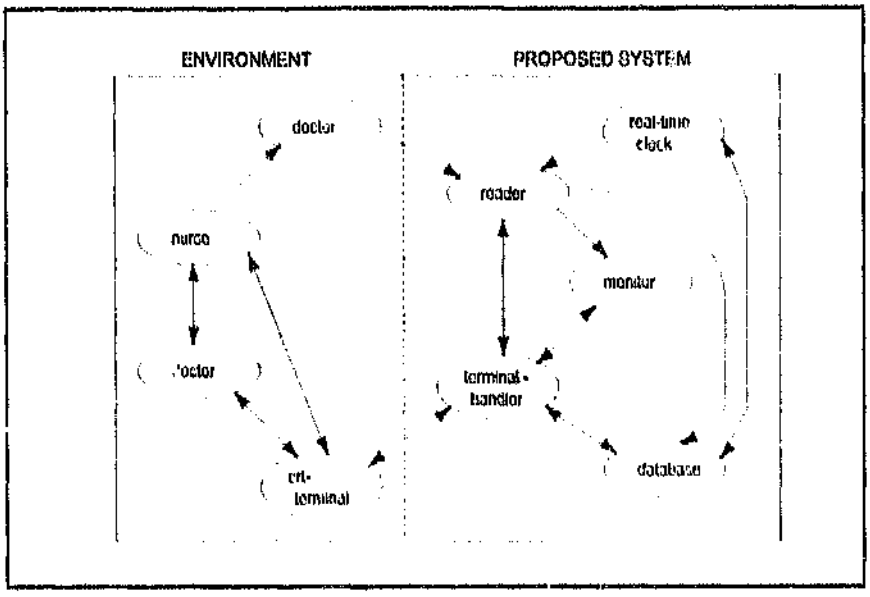


Figure 3: Zave's partial model of a patient-monitoring system

6.1.2 Dealing with Timing Issues

One of the concern of embedded systems developers is how to deal with the real-time issues. There is more and more pressure to deliver sophisticated real-time systems that can deal with complex, incompletely specified, dynamic environments in an increasingly autonomous fashion [19]. To do so requires that time-constrained problem-solving techniques be incorporated into real-time systems. As it is, there is currently no coherent theory of real-time problem-solving [20]. The existing real-time problem-solving systems are, at best, coincidentally real time [19]. Strosnider and Paul [20] presented a structured view of real-time problem-solving, based on Newell-Simon problem-space hypothesis. They look at the problem-level search space and at the knowledge retrieval level to reduce the search variations in real-time problem-solving. Although still purely theoretical, their approach is a constructive step towards reducing computation time in real-time problem-solving systems.

Always on the subject of meeting the time constraints on real-time embedded

6 Planning Embedded Systems Projects

6.1 Development Strategies

Little has been written on complete development processes specifically for embedded systems. The tendency is to use the existing, general software development processes like the ones mentioned in section 2.0 of SDE 102. What has been investigated, though, are techniques to make the different development phases more suitable for embedded systems. Many of those techniques have their roots in other domains closely related to embedded systems, like the artificial intelligence (AI) community or the real-time community. In this section we look at some of those planning and development strategies.

6.1.1 Formal Requirements Specifications Methods

A lot of research is devoted into capturing the characteristics of embedded systems down at the requirements level [1][16][17][18], where already they differ from other kinds of software developments. One such study was performed by Zave [1]. It is based on an operational approach to development. Unlike conventional requirements specifications methods, hers is strongly coupled with the system's environment. Including an explicit model of the environment has several advantages for requirements specification. It provides more insight into the interfacing of the system with the environment, which for embedded systems is often complex, asynchronous, highly parallel and distributed. Furthermore, assumptions and expectations on both sides of the boundary can be documented. The result is a specification which is far more precise and yet comprehensible than could be obtained by treating either side of the interface as a "black box", which is what happens when the environment is not modeled. Zave's approach is depicted in figure 3 which shows a partial requirements model for a patient monitoring system. The interacting environment-proposed system model is even supported by a specification language called PAISLey (Process-oriented Applicative and Interpretable Specification Language) [1].

The model proposed by Zave bears some similarities to the idea of object-orientation, in the sense that it captures the problem in terms of abstractions and not as functions. Notice that a similar integration of the environment into the specifications can also be achieved with object-orientation simply by looking at the problem at a higher level of abstraction. Furthermore, object-orientation too can easily encompass the hardware as well as the software components of the

In order to assist this critical testing phase, test tools, such as logic analysers and in-circuit emulators have been developed. Test tools and development tools in general are considered in section 9.

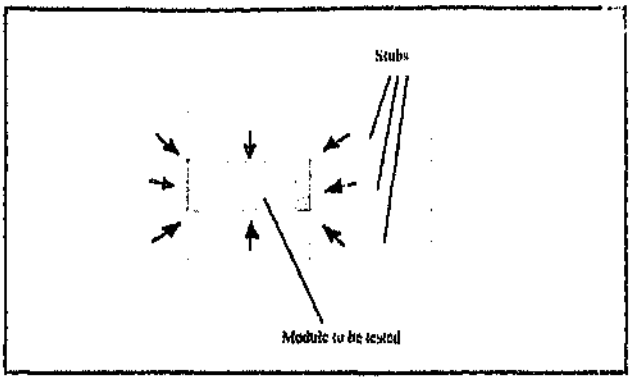


Figure 2 (a): Integration testing of modules using stubs - single module

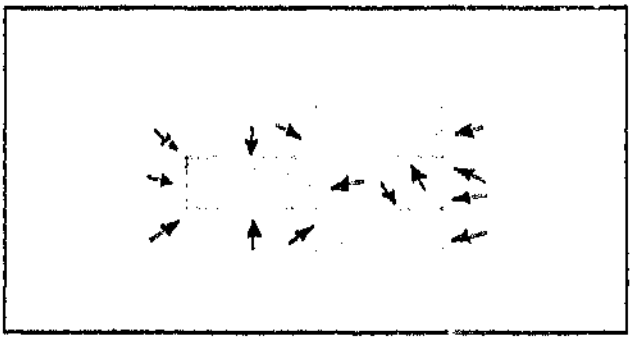


Figure 2(b): Integration Testing of modules using stubs - multiple modules

Modules or subsystems may perform correctly when tested alone but fail in combination. Consequently dynamic testing should always be carried out in depth and should include functional program testing such as:

- *Stress tests.* Stress tests (sometimes called endurance tests) are black box tests which impose a range of abnormal and illegal input conditions so as to stress the capabilities of the software. Input data volume and rate, processing time, utilisation of data and memory are all tested beyond the design capability. The length and depth of the test will depend on the complexity of the product.

- *Environmental tests.* These will also include tests under conditions of electromagnetic interference. Software systems are particularly prone to data corruption resulting from both mains and air-borne interference and thorough functional tests are needed.

different conditions. The following pattern is generally observed [10]:

a. *Code inspection and walkthrough.* This is the weakest level of test. It is subjective and relies on the application of past experience to the review of code.

b. *Symbolic evaluation.* A path is followed through the code and the effect of various inputs on the corresponding outputs is tested. This is a low level parametric test which validates the performance of the code in respect of particular variables rather than addressing the functional specification.

c. *Static analysis.* These are tests which do not actually execute the program but examine the logic and paths within it. To some extent a compiler performs a static test by checking for syntax errors and undeclared variables. Static analyzers examine the code algebraically without the use of actual input and output values. Static analysis packages are tests which assess the structure of a program. They operate at source code level and verify that the program is correct against its specification. Furthermore they validate the program algebraically. An example of such a package is MALPAS (MALvern Program Analysis Suite).

d. *Dynamic analysis.* In this type of testing 'live' data is used to exercise the inputs and outputs and thus test the actual performance of the code. It is the most important test, as it focuses on the performance of the product in conditions as close as possible to the real-type situation. It is at this stage that individual modules are tested and then integrated into the system. The tests rely on the executing of the program either with simulated or real inputs. The concept of integration testing of modules using simulated modules, known as stubs is depicted in figure 2(a). Stubs are necessary to provide a minimum simulated interface to replace each of the as-yet uncoded modules. As additional modules become available they replace the stubs as shown in figure 2(b).

e. *Test reports.* There should be a test report for each test or group of tests. The main benefit is that the report provides a medium for recording any actions which arise as a result of the faults revealed. The actions, having been formally recorded, can then be reviewed until they are discharged.

5.3 Limitations of testing

Test requirements as seen in the context of the life-cycle, evolves from the very beginning of the design process. Testing is the exercise that ultimately assesses the quality of the output product. However, testing itself, has some limitations and when analysing test results it is essential that they are viewed in the context of those limitations.

The main limitation to software testing is the inability to foresee all the combinations of external conditions and logic states in the program. Timing related faults prove the most difficult to reveal. Practical test limitations as exposed by Smith and Wood [10] include:

a. The range of test conditions and loads which can be supplied. It may only be possible to apply environmental conditions individually or in a limited number of combinations. This limitation may also apply to the loading of inputs which will depend on the test equipment and simulators provided.

b. The range of operator actions which can be foreseen and planned for in the test procedure. These should include misuse tests but they will be limited by the imagination of the test writer.

c. The extent of the operating instructions.

d. The choice of language and compiler which will have an effect on the difficulties actually experienced during testing.

Management constraints of time and cost will also limit the test effectiveness. Since testing, however soon it commences, is the last activity before despatch, and since schedules always slip then it will be the activity most likely to be curtailed in favour of delivery.

5.4 Test Strategy

Typically a test strategy involves viewing the product at different angles and under

development process. Frequently the cost associated with producing fault-free software, also presents itself as an indirect cause of faults. Sommerville [11] observes that developing fault-free software is very expensive and, as faults are removed from a program, the cost of finding and removing remaining faults tends to rise exponentially. It is therefore common to find organisations trying to reach a compromise between cost and residual faults.

5.2 Test Management

The whole structure and philosophy of the tests are dependent on the requirements specification and on the system configuration. The test plan and an outline of tests methods must therefore begin to evolve, along with the hierarchy of documents, right from the beginning.

The first document, as regard testing, will be the test strategy, discussed in section 5.4 below. This might be a separate document or may form part of the quality plan. It will outline the various levels of test and how these will build up to a final functional system test. Broad details of the test hardware and software which will be needed to carry out the various integration, simulation, loading and other tests will also be given. This, and the subsequent test documents, should not be produced by the designers but by a separate test authority. Other essential documents will follow as the design proceeds. These include [10]:

a. *Test specifications.* One for each separate test activity of which there may be several dozen. It will describe the functions to be tested and the test method to be employed, including the range of values which will be covered. The test equipment required is also described here.

b. *Test procedures.* The actual test instructions down to the details of connecting test equipment and the actual inputs to be applied and their sequence. A good test procedure should contain the anticipated result of each test and a record sheet on which to record the results.

c. *Test records.* There should be a test record for every test and, as mentioned above, this may be part of the test procedure document.

d. *Test utilities specification.* Both the hardware and computer facilities needed for all the tests are specified here. Any test software (i.e. program that provide test data so as to simulate modules or hardware items not yet designed) is also described here.

- Using a model that is not a good fit to the physical situation
- Incorrect document cross-references

Possible causes of inconsistent or incompatible requirements are:

- Having two references give conflicting information
- Inconsistent use of conventions
- Unclear or illogical requirements
- Omitted requirements (e.g.: handling of invalid inputs).

b. From the design phase, faults may be caused as a result of:

- Unstructured approach to the design breakdown (i.e. detail is considered first)
- Lack of proper reviews
- Lack of change control
- Specification was misunderstood

c. From the coding exercise, faults may occur due to:

- Semantic errors involving incorrect use of statements
- Logical errors in translating the design into code
- Detailed syntax errors which may have escaped detection by the compiler
- Poor data validation (e.g. no default condition after a data input)
- Variables not initialised, or used incorrectly
- Insufficient arithmetical accuracy
- Insufficient range checks. (e.g. divide by zero)
- Type mismatch (e.g. string used as a variable)
- Residual errors in compilers

Another possible source of error is the corruption of the program itself [3]. In most small real-time systems, the program is stored in some kind of read-only memory where it is relatively safe from harm. Instructions are transferred to the CPU for execution. Controllers often operate in electrically noisy environments, and occasionally interference can affect this transfer. The result is that the microcontroller executes the wrong instruction, and this may upset the program sequence. The watch-dog circuit will trap this kind of behaviour and restart the program from the beginning.

As the list above shows, at the origin of faults often lies a mismanagement of the

5 Software Testing

Software testing is often the last defense against disasters caused by faulty software development. When lives and fortunes depend on software, software quality and its verification demand increased attention [14]. Testing not only removes faults (the underlying source of software failure) and hence improves its reliability, it also demonstrates reliability and safety by running software under field conditions [15]. Software testing is an activity common to all software development processes. It will be discussed here in the context of embedded systems.

5.1 Faults

Prior to developing the subject of testing, it is worthwhile to examine the nature of the faults that the testing exercise aims at checking.

Faults may occur in both hardware and software. Software faults - often known as *bugs* - will arise as a result of particular parts of the code being used for the first time or because of corruption due to some outside influence [10].

A fault (bug) may lead to an *error*. This is the condition whereby the system is in an incorrect state. A data value or instruction is thus incorrect and only when that particular part of code is executed is the error revealed. However the presence of a fault in a program does not necessarily result in an error or failure. A long time may elapse before that portion of code is used under the circumstances which lead to failure.

An error may propagate to become a *failure* if the system does not contain some error recovery logic capable of dealing with and minimising the effect of the error. A failure, be it hardware or software related, is the termination of the ability of an item to perform its specified function.

5.1.1 Causes of Faults

The possible causes of faults are numerous; some of them are listed below.

a. The requirements specification is often a source of faults as a result of incorrect or inconsistent requirements. Possible causes of incorrect requirements are:

- *Reinitialisation.* This involves resetting the system to a known acceptable state and reinitialising the processing. Variables are reset to predetermined or known good values.

- *Alternate path.* In the case of mathematical routines a second path, or try, can be provided. If the result of a particular calculation is not satisfactory then an alternative calculation can be performed.

- *Recovery blocks.* These consist of blocks consisting of an acceptance test for the calculation with one or more alternate routines which are used if the test fails.

- *Exception handling.* These consists of identifying conditions which are defined as exceptional and which require additional code to carry out the processing.

- *Memorising executed cases.* Here a record is made during certification of code of the allowed paths for program execution. This information is stored in some way and when the program is executed in its real environment the actual execution is compared with this allowed set. If an uncertified path is executed then safety action is taken.

- *Error correcting codes.* Here we try to detect and correct errors in sensitive information by using different types of code, e.g. Hamming codes.

- *Manual recovery.* In the event of failure, control of the system is returned to an operator who attempts to control the system rather than leaving it to automatic means.

in any case a total solution to the problem [10]. We have already seen that a significant source of software faults is the activity of stating the requirements specification. As a result identical faults can propagate through each of the 'separate' design activities and manifest themselves in all the channels.

4.2 Error Management

As already mentioned, a fault-free software is often not achievable. At times the problem might not even be at the software level, e.g. a sensing circuit that suddenly delivers pulses at a faster rate than the microcontroller can read, causing some of the pulses to be missed completely. Reliability considerations demand that provisions are made to deal properly with residual errors. We have already covered different configurations that can improve the overall reliability. We shall now look at how the software itself should behave under an error condition. In any event, the response of the software must be controlled and should not cause the program to crash. As a first step, the software needs to be able to recognise an error condition. Error detection is achieved by a number of techniques:

- Watchdog timer techniques involve using the processor clock to monitor outputs to verify that they are not stuck in one state.

- A large proportion of the hardware in a programmable system consists of memory. It is possible to check the state of a memory by reread. This walking bit technique will minimise software corruptions faults by flagging up failed memory. In the case on ROMs checksum techniques are needed which will verify the contents (which should not change) against predetermined checksum values.

- An extension of this philosophy is to provide code which can, having diagnose that an error has been generated, correct the value or values in store so as to effect a recovery. The simplest example of error detection software is the parity digit, whereby an additional digit is included with some value.

- Range checking techniques, discussed in section 5 of SDE 113.

Once an error condition is detected there should be built-in procedures to recover from that condition with the least possible damage to the system. Seven approaches to error recovery are recognised by Smith and Wood [10]:

7 About Programmable Devices

7.1 The Devices

Embedded systems can be developed on different intelligent platforms. Below is a list of the major architectural types.

a. Von-Neuman Architecture

Microcontrollers based on the Von-Neuman architecture [31] [34] have a single "data" bus that is used to fetch both instructions and data. Program instructions and data are stored in a common main memory. When such a controller addresses main memory, it first fetches an instruction, and then it fetches the data to support the instruction. The two separate fetches slows up the controller's operation.

b. Harvard Architecture

Microcontrollers based on the Harvard Architecture [34] have separate data bus and an instruction bus. This allows execution to occur in parallel. As an instruction is being "pre-fetched", the current instruction is executing on the data bus. Once the current instruction is complete, the next instruction is ready to go. This pre-fetch theoretically allows for much faster execution than a Von-Neuman architecture, but there is some added silicon complexity.

c. CISC

Almost all of today's microcontrollers are based on the CISC (Complex Instruction Set Computer) concept. The typical CISC microcontroller has well over 80 instructions, many of them very powerful and very specialized for specific control tasks. It is quite common for the instructions to all behave quite differently. Some might only operate on certain address spaces or registers, and others might only recognize certain addressing modes. The advantages of the CISC architecture is that many of the instructions are macro-like, allowing the programmer to use one instruction in place of many simpler instructions.

d. RISC

Reduced Instruction Set Computers (RISC) [31][32], as their name suggests, have

Management frequently desires to see code as an indication of work accomplishment, however, starting the code phase before proper design and review usually leads to larger delays later in the project.

- Inadequate Time for Testing and Product Integration

In microcontroller software systems the time required for testing and product integration is often larger than with traditional software systems because:

- The hardware is developed as the software is developed. There is not a stable base for software debug, thus lengthening the debug time.
- The microcontroller software is a small part of a large electromechanical system. Software debug is constrained by the rate at which the rest of the system is debugged.

6.3 Pitfalls to Avoid When Planning

Various aspects of microcontroller software development require special attention during planning and execution to ensure that the project stays on schedule. Some of these are listed below.

- Because of the embedded nature of most microcontroller applications, requirements frequently change when the problems occur in other parts of the system. Developers must anticipate changes in their plans, or when changes are made, the schedule and cost estimates must be updated. The overall system architect should be aware of the cost changes to software requirements and should weigh carefully the merits and disadvantages of solving non-software problems by changing the software requirements late in the software development cycle. The COCOMO model[30] accommodates this effect with a factor called "requirements volatility", described in table 3.

Table 3: Requirements Volatility Effort Multipliers

Rating	Project Rework due to Requirements Changes	Effort Multipliers
Low	Essentially None	0.91
Nominal	Small, noncritical redirections	1.00
High	Occasional moderate redirections	1.19
Very High	Frequent moderate or occasional major redirections	1.38
Extra High	Frequent major redirections	1.62

- Parkinson's Law Corollary [7] - Code will expand to fill the memory available.

  Since the cost per instruction can increase dramatically as the utilisation of memory approaches the limit, it is necessary to resist the temptation to sneak in a few more features. In microcontroller system this is usually a severe constraint, unlike large computer systems where virtual memory is often a solution.

- How Much Code Syndrome

  Design and review steps should not be rushed to begin coding.

Most of what was already mentioned about project planning and software metrics in section 5 of SDE 102, applies for embedded systems. In fact, in our discussion about COCOMO in section 5.3.1 of that same document, the case of embedded systems was taken into account. Metrics can be very useful when planning for and making predictions on a software development, provided their limitations and assumptions are well understood.

