ALVEY PROGRAMME

SOFTWARE ENGINEERING

SOFTWARE RELIABILITY AND METRICS PROGRAMME

Prepared by The Centre for Software Reliability for the Alvey Directorate

July 1984

Centre for Software Reliability

The City University
Northampton Square
London EC1V 0HB

Computing Laboratory
University of Newcastle upon Tyne
Newcastle upon Tyne NE1 7RU

INDUSTRY
MOD
The Alvey Directorate
DTI
SERC

Alvey Directorate
Dept. of Trade & Industry
Millbank Tower
Millbank
London SW1P 4QU

SOFTWARE RELIABILITY AND METRICS PROGRAMME

Prepared by The Centre for Software Reliability for the Alvey Directorate

Contents

1. Introduction

The ability to produce software of known quality to a given cost is critical to the future of the UK IT industry. Together with other aspects of the Alvey software engineering programme, software reliability research aims to provide industry with the means of designing and building reliable, cost-effective software systems. It is therefore primarily aimed at improving the engineering procedures used in software production, although 'software' may be considered to be a generic term encompassing the design aspect of hardware systems.

It is apparent that software engineering will become a key world industry. If the UK is to retain or extend its share of the world IT market, then it must back its established excellence in innovative product-oriented research with a new appreciation of the commercial needs of industry. Furthermore it is vital that research in this area is begun as rapidly as possible. Already the USA is using the DoD STARS initiative as a means of funding work on software measurement activities, while the Japanese emphasis on product quality is a national preoccupation.

In order to remain competitive, UK industry will need to produce software of guaranteed high quality. This demands the ability both to understand and improve the software development *process* (i.e. the techniques, tools, tasks and procedures by which software is produced) and to evaluate quantitatively the *product* produced by that process. Only then will software managers and production staff be able to introduce the vital commercial factors of cost and resources into the production process and to determine a profitable balance between production costs, maintenance overheads and customer requirements.

It seems obvious that companies which are prepared to provide warranties on their software will have a major competitive advantage. The need to base warranties on accurate quality evaluation throughout the development process will require measurement to be incorporated into the development process, as well as the industrialisation of the development process envisaged in Information Systems Factories (ISFs). Furthermore, new standards of evaluation must be applied to the techniques, tools and procedures incorporated into ISFs.

UK industry cannot afford to wait for its competitors to provide the basic research in this area. Feedback from measurement and evaluation from the first generation of Integrated Project Support Environments (IPSEs) will determine the nature of the second and third generation products. It is not possible to design integrated environments which will take up the results of non-UK research, even if such work remains in the public domain. It would be folly to allow our competitors two or three years head start in feeding reliability research into their working methods and software products. The success of the Software Engineering initiative of the Alvey Directorate will rest on its ability to demonstrate the effectiveness of new methods of development; research into software measurement and evaluation is essential to that goal.

2. Software Reliability

2.1 What is Software Reliability?

Software reliability may be formally defined as the probability of failure-free operation of the software for a given time period in a specific environment. Alternative metrics, such as rate of occurrence of failures, may be of interest in certain applications. It must be stressed that these metrics are quantitative and not merely qualitative. They utilise the terminology of probability theory and statistics to yield a formal numerical measure of software reliability. Their use enables a rational and scientific approach to be adopted towards issues of quality in software engineering, a notable achievement in an area which is notorious for its soft and anecdotal nature.

The aim of software reliability modelling is to estimate the current reliability, and predict future reliability, of a software system via measurements of the system. A reliability model should, for example, permit a software manager or engineer to answer the following questions:
—what is the current reliability of the system?
—how long will it take for a given level of reliability to be achieved?
—what will it cost to maintain this system once it is released to customers?

The ability to model reliability successfully will permit a number of research issues to be addressed:
—what effect have different development techniques on reliability?
—what measurements of software in its various stages of development relate to reliability?
—what influence does customer use have on reliability?

These issues will determine to what extent it is possible to guarantee software reliability and how this may be achieved.

2.2 The Importance of Reliability

During the early years of programming, it was difficult enough to produce a system which provided the functionality required by its users, without worrying about reliability or any other quality considerations. Computers acquired the reputation of being unreliable and difficult to use. Members of the public ceased to be surprised by bills for ridiculous amounts, but the sphere of influence of computers and software was sufficiently limited for them not to feel unduly worried.

