Software Engineering in a British Defence Project in 1970

Ian Pyle(✉)

York, UK

Abstract. This software development project took five years from 1968, and engaged 50, rising to 130, people. They were programmers from a commercial company, together with scientists from a government laboratory. In the infancy of software engineering, completely new techniques were established to carry out the task, based on system theory, which are described here. The project was completed in 1973.

Keywords: Historical · Multi-computer · Radar · Linesman

1 Introduction

Linesman was the UK's Air Defence System for the 1970s. After all the hardware had been installed, including all the radars and about 10 specially-designed computers at the building called L1 at West Drayton (near Heathrow airport), it was found in about 1967 that the contractor could not make the software for the computers, called the Radar Data Processing System (RDPS). This is the story of how that software was developed. It explains the software engineering approach and techniques used to design and produce the software for the RDPS.

A team from Harwell was assigned to assist the contractor, and subsequently (after a major review) to lead the software development. Stimulated by the dramatic publication of the report [1] of the NATO conference at Garmisch,"Software Engineering", we decided to apply the principles of systems engineering to the development of the software for the RDPS.

1.1 Classification

Because it was essentially military, all work on the project was subject to the United Kingdom's Official Secrets Act, which meant that everything about it had to be treated carefully to prevent disclosure. Some of the rules were ridiculous (for example, the order code of the computers used was classified!) and the general limitation of information made it extremely difficult at the beginning to find out what the project was about and what the problems were. When there was a public crisis about it, we discovered more from the daily newspapers that we had from the internal documents.

I. Pyle—Formerly of AERE Harwell.

D. Kreps et al. (Eds.): HCC13 2018, IFIP AICT 537, pp. 16–30, 2018.
https://doi.org/10.1007/978-3-319-99605-9_2

Specifically, the Software Engineering approach and techniques had to be classified ("Restricted"), and the present paper, over thirty years after the documents were written, is probably the first public exposure of the details. (Earlier publications [2, 3] about techniques used had to be guardedly disguised to avoid disclosure).

1.2 Contractual Relationships

In common with most British Defence Procurement projects at the time, Linesman was carried out as a "cost-plus" contract, involving two principal parties: the Design Authority (a private company, in this case The Plessey Company) who had to develop and install the equipment, and the R&D Authority (a government laboratory, in the case the Royal Radar Establishment: RRE, Malvern) who had to monitor the technical work being done by the Design Authority, and confirm that it was appropriate for the situation. In addition, the prospective end-user (a military body, in this case the Royal Air Force) was involved in negotiations, particularly for training and eventual hand-over.

Harwell only became involved at a late stage in the project, initially in support of RRE, and with increasing despair as we discovered the inadequacy of the monitoring and software engineering, in spite of increasing the numbers of people involved. After a full review of the project, it was decided that Harwell should also take a leading part in the work of the Design Authority, as well as continuing to support the R&D Authority. The present paper is about the approach we took within the Design Authority to achieve the required behaviour of the RDPS.

2 Background

Previous attempts to make the software for the RDPS had been overwhelmed by the complexity of the requirements. In evidence to a parliamentary enquiry [4], a representative of Plessey said "The software task involves the writing of more than a quarter of a million 48-bit words and there existed nowhere any previous experience of the programming of so large and complex an on-line real time system for air defence purposes." The company specialised in electronics and telephone exchanges, and they recruited programmers who were only familiar with sequential programming. There were no system programmers.

"An unsuccessful attempt was made by the Company in 1967 to obtain assistance from software houses, but experience relevant to the Linesman task did not exist anywhere. It was only from AERE that it was found possible to obtain a substantial element of high grade programming effort." Staff from A.E.R.E. Harwell (particularly nuclear physicists, but initially from the computing group) were well experienced in using computers for data gathering and control of experimental apparatus with extensive electronics. A small team was allocated to the project in 1968, and we proposed a novel approach, which was strongly influenced by the NATO Software Engineering report; the style was essentially what was later described as the V-model.