As an example of how COCOMO can be applied to embedded systems, a "typical" product development activity for real-time embedded microcontroller products in large companies was assessed by Rauscher and Ott [7] and is summarized in appendix A. Note that the product of the effort multipliers is 4.53, which means that the effort required is over four times the effort required for the nominal case. Carrying the analysis one step further, consider the case of the development of 64 kilobytes of software (assuming that one higher level language statement generates on the average 2 bytes of code):

MM_{nom} $=2.8 \times KDSI^{1.20}$

$=2.8 \times 32768^{1.20}$ assuming 2 bytes per higher level language statement

$= 184$

$MM = \sum_{i=1}^{15} MM_{nom} \times P$ (effort multiplier)

$= 184 \times 4.53$

$= 835$

$TDEV = 2.5 \times MM^{0.32}$

$= 2.5 \times 8350.32$

$= 21.5$

The results show that the particular development requires:

- Over 69 man-years effort
- Over 21 months of calendar time

Understanding all the factors that affect microcontroller software development is thus critical to planning the schedule of a microcontroller-based product. Accordingly, one should not expect to estimate the cost or schedule for a project for which the requirements are unknown or unclear.

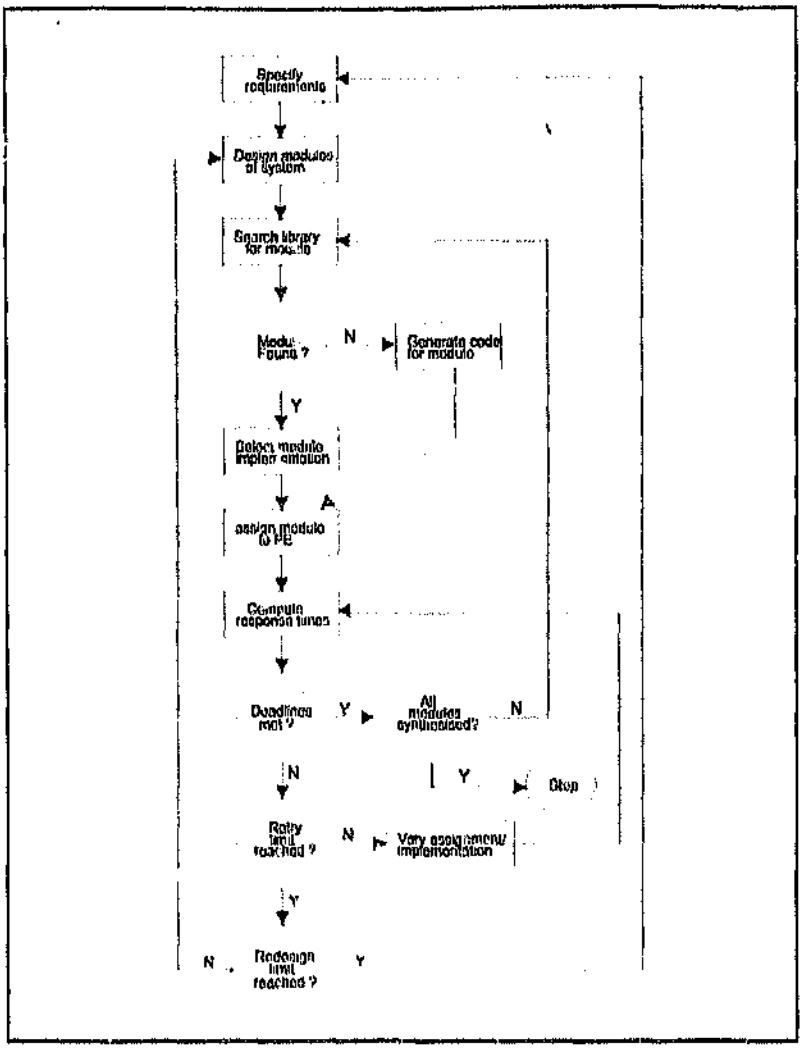


Figure 11: Software synthesis model for object-based, parallel embedded systems

a new module. To perform the analysis, an implementation of a module is chosen, and is assigned to a PE. The response time of the module are then tested. If the performance is acceptable the exercise is repeated with a different module, else the module is modified or the process is assigned to another PE. The process of varying implementations of components and varying the assignment is repeated until the timing criteria are satisfied, or until some fixed number of attempts have been made, at which time the system design "fails" to meet the requirements. Only a fixed number of designs is tried before the system requirements are deemed to be "unreasonable" and are therefore modified.

6.2 Applying Software Metrics to Embedded Systems Projects

the *me too* language. In the *validate step*, the specification resulting from the CSP/*me too* description is exercised. Feedback from this step can be used to improve the original model, either by correcting errors, by remedying omissions and inconsistencies or by exploring alternating designs.

A typical *me too* specification is shown in figure 10.

```
Specifications of Packets
objects
   Destinations = String
   Contents = String
   Packet
exported operations
   make_packet: Destination*Contents → Packet
   the_dest: Packet → Destination
   the_mess: Packet → Contents
   null_pkt → Packet
object representations
   Packet = record (dest: Destination, cont: contents)
definitions
    make_packet(d,m) == mk_packet (d,m)
    the_dest (pk) == dest (pk)
    the_mess (pk) == mk_packet ("nowhere")
end Packets
```

Figure 10: Example of a *me too* specification - Specification of the Packet object

Another object-based approach to the development of embedded systems is proposed by Stoyenko et al. [29]. This approach is restricted to parallel embedded systems; i.e. the operations can be assigned to more than one processing element (PE). The development model is outlined in figure 11.

The first stage of the development is the specification of the requirements. This is followed by the design stage. The model assumes that an object-orientation approach is chosen for the design, where physical objects from the application domain are modeled as abstract data objects (ADOs) or as abstract data types (ADTs). The output of the design stage is a set of ADO, ADT and process module specifications, and a description of the interactions of the modules. Given the specifications of the modules, libraries of previously implemented modules are searched to determine which modules can be reused first, before implementing

6.1.4 Object-Oriented Approach

There are attempts of adapting object-orientation more closely to the characteristics of embedded systems. One such approach presented by Clark [28] is based on the combination of Communicating Sequential Processes (CSP) and the *me too* method of software design. The final specifications are transformed into an outline concurrent Ada program. *me too* is used to express the bject representations and the operation definitions. It is a purely functional language which make it easier to reason about operations than in an imperative language.

The *me too* method of software design has three steps: the model step, the specify step and the validate step. The method is iterative with feedback from the specify and validate steps being used to refine the original method. The CSP/*me too* approach is the same, but deals with concurrent systems. The method is based on the operational approach (see section 2.0, SDE 102) to software development, where an executable specification of a complete system is constructed in terms of problem-oriented structures [28]. The method is outlined in figure 9.

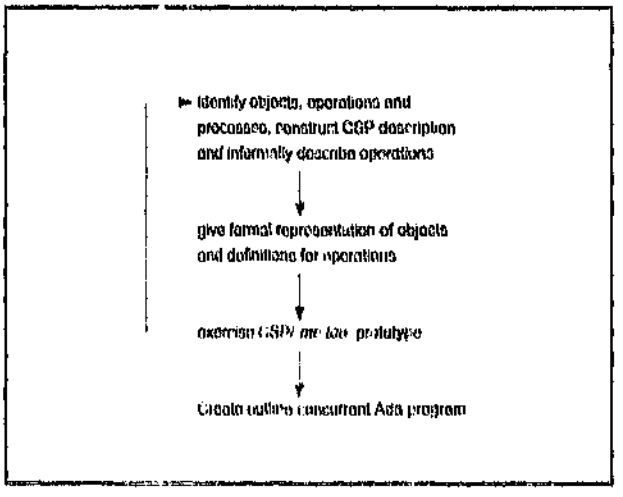


Figure 9: The CSP/me too method

In the *model step*, a model of a real-world situation is built by identifying a set of objects which can communicate with one another by message passing. In the *specify step*, each object in the model is given a formal representation in terms of sets, sequences etc. Operations on each object are defined constructively in

b. *Design Partitioning*: The second step of the design process is design partitioning, i.e. choosing the software or hardware implementation of each component of the system specification.

c. *Hardware synthesis*: The third step of the process is to implement each CFSM in the chosen style. The synthesis algorithm is as follows:

- Each hardware partition is implemented as a fully synchronous circuit.
- Each software partition is implemented as a C stand-alone program (with an operating system skeleton that includes the scheduler and I/O drivers) embedded in a microcontroller.
- All partitions have the same clock

d. *Software Synthesis*: The CFSM subnetwork is mapped into a software structure that includes a number of procedures and a simple operating system. A two-step process implements the reactive behaviour:

- Implementing and optimising the desired behaviour in a high-level, technology-independent representation of the decision process (called s-graph)
- Translating the s-graph into portable C code and using any available compiler to implement and optimise it in a specific, microcontroller-dependent instruction set.

e. *Modelling Data Flow*: Although general in terms of expressiveness, CFSMs are not specifically designed for computation-intensive tasks, but only for control-dominated ones. The idea is that a CSFM specifies the reactive part of the behaviour, whereas standard functions can specify the details of the algorithms associated with the actions invoked.

f. *Interfacing Implementation Domains*: Event emission and detection are implemented differently in each domain (software and hardware) so that an interfacing mechanism is needed. Chiodo et al. use a three-layer graphical block to represent the three different kinds of interfaces (hardware-to-hardware, software-to-hardware, hardware -to-software).

system, and not just the ROM, RAM and other immediate peripherals.

The methodology of Chiodo et al. is an extension of classical finite state machines called Codesign Finite State Machine (CFSM). In that respect it bears some resemblance to the other state machines representations like the one presented by Drusinsky [22] above (section 6.12). The codesign framework of Chiodo et al. is depicted in figure 8 below.

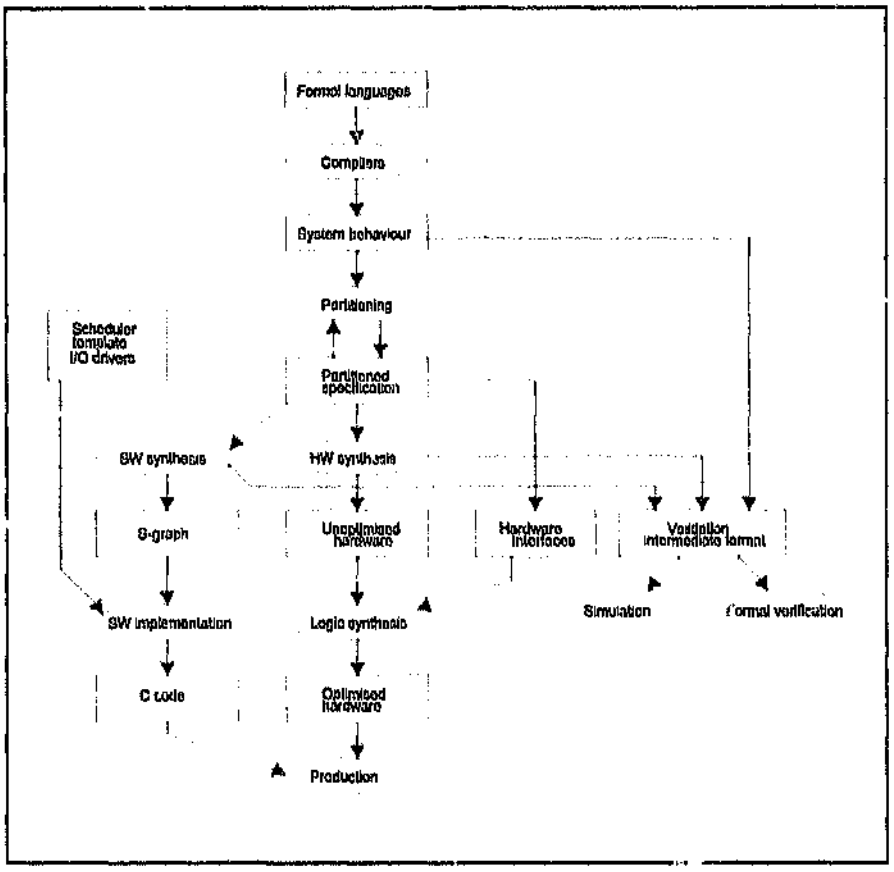


Figure 8: The CFSM framework

The main steps of the process are described below:

a. *Specification Language Translation*: Due to its relatively low-level view of the system, the CFSM model is not meant to be used directly by designers. Therefore the specifications are written in a higher level language -Esterel, StateCharts, or a subset of VHDL - that directly translates into CFSMs.

determines the range of available applications and, conversely, application requirements may dictate the size of the engine and the hardware design. Change one and the other is affected. For that reason the two should be co-designed, developed in concert to ensure that the completed system not only functions properly, but also meets performance, cost and reliability goals. More often, though, hardware and software systems are laid down separately. While this approach is possible, failure to consider tradeoffs between hardware capability and application requirements during development can lead to designs that are time-consuming, expensive and that ultimately fail to perform as intended [25].

Wolf [25] presents an approach to hardware-software co-design of embedded systems. The process is a top-down sequence of steps which starts with a common specification for both the hardware and software elements. After the system architecture is defined, design of the hardware and software components proceed separately before being integrated. Lastly the whole system is tested. The process is outlined in figure 7 below.

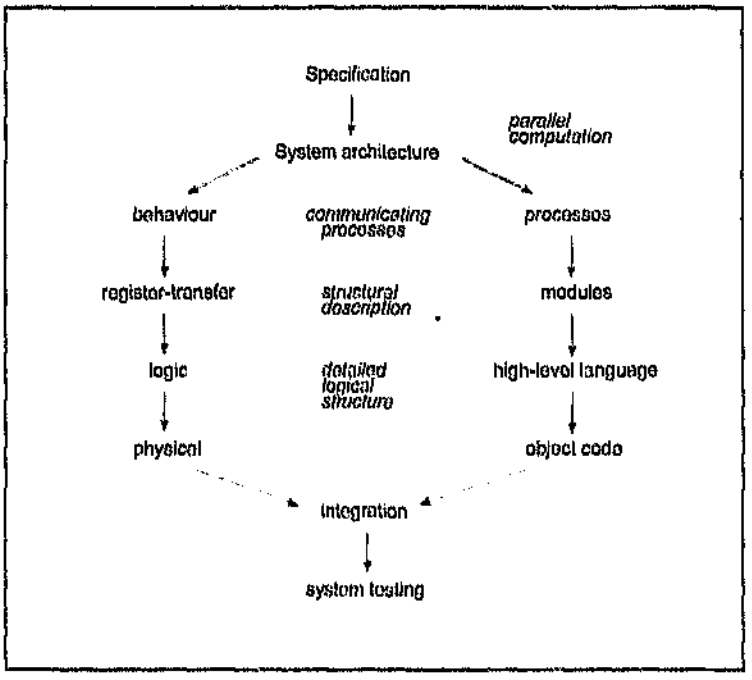


Figure 7: A top-down hardware-software co-design process for an embedded system

Another hardware-software co-design approach is proposed by Chiodo et al [27]. The approach of Chiodo et al. has for advantage of viewing the hardware platform as being the interfacing between the software and the rest of the application

diagrams. An example of a state diagram is shown in figure 6, where it is used to capture the events and states for an intelligent traffic-light controller. The extended diagram is very similar to the state transition diagram used by Booch [24].

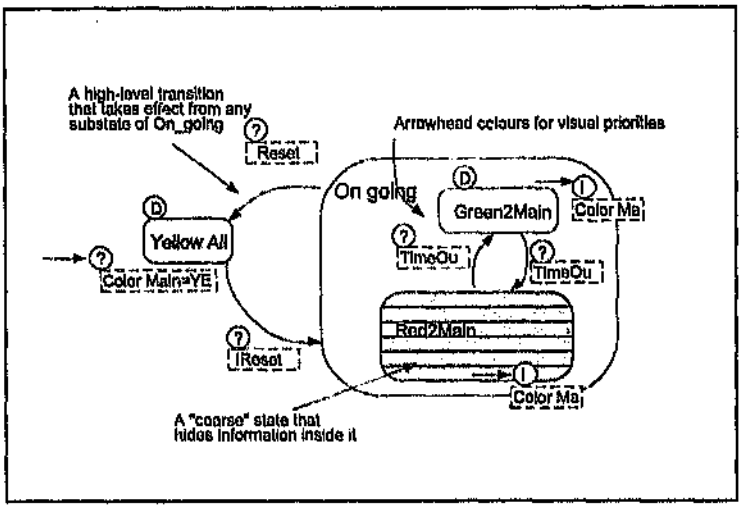


Figure 6: An extended state diagram

6.1.3 Hardware-Software Co-Design Techniques

The strong hardware interaction of software with hardware in embedded systems has pushed for strategies that encompass the hardware development along the software evolution. The existing hardware-software co-design techniques though, tend to focus on Very-Large Scale Integration (VLSI) technology [25][26] applications, where the term hardware is used to describe the direct platform on which the software operates, like the CPU, ROM and RAM. They do not really cover the exercise of interfacing the software with the broader structure of the application. This exercise typically involves hardware design considerations like sensing and actuators mechanisms, electromagnetic noise concerns, performance trade-offs and compatibility issues. However their approach is a very constructive step in bringing hardware design considerations together with software design decisions.

The argument for hardware-software co-design is as follows. By definition the hardware and software elements of an embedded system are closely intertwined: the size of the computing engine and the design of the hardware platform

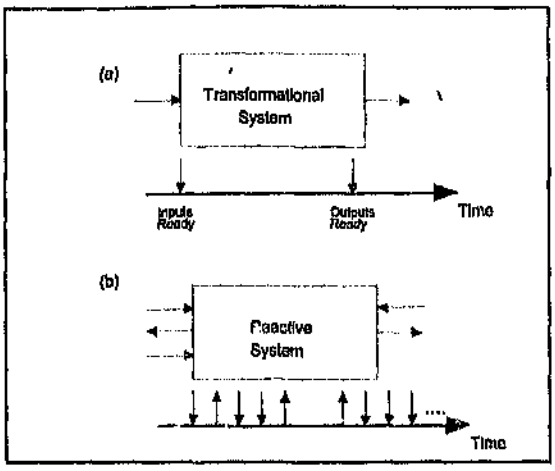


Figure 5: (a) Transformational systems (b) Reactive systems

Finite state machines (FSMs) and state diagrams (FSMs visual counterparts) have traditionally been used to specify and design reactive (sub)systems. They are well known, well accepted, highly visual, and intuitive. However FSMs and state diagrams have not changed much over the past 30 years and suffer from limitations when applied to today's reactive embedded systems [22]:

a. FSMs are flat. They do not cater for top-down design and information hiding.

b. FSMs are purely sequential, whereas applications are not. Modern controllers need to react to signal to and from a plurality of entities in the environment.

c. Text-based entry methods, which are by definition sequential, cannot effectively capture concurrent behaviour (a common occurrence in embedded systems). therefore, drawing state diagrams on paper and entering them textually is no longer effective.

d. Top-down design concepts require interactive software to enable the user to manipulate and browse through complex designs.