However, the last few years have seen a massive increase in the use of computers throughout society. Familiarity with computers has led the general public to expect higher quality software, and moves to computerise life-critical and cost-critical systems have heightened this new awareness.

Failure of a major company as a result of unreliable business software or any loss of life due to software failures in avionic systems or major chemical processing plants could cause a massive public reaction against the use of computers, which would have very serious consequences for all the industrialised countries. Events such as the software synchronisation failure which delayed the launch of the Space Shuttle, and the false report of an enemy attack proclaimed by the WWMCCS computerised defense control system, may be a harbinger of worse things to come if software reliability issues are not resolved.

It would be foolish to remain complacent about current levels of software reliability when advances in hardware and the requirements of industry are encouraging the development of ever larger and more complex software. Advances in the functional capability of software must be matched by improved methods of achieving reliability targets, and of knowing that they have been attained.

Unreliability of software, however measured, now accounts for an increasingly substantial proportion of life-cycle costs. This situation will become untenable unless present trends and practices are improved. Unfortunately the bulk of software reliability research has been relatively unproductive and has failed to produce useful and applicable results. The reasons for this include:

(a) The poor perception on the part of researchers of the real problems faced by project managers.

(b) The fragmented nature of software reliability research, which has led to research projects of little practical or theoretical importance.

(c) The fact that much research has been reported in a highly technical and often confusing way which has not encouraged a wider readership.

(d) The failure on the part of industry to provide objective feedback on the practical performance of theoretical results.

(e) The perceived costs and difficulties of collecting, storing and accessing information on software reliability behaviour.

(f) The secretive attitude adopted by many organisations towards the dissemination of data concerning software reliability.

2.3 Software Certification

Guarantees of software quality will have much greater credibility if they are based upon independent assessment. The proposed National Quality Certification Centre is the obvious agency for such independent assessment and would be a major user of new measurement and modelling techniques resulting from the software reliability programme.

The NQCC would provide software producers with formal certification of software products. However, a producer would need to base a warranty on commercial considerations. When certification confirmed that a product exhibited a certain level of reliability, the producer could make informed decisions about maintenance costs before providing a warranty to customers.

Thus a software guarantee would have two components: independent certification by the NQCC and a cost-based warranty provided by the producer (which may also need to be agreed with the NQCC).

Current practice in software quality assurance concentrates on the software development process. This involves agreeing the procedures necessary to complete the individual processes involved in each stage of software development (theoretically from requirements definition until replacement but in practice usually from specification until delivery), and checking that each process is completed in accordance with the agreed procedures. Such procedures usually involve conducting a series of design reviews or audits, adhering to agreed coding standards, producing a number of standard documents and performing certain classes of testing.

The two major deficiencies of this approach to quality assurance are:

(i) the relationship between system reliability, and the procedures and process at each stage of development, is completely unquantified and in some cases is no more than a pious hope;
(ii) there is no requirement to certify the product itself rather than the process that produces it: direct product measurement or assessment is ignored.

Accordingly, there is no way of assessing appropriate software development techniques for various reliability requirements and no way of evaluating whether reliability requirements have actually been achieved. The software reliability research programme presented here will provide the basic information and techniques needed to resolve these problems and place software certification on a scientific and quantitative basis.

2.4 Aims of the Research Programme

The objective of the software reliability research programme is to provide the capability to design and build reliable software systems in the most cost-effective way. Thereby, software producers in the UK will be in a position to deliver and warrant software of competitive quality.

In order to achieve this objective a number of co-ordinated research activities need to be initiated:

(a) to understand the causes of system failure, in order to develop and validate models of system reliability which permit software reliability to be specified, measured and controlled;

(b) to understand and measure the component processes of the software lifecycle in order to develop and validate quantitative models of software production;

(c) to improve and evaluate software development methods and techniques in terms of the reliability of resultant software;

(d) to produce software development rules and procedures such that given reliability requirements may be achieved;

(e) to develop resource models incorporating models of software development processes and software quality such that costs to the customer and producer can be predicted and controlled;

(f) to provide procedures and rules to enable the development of reliable software to be adequately managed.

The programme is divided into two themes, achievement and assessment, since it is important not only to *evaluate* the reliability of software but also to provide the techniques and tools which assist in the *production* of reliable software. The anticipated goals of these areas of research are summarised in Figure 1. It should be emphasised that the various categories that are used to partition the research goals will in practice be very closely interrelated as one part of the research programme feeds back information into another. For example, the results of technique evaluation should feed back into the development of handbooks to guide production and management; cost and quality considerations and model validation exercises will refine the original models of reliability and the development process.