The fully structured software system design applied System Theory to this problem, by insisting on well defined components and interactions between them, at each

element in a hierarchy. This brought out the importance of documentation for the project, and led to the specification of a documentation structure to describe the software, from design through implementation and testing to handover. A document called the Software Standard [5], was written to define the rules of operation for the software development, including comprehensive authorisation and change control. There were no software tools to support the development, so all the documentation was produced in a form that could be properly checked by human readers. Essentially, all the information needed to ensure consistency on each item was included in a single document. Between one document and another, relevant sections of text were identical.

3 Systems

From the totality of the RDPS, down to individual software modules, everything was treated as a "system" having two aspects: (a) the whole, with specific behaviour and specific allocated resources; and (b) a set of components, each of which could be a smaller system, which interacted to provide the behaviour of the whole.

The Software Standard introduced nomenclature for the significant kinds of software items in this project: "suites" for software involving several computers, concerned with a particular function; "packages" for software within a single computer dealing with a particular function; and "modules" as software components that are actually compiled. Completed working software (as it became produced, and on which other software would be built), was called a "presystem".

The review of the system requirements identified three major areas: primary (relating to the maintenance of a recognised air picture of the UK air space), secondary (relating to maintaining the computer system and intercommunication between its physical components in the presence of all possible failures), and tertiary (providing facilities for program production and maintenance). The software staff were accordingly divided into three departments to cover these areas.

The standard recognised that the requirements could not be stated in full before the design of the system began; the design had to be undertaken with the assumption that the requirements (particularly the primary requirements) would change in detail, although the secondary and tertiary requirements would be more stable.

4 Structure and Content of the Software Standard

Because the intended users of the software standard were experienced programmers with little knowledge of higher level software design or software engineering, the standard was written with great detail specific to the Linesman Radar Data Processing System. There were seven parts, covering different aspects of the process of software development (see Appendix A for the list of contents, showing the sizes of each section, and the date when the version was issued). The introduction explained the purpose of the standard, specifically identifying the particular objectives, as

(a) to assist individual programmers to produce programs which properly interface with others to make a coherent working whole;
(b) to provide tangible output for other designers and management before implementation;
(c) to become the reference basis for the final implementation;
(d) to provide the information needed for debugging and testing, and for training the eventual users about the total design of the system; and
(e) to accumulate information about effort for future estimation.

For the three main parts (Requirements, Design, Implementation) there are a number of sections, in each of which the standard explains the scope of the section, the activities to be done, the products that would result, the criteria for completion, and the possible need for repetition.

4.1 Requirements

Originally, the requirements for the RDPS had been expressed from the point of view of the operators, with detailed descriptions of their consoles, the messages they might receive, and the responses they might give to them.

Following the appointment of Harwell staff to oversee the project, the point of view was shifted to the computer system, and the requirements were re-cast in terms of the data that had to be held, and the processes that were to be applied to these data. Specifically, the standard called for detailed specification of the observed objects (which we called the "World Model"), historical background to be taken into account, the monitoring, assessment and control procedures, outline specification of the computing system, specification of operational events, and specification of transactions. For each of these, the standard explained the scope of the particular requirement, the activities to be carried out to write the relevant text, the document to be produced (and by whom it was to be checked), the criterion for completion of the work, and conditions under which some repetition of the work may have been necessary.

4.2 Design

The design of the RDPS software was explained in a sequence of sections, descending top-down in detail from the overall software system design, through presystems, suites and packages, to individual modules. For each level, the standard gave the issues to be taken into account, the decisions to be made, and the outputs to be produced. A co-ordinator had to be appointed to take continuing responsibility for the design through all the development. For example, for suite design, the design team had to decide how to distribute the suite's functionality between packages in different computers, and what communication was needed between these. They had to decide how the resources allocated for the suite were to be distributed among its constituent packages. At the bottom level, the module designer had to give identifiers for the module parameters, define the data structures needed, prepare a module algorithm including debug points, and prepare an outline test plan.

When each design document had been written to part 3 (see "Planning and Progress measurement" below), and approved by the appropriate authority, the design team had completed that stage of the work, and was assigned another item of work.