Because of such limitations, FSMs have been used sparingly in recent years [22]. Compensating for these limitations are extended state diagrams. While addressing the hierarchy, concurrency, priorities, and synchronisation within state diagrams, extended state diagrams retain the visual and intuitive appeal inherent to state

CORAL [35] is a block structured language which has been promoted by the British Ministry of Defence for real-time programming. It predates PASCAL, but has many of its features, being originally derived from ALGOL [38]. The language includes "bit" variables and sections of assembly code can be included within modules. It was not well marketed outside the UK defence industry and is rarely implemented on desk top computers, but still has its uses. Programs in CORAL are usually developed on large mainframes and then downloaded into the target machine for debugging.

e. ADA

ADA [38] [39] was developed by the United States Department of Defense to try to standardise the large range of languages used in their systems. By insisting that it is used on all NATO contracts, they have been able to disseminate it into many organisations [3], although the take up has been slower than intended. ADA is a very large, complex language requiring an investment in training that most organisations are reluctant to make. It is designed for real-time programming, but is not really suitable for small, embedded systems.

f. PL/M

PL/M is a language developed for the Intel range of microprocessors. It is block structured like most modern languages, and is similar to PASCAL in most of its features. PL/M is designed to take advantage of the features of the Intel microprocessors and microcontrollers, and runs on that company's development systems.

g. FORTH

FORTH, unlike most other high level languages, in neither compiled nor interpreted like BASIC [40]. Instead it is a threaded interpretive language. Most operations involve a stack so that, for example, an instruction will add the number on top of the stack to the number immediately below it, placing the answer on the stack. Programs are written by defining new keywords using combinations of existing instructions, gradually extending the language definition until it includes the application program. Because of its speed and flexibility, FORTH is of particular interest for real-time embedded systems (e.g. it has been extensively used by makers of video arcade games). However, it is quite a difficult language to use, at least until the programmer is thoroughly familiar with its idiosyncrasies. Maintaining software developed in FORTH is not an easy task. It is good to note

b. PASCAL

PASCAL [37] is a block-structured, compiled language which is widely used in industrial and commercial programs. PASCAL compilers are available for almost all microprocessors and will run on most computers, but they are however much less common for microcontrollers. As a compiled language, PASCAL can produce programs which execute much faster than an interpreter. The actual speed itself depends largely on the compiler used and on the microcontroller itself. However in general we note that PASCAL compilers tend to produce lengthy code [3], which does not encourage their use in embedded systems applications. In addition, because the language is designed to be run on a variety of different computers, the programmer is insulated from the machine-specific features. The same procedure call is used to write a line of text to the console device regardless whether it is a video display unit (VDU) or a teleprinter. The PASCAL system calls the appropriate device driver in each case without the programmer's intervention. The designer of embedded systems usually needs a closer involvement with the hardware. There may be timers to control, converters to drive and special interface requirements to be met.

c. C

C was originally designed as a tool for the development of computer operating systems but, because of its power and flexibility, it is used in many other applications. It is designed to be portable, so that only a small part of the compiler has to be rewritten to transfer it to another microprocessor. This in itself gives it a considerable advantage over the other languages for embedded systems applications, since it is often easier to find a C compiler for a particular microcontroller than any other high level languages. In fact most of the C compiler is itself written in C. The language tends to produce fast and compact code in addition to being widely used, available and supported, making it suitable for many real-time embedded applications.

Compared with PASCAL it is a more terse language. It uses terse symbols extensively, where PASCAL uses keywords. This makes the transition from the English language to C more difficult. C offers the programmer more freedom in writing and has far less restrictions on the programming style than PASCAL. This power of C means that the programmer has to be more careful about the way the program is written.

d. CORAL

call them from the higher level code. This technique can also be used to handle sections of the program for which execution speed is critical. Some languages such as ADA and versions of Pascal provide a pragma pseudo-op which allows for assembly code to be placed in-line with the higher-order language code [13]. A method sometimes used to reduce the tediousness of writing assembly codes is by writing a shell of the program in a high order language. This program is then compiled to an assembly file, which is finally massaged to obtain the desired effect [13].

To summarise, high level languages should be used whenever their drawbacks of memory size, execution speed and awkward interfaces will not impact on the success of the design. The speed of code production, enforced design discipline and relative ease of maintenance are valuable assets. Assembly language design is appropriate in small systems where memory space is severely limited, as is unfortunately the case for many embedded systems projects. Assembly language should also be used for time critical sections of code. In any case the choice between high level languages and assembly languages remains a difficult one. As Adamson [3] puts it, the difference between writing in high and low level languages is like the difference between working under a strict supervisor and being left in charge of the workshop. All things become possible, but not all things are wise!

8.3 Overview of some High-Level Languages in the Context of Embedded Systems

A brief evaluation of a few of the high languages with respect to embedded systems applications follows:

a. BASIC.

Although BASIC is widely known in the programming environment [35][36], it is rarely used in embedded systems. The reason is again efficiency-related. BASIC is an interpreted language. This means that the source code written by the programmer is turned into machine code each time the program is run. This contrasts with compiled languages which are translated into machine code by the compiler. The compiled code is then run whenever required. BASIC programs run slowly because this translation has to be performed each time. If a loop is executed ten times, the codes inside are interpreted ten times, and the comments are read and discarded ten times. The entire source code must be present at run time, and it will probably occupy much more memory its machine code equivalent.

to maintain the software or perform quality control activities over the program. On the other hand, too great a reluctance to change established ways of working can result in a design whose performance is not as good as it could have been.

8.2 High-Level Languages v/s Assembly Languages

The tendency is towards programming in high level languages like Pascal or C. High level languages offer a higher level of abstraction, demanding less knowledge about the inner structure of the microcontroller and its memory structure. They produce much more readable code, with fewer errors, leaving the innermost details of the implementation to the compiler. The overall result is faster development and easier maintenance.

On the other hand, writing in assembly languages is a lot more time consuming. Assembly languages lack the inherent structure of their higher level counterparts. In addition microcontrollers of different makes have different assembly languages. This will mean learning a new language for each new microcontroller.

From the above discussion, there would seem to be a powerful argument for using high level languages, however what is seen in practice is that a lot of embedded systems projects include codes written in assembly language. The reason is often efficiency related.

Compilers are general purpose programs, and the code they produce are longer and slower than that which can be written by a skilled assembly programmer. Adamson [3] observes that a good compiler may generate code 150% longer than the hand written equivalent. More efficient use of execution time, memory space and the microcontroller resources in general, often make assembly language a more suitable choice for real-time embedded systems. Very few compilers allow the programmer to specify that certain items should be kept in a register rather than a RAM location for speed of access. Multiple operation on the same data item can lead to the data continually being read from memory and then written back, when an assembly program would keep it in a register until processing was completed. Most microprocessors perform register-based operations faster than those which require RAM accesses (notice that C allows some variables to be defined as register resident). There can also be difficulties in using specialised parts of the microcontrollers from high level languages. The high level language definition knows nothing of on-chip timers or interrupt flags, and it may be necessary to define short driver modules in assembly language and

8 Language Issues

Coding should follow naturally from specification, since a computer program is simply a detailed specification of the actions the computer must perform [3]. How true this statement is in reality depends much on the methodology and the choice of language for the particular application. Design methodologies were treated in chapter 6 of SDE 102. Here we focus on the programming language issues in the context of embedded systems.

8.1 Factors to Consider When Choosing a Language

Strictly speaking most of the languages available for large systems, are applicable to embedded systems. However, in practice the peculiarity of embedded systems often considerably restricts the choice of languages and tends to give this choice a much higher importance. Factors that affect the choice of language are, in particular:

- Time constraints. For hard real-time systems the choice of language often decides of the fate a design.

- Space constraints. The resulting size of machine code is often of critical importance for embedded systems, especially when using single-chip microcontrollers.

- Compiler availability constraints. Even when time is not critical, the use of some languages might not be possible because either the compiler for that particular language and microcontroller is not very efficient - compilers can differ largely in the way they handle the instructions - or else simply the development tools for the microcontroller do not include a compiler for that language (e.g. the H8/3048 does not support Pascal)

- Compatibility constraints. It is always better to choose a language that is supported by other elements of the development environment, like the in-circuit emulator or the simulator, in other words a language that provides ease of interaction with the hardware platform.

- Programmers' previous experience. It may not be worth learning a new language for one small development project. Changes to the programming language affect not only the programmer, but also anyone who may have

units to achieve a higher dollar total. Figure 12 below shows the market share for the 16-bit microcontroller. It pictures Hitachi as being the leader in this product range, with $300M in shipments.

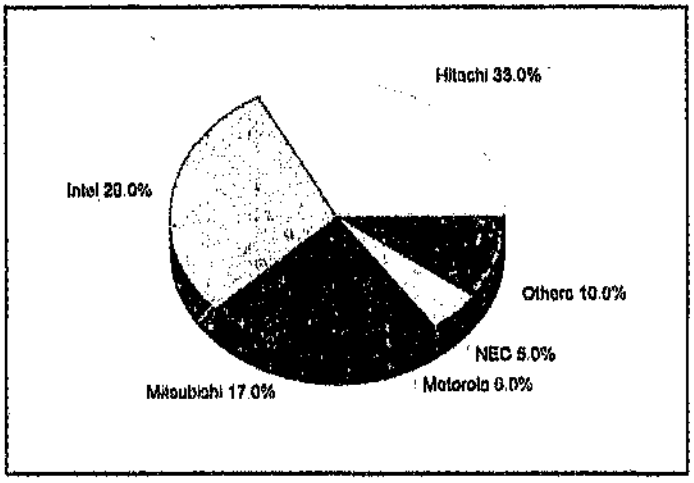


Figure 12: 1994 Worldwide 16-bit Microcontroller Merchant Market [47]

Among the popular microcontrollers are the 8048 (Intel), 8051 (Intel and others), 80c196 (Intel), 80186 (Intel), 80386 EX (Intel), 6805 (Motorola), PIC (Microchip), Z8 (Zilog), COP400 (National Semiconductor), HD64180 (Hitachi) and TMS370 (Texas Instruments).

The 8051 is probably one of the most popular 8-bit microcontroller. Consequently a considerable amount of software, documents and textbooks are available on this device and its applications. This is not a factor to overlook when choosing a microcontroller. PIC microcontrollers have RISC architectures. They are also very popular, especially among the hobbyists. They are characterised by low cost, small size and low power consumption. The 80386 EX is an embedded PC, meaning that it is based on the PC platform. In this case it is based on the 386 microprocessor. This allows the designer to use PC expertise, PC development tools, and desktop PCs for both software and hardware development.

support (like ICEs and evaluation boards) and many others. Even other products which do not appear to be derived form programmable devices at first glance, like for example Liquid Crystal Displays (LCDs), are actually closely interlinked with the microcontroller market.

7.2.2 The players

From the size of the market it comes to no surprise that many companies are involved in the field of embedded systems. As mentioned above the real size of the embedded systems market extends well beyond the simple production of microcontrollers. A few companies like Intel or Hitachi are even involved in different aspects of the industry; i.e. in addition to the microcontrollers, they also produce or sell the support chips, the ICE and other support products. Those companies tend to have a larger share of the market. Table 5 below provides an idea of the involvement of the major companies in the microcontroller industry.

Table 5: The major companies involved in the microcontroller industry - in terms of number of units sold. [4]

Company	Units (K) - 1993
Motorola	358,894
Mitsubishi	71,674
NEC	70,180
Hitachi	67,873
Philips	56,680
Intel	46,876
SGS-Thomson	37,350
Microchip	35,477
Matsushitta	34,200
Toshiba	32,205
National Semiconductor	31,634
Zilog	31,000
Texas Instruments	29,725
Siemens	20,874
Sharp	17,505

One should keep in mind that units sold does not necessarily equal dollars. Since some companies deal primarily in higher end devices, they need to sell fewer

Table 4(a): worldwide Microcontroller Shipments (in millions of dollars) [5]

	'90	'91	'92	'93	'94	'95	'96	'97	'98	'99	'00
4-bit	1393	1597	1596	1698	1761	1826	1849	1861	1856	1816	1757
8-bit	2077	2615	2862	3703	4689	5634	6553	7529	8423	9219	9715
16-bit	192	303	340	484	810	1170	1628	2191	2969	3678	4405

Table 4(b): worldwide Microcontroller Shipments (in millions) [5]

	'90	'91	'92	'93	'94	'95	'96	'97	'98	'99	'00
4-bit	778	906	979	1036	1063	1110	1100	1096	1064	1025	970
8-bit	588	753	843	1073	1449	1803	2123	2374	2556	2681	2700
16-bit	22	38	45	59	106	157	227	313	419	501	585

Although not listed above, 32-bit devices have also been recently introduced on the market. Owing mainly to cost considerations, their utilisation is still limited to a few applications where their increased computing power is necessary. Notice that even the relatively low-performance 4-bit device is popular. This is certainly a reflection of the popularity of microcontrollers in small control applications like washing machines or toaster ovens. 16-bit or 32-bit devices for such applications would certainly be an overkill. The choice of the device should be made in accordance with the size and complexity of the application. Also notice that the 8-bit market just keeps growing, and will probably continue to grow. 8-bit devices account for over half of the market, and will eventually grab even more. This explains why silicon manufacturers lay so much importance on their 8-bit microcontrollers, even if they are introducing high-end 32-bit devices. At times they would even upgrade their 8-bit microcontrollers, adding features to it, but still maintaining it as an 8-bit device, with more or less the same execution speed and the same memory limitations. This is very different to other field of software applications like desktop computers where the production of lower end microprocessors are either reduced or simply discontinued (like the Intel 286 microprocessor).

When judging the size of the market one should not be restricted to the microcontrollers. There are many other satellite industries that thrive from the main microcontroller industry. The other products include a whole range of software (compilers, simulators, debuggers, etc.), memory devices, hardware

keyboard handling, signal processing, video processing, and other tasks.

Because of their very general purpose and because of the large number of support hardware that they require, microprocessors are generally not used for embedded systems applications. Microcontrollers, on the contrary, are more adapted for embedded systems applications. They contain devices and peripherals which would be regarded as external in systems built around microprocessors. Those feature are very helpful to the control engineer and include watchdog timers, sleep/wakeup modes, power management, powerful I/O channels, and so on. Some also have a built-in universal asynchronous receiver/transmitter (UART) which transfers data using established protocols. Those features are not covered in detail here but are generally well documented in the hardware manual of the microcontroller. By keeping the instruction set specific and reduced (see SISC and RISC above), and thus saving valuable real estate, more and more of those features can be added, while maintaining the economy of the microcontroller.

Single chip microcontrollers (irrespective of the architecture) often combine many of the sections of a single board computer into one integrated circuit. They still require a few external components to make a fully functioning real time controller, but the number of support chips and consequently the space occupied is drastically reduced. This has an effect on the overall cost of the system, with the result that single-chip microcontrollers are often used in mass production applications. Small size, low power consumption, and flexibility make these devices ideal for unattended data monitoring and recording.

7.2 The Microcontroller Market

7.2.1 The Market Size and Product Range Distribution

The field of embedded systems generally presents itself as limited to small systems. We have already mentioned that this is by no means a reflection of the level of complexity it is called upon to solve or the level of safety and reliability it is expected to assume. In addition, we find that the size of the applications should also not be confused with the impressive size of the microcontroller market. A mere glance at the figures in the tables below should demonstrate the size of the market and in the same instance validates the amount of research work in the field of microcontroller applications. Table 4(a) and (b) also show the predicted figures for the years to come.

Flash provides a good better solution than regular EEPROM when there is a requirement for large amounts of non-volatile program memory. It is both faster and permits more erase/write cycles than EEPROM.

c. Battery backed-up static RAM

Battery backed-up static RAM is useful when a large non-volatile program and DATA space is required. A major advantage of static RAM is that it is much faster than other types of non-volatile memory so it is well suited for high performance application. There also are no limits as to the number of times that it may be written to so it is perfect for applications that keep and manipulate large amounts of data locally.

d. Field programming/reprogramming

Using nonvolatile memory as a place to store program memory allows the device to be reprogrammed in the field without removing the microcontroller from the system that it controls. One such application is in automotive engine controllers. Reprogrammable non-volatile program memory on the engine's microcontroller allows the engine controller program to be modified during routine service to incorporate the latest features or to compensate for such factors as engine ageing and changing emissions control laws (or even to fix bugs!). Reprogramming of the microcontroller could become a standard part the routine engine tune-up.

Almost every application could benefit from this type of program memory - if a modem's hardware supported it, you could remotely upgrade the modem, or incorporate new features such as voice control or a digital answering machine.

e. OTP - One Time Programmable

An OTP is a PROM (Programmable Read-Only-Memory) device. It uses standard EPROM, but the package has no window for erasing. This is usually used for limited production runs before a ROM mask is done in order to test code. Once the program is written into the device with a standard EPROM programmer, it cannot be erased or modified. Notice that strictly speaking any bit that is a one can be changed to a zero, but a bit that is a zero cannot be changed into a one.

In addition microcontrollers and CPUs may have 4-, 8-, 16- or even 32-bit data buses, depending on the complexity of the system to be controlled. Specialised processors are also available which include features specific for communications,

math-intensive algorithms. Today many embedded applications require both type of processors, and semiconductor manufacturers have responded by introducing microcontrollers with on-chip DSP capability and DSPs with on-chip microcontrollers [4]. Where parallel processing is essential, transputers can be used. However this usually means learning a new language like Occam [33].

Typically a given microcontroller is available with different memory options - both for ROM and RAM. Mask ROM is the most popular memory option for storing instructions for large production volumes. One problem with mask ROM is that programming, set-up, and engineering changes make it economical only when the systems manufacturer purchases large quantities of identically programmed micros. Even then, one has to consider the cost of making changes when additional bugs are discovered. Using mask ROM can lead to a lot of waste, where large quantities of microcontrollers running older software versions are thrown away. Another factor is that of lead time (the time when the code is submitted to the micro manufacture, to the time the programmed device is received) can be very long (can range form 8 weeks to as bad as 44 weeks!)

However a major advantage of mask ROM is the protection it offers to the software. Many of the other memory options below are not protected against unauthorised intrusion (reverse engineering, modifications, piracy, etc.) Some of the devices like OTPs (see below) offers additional protection as an option, either by encryption or fuse protection. On masked ROM devices, security is not needed; - the only way to read the implanted code would be by using a scanning electron microscope - not a commonly available equipment.

In addition to the common RAM and ROM options there is now some new methods of storing instructions and data. Below is a list of the most common advanced memory options and their characteristics.

a. EEPROM - Electrically Erasable Programmable Read Only Memory

Many microcontrollers have limited amounts of EEPROM on the chip. EEPROM seems more suited (because of its economics) for small amounts of memory that hold a limited number of parameters that may have to be changed from time to time. This type of memory is relatively slow, and the number of erase/write cycles allowed in its lifetime is limited.

b. FLASH (EPROM)

fewer instructions than most microprocessors. It is difficult to write compilers that take full advantage of the powerful instructions available in a microprocessor. Even if assembly language is used, most programmers concentrate on a subset of the instructions. By designing the hardware to use a simplified range of instructions, programming becomes more straightforward. Without the hardware necessary for complex but rarely used instructions, execution cycle time can be reduced even if more instructions are needed to perform a given function. In fact, high-performance, 32-bit RISC microprocessors were designed for use in workstations [8]. Now, they have penetrated the embedded market. The benefits of RISC design simplicity are a smaller chip, smaller pin count, and very low power consumption. Among some of the typical features of a RISC processor are:

- Harvard architecture, (separate buses for instructions and data) allows simultaneous access of program and data, and overlapping of some operations for increased processing performance

- Instruction pipelining, which increases execution speed

- Orthogonal (symmetrical) instruction set for programming simplicity; which allows each instruction to operate on any register or use any addressing mode; instructions have no special combinations, exceptions, restrictions, or side effects.

e. SISC

Specific Instruction Set Computer (SISC) are microcontrollers where more of the general purpose instructions have been eliminated (similar to RISC). The original idea behind the microcontroller was to limit the capabilities of the CPU itself, allowing a complete computer (memory, I/O, interrupts, etc) to fit on the available real estate. At the expense of the more general purpose instructions that make the standard microprocessors (8088, 68000, 32032) so easy to use, the instruction set was designed for the specific purpose of control (powerful bit manipulation, easy and efficient I/O, and so on).

Alternatively embedded systems can make use of programmable controllers and processors developed for particular applications. Digital Signal Processing (DSP) is an expanding field, with specialised integrated circuits available from several sources. A DSP is an example of an Application-Specific Processor (ASIP) [25], which is a CPU optimised for a particular application. Whereas microcontrollers react to and control events, Digital Signal Processors execute repetitive

2 Background Information

As a first step the general architecture of an on-line UPS will be discussed. This serves to situate the inverter within the broader context of its environment.