Figure 1

Strategy	Innovation and Understanding	Integration and Implementation	Exploitation and Evaluation
Achievement	Models of the software development process	Procedures for management, and for production staff e.g. Handbooks for reliable software development	—Database of software reliability and development —Procedures for software certification e.g. data collection from IPSEs
Assessment	Models of software reliability	—Cost and quality models —Procedures for quality assurance	—Software development technique and tool evaluation —Model evaluation

3. Relationship to Other Alvey Areas

Technical innovation, particularly on a scale anticipated by the Alvey report, will create many opportunities to undertake tasks previously considered impossible by virtue of their complexity, or to find more efficient ways of achieving current objectives by using the new enabling technologies. Hopefully, as these opportunities present themselves to UK industry, there will be enthusiasm for their commercial exploitation.

Initially it may be sufficient to have an innovative or novel product to secure a foothold in world markets. However, to secure these markets it is necessary for UK industry to produce competitive products of known and demonstrable reliability and quality, delivered to the customer on schedule. Considerable attention must be given to areas not traditionally thought of as being research interests, for example: resource management, project managment, design control, production engineering, quality control and assurance, reliability measurement and demonstration.

Whilst this report addresses those reliability research areas necessary for the creation and support of innovative software products, it is clear that a concern for reliability should imbue the other parts of the Alvey programme. The following sections discuss this issue in more detail.

3.1 Software Engineering

Other specialist panels will support the SE strategy. These will be concerned with *process/management* and *formal methods*.

Since the software reliability programme is primarily concerned with quantification of the software development process, there must be strong interfaces with the management and process research programme. The introduction of quality and productivity metrics, with associated models of the development process, will facilitate the development of management support tools. Such tools will provide managers with quantitative monitoring and control facilities. For example, cost models incorporating quality considerations will improve product planning and estimation. Improved reliability models will permit program certification techniques to be established. They will also provide information about the suitability of a product for release. Models are needed to predict maintenance and support costs, to complement those for development and production costs, since the former are usually a greater proportion of life-cycle costs.

The most important contribution the reliability programme will make is in the evaluation of the effectiveness of software development techniques and tools. This will necessitate close co-operation between the *reliability* and *process* panels so that appropriate information is collected by the latter to feed into the models and metrics of the former. It is vital, for meaningful statistical analysis, that this co-operation should begin before the research programmes start. Many previous evaluation attempts of this kind have been rendered worthless by collecting inappropriate data, expensively, at the wrong time.

Similar considerations apply to the formal methods area. Formal methods of software development bring mathematical precision and rigour to efforts to enhance software quality. It is vital that the level of reliability such methods can establish is evaluated, as well as the cost of applying them, since they should eventually form part of the tool-kit of the practising software engineer.

Furthermore, formal methods will need to be extended to permit the specification of reliability and quality requirements.

IPSE development, which is a major thrust of the SE strategy, will be carried out using input from the reliability panel in its early stages and will be sensitive to advances in the reliability area. The results of metric and model development, for example, must influence the automatic data collection mechanisms incorporated into project support environments. Additionally, at a later stage, the requirement to engineer and manage for reliability implies that the analysis tools provided in the IPSEs be those which have been proven in the reliability programme.

Clearly, much of the work outlined in the reliability programme will feed into the development of procedures and methods required for software certification. This work will be of direct relevance to the proposed National Quality Certification Centre (NQCC).

3.2 VLSI

Most current hardware reliability models concentrate on component failure due to physical causes and are unable to address the problem of hardware design faults. It is likely that these design faults will become increasingly significant with the introduction of VLSI/VHPIC technologies and non-von Neumann architectures, since the complexity of these designs is increasing by orders of magnitude. The problems in this area show great similarity to those encountered in software: for example, the practical impossibility of exhaustive testing and the intellectual challenge of formal verification. It is probable that many of the results emanating from the software reliability research programme will be directly relevant to VLSI design reliability. Indeed, the currently available techniques for measuring and predicting software design reliability are superior to any in the hardware field and should be more widely used there.