4.3 Implementation and Testing

Implementation was bottom-up, with section dealing with ascending levels of complexity, from modules to packages, suites, presystems and the whole system. For each, the standard identified the focus of the work, as the element was assembled from its constituents and tested to show that it conformed to its specification. For example, for a package, the activities included deciding the sequence of modules to be assembled, describing the test environment for off-line and on-line testing, labelling of all test points with in the package, preparation of test data and expected trace points; the results included corresponding output values, package execution times and module use counts, together with the overall package size. A log was produced recording all significant activities.

Presystem implementation and testing formed a significant part of the overall demonstration of progress of the development project.

4.4 Resources

The standard recognised that the utilisation of resources and maintenance of records was vital for successful engineering, and defined the duties of a specific group that had responsibility for this work; they were called the System Keepers. Their work involved both clerical and analytical activities. Records were kept of the system, the presystems, and the various computer roles, including lists and software logs. The system keepers were also responsible for the preservation and maintenance of paper tapes and magnetic tapes for the system.

They also monitored, analysed and assessed the utilisation of resources within the system; specifically, for the use of core stores, CPUs, and the inter-computer communication highways. The system keepers had to liaise with coordinators for each element of the design.

4.5 Approval of Documents

Every document had to be approved in some way before it was considered to be satisfactory. There were several levels of approval, depending on the kind of document. Every document began as "Draft," when its author could modify it arbitrarily. Other levels were "Edition," "Issue," and "Certified," following checking and approval by appropriate bodies, including an Internal Approval Authority and an External Approval Authority. The standard laid down the criteria (depending on the type of the document) and corresponding approval authorities.

4.6 Appraisal and Preparation for Handover

In preparation for handover of the developed system to the intended users, particular documents had to be prepared and checked, covering manuals and software details for subsequent use and maintenance of the system. The standard specified what these must contain, and how they were to be checked. For example, the final part (part 7) of each document defining a software item had to be prepared by the System Keepers based on material from periodic appraisals, including a summary of the usage of the item, the faults attributed to it, details of amendments to it, together with an appraisal of the item's effectiveness and notes on any foreseen modifications or extensions.

5 Results

The application of the software standard resulted in the production of a very large number of documents, of which only a few can be mentioned here.

5.1 Requirements

The first and most significant challenge was to find a form of words which everyone would agree was the purpose of the RDPS. What was agreed was very different from anything written previously (although the words had been spoken). This was Part 1 of the top-level design document [6]. The text is given in Appendix B. Notice that the overall purpose was developed there into layers of facilities, called primary (Application Software), secondary (System Foundation) and tertiary (Program Preparation). By analysing the dependencies between these layers, it became clear that, in contrast with earlier priorities, the development of the RDPS software had to focus first on Program Preparation, then on the System Foundation, and, only after those had been designed, on the Application Software. Until then, effort had been concentrated on the sequence of primary facilities needed, and got nowhere.

5.2 Planning and Progress Measurement

The documentation structure identified in the Software Standard provided the basis for planning the development, and for measuring progress, by the stages of parts of the various documents.

For each item, whatever its size, the documentation was produced as a sequence of parts, with time-gaps between the writing of specific stages. In general, Part 1 specified the behaviour of the item and the resources allocated for it. Part 2 specified the constituents of the item, defining the behaviour and resources for each constituent. Part 3 described the interactions between the constituents, explaining how they jointly produced the behaviour given in Part 1. On completion of Part 3, the document was checked and then distributed for use as a basis for the design of the constituent systems. In parallel, Part 4 was written to specify the order of assembly of the constituents, with the rationale. Part 5 specified the tests to be carried out (after the constituents had been produced) which confirm that the interactions of Part 3, and the behaviour of Part 1,

were achieved. When the constituents had been produced, Part 6 was written to record the results of the tests identified in Part 4. Part 7 recorded the handover of this item as a constituent of the next higher system.