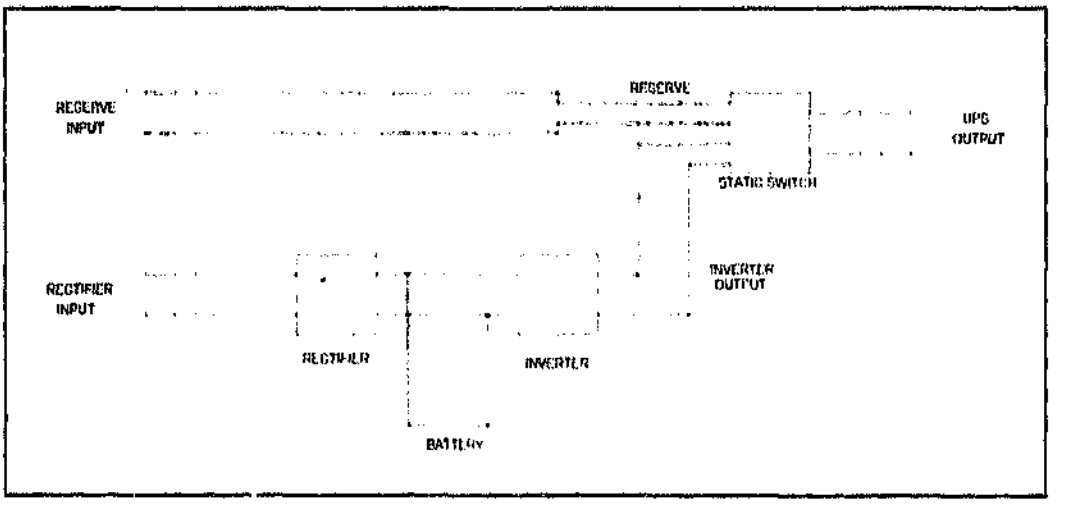


Figure 1: Basic structure of the UPS

The system consists primarily of a solid-state inverter, a rectifier/battery charger, a storage battery and a static transfer switch.

Under normal running conditions, in an on-line system the critical load is continuously supplied by the inverter. The rectifier/battery charger derives power from the commercial ac source and supply power to the inverter while simultaneously float charging the battery.

Upon failure of the commercial ac source the critical load is supplied by the inverter, which without any switching, obtains its power from the storage battery. There is to be no interruption to the critical load upon failure or restoration of the commercial ac source.

Upon restoration of the ac commercial ac source, the rectifier/charger powers the inverter and simultaneously recharges the battery. This must be an automatic function and should not cause any interruption to the critical load.

The very purpose of the UPS implies that it should constantly monitor the Reserve supply. In addition, the critical function of the UPS indicates that it has

the SEAL Quality Management System

- Engineering Manager, Meissner
- Product Developer
- Product Manager

1.5 Applicable Documents

1.5.1 Specifications

None

1.5.2 Standards

a. SEAL QMS Document Creation Template, QS 002, Revision 0.01, 5 April 1994

b. SEAL QMS Document Layout, Presentation and Typesetting Guide, QS 003, Revision 0.01, 1 April 1994

1.5.3 Procedures

None

1.5.4 Guidelines

None

1.5.5 Other Documents

a. SDE 102, Software Development - A General Literature Survey

1.6 Assumptions

None

1.7 ISO 9001 Requirements Traceability

1 Scope

1.1 Introduction

In order to provide practical insight into the development of embedded systems and as a means of investigating the applicability of formal development processes and design methodologies on such systems, a case study is examined. The case study chosen investigates the possibility of using a microcontroller circuit to control a three-phase inverter for an on-line UPS. This example was chosen mainly because it captures well the main features of a typical embedded system; features like real-time constraints, reliability concerns and a strong hardware interaction.

1.2 Purpose

This document defines the nature of the project and its scope. Typically, the requirements analysis exercise is performed by the customer.

1.3 Definitions

SDES	: Software Development for Embedded Systems
UPS	: Uninterruptible Power Supply
IGBT	: Insulated Gate Bipolar Transistor
PLL	: Phase-Locked Loop

1.4 Audience

The audience for this document comprise the various stakeholders of the SEAL, including:

- Full-time staff members of SEAL
- All full-time and part-time post-graduate students associated with SEAL
- Members of the SEAL Management Board
- Head of the Department, Electrical Engineering
- Individuals who will perform internal and external surveillance audits of

Change History

Configuration Control

Project:	SDES
Title:	Requirements Analysis
Doc. Reference:	SDE 104
Created by:	H. Bapoo
Creation Date:	22 June 1995

Document History

Version	Date	Status	Who	Saved as:
0.01	95/06/22	Draft	H.Bapoo	\1995-13\TP\SDE104WP.001
0.02	95/07/28	Draft	H.Bapoo	\1995-13\TP\SDE104WP.002
0.03	96/07/16	Draft	H.Bapoo	\1995-13\TP\SDE104WP.003
1.00	96/08/25	Approved	H.Bapoo	\1995-13\TP\SDE104WP.100

Revision History

Version	Date	Changes
0.01	95/06/22	New Document
0.02	95/07/28	Further development of the specifications
0.03	96/07/16	Further development of the specifications and change of text format to comply with SEAL document presentation guide, QS 003
1.00	96/08/25	Update to revision number and status following Board Approval

Management Authorisation

Version	Date	Status	Management Board Minute Reference
1.00	96/08/25	Approved	Section 2.3 of SDE 592

Change Forecast

This document will be changed whenever there is an amendment to the requirements specifications of the case study.

Table of Contents

Table of Contents ii

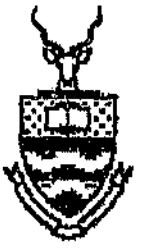
Change History iii
Configuration Control iii
Document History iii
Revision History iii
Management Authorisation iii
Change Forecast iv

1 Scope 1
1.1 Introduction 1
1.2 Purpose 1
1.3 Definitions 1
1.4 Audience 1
1.5 Applicable Documents 2
1.6 Assumptions 2
1.7 ISO 9001 Requirements Traceability 2

2 Background Information 3

3 Scope and Requirements of the Case Study 5

Appendices
Appendix A: Sensed Parameters in an On-Line UPS 7
Appendix B: Limit Detection Activities in an On-Line UPS 10



SDES QMS

Requirements Analysis

Technical Product

Version 1.00

Document Status: Approved

Appendix A: Example of Applying COCOMO to an Embedded System Project

Table A.1:Effort Multipliers for a "Typical" Real-Time Embedded System Development Project

Cost Driver	Rating	Characteristics	Effort Multiplier
Reliability	High	Major financial loss if product recall is caused by software failure	1.15
Data Base Size	Low	Most microcontroller software uses small data bases	0.94
Product Complexity	Very High	Reentrant code, interrupt-handling, communication, command processing	1.30
Execution Time Constraint	High	70 percent of available execution time used	1.11
Main Storage Constraint	Extra High	95 percent of available main storage used	1.56
Virtual Machine Volatility	High	Major change in virtual machine every two months during product development	1.15
Computer turnaround time	Nominal	Computer turnaround time less than four hours	1.00
Analyst Capability	Nominal	Average capability of the team in analysis ability, efficiency, and communication ability	1.00
Applications Experience	Nominal	Three years average experience	1.00
Programmer Capability	Nominal	Average capability of team in programming ability, efficiency, and communication ability	1.00
Virtual Machine Experience	Low	Less than four months average experience with virtual machine	1.10
Prog. Language Experience	Nominal	One year average experience with programming language	1.00
Use of Modern Programming Practices	Low	Experimental use of some modern programming practices	1.00
Use of Software Tools	Very Low	Basic microcontroller tools: assembler, linker, monitor, batch debug aids	1.24
Required Development Schedule	Low	15 percent acceleration compared with TDEV formula	1.08
Product of Effort Multipliers			4.63

possible to test any section of code within the module. It is also possible to alter condition flags to test routes through a program without setting up a complicated set of starting conditions or stepping through a loop a large number of times. By writing data to the output ports, the external circuit can be stimulated in various ways to test its performance. The effects of interrupts (external or internal) can also be tested, although this can only realistically be done at full speed.

Other less common embedded development tools are software performance analyzers like the Hewlett-Packard HP B1487 [44], simulation-based systems like the MOSIM test environment [45] and integrated environment tools like Xray MasterWorks [46]. A software performance analyser usually interfaces with an in-circuit emulator. From the emulation activity the analyser can provide information on the reasons a program is taking too long to execute. It can identify the modules that are slowing down the program and it can determine how intensive is the program's use of data and I/O ports. Simulation-based systems promotes testing in the host environment, i.e. as much as possible of software testing is moved from the target environment to the host environment. That might include unit testing and apart of integration and system testing. Integrated environment tools attempt to extend the automation or semi-automation of the software development for embedded systems to the development, generation and debugging of the codes.

However even with the existing tools software testing remains a tedious exercise. This can reduce the concentration and enthusiasm of testing operators and considering the importance of testing in embedded systems, this can be fatal. Repetition and tedium seems to be two elements that discourage human operators to perform exhaustive software testing [47]. Creative people such as programmers instinctively balk at the idea of spending long hours exercising every function of an enormous program over and over with the only variety being different input values to be compared with different expected outputs. As expressed by Williams [47], if anything cries out for even further automation, it is a way to thoroughly test software.

resources are available it is certainly an excellent investment. The ICE will be discussed in more details.

When using an ICE, the microcontroller is removed from its socket on the target hardware and is replaced by a plug (or pod) and lead connected to the emulator. This in turn is connected to a computer which runs the application program. The signals on the pins of the plug are intended to be identical to those which occur when the target microcontroller is used, so that external interfaces and other components on the circuit board behave as normal. There are generally two types of in-circuit emulators. The first type is based on a simulation of the microcontroller from discrete logic circuits. Various registers, memory blocks, etc. are put into separate circuits which can then be monitored. This approach is quite difficult and often involves a lot of circuitry [3]. It is also more liable to produce inexact or inaccurate results. The second approach uses a bond-out chip, which is simply a version of the microcontroller with many more connections. Extra points are connected to internal data and address buses and other points which can then be monitored as they operate. Some in-circuit emulators interface to standard desk top computers such as the IBM PC or compatible machines. Others use proprietary microprocessor development work stations, with part of the emulation circuitry contained in the unit. Most sophisticated systems will have full desk top computer facilities, including interactive screen displays, disk drives, printer ports and access to shared resources via a network.

A good in-circuit emulator can save a great deal of time when developing software. Because of its interactive nature it can also lead to better designed code and more thorough testing, thus adding to the efficiency of development process [3]. The facilities provided by in-circuit emulators vary. Some allow full speed non-intrusive emulation with all external signals handled exactly as in the real device. Others, not necessarily cheaper, run at reduced speed or do not provide exact emulation of certain features. The features common to all emulators are that they substitute the target microcontroller with their own circuits and use their own memory for the target program. Facilities are provided for the user to insert break points into the code and to display the contents of registers and memory. There is also means of modifying the contents of specific locations. Further control is provided by the single-step mode where the system executes one instructions and then breaks to await a command.

Using these facilities, code modules can be written, prepared and tested separately or together, setting up initial conditions before running the module and observing the results. By running from a labelled address to a break point, it is

- Structure-coverage analysers: Identify the statements, branches and paths in the code that have been exercised.

In addition to those general purpose testing tools, are tools that are particularly pertinent to the embedded systems environment. Those are discussed below:

a. Logic Analyzers

Logic analyzers allow logic circuits to be monitored. They can be used to capture data or events, to measure individual instruction times, or for timing sections of code [13]. In microcontroller applications they are at times used to sample a number of input lines at high speed, storing the result and then display it either as a multi-channel timing diagram or as a state table.

b. Simulators

A simulator runs the microcontroller software on a host machine (such as the PC). Using a simulator, the program can be stepped through while variables values and registers contents are monitored. It can also allow contents of registers and variables to be changed, thus altering the way the program runs. A simulator cannot support real interrupts or devices and often it runs much slower than the real device the program is intended for.

c. Resident Debuggers

A resident debugger runs the program on the microcontroller itself, while showing the progress on the host machine (such as the PC). It has many of the same advantages of the simulator, with the additional benefit of providing insight as to how the program runs on the real target machine. A resident debugger needs to "steal" some resources from the target machine, including: a communications port to communicate with the host, an interrupt to handle single stepping, and a certain amount of memory for the resident part (on the target) of the debugger.

d. In-Circuit Emulator

In-circuit emulators (ICEs) permit both monitoring and interaction with the software. They provide full and total control over the target, while at the same time not requiring any resources from the target. The emulator can either be a stand alone with its own display, or it can interface with a PC. If the financial

9 The Development Tools

Almost every phase of the software development is supported by tools. Poston [49] lists some of the types of testing tools:

- Requirements recorders: Capture requirements in formal specification languages
- Requirements verifiers: Automate the verification of requirements, i.e. checks for ambiguity, inconsistency and statement completeness in the requirements.
- Specification-based test-case generators: Tests how closely the product meets the specification through test cases based on statistical, algorithmic or heuristics methods.
- Requirements-to-test tracers: Automate the requirements-to-test tracing activity.
- Expected-output tools: Predict expected output values prior to testing.
- Metrics reporter: Derive and displays metrics information from source code data.
- Code checker: Identify the fuzzy areas in the program that are error-prone, like misplaced pointers, uninitialised variables and deviations from programming standards.
- Product-based test-case generators: Derive test cases from the analysis of source code.
- Code instrumentor: Insert extra line of codes into the program at strategic points to allow for measurement of structural coverage.
- Capture relay tools: Captures and replays the information flow through a software system.
- Test-harness: Test the interface of each module (through stubs or drives that can simulate the parts of the software not available when testing).

that this very obscurity feature of program listings in FORTH, can be argued as constituting a useful form of encryption for commercial security.

h. Other Languages

COBOL [41], although popular, is oriented towards data processing rather than control. FORTRAN [38] is still used on some embedded systems, although it is being replaced by more structured languages in new designs. RTL/2 [42] and MODULA-2 [43] are used less frequently, but like any language, each has its advocates. Some manufacturers of embedded control systems use their own proprietary languages, often for historical reasons. This practise forces the customer to return to the manufacturer for maintenance or invest in training courses for their own staff. It also makes it difficult for the manufacturer to recruit experienced programmers, or to retain those who see their career prospects being limited.

Very small programs are at times written directly in machine code or by entering hexadecimal codes through array of switches, but this is tedious and unreliable for anything longer than a couple of bytes.

- Individuals who will perform internal and external surveillance audits of the SEAL Quality Management System
- Engineering Manager, Meissner
- Product Developer
- Product Manager

1.5 Applicable Documents

1.5.1 Specifications

None

1.5.2 Standards

a. SEAL QMS Document Creation Template, QS 002, Revision 0.01, 5 April 1994

b. SEAL QMS Document Layout, Presentation and Typesetting Guide, QS 003, Revision 0.01, 1 April 1994

1.5.3 Procedures

None

1.5.4 Guidelines

[1] Bose B. K., "Modern Power Electronics", IEEE Press, USA, 1992

[2] Sen P. C., Premchandran G., "Improved PWM Control Strategy for Inverters and Induction Motor Drives", IEEE IAS Annual Meeting 1982, Paper CH1817-6/82/0000-0823, pp 823-830

[3] Vadivel S., Bhuvaneswari G., Sridhara Rao G., " A Unified Approach to the Real-Time Implementation of Microprocessor-Based PWM Waveform", IEEE Transactions on Power Electronics, Vol. 6, No. 4, October 1991, pp 565-575

[4] Blumenthal M. K., " Why PWM Inverters Beat Other Drive Actuators when it comes to Flexibility and Distribution System Compatibility", IEEE IAS Annual Meeting, 1982, Paper CH 1817-6/82/0000-0536, pp

1 Scope

1.1 Introduction

The nature of the problem, as exposed in the Requirements Analysis (SDE 104), calls for an investigation of the methods available for controlling three-phase inverters. Only when a suitable controlling technique has been selected can we proceed in defining the functional specifications of the software. In addition, in order to implement a phase-locked-loop (PLL) operation in software, its mechanism has to be further investigated.

1.2 Purpose

This document serves to capture the result of a brief survey of methods available for controlling the three-phase inverter. At the end of the survey, a suitable method is chosen. The document then proceeds in laying down the main characteristics of the proposed technique. Lastly, the principles of PLL are introduced and its main characteristics are revealed.

1.3 Definitions

PLL : Phase-Locked-Loop
PWM : Pulse-Width Modulation
SVM : Space Vector Modulation
VCO : Voltage Controlled Oscillator

1.4 Audience

The audience for this document comprise the various stakeholders of the SEAL, including:

- Full-time staff members of SEAL
- All full-time and part-time post-graduate students associated with the SEAL
- Members of the SEAL Management Board
- Head of the Department, Electrical Engineering

Change History

Configuration Control

Project:	SDES
Title:	Inverter Control Techniques and Phase-Locked-Loop
Doc. Reference:	SDE 105
Created by:	H. Bapoo
Creation Date:	22 June 1995

Document History

Version	Date	Status	Who	Saved as:
0.01	95/06/22	Draft	H.Bapoo	\1995-13\TP\SDE105WP.001
0.02	96/07/24	Draft	H.Bapoo	\1995-13\TP\SDE105WP.002
1.00	96/08/25	Approved	H.Bapoo	\1995-13\TP\SDE105WP.100

Revision History

Version	Date	Changes
0.01	95/06/22	New Document
0.02	96/07/24	Document updated to include additional information of inverter control techniques and to comply with SEAL QMS document presentation guide, QS 003

Management Authorisation

Version	Date	Status	Management Board Minut Reference
1.00	96/08/25	Approved	Section 2.3 of SDE 592

Change Forecast

None

Table of Contents

Table of Contents ii

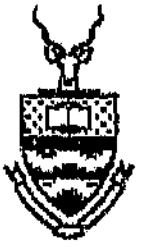
Change History ii
Configuration Control iv
Document History iv
Revision History iv
Management Authorisation iv
Change Forecast iv

1 Scope 1
1.1 Introduction 1
1.2 Purpose 1
1.3 Definitions 1
1.4 Audience 1
1.5 Applicable Documents 2
1.6 Assumptions 5
1.7 ISO 9001 Requirements Traceability 5

2 Overview of Inverter Control Techniques 6

3 Space Vector Modulation 9
3.1 Underlying Concepts 9
3.2 Principle of SVM 9

4 Phase-Locked-Loop 12
4.1 Principle of Phase-Locked Loop 12
4.2 PLL in the Context of the UPS 12



SDES QMS

Inverter Control Techniques and Phase-locked-loop

Technical Product

Version 1.00

Document Status: Approved

Appendix B: Limit Detecti·· ctivities in an On-Line UPS

Once the analogue and digital signals listed in appendix A are scanned, the system should be able to recognise the following conditions:

1. Reserve AC Input Overvoltage (+15% of nominal)
2. Reserve AC Input Undervoltage (-15% of nominal)
3. Rectifier AC Undervoltage (-30% of nominal)
4. Inverter AC Overvoltage (+10% of nominal)
5. UPS Output AC Undervoltage (-10% of nominal)
6. DC Overvoltage[1] (2.35V/Cell)
7. DC Undervoltage (1.72V/Cell)
8. Phase Rotation Error on Rectifier Supply
9. Phase Rotation Error on Reserve Supply
10. 125% Overload Condition
11. 225% Overload Condition
12. Out of Sync Condition (+/- 1Hz from nominal)
13. Power Supply Fail
14. Battery Discharge Condition
15. Low ittery (1.90V/Cell)
16. Imm Shutdown (1.79V/Cell)
17. Battery Shutdown Point[2](1.72V/Cell)
18. Inverter Frequency Fail
19. Inverter OK

[1]Two separate protection systems are required here; the first one (2.35V/Cell) operates with a delay and the second one (2.41V/Cell) is a fast operating one.