Even the physical-cause failures in VLSI devices might need to be modelled using techniques which derive from software work. For example, it is known that as the logic density becomes very high the devices are susceptible to quantum effects, diffusion, interference, cosmic and particle radiation, etc., which manifest themselves as hard, soft and intermittent failures. Equally, for a very high density device (even with a 'naive' and correct design) there is a high probability of manufacturing defects being present. To overcome these problems of vulnerability and low yield, defensive design strategies will need to be used. These can increase design complexity and hence susceptibility to design faults.

Finally, the increasing use in future system design of CAD techniques raises the possibility of systematic design faults induced by faults in design software.

All these problems have close similarities to ones in software. Close contact between the two research areas is important in order to prevent duplication of effort on similar research programmes. This can best be achieved in the short term by workshops in which both groups describe problems and what is currently achievable. It will be the responsibility of the two expert panels to meet regularly and ensure that technology transfer continues throughout the development of the Alvey programme.

3.3 IKBS and MMI

The development of management decision support and advice systems for software production will depend upon the availability of quantitative models of the development process.

In addition the requirement to develop systems based on sophisticated programming languages and novel system architectures may necessitate the development of new classes of reliability models.

4. Research Programme

4.1 Achievement

4.1.1 Software Management

4.1.2 Engineering for Reliability

4.1.2.1 The Software Development Process

—requirements
—specification
—design
—implementation
—release
—maintenance

4.1.2.2 Techniques and Tools

—validation, verification and testing
—fault tolerance
—reusable software
—metrics and models

4.1.3 Systems Issues

—system integration
—security and integrity
—safety
—vulnerability

4.2 Assessment

4.2.1 Understanding Causes of Unreliability

—understanding the development process
—cognitive processes
—human factors

4.2.2 Data Issues for Support of Reliability Research

4.2.3 Reliability Models and Tools

—reliability growth models
—criteria for judging reliability measures
—structural models
—process and environment
—development methods
—testing strategies

4.2.4 Cost/Resource Models

4.2.5 Quality Engineering

4.2.6 Technique and Tool Evaluation

4.1 Achievement

4.1.1 Software Management

4.1.1.1 Relationship between Reliability and Good Management

There is ample evidence to show that there is a direct relationship between management techniques and reliability, both for hardware and software. Positive management of reliability for hardware projects is well established and has had a dramatic effect in increasing reliability levels of equipment in service and in reducing life cycle costs.

Management of software can be regarded at two levels. The first is related to the engineering disciplines applied during the actual production of software, that is, starting with an agreed software specification and finishing with a tested and working 'package' which conforms to the user's requirements. This aspect of management has received considerable attention and, while not perfect, does have available many tools and techniques, e.g. structured programming, modular approaches, etc. Careful selection and use of these techniques by software development managers has been found to result in better and more reliable programs, although quantification of the improvement and evaluation of the scope and applicability of these techniques is rare.

However, when the managing of total systems is considered, the situation is very different. Management techniques for producing reliable software systems are not well established. As software gradually replaces more and more hardware within systems, the unreliability of software becomes more important both from a functional and a cost point of view.

The purpose of this area of software reliability research is to provide the techniques and tools which a project manager, who may not be a software specialist, can use to monitor, assess and control the requirements, design, resources, costs, scheduling and implementation of a project.

4.1.1.2 Management Needs

Overall a strategy is needed which can be applied across all phases of the project. Specifically, managers need procedures and tools to assist in monitoring, control and assessment in the following areas:

Agreement of requirements definitions between users and developers

Generation of specifications from requirements

Design integration issues:
hardware/software apportionment

computer architectures and languages

Cost and resource estimation

Modelling and predicting system reliability and reliability growth

Interrelationship of hardware and software reliability

Managing development:
preparation of integrated development plans

schedule and resource monitoring

identifying the 'milestones'

documentation and data

multiple sub-system control

Integration and test—metrics

Reliability demonstration, growth and assurance

Transition to service and monitoring in service

System maintenance and configuration control

System change and redesign

4.1.1.3 The Objectives

Detailed consideration of many of the specific needs will be given within other parts of this document.

The work in support of management tools and techniques will be to link all the relevant parts of the total programme into an integrated set of management procedures and techniques. This will entail research into:

(a) Identification and evaluation of criteria associated with software creation which must be positively addressed within the integrated project management of a system.

(b) Assessment of existing managerial tools, techniques and procedures, and identification of missing or inappropriate elements.

(c) The review and co-ordination of new tools and techniques emerging from the Alvey programme which will reinforce any existing work.