A different principle was used for large-scale progress measurement, in terms of demonstrated facilities on the computers; these were called "Presystems" (see below).

6 Presystems

Seven presystems were defined, mostly chronological, but for the infrastructure distinguishing between "basic" facilities need to progress, and "advanced" facilities to give additional functionality.

6.1 Programming Support

Presystem 1 provided the Basic Programming Support, i.e. compilation and assembly of modules, preparation of magnetic tapes for loading into on-line computers, on-line debugging aids, and simple reporting from post mortem dumps. This used an XL4 computer with an operating system called OS090.

Presystem 2 provided additional power for the workload of developing the RDPS software, as a computer bureau, using an additional computer, an ICL 1902A, with disk files and fast input/output. The operating system was called XANOS.

6.2 Foundation

Presystem 3 was the Basic System Foundation, providing scheduling of tasks on the on-line computers, communication between them, loading of software from magnetic tapes, control of the allocation of computers to specific rôles, on-line debugging facilities, peripheral handling and regular checking for faults.

Presystem 4 was for Advanced System Foundation, for responding to detected faults and making appropriate changes to allocations, for reconstituting data after a reconfiguration, for diagnosing reported faults, and for driving special equipment to investigate faulty electronic units. Subsequently, Presystem 4 was split, because a major hardware upgrade to the computers affected testing. The revised Presystem 4 contained the facilities independent of the upgrade, and a further presystem, called 4T, was defined to contain the additional facilities dependent on the upgrade.

6.3 Applications

Presystem 5 handled the various kinds of buttons and lamps at operators' consoles, including a command language interpreter for recognising operators' key sequences and taking appropriate actions.

Presystem 6 dealt with autonomous processes (detecting relevant changes) and maintaining the world model (both live and simulated).

Presystem 7 handled height finder equipment and secondary radar interactions.

7 Management

During the early period, before the design had made much progress, the management were extremely uncomfortable, because there was no way of estimating the amount of effort or time that would be required. However, once the structure in terms of presystems and suites had been identified, an overall plan could be prepared, and priorities identified for the allocation of staff and computer resources. Then as work on implementation progressed, confidence increased, although there were still problems of interaction between the hardware changes found necessary and the availability of the computers for debugging.

8 Quality

Surprisingly, the Software Standard did not mention quality: there was nothing about Quality Control or Quality Assurance, and I been unable to find any occurrence of the word "quality" in the document.

In retrospect, I think that the reason was that we considered quality to be intrinsic to the structure and procedures in the standard, not an "add-on": by having text copied verbatim from one document to another, and by insisting that each document had a clearly-defined focus, with well-structured contents and checking by all relevant parties, we presumed that the quality would automatically be there.

In practice, we did set up a quality control unit (a good management decision!), but its responsibilities were administrative rather than technical: they had to confirm that each document had been written and checked in accordance with the rules set down in the Software Standard, according to the type of the document.

9 Education

No-one in the project was a software engineer as we now understand the term. We were working from first principles, and disseminating our experience and insights as the project progressed. Some of the ideas of software structure were unfamiliar to the programmers, who were experienced mainly in a dialect of Coral 66 [7]: supposedly for real-time programming, but with no features (such as multi-programming, or interrupt handling) that are now known to be required for that field (see Jackson [8]). The ideas of the software standard were spread by example and mentoring. (I ordered fifty copies of the NATO Software Engineering report for distribution to staff and management).

9.1 Tasks

A particular problem was that most programmers did not understand the concept of a task, or process, in a multiprogramming environment. (The seminal description by Wirth [9] was still years ahead). Only after the successful implementation of the OLOS suite (On-Line Operating System) were most staff convinced that this was a viable and essential feature of the software.

10 Handover

The RDPS was handed over to the Royal Air Force in July 1973, at a formal meeting which reviewed all the software according to the structure and terminology of the Software Standard. The R.A.F. gave a demonstration of the completed system to invited guests on 18th December 1973.