[2]Same as DC undervoltage. Ideally two Shutdown Points (e.g. 1.72V/Cell and 1.65V/Cell), controlled by two independent systems, should be used to protect the battery. The second protection system should cause the battery breaker to trip

Purpose: To send a signal on a fuse fail condition.
Sensing Method: sensing wires across the inverter fuse.

c. Over-temperature Condition *[F]*
Purpose: To send a signal under a high temperature condition of the power devices. The signal will be a high or low digital signal.
Sensing Method: thermostat located on the heat-sink of the power switching devices.

d. 300% Overload Condition *[I]*
Purpose: To send a signal under a 300% overload condition.
Sensing Method: By monitoring the Drain to Source voltage of each IGBT. This circuit typically has fast acting optocouplers with high noise rejection.

This sensed current is usually used purely for display purposes.

Sensing Method: Current Transformers (with burden resistors) around the UPS output cables.

g. DC Link Voltage *[F]*
Purpose: Provides feedback of the DC Voltage level.
Sensing Method: sensing wires across the DC bus

h. Battery Current *[F]*
Purpose: Provides feedback of the magnitude and direction of the current flowing into or out of the battery.
Sensing Method: sensing wires across a shunt, in-line with the battery.

i. Input Frequency *[F]*
Purpose: Used by the frequency synchronisation algorithm.
Sensing Method: uses the phase to neutral voltage obtained from the Reserve voltage sensor to obtain the frequency of the Reserve supply.

j. Inverter Output Frequency *[F]*
Purpose: Used by the frequency synchronisation algorithm.
Sensing Method: uses the phase to neutral voltage obtained from the inverter output voltage sensor to obtain the frequency of the inverter output.

k. DC Logic Voltages *[M]*
Purpose: For the proper functionning of the logic circuitry the DC logic voltages (High and Low) should be within the specified tolerance for each device affected.
Sensing Method: Use sensing wires (or copper tracks) at the digital power supplies and at key test points.

Digital Signals

a. DC Cap Fuse Fail *[F]*
Purpose: To send a signal on a fuse fail condition.
Sensing Method: sensing wires across the DC-bus fuse.

b. Inverter Fuse Fail *[F]*

Appendix A: Sensed Parameters in an On-Line UPS

The sensed parameters can be grouped into two parts, analogue signals and digital signals. Both types are listed below. In addition the relative sampling rate required for each sensed signal is indicated by a label in square bracket. The labels are as follows: *[M]*: Moderate (order of seconds); *[F]*: Fast (order of milliseconds); *[I]*: Immediate (order of microseconds).

Analogue Signals

a. Reserve Voltage, phase UV, VW, WU and UN *[M]*
   Purpose: Provides feedback of the Reserve phase to phase and phase to neutral voltage.
   Sensing Method: sensing wires and differential amplifier circuits.

b. Rectifier Voltage, phase UV, VW, WU and UN *[M]*
   Purpose: Provides feedback of the Rectifier phase to phase and phase to neutral voltage.
   Sensing Method: sensing wires and differential amplifier circuits.

c. Inverter Output Voltage, phase UV, VW, WU and UN *[F]*
   Purpose: Provides feedback of the Inverter output phase to phase and phase to neutral voltage.
   Sensing Method: sensing wires and differential amplifier circuits.

d. UPS Output Voltage, phase UV, VW, WU and UN *[F]*
   Purpose: Provides feedback of the UPS output phase to phase and phase to neutral voltage.
   Sensing Method: sensing wires and differential amplifier circuits.

e. Inverter Output Current *[F]*
   Purpose: Provides feedback of the current drawn from the inverter. This sensing is used to recognise 125% and 300% overload conditions.
   Sensing Method: Current Transformers (with burden resistors) around the inverter output cables.

f. UPS Output Current *[M]*
   Purpose: Provides feedback of the current drawn from the UPS.

b. The PWM output should correspond to a balanced set of three-phase voltages, displaced by 120^0.

c. Possible fluctuations in the actual inverter output voltage should be accounted for by the software; i.e. the system should provide voltage feedback and regulation.

d. The inverter should be locked in phase to the reserve supply when the latter is within a predefined frequency window of 49Hz to 51Hz; i.e. the microcontroller system should provide a phase-lock-loop (PLL) control.

e. The microcontroller system is to issue a signal to inhibit the static switch whenever Reserve is out of frequency limit or is out of phase with the inverter output.

f. The control system should be capable of switching off all the IGBTs of the inverter bridge IMMEDIATELY upon either an overload condition or upon receiving of an 'inverter-off' signal. Under both conditions the inverter switches are to be maintained off until the microcontroller is reset. This will allow for the cause of the fault condition to be investigated by the user.

3 Scope and Requirements of the Case Study

The main role of the microcontroller circuit should be to carry out the critical function of driving and controlling the power devices of the inverter bridge. Figure 2 pictures a typical three-phase inverter bridge. The microcontroller should generate the PWM drive pulses that will control the inverter bridge. The drive pulses are fed to the gate inputs G0-G5 of the IGBTs SW0-SW5. The project aims essentially at investigating the design of embedded systems, and as such it will be restricted to the software aspects and the integration of the software system into the existing UPS system. Hardware design and live testing will be restricted to the necessary exercises that will ensure that the system will interface correctly with a typical UPS system. The project will therefore not cover such aspects as transformer designs, filter circuit designs and high voltage testing in general. Those aspects although important in the design of a complete UPS system are outside the scope of the present case study.

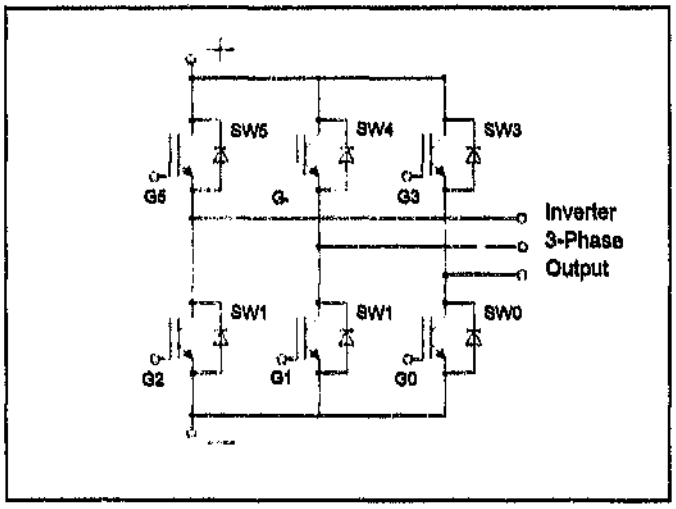


Figure 2: The Three-Phase Inverter Bridge

A breakdown of the main requirements of the system is provided below.

a. The microcontroller should produce a PWM set of signals at the output port.

to perform constant monitoring of all parameters liable to cause failure of the unit itself. Appendix A provides a list of some of those parameters. It reveals the importance of the sensing activity for a UPS. Following the sensing exercise the system must be able to recognise error conditions, so that corrective actions can be taken quickly. Appendix B provides a list of typical conditions that are monitored on an on-line UPS.



SDES QMS

Functional Specifications

Technical Product

Version 1.01

Document Status: Approved

locked loop mechanism described above and can be tackled in a similar way. The role of the microcontroller circuit would be to carry out the PLL action. It will have to capture the necessary information from the system and from those information it will then adjust the PWM drives so that the inverter output follows that of Reserve within a pre-defined frequency window.

4 Phase-Locked-Loop

4.1 Principle of Phase-Locked-Loop

A phase-locked loop (PLL) is a mechanism by means of which the phase of a frequency-modulated oscillator output signal is forced to follow that of the input signal [24] [25].

The essential elements that comprise the PLL, as described by Klapper [26], are the input phase detector, loop filter and amplifier, and the VCO (Voltage Controlled Oscillator), as shown in figure 2.

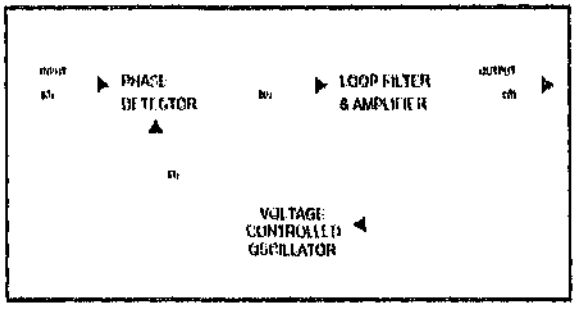


Figure 2: Block diagram of basic PLL operation

The basic operational requirement for the loop is to track the phase of the input signal. This is accomplished by producing in the phase detector a voltage b(t) proportional to the phase difference between the input signal and the VCO signal. This error signal is then amplified, filtered to remove excess noise and other undesirable components, and later applied to the VCO to effect input signal phase tracking. By tracking signal phase, frequency synchronism and frequency tracking are achieved.

4.2 PLL in the Context of the UPS

Any transition between Inverter to Reserve or Reserve to Inverter has to be done with the smallest possible phase change, otherwise a high potential difference will appear at the load during the transition. A smooth transition can be ensured by having the Inverter output following Reserve in phase and frequency; hence the importance of monitoring synchronisation status, as expressed in section 2.d and 2.e of the Requirements Analysis, SDE 104. The principle is similar to the phase-

provide the appropriate switching voltage vectors and remaining in them for the pre-determined times ta, tb and t_0, the inverter output will therefore closely resemble the desired voltage vector. It must be realised that the inverter cannot respond instantaneously to a command of change in $\underline{U}s^*$ and must wait until the completion of a sampling interval to do so.

If the PWM pulses are to be generated using SVM techniques, then the function of the microcontroller will be to generate the appropriate output pulses (high and low) for the appropriate switching times ta, tb and t_0. This together with the additional control features laid in the requirements analysis (see section 2, SDE 104) are in essence the technical requirements of this case study.

$\underline{U}s^*$ (representing the desired three-phase ac voltage set) can be synthesized by switching the relevant voltage vectors Ua and Ub, for appropriate periods of time. This in essence is the principle of SVM.

For example, to produce the desired voltage vector $\underline{U}s^*$ in figure 1, the following switching set will be used:

$$U_0^- \ ,U_1 \ ,U_2 \ ,U_0^+ \ ,U_0^+ \ ,U_2 \ ,U_1 \ ,U_0^-$$

$$|\leftarrow\text{-------}\ \Delta t\ \text{-------}\rightarrow|\leftarrow\text{-------}\ \Delta t\ \text{-------}\rightarrow|$$

A switching set is comprised of a forward switching sequence (first half) and a reverse switching sequence (second half). Δt is known as the sampling interval. U_0^- and U_0^+ are two redundant voltage vectors (of zero magnitude) introduced to ensure a binary Gray code switching sequence. This has for effect to reduce the harmonics in the inverter output waveforms [7].

Since it is the time average of the two switching voltage vectors $\underline{U}a$ and $\underline{U}b$ taken over the sampling interval Δt, that yields the desired voltage vector $\underline{U}s^*$ we can write,

$$\underline{U}ata + \underline{U}btb + \underline{U}_0t_0 = Us^*\,\Delta t \qquad (2)$$

$$ta + tb + t_0 = \Delta t \qquad (3)$$

ta, tb and t_0 are the respective on-times for the output voltage vectors $\underline{U}a$, $\underline{U}b$ and $\underline{U}_0$.

Van der Broeck et al. [22] provide the solution to equations (2) and (3) as,

$$ta = 9/\Pi^2\ UsMag\Delta t\ (Cos\alpha - 1/\sqrt{(3)}Sin\alpha) \qquad (4)$$

$$tb = 6\sqrt{(3)}/\Pi^2\ UsMag\Delta tSin\alpha \qquad (5)$$

$$t_0 = \Delta t - ta - tb \qquad (6)$$

Where α is the angle between the desired vector $\underline{U}s^*$ and the switching voltage vector $\underline{U}a$ of the generalized sector. UsMag is the magnitude of $\underline{U}s^*$.

The on-time for each different inverter switching voltage vector is given by equations (4) to (6) which show that if the magnitude UsMag of the desired voltage $\underline{U}s^*$ is defined along with angle α, the respective on-times ta, tb and t_0 may be calculated. By switching the inverter into the relevant switching states to

3 Space Vector Modulation

3.1 Underlying Concepts

The underlying concepts of SVM are:

- A balanced set of three-phase ac voltages can be represented by a single composite voltage vector, $\underline{U}s$, rotating in the complex-plane [22]; where,

  $$\underline{U}s = Us \exp(j\omega t) \qquad (1)$$

- There are only eight possible switching states for a three-phase inverter, which translates into eight discrete voltage vectors ($\underline{U}_0$ - $\underline{U}_7$) in the same complex-plane.

Those concepts are illustrated in figure 1.

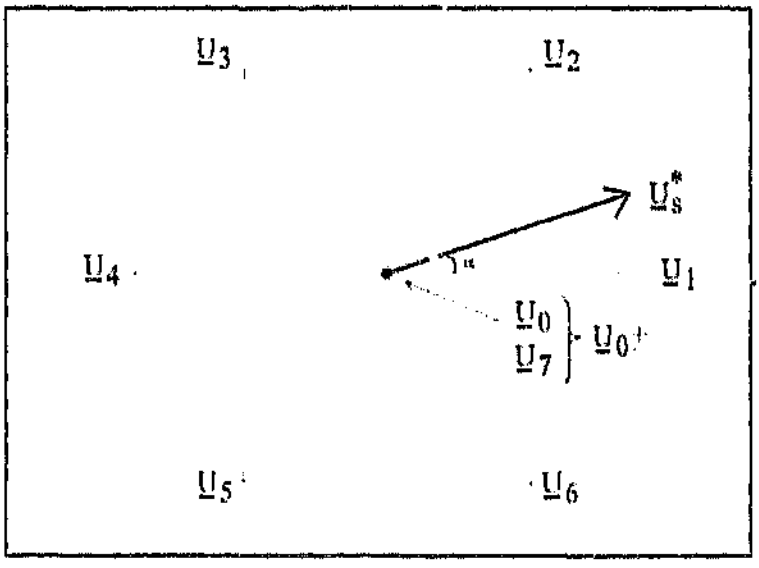


Figure 1: Phasor representation of the switching vectors and the desired vector $\underline{U}s^*$

3.2 Principle of SVM

From figure 1, it is easy to visualise that, at any instant of time, the desired vector

techniques using transputers [19]. It has also been implemented successfully in mixed look-up tables /on-line computations systems [20]. SVM has become prominent, especially in high performance Field Oriented Controlled AC drives [7]. It has also been used in fully software-controlled PWM rectifiers [21]. This is a considerable advantage in the present study since it offers the possibility of using the same technique to control both the rectifier and the inverter of the UPS. The possibility for software re-use in such cases is considerable. Comparison of sinusoidal PWM and space vector PWM [22] shows that the inverter output voltages and currents is less for the space vector method.

recognised that microcontrollers can offer significant advantages to the PWM control in terms of parameter repeatability, protection, advanced diagnostics and maintainability [13],[14]. The microcontroller makes possible monitoring and diagnostic functions that were too costly to provide with hard-wired circuits. These additional functions justify the cost of the microcontroller [14]. However, of the wide array of PWM techniques, few lend themselves to the application of microcontrollers and fewer still can be implemented into on-line and real-time computation algorithms.

For instance the naturally sampled PWM method (or the sinusoidal PWM method), one of the most popular method for industrial applications, is based on non-linear equations which cannot be solved on-line and in real-time. Also in this category is optimal-PWM which requires complex on-line numerical optimisation techniques [7]. The solutions in such cases, if microcontrollers are to be used, are either to use hybrid software/hardware systems [15],[23] where the hardware is responsible to handle the computation-intensive process or else to use EPROM look-up tables. The problem with those approaches is that the computations are not on-line. The system is not flexible and maintainable and in addition it fails to exploit the full capabilities of the microcontroller. The justification of using a microcontroller in such systems is usually not very strong. In addition, a purely look-up table method has large memory requirements, especially at lower fundamental frequency [10]. The few attempts at using sinusoidal PWM in on-line computation techniques (combined with look-up tables) had very low switching frequency limitations (lower than 1 kHz), making them not suitable for the control of devices with high-frequency switching capabilities, such as GTOs and IGBTs [15].

Some of the other techniques devised to circumvent the large computations are based on linear approximations [16]. Others try to substitute the nonlinear equations with less computation intensive equations, like the orthogonal Walsh series [17]. In the former case, although these methods allow on-line computations they usually give rise to high harmonic content, especially at low frequencies, as a result of the approximations they make. The method of equation substitution on the other hand gives rise to highly complex software and is not commonly adopted.

Space Vector Modulation (SVM), the method proposed for this case study, is a low harmonic generation method that is suitable for microcontroller implementation [7]. SVM has been successfully implemented both in applications using look-up tables to store the operating conditions [7] [18] and in on-line, real-time

2 Overview of Inverter Control Techniques

Converters that convert dc to ac are known as inverters. Inverters can be classified into voltage-fed and current-fed types. A voltage-fed inverter is fed by a stiff dc voltage, whereas a current-fed inverter is fed by a stiff dc current source [1]. The present survey focuses on PWM voltage-fed inverters, the type of inverter used on the UPS under consideration.

PWM inverters are used in a wide variety of industrial processes such as uninterruptible power supplies, static frequency changers and variable speed drives [2]. This popularity is mainly due to the capability of these inverters which permit the control of voltage, frequency, and harmonic content in a single power stage employing only one DC source [3]. Based on those criteria the performance of PWM inverters betters other methods of controlling the inverter [4].

Different PWM methods have been developed over the years The objectives have been mainly to obtain maximum voltage gain [5] and to increase the "dead band" between the wanted (fundamental component) and the unwanted spectral components [6] .Among the most common methods for voltage source inverters are [7]:

- naturally sampled PWM
- regularly sampled PWM
- synchronous sampling
- harmonic elimination
- optimal PWM
- Space Vector Modulation

The advantages of PWM techniques over more simple techniques (like the square wave generation) are usually gained at the expense of more complex control and power circuit configurations [8]; hence the trend in capturing the inverter control techniques into microcontroller systems [9],[10],[11]. Microcontrollers were first used in inverters to create novel methods of waveform generation in PWM voltage source inverters [12]. These were mainly academic exercises which strived for output waveform perfection and did not materialise into production units [13]. The main reasons for this were the limitations imposed on instruction speed by earlier microcontrollers, the unit costs, and limited user advantages. With the evolution in microcontroller technology and as the price of the devices goes down, the benefits of microcontrollers to the user becomes more apparent. It is now

Comparison of Real-Time Sine-Wave Generation for PWM Circuits", IEEE Transactions on Power Electronics, Vol. 8, No. 1, January 1993, pp 46-54

[24] Blanchard A., "Phase-Locked Loops", John Wiley and Sons, USA,1976

[25] Manassewistch V., "Frequency Synthesizers Theory and Design", Second Edition, John Wiley and Sons, USA, 1980

[26] Klapper J., Frankie J., T., "Phase-Locked and Frequency Feedback Systems", Academic Press, Inc., 1972, U.S.A

1.5.5 Other Documents

a. SDE 104, Requirements Analysis

1.6 Assumptions

It is assumed that the reader is familiar with inverter systems in general.