(d) Preparation of management guidelines and procedures which can be used by project officers within an integrated management system.

4.1.1.4 Education

Education and technology transfer will be discussed in more detail in section 5. However, there are specific problems associated with persuading managers to use novel development methodology. As the efficacy of procedures and techniques becomes known from the work being carried out in the main body of the research programme, it is vital that this knowledge is used in real development projects. This will involve a selling operation. Managers understandably adopt a hard-headed approach and need to be shown that new techniques actually work. The SE programme will therefore support prototype development environments which can be used to dramatically exemplify the effectiveness of the new methodologies by applying them in the construction of realistic systems.

4.1.2 Engineering for Reliability

4.1.2.1 The Software Development Process

The development of software can be viewed as comprising a series of stages, each of which could benefit from further investigation and research. The stages are:

(a) *Requirements,* that is the discovery, collation and precise statement of the nature, qualities and functionality required of the software item, within a specific environment. When documented, requirements are usually written in natural language and are often relatively informal. They should include the quality and performance levels which must be met if the product is to be acceptable and useful.

(b) *Specifications,* that is the formal statements of functionality required. They may be wholly or partly presented in mathematical notations and formally defined languages specific to the nature of the task and/or to the methodology (procedures and tools) which will be applied in the next transformation of the specifications. However, for many practical systems the specification is written in natural language, albeit with considerable concern for precision and clarity.

(c) *Design.* This is the process of transforming requirements and specifications into an implementable form. It includes the identification of computing functions needed, their grouping into manageable and testable units, the creation (or selection) of algorithms, the identification and description of internal and external interfaces, and the documentation of the validation procedures and tools which must be applied to confirm the accuracy with which the design matches the specification and requirements.

(d) *Implementation.* This is the process of translation from design to executable software. Additional testing needs will become apparent, related to the exact nature of the code produced and the relationships between code units. Validation conducted during each of the preceding stages will be supplemented by an intensive examination of the structure and functions of internal modules, and of the functionality, quality and performance of the software as a whole.

(e) *Release.* At some point the software must be released by its builders and handed over to the customer. Special tests may take place at this point, with the aim of establishing that the product conforms to standards of functionality and performance which are acceptable.

(f) *Maintenance.* Once accepted and released a product is still likely to need attention. Faults are repaired, omissions filled and new functions added. Each of the changes must be considered for its effects on the current state of the product and, if made, should be followed by full testing of all affected areas and re-measuring of relevant reliability metrics.

4.1.2.2 Techniques and Tools

Numerous techniques and tools have been devised for the various stages of software development: methodologies to assist in design; notations for specification, design and implementation; testing strategies, etc. Their effectiveness is expected to be enhanced by making them available in a coherent and uniform way by means of Integrated Project Support Environments. The mechanisation inherent in the use of a tool can itself have a significant impact on reliability. An IPSE should facilitate both mechanisation and consistent modelling, and measurement of techniques. This should in turn assist in the comparative evaluation of different techniques, and the control and monitoring of individual software developments.

Four particular categories of these techniques can be expected to have a direct impact on software reliability, and these are summarised below.

A. *Validation, Verification and Testing*

The process of software development is basically one of representation and transformation, whereby a first informal idea is translated into progressively more detailed and concrete terms until it is implemented as an executable system or program. At each stage in the process, each intermediate representation needs to be checked to ensure that it corresponds to the proceeding representation, that its characteristics are consistent with the stage of development it has reached and that it is suitable for entry to the next stage of processing. This process of checking is variously called validation, verification or testing with no uniformly accepted definition of any of these terms.

In common usage, the term verification tends to be associated with checking procedures based on mathematical logic whereas testing tends to be associated with checking that a program correctly executes with a number of specific inputs (test cases or trials). Thus verification has tended to be associated with the early stages of software development and testing with the later stages. Validation tends to be used to indicate an assessment of the general worth of quality of a program and is not usually confined to a specific checking method.

Thus, it seems appropriate to use the term validation as a general term for all checking procedures, and to use verification to refer to formal checking procedures and testing to indicate validation by program execution methods. This convention will be followed throughout this document.

Research issues in validation centre around identifying efficient and cost-effective checking procedures for each stage in the development process, and establishing the inferences which can be made about the quality of a given software product after validation procedures has taken place. However, validation procedures which detect errors efficiently and cheaply may be incompatible with those aimed at obtaining information relating to product quality. Thus, there is also a need for research aimed at identifying and reconciling conflicts between validation techniques aimed at defect removal and those aimed at product assessment.