11 Origin of Software Standard

The ideas behind the document were largely derived from experience in the design of software at Harwell (e.g. the Fortran compiler for the Atlas computer, and the HUW system [10]) and "systems" thinking, helped by the NATO report on Software Engineering. However, for the context of the Linesman RDPS, it was recognized that principles were not enough, and great care was needed to express the ideas in concrete form for this project.

A group of three people: myself, J.R. Taylor (Harwell), and D.M. England (Plessey), spent about three months in the spring of 1970 carefully writing the document, in preparation for the eventual decision to change the direction of the software development to the method explained here. The edition of the document that I preserved is dated July–November 1970.

12 Conclusion

This was an innovative software engineering project, on a larger scale than had previously been encountered in the U.K. Over a hundred people were employed on the development of software for a major defence requirement. The software engineering principles used were simple (and, some said, boring!), without any particular "method" or silver bullet: just carefully-focussed well-structured writing, with extensive appropriate checking. The constraints were severe, yet a system was produced which was handed over and accepted by the military. Because of its military nature, little has been published about it hitherto.

'This changed everything' because the work reported here established basic principles for Software Engineering: in System Theory. The idea of a system as a set of interacting components, whose properties exceeded those of the individual components, was reified here to provide an effective method of developing software. The success of the project was attributable to the many people involved in the development of the ideas as well as in the actual software development. The pressure on us all was immense, conscious of the political and military implications of the project. This was a major team effort, and the experiences of those involved enabled them to carry out subsequent (classified) projects with great success. The major failure was the consequence of the Official Secrets Act: details about the method could not be widely disseminated. The learning was passed on to other developments only by the people who had been involved (from Harwell and Plessey, and, to a lesser extent, the Royal Air Force). I recognised that that the resulting system would be unlikely to be fully satisfactory, and (as a result of experience with implementing this approach to software

engineering) in an Infotech State of the Art lecture [11], I proposed the acronym "DITHER" to express the overall process: Design, Implement, Test, Evaluate, Replace. Thus hoped that this work would enable the commissioning authority (i.e. the British Ministry of Defence) to do a better job next time. Unfortunately it did not. After Linesman, the United Kingdom Air Defence Ground Environment (UKADGE) was out-sourced to a different company which encountered problems that were different but with no significant improvement in outcome. But Harwell continued to develop software systems for sensitive projects, with great success.

What had previously been a failing project, severely criticised in the press and the subject of parliamentary questioning [12], dropped out of the news. It was working.

13 About This Document

Although the work described here was carried out in the years around 1970, it could not be published then. The present document was started in 2010, as a "memoir" recording my recollections about the project. Because it referred to classified information, it was submitted for security clearance in May 2017, and was cleared for publication. Further details are available from the author at <ian.pyle@cantab.net>.

Appendix A – Contents of the Software Standard

			Date	Pages
1	INTRODUCTION		24.7.70	4 pp
	1	Function of Document		1/1
	2	Design Environment		1/1
	3	Objectives of Documentation		1/1
	4	Structure of the Standard		1/1
	5	Designation and Approval of Documents		1/4
	6	Status of Standard and Revision Procedures		1/4
2	SPECIFICATION OF OPERATIONAL REQUIREMENTS		26.10.70	10 pp
	1	Introduction		2/1
	2	Outline Requirements		2/1
	3	Detailed Specification of Observed Objects		2/2
	4	Historical Assessment		2/3
	5	Specification of Monitoring, Assessment and Control Procedures		2/3
	6	Outline Specification of Computing System		2/5
	7	Specification of Operational Events		2/7
	8	Specification of Transactions		2/8
3	DESIGN		24.7.70	14 pp
	1	Introduction		3/1
	2	Overall Software System Design		3/2
	3	Presystem Design		3/5
	4	Suite Design		3/6
	5	Package Design		3/8
	6	Module Design		3/13

(*continued*)

(*continued*)