1.7 ISO 9001 Requirements Traceability

- ISO 9001 Clause 4.4.4 - Design Input

[14] Kusko A., Galler D., " Survey of Microprocessors in Industrial Motor Drive Systems", IEEE IAS Annual Meeting 1982, Paper CH1817-6/82/0000-0435, pp 435-438

[15] Richardson J., Kukrer O., Implementation of a PWM Regular Sampling Strategy for AC Drives, IEEE Transactions on Power Electronics, Vol. 6, N0.4, October 1991, pp 645-655

[16] Varnovitsky M., "A Microcontroller-Based Control Signal Generator for a Three-Phase Switching Power Inverter", IEEE IAS Annual Meeting, 1982, Paper CH 1817-6/82/0000-0836, pp 836-840

[17] Asumadu J. A., Hoft R. G., "Microprocessor-Based Sinusoidal Waveform Synthesis Using Walsh and Related Orthogonal Functions", IEEE Transactions on Power Electronics, Vol. 4, No. 2, April 1989, pp 234-241

[18] Kazmierkowski M. P., Dzieniakowski M. A., Sulkowski W., "Novel Space Vector Based Current Controllers for PWM-Inverters", IEEE Transactions on Power Electronics, Vol. 6, No. 1, January 1991, pp 158-165

[19] Webster M., Harley R.G., Diana G., Levy D.C., "Space Vector Modulation - A Real-Time Application", The Transactions of the SAIEE, September 1989, pp 38-42

[20] Fukuda S., Iwaji Y., Hasegawa H., "PWM Technique for Inverter with Sinusoidal Output Current", IEEE Transactions of Power Electronics, Vol. 5, No. 1, January 1990, pp 54-60

[21] Kwon B., Min B., "A Fully Software-Controlled PWM Rectifier with Current Link", IEEE Transactions on Industrial Electronics, Vol. 40, No. 3, June 1993, pp 355-363

[22] Van der Broeck H., Skudelny H., Stanke G., "Analysis and Realisation of a Pulse Width Modulator Based on Voltage Space Vectors", IEEE IAS Annual Meeting, 1986, Paper CH 2272-3/86/0000-0244

[23] Mirkazemi-Moud M., Green T.C., Williams B.W., "Analysis and

536-543

[5] Joos G., Ziogas P., Vincenti D., "A Model Reference Adaptive PWM Technique", IEEE Transactions on Power Electronics, Vol. 5, No. 4, October 1990, pp 485-494

[6] Boost M., Ziogas P., " State-of the-Art Carrier PWM Techniques: A Critical Evaluation", IEEE Transactions on Industrial Applications, Vol. 24, No.2, March/April 1988, pp 271-280

[7] Granado J., Harley R.G., Diana G., "Understanding and Designing a Space Vector Pulse-Width-Modulator to Control a Three-Phase Inverter", The Transactions of SAIEE, September 1989, pp 29-27

[8] Acharya G.N., Rao U. M., Shekhawat S.S., Varma R., Perlekar S.D., "A Pulse Width Modulated AC Motor Drive for Mining Locomotives", IEEE IAS Annual Meeting 1982, Paper CH1817-6/82/0000-0500, pp 500-505

[9] Le-Huy H., Dessaint L. A., "An Adaptive Current Control Scheme for PWM Synchronous Motor Drives: Analysis and Simulation", IEEE Transactions on Power Electronics, Vol. 4, No. 4, October 1989, pp 486-495

[10] Bose B. K., Sutherland H. A., "A high-Performance Pulsewidth Modulator for an Inverter-Fed Drive System Using a Microcomputer", IEEE Transactions on Industrial Applications, Vol. IA-19, No. 2, March/April 1983, pp 235-243

[11] Pacaut R., Perret R., Le-Huy H.. "Microprocessor Control of a Current-Fed Induction Motor", IEEE IAS Annual Meeting 1982, Paper CH1817-6/82/0000-0457, pp 457-461

[12] De Carli, " Direct Computation of Optional PWM Voltages" IPEC Tokyo 1983, pp 367-383

[13] Iwanciw P., " An Improved Intelligent A.C. Drive for Industrial Applications", Power Quality Proceeedings, November 1990, pp 118-124

The six switching vectors, U1-U6, are positioned at 60 degrees intervals in the complex plane, giving rise to six sectors, each flanked by a pair of switching vectors. The sector location of Us* at any instant, determines the switching vectors at that particular instant. We can label the switching vectors in a general sector as Ua and Ub, where Ua represents the lower switching vector and Ub represents the upper switching vector. Ua and Ub will take on different values according to the sector as depicted in table 1 below.

Table 1: Distribution of Ua and Ub according to sectors

SECTOR	Ua	Ub
1	U1	U2
2	U2	U3
3	U3	U4
4	U4	U5
5	U5	U6
6	U6	U1

4.4 Vector Magnitude UsMag

The inverter output voltage is controlled in a feedback loop where usMag is the command magnitude of the desired or command vector Us*. usMag is itself determined from the actual inverter output magnitude, vInvActual, which is fed back to the microcontroller. The loop is to compensate for variations in the DC supply or in the load itself, to ensure a constant inverter output voltage. usMag can take any value between usMagMin and usMagMax, which correspond to the highest inverter output voltage and the lowest inverter output voltage permissible, respectively.

4.5 Angle Alpha

alpha is the angle Us* makes with the vector Ua or the lowest switching vector in the particular sector Us* is found. The rate of change of alpha determines the fundamental frequency of the inverter output voltage.

4.6 Sampling Time and Cycle

The position of the desired vector Us* should be adjusted at regular intervals in order to ensure that it revolves in the complex plane at the desired frequency.

4 Specifications for the SVM Algorithm

Space Vector Modulation is the adopted algorithm for the control of the inverter. This choice follows from the technical appraisal documented in the SDE 105.

This section looks at SVM from a software design perspective. The SVM algorithm is broken down to help distinguish its main parameters and the functions or roles of those parameters.

4.1 The Desired Vector Us*

Us* is a vector representation of a balanced set of three phase ac voltages. Us* revolves in the complex plane at the fundamental frequency of the balanced set of three phase voltages. Us* is represented by a magnitude usMag, and the angle alpha it makes with the lowest switching vector. Typically the actual voltages as observed at the inverter output will differ from Us* because of variations in the DC supply and in the load.

4.2 The Switching Vectors

Eight switching vectors are identified, of which two, U0 and U7, are zero vectors and do not contribute to the magnitude of the desired vector Us*. The remaining six vectors, U1 to U6, determine the magnitude and direction of the vector Us* at any point in time. They are displaced at 60 degrees intervals in the complex plane. The switching vectors represent the eight possible switching states of a three-phase inverter bridge. If a digital '0' represents an off position (of the switch) and a digital '1' represents an on position, then the switching vectors can be assigned binary numbers as follows:

U0: 111 000
U1: 100 011
U2: 110 001
U3: 010 101
U4: 011 100
U5: 001 110
U6: 101 010
U7: 000 111

4.3 The Sectors

3 Specifications of the Microcontroller

The Hitachi H8/3048 microcontroller has been chosen for this project. This choice is discussed below.

3.1 Justifying the Choice of Microcontroller

The main factors that lead to the choice of the microcontroller were, in order of importance:

c. Its availability. The microcontroller, as well as its accompanying support tools and documentation, are available in the company.

c. Its high performance features - see appendix 1. Its high proportion of ROM and RAM combined with the fact that it is a 16-bit microcontroller and has a very high clock speed makes it a suitable choice for high-level language programming in a real-time environment.

c. It supports a high-level language, namely C. (This feature is however now common to many microcontrollers)

d. Its price is comparable to other 16-bit microcontrollers on the market.

e. Hitachi is represented locally.

3.2 Adverse Aspects of the Microcontroller

Disadvantages that accompany the choice of that particularly microcontroller can be grouped as follows:

a. It is expensive as compared to the 8-bit microcontrollers commonly used in embedded systems, like the Intel 8048 or 8051.

b. It is a fairly new product. That means that documentation (outside the ones provided by the manufacturer) concerning its use and applications are scarce.

c. It does not support C++ or any Object-Oriented language.

Where a microcontroller is assigned to the MMI, it will typically have the following responsibilities:

a. Data Presentation: Provide display information to drive a Liquid Crystal Display (LCD) display and also to control the RS-232 serial port, if one is fitted.

b. Data Storage: Maintain local storage of selected data. This is particularly important on faults conditions, when recording the time of occurrence, the signal name, and when values are stored. Those data are generally stored in a separate non-volatile type memory (accessed by the microcontroller) as an alarm condition.

2.2 The Inverter Bridge

Both a rectifier bridge and an inverter bridge are required. The subject of this case study is the inverter. The inverter bridge will be a standard DC-to-AC inverter, consisting of an IGBT pack of six IGBTs with free-wheeling diodes (see figure 2, SDE 104). It will be controlled by the microcontroller via a drive card. The drive card will act as an interface between the inverter bridge and the microcontroller card. It will provide isolation between the two circuitry. Smoothing of the PWM output will be achieved by means of capacitors across the bridge and an output transformer.

2 Process Abstraction

Process abstraction involves the partitioning of the hardware elements of the system. It is used to situate the inverter controller within the broader context of the UPS and to identify the general potential functions of the microcontroller. This exercise is particular useful when applying object-oriented techniques which attempt to identify the potential areas of re-use. In particular, it helps in the capture of abstractions during the object oriented analysis (OOA) exercise.

2.1 Processor Activities

Three main potential use of the microcontroller are identified:

a. Data Acquisition: Capture analogue and digital signals from the sensors (with possible preprocessing to eliminate signal artifacts).

b. Data Analysis: Analyse sensed data to assess their meaning; i.e. to determine whether the received data are within the specified limits or not.

c. Data Processing: Manipulate the received data to provide intelligent output to drive the actuators.

In addition, in a fully microcontroller-controlled UPS the man-machine interface (MMI) will be controlled by another microcontroller. Although outside the scope of the present study, the MMI is an important consideration for the UPS. It also affects the way that some of the parameters used by the inverter controller are obtained and calibrated. In particular we note the following basic elements of the MMI:

a. A Clock: The clock will simply provide time and date tracking and will be a standard on-board clock.

b. A Keypad: It will enable data inputs, such as limits for sensed parameters. It will also allow selection of a particular parameter to be displayed. It can be a standard off-the-shelf keypad.

c. An LCD Display: It will display the current selection and the current value of the selected parameter. A 32x8 character LCD display should be adequate.

b. SEAL QMS Document Layout, Presentation and Typesetting Guide, QS 003, Revision 0.01, 1 April 1994

1.5.3 Procedures

None

1.5.4 Guidelines

[1] Hitachi H8/3048 Series Hardware Manual, Semiconductor and IC Division, Hitachi Ltd., 1995, Japan. pp 2 17

1.5.5 Other Documents

a. SDE 104, Requirements Analysis

b. SDE 105, Inverter Control Techniques and Phase-Locked-Loop

1.6 Assumptions

It is assumed that the reader is familiar with inverter systems and the concept of Space Vector Modulation.

1.7 ISO 9001 and ISO 9000-3 Requirements Traceability

- ISO 9001 Clause 4.4- Design Control
- ISO 9000-3 Clause 5.3 - Purchaser's Requirements Specification

usMag: command magnitude value for the desired vector Us*

usMagMax: higher limit for usMag

usMagMin: lower limit for usMag

vInvActual: Actual Inverter voltage, sensed and sampled by the microcontroller

1.4 Audience

The audience for this document comprise the various stakeholders of the SEAL, including:

- Full-time staff members of SEAL
- All full-time and part-time post-graduate students associated with SEAL
- Members of the SEAL Management Board
- Head of the Department, Electrical Engineering
- Individuals who will perform internal and external surveillance audits of the SEAL Quality Management System
- Engineering Manager, Melssner
- Product Developer
- Product Manager

1.5 Applicable Documents

1.5.1 Specifications

None

1.5.2 Standards

a. SEAL QMS Document Creation Template, QS 002, Revision 0.01, 5 April 1994

1 Scope

1.1 Introduction

Following the Requirements Specification layout, the problem of identifying the actual processing functions is addressed. This document looks at issues such as the programming language, the partitioning of the system into subsystems, the inverter control technique, inputs, outputs, interface communications and data flow.

1.2 Purpose

The purpose of the Functional Specifications is to describe how the design will be achieved. It forms part of the contractual documentation against which acceptance is ultimately reviewed. It bridges the gap between the requirements analysis and the design. It translates the requirements analysis into a format comprehensible by both the user and the developer.

1.3 Definitions

fInv: command Inverter frequency value

fInvActual: actual Inverter output frequency

fMax: higher frequency limit for the Inverter output, 51Hz for a 50Hz system

fMin: lower frequency limit for the Inverter output, 49Hz for a 50Hz system

fNominal: nominal output frequency of the Inverter, typically 50 Hz

fRes: actual Reserve frequency

fWindow: frequency window delimited by fMin and fMax

Inverter failure: a condition where either vInvActual or fInvActual is out of limits or both

tSampling: sampling time

Change Forecast

This document will be updated each time changes to the functionality of the system are effected.

Change History

Configuration Control

Project:	SDES
Title:	Functional Specification
Doc. Reference:	SDE 106
Created by:	H. Bapoo
Creation Date:	31 August 1995

Document History

Version	Date	Status	Who	Saved as:
0.01	95/08/31	Draft	H.Bapoo	\1995-13\TP\SDE106WP.001
0.02	96/03/19	Draft	H.Bapoo	\1995-13\TP\SDE106WP.002
1.00	96/03/30	Approved	H.Bapoo	\1995-13\TP\SDE106WP.100
1.01	96/07/18	Approved	H.Bapoo	\1995-13\TP\SDE106WP.101

Revision History

Version	Date	Changes
0.01	95/08/31	New Document
0.02	96/03/19	Further development of the specifications
1.00	96/03/30	Update to revision number and status following Board approval and change of text font to comply with SEAL document presentation guide, QS 003
1.01	96/07/18	Refinement of the specifications. In particular, addition of tables to summarise the specifications

Management Authorisation

Version	Date	Status	Management Board Minute Reference
1.00	96/03/30	Approved	Section 2.4 of SDE 566
1.01	96/07/18	Approved	Section 2.3 of Sde 592

4.8.3 Transitory States 11
4.9 Short Circuit Considerations 13

5 Phase-locked-loop 14
5.1 Interfacing Phase-locked-loop with SVM 14
5.2 Frequency Window 14
5.3 Inverter Failure 14
5.4 Phase-locked-loop Specifications 15

6 Summary of the Functional Specifications 17

Appendix
Appendix A: Main Features of the Microcontroller 22

Table of Contents

Table of Contents ii

Change History iv
Configuration Control iv
Document History iv
Revision History iv
Management Authorisation iv
Change Forecast v

1 Scope 1
1.1 Introduction 1
1.2 Purpose 1
1.3 Definitions 1
1.4 Audience 2
1.5 Applicable Documents 2
1.6 Assumptions 3
1.7 ISO 9001 and ISO 9000-3 Requirements Traceability 3

2 Process Abstraction 4
2.1 Processor Activities 4
2.2 The Inverter Bridge 5

3 Specifications of the Microcontroller 6
3.1 Justifying the Choice of Microcontroller 6
3.2 Adverse Aspects of the Microcontroller 6

4 Specifications for the SVM Algorithm 8
4.1 The Desired Vector Us* 8
4.3 The Sectors 8
4.4 Vector Magnitude UsMag 8
4.5 Angle Alpha 8
4.6 Sampling Time and Cycle 8
4.7 Switching Times 9
4.8 Sequence Considerations 9
4.8.1 Within A Sector 10
4.8.2 On A Change of Sector 10

Table 8: Functional Specifications Matrix on Control Features - Scenario II: Out of Phase Condition

Requirements Traceability: Section 3.d, 3.e, SDE 104			
Step	**Action**	**Expected Observation**	**Requirement Met: ✓/ ✗**
1	Set fRes below the Nominal value	-	
2	Adjust vRes or vInv so as to create about 1 V difference between the two; i.e. simulate a large voltage difference between Reserve and Inverter	Static Switch Inhibit Signal must be Set (low signal) - Inhibit	
		Inverter output frequency should be at the higher limit of the frequency window	
3	Reduce the difference between vRes and vInv; i.e. return to Normal conditions	Static Switch Inhibit Signal must be Off (high signal) - Inhibit lifted	
		Inverter output frequency should follow fRes command input again	
4	Repeat steps 2 and 3 with fRes set above the Nominal value	The same behaviour should be observed, except that this time the inverter output frequency should shift at the lower limit of the frequency window	
5	Move vRes and vInv slightly on opposite side of the Nominal value; e.g. 2.45 and 2.55 - i.e. simulate an opposite direction condition	The same behaviour as in steps 2, 3 and 4 should be observed; i.e. the inverter frequency moves to the limiting value opposite to the side fRes is of the Nominal value. Again, the inverter should be as according to fRes command, when the parameters are adjusted back to the Normal Conditions	

Table 7: Functional Specifications Matrix on Control Features - Scenario I: Inverter Failure Condition

Requirements Traceability: Section 3.c, 3.f, SDE 104			
Step	**Action**	**Expected Observation**	**Requirement Met: ✓/ ✗**
1	Move fRes out of limits (test both limiting cases)	Static Switch Inhibit Signal must be Set (low signal) - Inhibit	
		Inverter output frequency should be at the nominal value, on all three phases	
2	Vary vRes	Waveform pattern should not change	
3	Adjust vRes outside the 2.4V - 2.6V range	-	
4	Move vInv out of limits; i.e. simulate an Inverter Failure	Static Switch Inhibit Signal must be still be Set (low signal) - Inhibit	
		Drive pulses must be off - check at microcontroller output	
5	Move vInv back within limits	Drive pulses should NOT be restored	
		Static Switch Inhibit Signal remains Set (low signal)	
6	Move vRes within 2.4V and 2.6V range - to simulate a zero-crossing	Static Switch Inhibit Signal must be Off (high signal) - Inhibit Lifted	
		Drives pulses should remain off	
7	Reset the microcontroller and repeat steps 1, 3, 4, 5 and 6, but instead of adjusting vInv, adjust fInv	The same behaviour as when vInv was adjusted should be observed	

Table 5: Functional Specifications Matrix on General Output Features

Requirements Traceability: Section 3.a, 3.b, SDE 104			
Step	**Action**	**Expected Observation**	**Requirement Met: ✓/ ✗**
1	Reset the microcontroller	PWM pulses should appear across the gate and source of each of the six IGBTs of the inverter bridge	
		A 50 Hz sinusoidal waveform should appear at each output phase of the inverter	
		The PWM pulses should correspond to a balanced set of three-phase waveforms displaced by 120°	
2	Change the nominal frequency to 60Hz and run the program - this test can be limited to the ICE testing	The same output as in step 1 should be observed, except that the output frequency should now be at 60 Hz	

Table 6:Functional Specifications Matrix on General Feedback Features

Requirements Traceability: Section 3.c, 3.d, SDE 104			
Step	**Action**	**Expected Observation**	**Requirement Met: ✓/ ✗**
1	Vary fRes within the frequency window	Inverter output frequency should follow that of fRes command, on all three phases	
		No change in output voltage magnitude should be observed	
2	Vary vInv within the voltage window	As vInv is adjusted below the nominal value the inverter output magnitude should increase	
		As vInv is adjusted above the nominal value the inverter output magnitude should decrease	

6 Summary of the Functional Specifications

For the purpose of testing, the functional specifications established in this document are captured in scenarios listed in the tables below. Those scenarios provide an easy way of performing an accurate verification and validation of the design without getting into the details of the underlying principles. The tables are to be completed at the end of the testing phase when the final verification and validation are performed. To satisfy all the expected observations in the tables below will be analogous to meeting all the requirements of the case study. fRes and finv refer to the Reserve and Inverter output sensed frequency respectively, and vRes and vInv refer to the Reserve and Inverter output sensed voltage respectively.

microcontroller. An inverter inhibit occurs either:

a. on an inverter failure

b. or under a short-circuit condition

a transfer be allowed. This procedure ensures that the phase change in the UPS output does not exceed 90 degrees. Unfortunately it is achieved at the expense of a break in the output, but this break will be less harmful to the load than a 180 degrees phase change in the supply voltage.