Validation techniques which are of particualr importance for the production of reliable software include:

(i) *Animation and prototyping*
Animation techniques aim to provide a simulation or a model of a system early in the development process. Prototyping is a form of animation with an implication that the simulation is of a similar nature to the final system. Animation is more general, encompassing non-computer based activities such as scenario-building at one extreme, and computer execution of specifications at the other.

Animation is meant to reduce the uncertainty which normally exists between a user's perception of a system which may be expressed in informal requirements and a systems designer's view which may be expressed in more formal and abstract terms.

(ii) *Verification*
Software verification attempts to confirm that the software and its specification are consistent, usually by the application of the mathematical techniques. Formal verification can only be performed when a formal specifiation is available but then offers a stringent means of validation which can, in principle, be checked mechanically.

(iii) *Reviews and Inspections*
Reviews and inspections are used to validate representations found early in the software development process without using mathematical techniques. Empirical results indicate that both techniques provide effective and economical means of error detection although definitions of the two techniques vary and are sometimes contradictory.

(iv) *Analysis*
Once code exists, it may be analysed, usually with automated assistance, to provide information on measurable properties of the software (such as size and complexity) and to locate any occurrences of certain specific faults (such as unreachable code, non-terminating loops, illegal addresses, etc.). Analysis of code structure provides measures of test efficiency (path coverage, proportion of code executed, etc.).

(v) *Testing*
Testing conventionally consists of determining a set of input test data, executing the software on each element of the data set and checking that the outputs from the system agree with the tester's expectations. Testing usually takes place at a series of levels of completeness of a product, as the components of a product are integrated into larger structural units (module, subsystem and system).

Testing is an area for which there are many techniques and little background theory. There are a number of problems involved in interpreting the results of testing. There are also practical problems involved with the conflict between testing aimed at error detection and testing aimed at demonstrating product quality.

Research into the techniques themselves will also be conducted by the formal methods and process panels. However, a number of research topics relate directly to reliability and measurement:

(i) evaluation of the cost-effectiveness of the various techniques;

(ii) investigation of relationships between measurable aspects of the techniques and system reliability, and the implications for product certification;

(iii) overall design of a validation process in terms of cost and reliability (this must identify the relationships between various techniques, the development process and cost and quality factors);

(iv) experimental evaluation of comparable techniques (i.e. different specification languages, or reviews and inspections).

B. *Fault Tolerance*
One of the major factors contributing to the very high levels of hardware reliability which can now be achieved is the use of component redundancy to provide tolerance to physical faults. In recent years, more general techniques (such as recovery blocks and N-version programming) have been proposed which aim to provide tolerance to the faults of design which cause unreliability in software systems. Software fault tolerance techniques are based on the implementation of replicated modules of diverse design, with provision for either state restoration or replication, and of some means for adjudicating between the outputs from the replicated modules.

Some highly critical systems have relied on the construction of independently designed versions of the entire software system (e.g. Space Shuttle, A310 Airbus, railway signalling). Many database systems and telephone switching systems employ sophisticated recovery techniques which can prevent corruption of data by certain categories of software fault. Less comprehensive strategies for software fault tolerance are often referred to as defensive programming, which can range from this disciplined use of exception handling notations to the ad-hoc inclusion on run-time assertions.

The increasing complexity of hardware designs made possible by VLSI techniques implies that design faults can be expected to have substantial impact upon the reliability of future hardware systems. Software fault-tolerance techniques will be of direct relevance to these hardware design problems.

Research in fault-tolerance is needed to:

(i) develop more general approaches to design fault tolerance than the specific notations currently available;

(ii) ensure that these more general approaches can still be expressed in notations which allow them to be exploited effectively by the designers of complex systems;

(iii) devise techniques for use in concurrent, real-time, distributed and embedded systems;

(iv) establish methodologies for the guidance of the designers of fault tolerant software;

(v) harmonise fault tolerance and exception handling approaches;

(vi) evaluate the relative effectiveness of different fault tolerance approaches;

(vii) compare the effectiveness of fault tolerance techniques with other ways of improving software reliability.