			Date	Pages
4	IMPLEMENTATION AND TESTING		24.7.70	14 pp
	1	Introduction		4/1
	2	Module Implementation and Testing		4/1
	3	Package Assembly and Testing		4/3
	4	Suite Assembly and Testing		4/7
	5	Presystem Assembly and Testing		4/9
	6	System Assembly and Testing		4/12
5	UTILISATION OF SYSTEM RESOURCES AND MAINTENANCE OF RECORDS		26.10.70	10pp
	1	Introduction		5/1
	2	Maintenance of Records		5/2
	3	Monitoring, Analysis, and Assessment of Resource Utilisation		5/6
6	APPROVAL of DOCUMENTS		26.10.70	4 pp
	1	Introduction		6/1
	2	Levels of Approval		6/1
	3	The Internal Approval Authority		6/3
	4	Technical Editing		6/4
7	APPRAISAL AND PREPARATION FOR HANDOVER		27.11.70	5pp
	1	Introduction		7/1
	2	User Manuals and User Guides		7/1
	3	Maintenance		7/3
	4	Appraisal		7/4
Appendices			Date	Pages
A	DOCUMENT DESIGNATION, STATUS AND HANDLING		24.7.70	17 pp
	A1	Document Designation		11
	A2	Document Status and Handling		5
	A3	Integration of Existing Documents into the Document Structure		1
B	DOCUMENT PREPARATION		24.7.70	40 pp
	B1	Document Style and Layout		11
	B2	Design Document Formats		29
C	DIAGRAMMATIC AIDS		31.8.70	40 pp
	C1	Flowcharts		19
	C2	Decision Tables		2
	C3	Hierarchy Diagrams and Matrices		6
	C4	Communication Diagrams and Matrices		5
	C5	Data Structure Diagrams		3
	C6	Timing and Sequence Charts		2
	C7	Core Maps		
D	RECORD PREPARATION			
	D1	Lists, Indexes, and Glossaries		
	D2	Load Lists		
	D3	Data Lists		
	D4	Software Logs		

(*continued*)

(*continued*)

Appendices			Date	Pages
E	TECHNIQUES			
	E1	Programming in Minicoral		
	E2	Programming in XAL		
	E3	Off-line Testing		
	E4	On-line Testing and Field Trials		
F	REFERENCE MATERIAL			
	F1	Glossary		
	F2	Bibliography		

Appendix B – The RDPS Software System: Overall Requirements

The following is the text of Part 1, Sect. 1, of SDA 000000/3, edition 5, which was distributed for review, with the intent of raising its status to Issue 2 on 6th September 1971.

1. FUNCTION

The Radar Data Processing System (RDPS) provides the information for the central co-ordination and controlling element of the Linesman U.K. Air Defence System. Its function is to maintain and display representations of the airspace activity of defence interest in the U.K. Air Defence Region and approaches (both live and simulated) as a 'Recognised Air Picture', on the basis of information received from radar, data links, operator injections, and an environment simulator. The information is to be displayed on equipment in the L1 building, and transmitted over data links for use elsewhere, especially to Continental Early Warning Stations.

1.1 PRIMARY FACILITIES

The facilities to be provided by the RDPS (Hardware and Software) are defined in document SRA 000099. The following list summarises the primary facilities that are required to fulfil the function defined above.

Automatic input/output of information over data links from and to equipments external to the RDPS, in locations outside the L1 building, with some checking of content.

- Input of information from operators by means of keys and rolling balls at their consoles, using displays to assist the construction of data fields and assist in checking prior to injection.
- Output of information to operators by means of various displays: Electronic Data Display (EDD), Marked Radar Display (MRD), Label Plan Display (LPD), Higher Formation Display (HFD), General Situation Display (GSD) and Totes,

The information processing needed to provide these facilities calls for:

- construction, maintenance and updating of a world model, including transformation and vetting of inputs;
- periodic checks to determine whether significant or critical conditions have developed in the world model;
- messages played out by the system indicating condition known to be of interest;
- routine playout of data from the world model for display or transmission to another system;
- playout of specific data from the world model to console operators on demand;
- computations performed on the world model on request from console operators.

The software which provides these primary facilities is called Applications Software (or functional suites).