5.4 Phase-locked-loop Specifications

The phase-locked-loop specifications are summarised below:

1. Under normal running conditions the Inverter frequency should follow that of Reserve within fWindow; i.e. fInv should be equal to fRes.

2. In case the Reserve frequency runs out of fWindow then:
   a. fInv should be set to fNominal
   b. the static switch should be inhibited (to prevent transfer to Reserve)

The static switch is inhibited by a signal, sSwitchInhibit, sent on one of the output port of the microcontroller.

3. If an Inverter failure occurs under the above state (2) :
   a. the Inverter should be inhibited
   b. the inhibition on the static switch should be lifted on the next zero crossing of the Reserve supply.

4. If fRes gets back within fWindow after condition (2) has occurred then, subject to fInv being also in phase with fRes,
   a. fInv should be made to follow fRes again
   b. the inhibition on the static switch should be lifted

5. At all times, when fRes is within fWindow, either under normal running conditions or after it has run out of range and come back again, the phase difference between the Reserve frequency, fRes, and the Inverter actual frequency, fInvActual, should be closely monitored.

6. Under an out of phase condition the following steps should be taken:
   a. the static switch should be inhibited
   b. a phase difference reduction mechanism should be initiated to get the two signals, fRes and fInvActual, in phase again

7. The only way to lift the inhibition on the Inverter is by resetting the

5 Phase-Locked-Loop

5.1 Interfacing Phase-Locked-Loop with SVM

The PLL implementation method devised will have to blend with the SVM algorithm. Consequently this factor needs to be build in the conception of the PLL algorithm, right at the beginning.

SVM has no parameter to control the phase angle directly. However it has direct control over the output frequency through the command Inverter frequency, fInv. This means that any phase or frequency changes requested by the PLL algorithm, will have to be reflected by changes in fInv. The PLL algorithm will therefore need to have direct control over fInv.

From this factor and the discussion in the previous section we can already shape the structure of the PLL software algoritm. It will need to have a *phase detection mechanism* and a *phase reduction mechanism*. How those mechanism are implemented is a design issue and will be tackled in the Design Specifications document.

5.2 Frequency Window

In order to ensure a high quality output, the Inverter should only follow the Reserve within a predetermined frequency window (typically +/-1Hz of the nominal value, fNominal). As soon as the Reserve frequency goes out of this window determined by fMin and fMax, the inverter frequency should jump to the nominal value and stay there until the Reserve frequency comes back within the window.

5.3 Inverter Failure

When the Inverter output is phase-locked to Reserve, a failure condition of the Inverter should cause immediate transfer of the UPS output from Inverter output to Reserve. By Inverter failure we understand that either the Inverter voltage or frequency has run out of their respective limits.

However during an out of phase situation, when the Inverter is running at the nominal frequency value, a failure of the Inverter should be followed by a break until the next zero crossing of Reserve is reached. Only at that particular time can

Table 4: Complete forward and reverse switching sequences of the first sector

SECTOR 1 - FORWARD SEQUENCE							
U7	U7-1	U1	U1-2	U2	U2-0	U0	(NEXT INTERVAL) U0
000111	000011	100011	100001	110001	110000	111000	111000

SECTOR 1 - REVERSE SEQUENCE							
U0	U0-2	U2	U2-1	U1	U1-7	U7	(NEXT INTERVAL) U7
111000	110000	110001	100001	100011	000011	000111	000111

In this example, one can again observe how the transitory states ensure that only one bit changes when moving from one state to the other. The first state of the next sampling interval has been included in each case to show clearly that there is no transitory state when moving to another sampling interval.

The observations from this example lead to important specifications that can be summarised as follows:

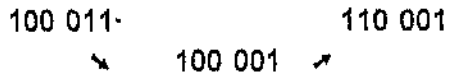
a. There is no transitory state between sampling intervals.
b. The middle transitory state (U1-2 in the example) is the same for a forward and a reverse sequence.
c. The transitory states on the sides, U7-1 and U2-0 in the example, swap positions when the sequence changes.

4.9 Short Circuit Condition

Upon a short circuit condition, identified by the 300% overload detection circuit, all switches are to be put immediately into the zero switching state. This idle state can then only be changed by resetting the microcontroller. This is in accordance with section 3.f of the requirements analysis, SDE 104.

the states U0, Ua, Ub and U7. As such, no switching time is allocated to the transitory states.

For example, with the transitory state included, the transition between U1 and U2 will be as follows:



A simple analysis yields the remaining transitory states, listed in table 3. They are labelled according to the state transition they are bridging; e.g.: U1-2 will be the transitory state from switching vector U1 to switching vector U2, similarly U2-1 will be the transitory state from U2 to U1.

Table 3: The transitory states - Forward Switching Sequence

TRANSITORY STATE	BINARY REPRESENTATION
U1-2	100 001
U2-0	110 000
U2-3	010 001
U3-7	000 101
U3-4	010 100
U4-0	011 000
U4-5	001 100
U5-7	000 110
U5-6	001 010
U6-0	101 000
U6-1	100 010
U1-7	000 011

It can easily be shown that the corresponding transitory states, of a reverse switching sequence, are the same as their forward sequence counterparts, i.e. U2-1 is the same as U1-2, etc. The following example depicts this behaviour.

MOVING TO SECTOR 2 FROM POSITION *(a)		
Pattern Flow - Binary Representation	111 110 010 000	000 010 110 111
Pattern Flow - Switching Vectors	U0 U2 U3 U7	U7 U3 U2 U0
Pattern Flow - (Ua/Ub)	U0 Ua Ub U7	U7 Ub Ua U0
Sequence	Forward	Reverse

MOVING TO SECTOR 2 FROM POSITION *(b)		
Pattern Flow - Binary Representation	000 010 110 111	111 110 010 000
Pattern Flow - Switching Vectors	U7 U3 U2 U0	U0 U2 U3 U7
Pattern Flow - (Ua/Ub)	U7 Ub Ua U0	U0 Ua Ub U7
Sequence	Reverse	Forward

From the above analysis, the following observations can be made, when a change of sector occurs:

a. Ua and Ub are not interchanged
b. U0 and U7 are interchanged as before

From that analysis we further deduce that U0 and U7 are always interchanged after a switching interval.

4.8.3 Transitory States

In the previous section only the first half of the binary representation of the switching states were considered. Once we consider the full binary representation it becomes apparent that contrary to the rule stated previously, more that one bit changes when going from one state to the other; e.g.: moving from U1 to U2: 100 011 → 110 001

In order to correct this discrepancy transitory switching states should be included between each switching state transition. The sole purpose of the transitory states is to ensure a Gray code pattern. They only constitute a brief transition between

We note that a switching set covers two switching intervals, with one interval corresponding to a forward switching sequence and the second interval corresponding to a reverse switching sequence. In the forward switching sequence, the switching state moves from one where all the bottom switches in each inverter leg are all ON and the top ones are OFF (U0) to a state where the bottom switches in each inverter leg are all ON (U7). The reverse switching sequence is just the implementation of the forward switching sequence in the reverse order; doing so ensures a binary Gray code switching sequence.

Some important specifications are retained from the above analysis of the switching sequence. They are exposed below.

4.8.1 Within A Sector

For a normal switching pattern flow, i.e. within a sector, if Ua and Ub represent the first and second vectors of a given sector, then after each sampling period tSampling:

a. the position of Ua and Ub must be interchanged
b. the position of U0 and U7 are also interchanged

4.8.2 On A Change of Sector

The transition from one sector to another deserves special considerations and will be approached by taking an example. Remember that in any transition, only one switch change is allowed. In the following example only the first three digits of the binary representation of the switching vectors, i.e. only the bottom half switches, are considered to simplify the issue.

Table 2: Analysing the effect of sector change ... switching sequence

IN SECTOR 1		
Pattern Flow - Binary Representation	000 100 110 111*(a)	111 110 100 000*(b)
Pattern Flow - Switching Vectors	U7 U1 U2 U0	U0 U2 U1 U7
Pattern Flow - (Ua/Ub)	U7 Ua Ub U0	U0 Ub Ua U7
Sequence	Forward	Reverse

This time interval corresponds to the sampling time tSampling. In any one of such a cycle several activities are to be performed and can be divided into three categories:

a. parameter capture, where values of monitored parameters are sampled

b. information processing, where calculations are performed and conditions tested

c. output operations, where the values of output parameters are updated and/or flags are set (if for example certain limiting conditions were determined in b).

4.7 Switching Times

The projections of Us* onto the bordering two vectors Ua and Ub give the average values that must be produced by each switching voltage vector during the sampling interval. Those values in turn represent the average time that the inverter bridge should spend in each switching state. The time spent in each switching state, Ua, Ub and the zero switching state respectively, are as follows (see section 3.2, SDE 105):

$ta = 9/\Pi^2 * usMag * tSampling(Cos\alpha - 1/\sqrt{3} * Sin\alpha)$ (1)

$tb = 6\sqrt{3}/\Pi^2 * usMag * tSampling * Sin\alpha$ (2)

$t0 = tSampling - ta - tb$ (3)

4.8 Sequence Considerations

It is important to ensure that the switching process is well defined as this helps in the reduction of harmonics in the inverter output waveforms. To guarantee this, SVM uses the two redundant switching voltage vectors U0 and U7 and selects a switching sequence which ensures that each inverter-bridge switch needs only switch once during each sampling interval; in other words a binary Gray code pattern is to be used, like the following switching set shows:

U_0, U_1, U_2, U_7, U_7, U_2, U_1, U_0

|<-------- tSampling ---------->|<-------- tSampling --------->|

where tSampling = sampling interval

vResMax: Maximum allowable value of vRes.

1.4 Audience

The audience for this document comprise the various stakeholders of the SEAL, including:

- Full-time staff members of SEAL
- All full-time and part-time post-graduate students associated with SEAL
- Members of the SEAL Management Board
- Head of the Department, Electrical Engineering
- Individuals who will perform internal and external surveillance audits of the SEAL Quality Management System
- Engineering Manager, Meissner
- Product Developer
- Product Manager

1.5 Applicable Documents

1.5.1 Specifications

None

1.5.2 Standards

a. SEAL QMS Document Creation Template, QS 002, Revision 0.01, 5 April 1994

b. SEAL QMS Document Layout, Presentation and Typesetting Guide, QS 003, Revision 0.01, 1 April 1994

1.5.3 Procedures

None

1.5.4 Guidelines

usMagNominal: corresponding value of usMag at vNominal

uZero: second zero switching vector of a sampling interval and also the first zero switching vector of the next interval.

uZeroTemp: temporary value of uZero; also the next value of uZero

vDiff: voltage difference between Reserve and Inverter; i.e. vInvActual - vRes

vDiffMax: Maximum allowable value of vDiff, set to 5% of vNominal

vInv1: value of vInvActual at the previous sampling interval

vInv2: present sampled value of vInvActual

vInvDiff: difference between vInv2 and vInv1; i.e. vInv2 - vInv1

vInvDirection: Direction of vInvActual. (0 = negative direction, 1 = positive direction)

vNominal: Nominal voltage of the system, typically 230V

vRes: Reserve voltage

vRes1: value of vRes at the previous sampling interval

vRes2: present sampled value of vRes

vResCross: a very small value close to zero, typically 0.01VNominal. It represents a value close to the zero crossing point of the Reserve.

vResDiff: difference between vRes2 and vRes1; i.e. vRes2 - vRes1

vResDirection: Direction of vRes. (0 = negative direction, 1 = positive direction)

vResMin: Minimum allowable value of vRes. Below that value a mains failure condition is recognised and power is drawn from the batteries

tOMinHalf: Lower limit for tOHalf. tOMinHalf = tUpdate + a safety margin (typically 1 µs). tOMinHalf sets an upper limit on usMag

ta: computed switching time; ta is assigned either to tjTemp or tkTemp according to the sequence

tb: computed switching time; tb is assigned either to tjTemp or tkTemp according to the sequence

tClock: clock frequency of the microcontroller

tDelay: delay time to compensate for response time of the IGBTs

tj: time allocated to uj

tjTemp: temporary value of tj; also the next value of tj

tk: time allocated to uk

tkTemp: temporary value of tk; also the next value of tk

tSampling: sampling (or switching) time

tUpdate: Time required for the microcontroller to execute the last assignment operations - after the last subroutine of the *interrupt cycle* has been executed

uj: first non-zero switching vector of the sampling interval.

ujtemp: temporary value of uj; also the next value of uj

uk: second non-zero switching vector in the sampling interval

ujtemp: temporary value of uk; also the next value of uk

usMag: command magnitude value for the desired vector Us*

usMagMax: higher limit for usMag

usMagMin: lower limit for usMag

ITU_TIOR3: Timer I/O Control Register - Channel 3

ITU_TSR0: Timer Status Register - Channel 0
ITU_TSR1: Timer Status Register - Channel 1
ITU_TSR2: Timer Status Register - Channel 2
ITU_TSR3: Timer Status Register - Channel 3

ITU_TSTR: Timer Start Register

lamda: one increment of alpha at fNominal

maxSectorDiv: Number of subdivisions of a sector. This number determines the value of alphaMin, used by the look-up-table generation algorithm. The higher maxSectorDiv, the smaller is alphaMin and the more accurate becomes the switching times calculation algorithm.

P1DDR: Port 1 Data Direction Register
P1DR: Port 1 Data Register

pheta: actual angular displacement from the switching vector U1.
(pheta = w*t)

sector: a 60 degrees sector in the complex plane. The complex plane is divided into 6 sectors, starting from the horizontal or switching vector U1.

sequence: determines the direction of the switching vectors sequence within a sampling interval. +1 if sequence is forward, -1 if sequence is reverse.

SYSCR: System Control Register

tActual: time variable (in seconds). tActual is the time displacement of Us* with respect to the switching vector U1.

t0Half: half the time allocated to the zero switching vectors
t0Half = (tSampling - tj -tk)/2

t0HalfTemp: temporary value of t0Half; also the next value of t0Half

dissolved, following the design stage and the iterative software development. Consequently, although the choice of classes need to be tested against the factors of state, behaviour and identity, this validation exercise does not have to be rigorous, at this stage.

1.3 Definitions

For the purposes of this document, the definitions given in document SDE 106 apply, together with the following:

alpha: angular displacement of Us* from the lowest vector (Ua) of the sector in which it is found (alpha = phata - sector*60)

alphcMin: alphaMin = 60/maxSecDiv

counter: holds the number of times the *main cycle* is executed.

gamma: alphaMin/2

IPRA: Interrupt Priority Register A
IPRB: Interrupt Priority Register B

ITU_GRA0: Integrated Timer Unit General Register - Channel 0
ITU_GRA1: Integrated Timer Unit General Register - Channel 1
ITU_GRA2: Integrated Timer Unit General Register - Channel 2
ITU_GRA3: Integrated Timer Unit General Register - Channel 3

ITU_TCR0: Timer Control Register - Channel 0
ITU_TCR1: Timer Control Register - Channel 1
ITU_TCR2: Timer Control Register - Channel 2
ITU_TCR3: Timer Control Register - Channel 3

ITU_TIER0: Timer Interrupt Enable Register - Channel 0
ITU_TIER1: Timer Interrupt Enable Register - Channel 1
ITU_TIER2: Timer Interrupt Enable Register - Channel 2
ITU_TIER3: Timer Interrupt Enable Register - Channel 3

ITU_TIOR0: Timer I/O Control Register - Channel 0
ITU_TIOR1: Timer I/O Control Register - Channel 1
ITU_TIOR2: Timer I/O Control Register - Channel 2

1 Scope

1.1 Introduction

This document is presents the result of Object-Oriented Analysis (OOA) applied to the case problem. In OOA the functional specifications are translated into a format that befits an Object Oriented Approach.

Analysis will be restricted to the control of the Inverter and will be performed using Booch techniques and notation.

The strategy that will support the present analysis comprises of three steps:

a. A study of the functional specifications and the identification of the main features of the problem domain.

b. For each such feature, attach a scenario that will provide insight as to its role.

c. At the end of each discussion, capture a class or classes for each feature, summarise its main responsibilities and identify its collaborators. This conforms to the CRC (Class, Responsibilities and Collaborators card method mentioned by Booch [1]).

1.2 Purpose

The objectives of the analysis phase are to:

- Identify the classes and objects that are common to the problem domain
- Identify the roles and responsibilities of those classes
- Identify the primary and secondary scenarios
- perform a risk assessment

Note that the analysis phase does not have to be exact. It is more intended to sketch a primary skeleton of classes that will later be reinforced or

Change History

Configuration Control

Project:	SDES QMS
Title:	Software Design Specifications - Part 1
Doc. Reference:	SDE 107
Created by:	H. Bapoo
Creation Date:	20 March 1996

Document History

Version	Date	Status	Who	Saved as:
0.01	96/03/20	Draft	H.Bapoo	\1995-13\TP\SDE107WP.001
1.00	96/07/17	Approved	H.Bapoo	\1995-13\TP\SDE107WP.100

Revision History

Version	Date	Changes
0.01	96/03/20	New Document
1.00	96/07/17	Update to revisison number and status following Board approval and change of document format to comply with SEAL document presentation guide, QS 003

Management Authorisation

Version	Date	Status	Management Board Minute Reference
1.00	96/07/17	Approved	Section 2.4 of SDE 596

Change Forecast

This document will be updated each time there is a change in the functional specifications, either in terms of the elements of the system or in terms of the scenarios.

5.2 Phase-locked-loop 18
5.3 The Magnitude of the Desired Vector, usMag 19
5.3.1 Choosing A Higher Limit for usMag 19
5.3.2 Choosing A Lower Limit for usMag 20
5.3.3 Algorithm to Determine the Correct Value for usMag ... 20
5.3.4 Identified Classes 21
5.4 Sector Location 22
5.4.1 Description 22
5.4.2 Identified Classes 23
5.5 Sequence Update 25
5.5.1 Description 25
5.5.2 Identified Classes 25
5.6 Switching Times 26
5.6.1 Practical Observations About Process Time 26
5.6.2 The Switching Time Order 27
5.6.3 Identified Classes 28
5.7 Switching States 28
5.7.1 Description 28
5.7.2 Identified Classes 29

6 Phase-locked-loop 30
6.1 Phase Difference Reduction Mechanism 30
6.2 Phase Detection Mechanism 33
6.3 The Zero Crossing Point 34
7.4 Identified Classes 35

7 Synchronising the Main Cycle to the Interrupt Cycle .. 36
7.1 Description 36
7.2 Identified Classes 37

8 Transitory States and Igbt Turn-Off Time Considerations . 39

Appendices

Appendix A: Code Listings for SDE 0201.C 41
Appendix B: Code Listings for SDE 0601.M 42

Table of Contents

Table of Contents ii

Change History iv
Configuration Control iv
Document History iv
Revision History iv
Management Authorisation iv
Change Forecast iv

1 Scope 1
1.1 Introduction 1
1.2 Purpose 1
1.3 Definitions 2
1.4 Audience 6
1.5 Applicable Documents 6
1.6 Assumptions 7
1.7 ISO 9001 and ISO 9000-3 Requirements Traceability 7

2 Time and Space Semantics 8

3 The Interrupt Cycle 11
3.1 Description 11
3.2 Identified Classes 13

4 Initialisation Procedure 14
4.1 Configuring the Microcontroller 14
4.2 Initiating the Interrupt Cycle Run 14
4.3 Computing the Constant Terms 14
4.4 Initiating the Phase-locked-loop Process 15
4.5 300% Overload Interrupt Routine 16
4.6 Identified Classes 16

5 The Main Cycle 17
5.1 Sensing Parameters 17
5.1.2 Description 17



SDES QMS

Software Design Specifications
Part 1

Technical Product

Version 1.00

Document Status: Approved

Appendix A: Main Features of the Microcontroller

Below is a list of main features of the Hitachi H8/3048 microcontroller[1]:

a. 128 Kbytes of ROM (Internal) [16 Mbyte ext.]
b. 4 Kbytes of RAM (Internal)
c. Max. Clock Rate of 16 MHz
d. Sixteen 16-bit general registers
e. 16-bit Integrated Timer unit
f. 70 Input/Output pins
g. 8 Input-only pins
h. 30 Internal Interrupts, 7 External Interrupts
i. 10-bit resolution A/D Converter
j. 8-bit resolution D/A Converter
k. Serial Communication Interface
l. Watchdog Timer
m. PWM generation feature

Table 9: Functional Specifications Matrix on Control Features - Scenario III: Inverter Shutdown Condition

Requirements Traceability: Section 3.f, SDE 104			
Step	**Action**	**Expected Observation**	**Requirement Met: ✓/ ✗**
1	Trigger the Inverter-Off signal	All d e pulses should be off	
2	Cancel the Inverter-Off signal	All drive pulses should remain off	
3	Reset the microcontroll.	-	
4	Trigger the 300% Overload signal	All drive pulses should be off	
5	Cancel the 300% Overload signal	All drive pulses should remain off	

sensing parameters were identified in the Requirements Analysis, SDE 104. Therefore it will be judicious to unify our sensor abstractions under a single abstract class sensor which will serve as the immediate base class for all sensing classes like vSensor and fSensor, which sense the voltage and the frequency respectively.

sensor

Responsibilities:

- calibrates the sensor
- computes and returns the current value of the sensed parameter

Collaborators:

- will be a superclass to all classes responsible of sensing activities

vSensor

Responsibilities:

- keeps track of the current voltage

Collaborators:

- is derived from superclass sensor
- will be used by the phase-locked-loop algorithm
- the inverter output voltage will be required in determining usMag

fSensor

Responsibilities:

- keeps track of the current frequency

Collaborators:

- is derived from superclass sensor
- will be used by the phase-locked-loop algorithm
- the inverter output voltage will be required in determining usMag

5.2 Phase-locked-loop

The phase-locked-loop is a very important issue in the development of an inverter design for UPS applications. It remains though, a separate concern to SVM and as such warrants a separate treatment. We choose to include it in the present design because of its vital role in the UPS and because of its role in determining the command inverter frequency, finv, used in the SVM algorithm. Phase-locked-loop is treated in more detail in section 6.0.