C. *Reusable Software*
It is widely expected that techniques for the re-use of software modules will have a significant effect on the reliability and cost of new software products. Although an immediately attractive idea, there are a number of problems to be solved by research, design and standardisation methods. For reusability, the modules must be ones which are widely required and unambiguously specified. Their design and implementation must be such that they can be automatically incorporated into programs created using different methods and tools. This will be facilitated by adherence to programming standards. Their documentation must be both complete and flexible, to enable its inclusion into larger documents of various kinds. The SE programme will set up a database of components (which will include, for example, hardware designs as well as reusable software modules). This database will be used in the short to medium term to acquire experience within the research community of the use of these techniques, and also to support modelling research. In the longer term, there is a need for the information base to be made available to the commerical community, perhaps via IKBS techniques.

It can be hoped that trends in modern programming languages (modularity and encapsulation) and in system design (object based systems) will give a sorely needed impetus to the development and application of reusable software. Special reliability modelling techniques, as discussed in 4.2.3.3.1, may need to be developed to reflect the structural properties of software built up from reusable modules.

Important unresolved issues in this area include metrics. How, for example, should reliability information about software modules be combined to form reliability metrics for the overall programs of which they will be parts? Only when questions of this kind can be answered will reusability begin to yield economic benefits. Without such techniques it would be necessary to treat the module-based new system as a black box program and obtain reliability measures by conventional methods. This issue is discussed in more detail in 4.2.3.3.1.

It should be noted that current notions of structure might have to be revised if reusability becomes widespread. For example, the existence of reusable software, coupled with the possibility of embedding this software in a VLSI hardware system, might result in cost-effective system architectures radically different from those currently advocated.

Equally, the structures which are advantageous from a reliability standpoint may not be those which system designers would regard as most natural. The importance of achieving reliability benefits (or at least not incurring reliability penalties) in cost-effective ways via reusability, suggests that there should be close co-operation between workers in the reliability modelling area and those dealing with the wider feasibility studies of reusability. It is possible that the latter may have to pay the price of dealing with unfamiliar structures to gain the advantage of having easily obtained reliability measures.

Formal requirements concerning the reliability data to be collected on modules (and available in the database) will be agreed with the reliability panel.

D. *Metrics and Models*
The development of metrics and models relating to all phases of software development is a major requirement of any programme which aims to provide quantitative methods of evaluating and managing the development process and assessing the quality of software products. This research area forms the basis of the whole of section 4.2 of this report.

4.1.3 System Issues
Although this report emphasises the more formal aspects of reliability, users' perceptions of the reliability of systems generally embrace other, more informal, qualities. There is a need to investigate interactions between these various wider reliability issues (such as availability, maintainability, security, integrity and safety) as they apply to both systems and software. Different applications may make greater demands on some of these than on others. For example, nuclear power plant control needs to be safe, databases may need to be secure. Availability, which concerns the amount of downtime resulting from failures, may not be as relevant for software as it has been traditionally for hardware.

Research is required to investigate the suitability of various tools and techniques for the achievement of reliability, availability, safety, maintainability, etc. The tool mix may differ for systems whose requirements place different emphases on these different qualities. This observation has implications for the development of IPSEs.

4.1.3.1 System Integration
There is a need for research into the reliability problems caused by incompatabilities between hardware and software aspects of systems, and into systems requirement, specification and design disciplines which serve to reduce or eliminate such problems. Since the ultimate aim of software reliability is reliability of systems, there is a need for work aimed at unifying measures and goals for reliability of hardware, of software and of systems.

4.1.3.2 Security and Integrity
There is a need to investigate means of ensuring and assuring the security and integrity of computer based systems. This is an area where there is already considerable public concern, particularly as far as data protection and confidentiality is concerned. It will be necessary to consider both accidental breaches of security caused by unreliability or other deficiencies, and seek means of guarding against deliberate attempts by outside agencies to breach security.

4.1.3.3 Safety
System safety is, in certain application areas, one of the major driving forces for reliability of software. There is a need for research in this area focusing attention on the achievement of system safety through the use of suitable software development methodologies.

4.1.3.4 Vulnerability
Complex systems containing many diverse components are vulnerable in different ways to the loss of functionality of one or more of these components. Different structures may have different vulnerabilities, for example by having different probabilities of retaining particular important functions under similar component loss. These important structural issues need further study.

4.2 Assessment

4.2.1 Understanding Causes of Unreliability
Research is essential into the causes of unreliability in software: it is only by understanding and avoiding practices which lead to unreliable software that software engineering tools and techniques leading to reliable software can be recognised. This research needs to be very practical, involving the detailed examination of the background to, and development of, software systems which are known to be unreliable, thus identifying properties which differ from those of reliable systems.