1.2 SECONDARY FACILITIES

In order that the primary facilities of the RDPS may be fulfilled, there is a secondary function, namely to enable the RDPS to operate continuously; giving service in the face of the inevitable faults (or other anomalous occurrences) to hardware and software, informing the system controller when any malfunction is suspected, and taking such automatic recovery procedures as are possible.

A discussion of the availability to be expected is given in Part 3. The secondary facilities therefore are:

(a) detecting and reporting suspected faults,
(b) degrading gracefully and recovering rapidly in the event of a fault occurring,
(c) assisting engineers in the investigation and cure of faults,
(d) reacting to instructions from the system controller concerning the hardware of the RDPS.

The information processing needed to provide these secondary facilities calls for:

- construction, maintenance and updating of system records of the hardware state and the roles occupied;
- reloading of programs into computers when needed;
- reconstitution of data for programs between loading and use;
- using a healthy computer to obtain information for diagnosis of hardware faults;
- checking hardware for presence and correct functioning;
- communication with system controller about suspicious events;
- providing an on-line debugging environment.

The software which provides these secondary facilities is called Foundation Software (or system foundation).

1.3 TERTIARY FUNCTION AND FACILITIES

The tertiary function needed is to prepare information to be transferred into the main system, and post process information transferred out of it. During the development stages, the tertiary function is very important, as it calls for program preparation facilities. The same facilities are also needed during the operational use of the RDPS, in order to repair software errors and make enhancements, although the level of activity

will be lower. This function calls for a general sequential job execution facility, which is defined in Appendix D.

1.4 OTHER FUNCTIONS AND FACILITIES

Finally there are number of administrative and documentation tasks which will be processed by computer in the interests of accuracy and efficiency. These include planning and maintaining: resource utilisation records, document numbering schemes, and indexes.

References

1. Naur, P., Randell, B. (eds.): 1968 NATO Conference, "Software Engineering", Report on a Conference Sponsored by the NATO SCIENCE COMMITTEE, Garmisch, Germany, 7–11 October 1968. Scientific Affairs Division NATO, Brussels, Belgium (1968). http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF
2. Pyle, I.C.: Some techniques in multi-computer system software design. Softw. Pract. Experience **2**, 43–54 (1972)
3. Stenson, J.: Reconfiguration of computers in critical systems. In: Computing with Real Time Systems: Proceedings of First European Seminar on Real-Time Programming, A.E.R.E. Harwell, 5–7 April 1971. Transcripta Books (1972)
4. Fourth report from the Select Committee on Science and Technology, Session 1970–71: The prospects for the United Kingdom Computer Industry in the 1970's, vol. 3: Appendix 42: The Linesman and Mediator Projects (Plessey). HMSO 621-III (1971)
5. Plessey document ("Restricted"): "Software Standard", reference number SSJ 000001 of 27.11.70, later changed to MSJ 000001 (1970)
6. Plessey document ("Restricted"): "The RDPS Software System" reference number SDA 000000 (1971)
7. Woodward, P.M.: Official Definition of CORAL 66. HMSO, November 1970. (ISBN 0114702217)
8. Jackson, K.: Adding real-time features to CORAL 66 via the operating system. In: Computing with Real Time Systems: Proceedings of First European Seminar on Real-Time Programming, A.E.R.E. Harwell, 5–7 April 1971. Transcripta Books (1972)
9. Wirth, N.: Toward a discipline of real-time programming. Commun. ACM **20**(8), 577–583 (1977). (ISSN: 0001-0782)

10. McLatchie, R.C.F.: HUW, an interactive computer system on IBM System 360/65. In: SEAS XIV, Conference, Grenoble (1969)
11. Pyle, I.C.: Hierarchies: an ordered approach to software design. Infotech State of the Art lecture 15 June 1971, in Infotech State of the Art report, Software Engineering, pp. 255–268 (1972)
12. Select committee on Science and Technology (Subcommittee A) Minutes of evidence: Wednesday 31st March 1971; Annex E: Question 5. Military/Civilian systems for Air Defence and Air Traffic Control (Linesman/Mediator)