5 The Main Cycle

The main functions of the *main cycle* are:

a. To sample the actual inverter output voltage and the Reserve frequency.

b. To perform the phase-locked-loop action.

c. To compute the correct value of usMag, the magnitude of the desired vector.

d. To determine the sector location of the desired vector Us* in the complex plane.

e. To determine the next switching sequence.

f. To compute the switching times and to choose their order of occurrence according to the sequence.

g. To choose the correct switching states and their order of occurrence according to sequence and sector.

Those issues will be considered separately.

5.1 Sensing Parameters

5.1.1 Description

Sensing of the Inverter output voltage, vInvActual, can be performed by a differentiator circuit that feeds an analogue-to-digital converter. This output can then be fed to one of the input ports of the microcontroller. Alternatively the analogue-to-digital conversion operation can be taken care of by the microcontroller itself. Sensing of the Reserve frequency, fRes, can be performed via a frequency counter. Again the data can be captured at one of the input ports of the microcontroller.

5.1.2 Identified Classes

Moving at the UPS level, we find that sensing is an important activity. Several

interpreted as an out of phase status.

4.5 300% Overload Interrupt Routine

Considering the potential damage of short circuits, it is best to declare and make the 300% interrupt routine available right at the beginning. Typically upon a 300% overload condition the software will go into an infinite loop whereby the output port is set to zero. This will switch off all the switches, as specified in section 4.9 of the Functional Specifications, SDE106. The only way to exit the loop will be to reset the microcontroller, meaning that the cause of the short circuit will have to be investigated.

4.6 Identified Classes

From the above discussion, two prominent abstractions appear :

- the start up procedure
- the generation of a look-up-table

The CRC representation for the above two abstractions will be as follows:

startUp	lookUpTable
Responsibilities:	Responsibilities:
- Initialises the microcontroller registers - Computes the constant terms - initiates the *interrupt cycle* - provides an Interrupt routine upon an NMI due to a 300 % overload	- generates a look-up-table of switching times - makes these switching times available
Collaborators:	Collaborators:
- the compute constant terms will have to be made available to all the relevant classes where they are used.	- will need to pass these values to the class responsible of computing the final switching times, which we will name switchingTimes

- uZero = 000 111 (7)
- uj = 110 001 (49)
- uk = 100 011 (35)

Strictly speaking a reverse sequence starts with uZero = 111 000 (56), but in order to ensure correct order for the next sequence, which is a forward sequence, uZero is initialised to 7. Remember that going from one sampling interval to the other, uZero stays the same, therefore initialising uZero to 7 ensures that the next sequence will start correctly with uZero = 7. In other words the first sequence does not respect the SVM algorithm and is only used to ensure correct starting values for the next sequence; the parameters of which are appropriately calculated in the first run of the *main cycle*. As far as the initial switching times are concerned, we choose an arbitrary value of a quarter of the sampling time for each of the four switching states. Again this is just to start the real algorithm.

4.4 Initiating the Phase-locked-loop Process

Since at start up nothing is known about the phase of the Reserve supply with respect to the Inverter supply, an important safety measure will be to assume and simulate an out of phase condition. In this way the phase-locked-loop mechanism, described further down, will be initiated right at the beginning to lock the Inverter supply to Reserve.

An out of phase condition can be initiated by:

a. initiating the static switch inhibit signal, sSwitchInhibit, to 'true'

b. feeding the phase-locked-loop algorithm values such that it will see the Reserve supply and Inverter output as having opposite directions, no matter what the actual state is.

As will be explained in section 6.2, the directions of the two waves, Reserve and Inverter, are important in determining the phase difference. Wave direction is determined by comparing the present value of vRes and vInvActual with the values they had in the preceding sampling interval, vRes2 and vInv2 repectively. At start the phase-locked-loop algorithm will have no real values for vRes2 and vInv2. By initiallising vRes2 to a large value, vBig, larger than its higher limit value, vResMax, and by initiallising vInv2 to a small value, vSmall, smaller than its lower limit value, vInvMin, we will ensure an initial opposing wave direction condition, which will be

4 Initialisation Procedure

The initialisation procedure will be responsible for configuring the system properly to ensure a smooth running of the operations. Five main tasks are recognised and are further developed below.

4.1 Configuring the Microcontroller

Before getting into the *main cycle* or the *interrupt cycle* the microcontroller and its memory should be configured appropriately and memory space provided for the variables. This is all done in the initialisation section, where constant terms are defined, like the clock frequency tClock, e.g. `#define tClock 62.5E-9`, the global variables used by the program are defined and register variables specific to the microcontroller, like SYSCR, are initialised to configure the microcontroller according to its present application. The relevant registers are defined in section 1.3 above. Also in the initialisation procedure, a look-up table storing the switching times, as described further down in section 5.6.1, should be generated.

4.2 Computing the Constant Terms

Considerable savings in execution time can be achieved by pre-evaluating the constant terms (like ka1, kb1, etc.) used throughout the program; especially those used in loops. Those terms could have been pre-calculated and included as pre-processor directives (*#define*). However this would reduce the flexibility of the software, making it more difficult to maintain. A better solution is to compute those terms in the initialisation phase.

4.3 Initiating the Interrupt Cycle Run

Another function of the initialisation procedure is to provide access to the closed-loop of the *interrupt cycle*. For the *interrupt cycle* to run it needs to be fed the appropriate switching states and switching times. Correct values of those parameters will only be obtained after a complete run of the *main cycle*. The initialisation procedure will only be responsible to set it running and not to provide it with correct values.

We choose to start in sector 1. If we want to start on a forward sequence after the first run of the *main cycle*, then the initialisation step should feed the *interrupt cycle* with reverse sequence values as follows:

3.2 Identified Classes

Following the above discussion, each timer interrupt will be governed by a separate class, which leads to the following CRC representation:

timerT3Interrupt

Responsibilities:

- stops timer T3
- sets timer T0
- starts timer T0
- returns control to the *main cycle*

Collaborators:

- will have to obtain the new value for the timer register from the *main cycle*

timerT0Interrupt

Responsibilities:

- stops timer T0
- sets output port to state uj
- sets timer T1
- starts timer T1
- returns control to the *main cycle*

Collaborators:

- will have to obtain the new values for the timer register and for the port register from the *main cycle*

timerT1Interrupt

Responsibilities:

- stops timer T1
- sets output port to state uk
- sets timer T2
- starts timer T2
- returns control to the *main cycle*

Collaborators:

- will have to obtain the new values for the timer register and for the port register from the *main cycle*

timerT2Interrupt

Responsibilities:

- stops timer T2
- sets output port to state uZero
- sets timer T3
- starts timer T3
- returns control to the *main cycle*

Collaborators:

- will have to obtain the new values for the timer register and for the port register from the *main cycle*

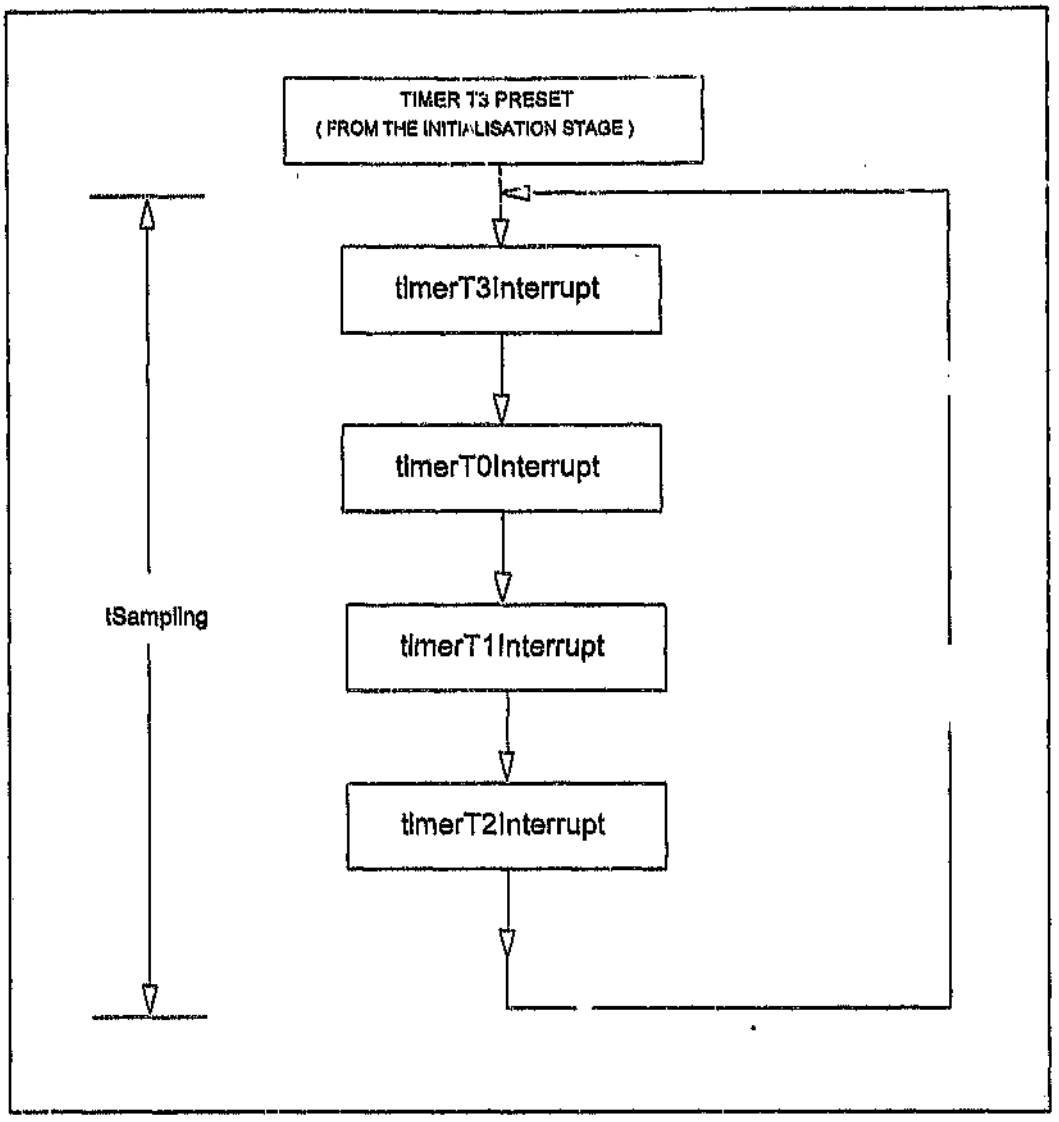


Figure 2: Flow Diagram - The *interrupt cycle*

Lastly, as stated previously, the *interrupt cycle* covers the whole sampling interval which it subdivides into four intervals, each one corresponding to a different switching state and each one initiated by a timer interrupt. One should not confuse the time span which the *interrupt cycle* controls, which is the same as the sampling interval, with the execution time spent by the four subroutines, which represents only a fraction of the sampling interval. Each time a subroutine is completed, i.e. each time another timer is set, execution passes back to the *main cycle*, where most of the sampling time is spent.

3 The Interrupt Cycle

3.1 Description

The *Interrupt cycle* will control four of the five timers of the Integrated Timer Unit of the H83048 and will consist of four subroutines which we will label timerT0InterruptRoutine, timerT1InterruptRoutine, timerT2InterruptRoutine and timerT3InterruptRoutine.

Each subroutine, with the exeception of timerT3InterruptRoutine, performs the following tasks:

a. Clear or stop the timer responsible for the last interrupt.
b. Change the output port configuration to the next switching state.
c. Set the next timer for the appropriate time.
d. Return control to the *main cycle*.

The switching time and switching state into which the subroutine configures the microcontroller are those calculated in the previous run of the *main cycle*. This is in accordance to section 3 above.

In addition, to protect the microcontroller against short circuits due to overlapping of the switching waveforms, all the switches are to be first opened for a time delay period before any switching transition is performed. Only then is the next switching state output. The time delay is slightly greater than the turn-off time of the device, which for an IGBT is typically 0.2 microseconds.

The timerT3InterruptRoutine does not have to set the output port because there is no change in the switching state when going from one sampling interval to the other. This was shown in section 4.8 of the Functional Specifications, SDE 104.

A flow diagram of the *Interrupt cycle* is shown in figure 2. Note that the *interrupt cycle* has a closed-loop configuration. The last interrupt routine of the sequence, timerT2InterruptRoutine, sets back the first timer of the sequence, timer T3. This ensures that one switching sequence follows the other endlessly. Direction of sequence, switching states and switching times are controlled by the choice of the state variables uj, uk, uZero and the values of the time variables tj, tk and t0Half.

One turn around the *main cycle* loop has to take less than the specified sampling time, tSampling, in order to ensure a totally new and correct set of parameter values after each sampling interval.

The background activities, will be contained in a different loop designated by the term *interrupt cycle*. It will consist of output operations regulated by timer interrupts. The values of the output parameters will be updated every loop.

A single timer can be used for this problem. It can be set and reset at different values at each interrupt. However, since we do not expect any of the other timers to be used at this stage, the *interrupt cycle* will be driven by four different timers, one for each of the four switching states in a sampling interval. This decision has for advantage a better separation of concern and a more loosely coupled system. The *interrupt cycle* has to cover the whole sampling time, tSampling, in order to ensure a faithful implementation of SVM.

This first exercise served to assess the time and space risks. Identifying the technical risks early in the development helps to avoid some of the common pitfalls and enables a better management of future strategic and tactical trade-offs of design.

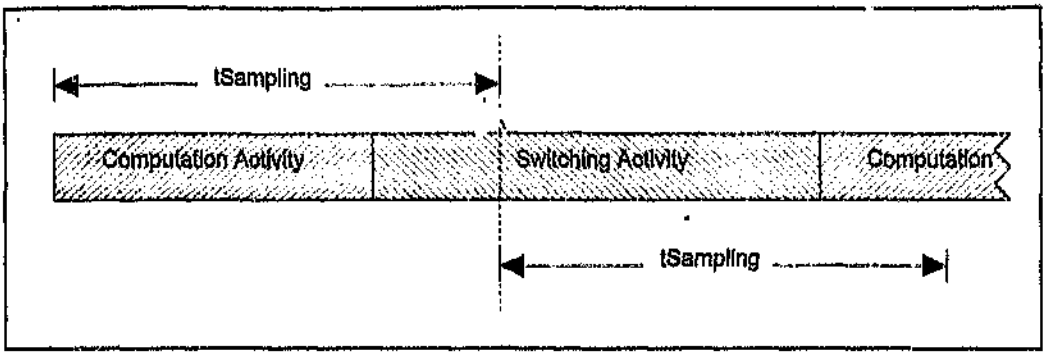


Figure 1: Conflict between the Computation and Switching activities

Two problems are uncovered through this analysis. First, the output ports of the microcontroller are idle during computation activity because at this stage the parameter values for the switching execution is not known yet. Second, since the switching execution has to cover a full sampling time, it overlaps onto the next sampling interval. This is a direct consequence of the time spent on computational activities.

Both observations expose the impracticality of a purely sequential approach.

The solution is twofold:

- First the switching times and states used or output to the ports in any sampling time interval should be the ones calculated in the previous interval. Hence the switching times are ready right at the beginning of a sampling time interval.

- An interrupt-driven, background-foreground activity setup should be used to ensure that the output port is always active, even during computation activities of the microcontroller.

The foreground activities, will be contained in that section of the program that we will call *main cycle*. It will be responsible for:

- parameter capture, where values of monitored parameters are sampled

- information processing, where calculations are performed and conditions tested

2 Time and Space Semantics

Two main activities are recognised in the implementation of SVM in real-time, one is the computational process whereby the switching times and sequences are calculated, the second is the control of the output ports where the calculated values are mapped onto the output port.

Having established the existence of the main operations of the process and having defined their roles, we have to decide upon their time and space semantics. Space and time limitations are strong elements of concern and risks in real-time embedded systems.

As far as space considerations are concerned our first attempt will be to limit the whole program code to fit onto the 128 kbytes of on-chip ROM and the data to be stored in the 4 kbytes of on-chip RAM, i.e. our first attempt will be to operate in single-chip mode. The advantage being a compact system that does not require external memory.

The decision about the time to complete the operations will primarily be governed by the sampling rate or switching frequency. Higher switching frequency will assume a faster execution of the operations. For the present problem, a high switching frequency is desirable in order to reduce the low order harmonics content of the Inverter output. The execution speed of the operations itself will depend upon the efficiency of the compiler and the processing speed of the microcontroller.

The first step is to adopt an appropriate time frame that will take into consideration the computational time and the output operations.

Adding the three switching times ta, tb and t0 should give exactly tSampling. From this observation it becomes obvious that, assuming parallel processing is not used, it is not possible in a given cycle, to determine the switching states and times as well as to have the inverter bridge to stay in those states during the same calculated times. Figure 1 depicts the situation.

[1] Booch G., Object-Oriented Analysis and Design, Second Edition, Benjamin/Cummings Publishing Co., 1994

1.5.5 Other Documents

a. SDE 104, Requirements Analysis
b. SDE 105, Proposed Technique For The Control Of The Three-Phase Inverter
c. SDE 106, Functional Specifications

1.6 Assumptions

It is assumed that the reader is familiar with:

- Inverter systems and the concept of Space Vector Modulation.
- Object Oriented Analysis techniques

1.7 ISO 9001 and ISO 9000-3 Requirements Traceability

- ISO 9001 (1994) Clause 4.4 - Design Control
- ISO 9000-3 (1991) Clause 5.6.2 - Design