4.2.1.1 Understanding the Development Process
A programme of research is necessary to investigate the processes involved in the software life-cycle. Two levels of investigation are required, one at the micro and one at the macro level. The micro level should aim to provide a detailed study of the component sub-processes associated with the conventional phases of the software life-cycle. The macro level should provide information about the gross economic processes involved in software production.

The aims of each part of the programme should be:

(i) To provide a basic understanding of:
—the processes involved in software development
—their relationships to one another
—the relationship between each process and the input to and output from that process.

(ii) To identify factors which could explain such relationships.

(iii) To progress towards predictive and strategic planning models of software development.

The initial investigation should concentrate on descriptive models but should provide the basic understanding necessary to identify quantifiable features of product and process.

It is not likely that the micro and macro level studies will link-up initially but their integration should be a long term goal.

This programme should be initiated early in the overall programme because it will provide necessary input into other parts of the programme.

The major outputs should be:

(i) Standards and definitions of the sub-processes of software development and their respective product inputs and outputs.

(ii) Macro and micro level descriptive models of software development.

(iii) Identification of areas of research needed to link the micro and macro view points.

4.2.1.2 Cognitive Processes
Since unreliability of software is caused by human errors, explanations of unreliability necessitate investigation of the reasons for human error. This implies that a programme of research is required which will consider the psychological implications of software development as a problem solving activity. Investigations aimed at identifying methods for restricting the occurrence of errors during problem solving tasks will have implications for many parts of the software engineering programme. Factors which should be considered are:

(i) The relationship between language design, problem representation and problem solving strategies.

(ii) The relationship between problem representation, problem transformation and learning strategies in the context of the software development process.

(iii) The implications of psychological research for the validation of new and existing software development methods and techniques.

The programme should be in two parts. Firstly, the implications of current work in cognitive psychology should be assessed in the light of current software development methods. Then, a programme of research into the problem solving aspects of computer programming should be instigated.

The major outputs of the programme should be:

(i) An evaluation of current software development tools and procedures in terms of current cognitive psychology theory.

(ii) An evaluation of software development as a problem solving activity, identifying methods of reducing the opportunity for error within the development process.

(iii) Procedures for evaluating new development tools and methodologies.

4.2.1.3 Human Factors
4.2.1.3.1 Group Factors
The people involved in the various aspects of the development process may have conflicting views of the software being evolved. Means of resolving these conflicts are required.

A programme of research aimed at investigating these influences will probably involve technology transfer from management sciences and social psychology aimed at providing techniques for the project manager and insight and understanding for group members.

4.2.1.3.2 Motivation
Motivation is primarily a management responsibility. The development of understanding and techniques to assist management require research aimed at the following issues:

(a) Motivating factors and techniques.

(b) Demotivating factors and avoidance methods. Current psychological methods need to be assessed in a computing environment.

(c) Influence of motivation on productivity, quality and reliability.

4.2.1.3.3 Productivity
Assessment and evaluation of project members is an important part of project management. Influences on motivation and group factors, and thus on productivity, must be considered to be part of any research programme.

4.2.1.3.4 Man-machine Interfaces
A research programme is necessary to ensure that the new generation of programme tools which will become part of the IPSEs have well designed man-machine interfaces. It is important to ensure that interfaces are compatible with human cognitive processes and productivity aims. First-generation new tools wil be based to a large extent on current tools. Part of the programme should therefore be directed towards the evaluation of current techniques. This will link into other aspects of the programme.

4.2.2 Data Issues for Support of Reliability Research
The success of the entire software reliability programme depends upon a close integration of the theoretical issues discussed here and real software development programmes. Just as, in all projects, reliability should not be an optional extra which can be invoked when convenient, so in this work an awareness of reliability issues must pervade the whole programme.

4.2.2.1 Data Requirements
There is a need to obtain data to support a number of activities within the software reliability section of the Alvey programme. These include the assessment of effectiveness of various tools, techniques and procedures in achieving reliability and other related goals. Data is also required to assist in understanding the causes of unreliability and in identifying and quantifying factors which explain unreliability. Specific kinds of data are also required to support the various modelling and reliability prediction activities.

Data of various kinds will be required by other areas of research within the Alvey programme, as well as the important topics of software metrics and quality. Although this data will be used for different purposes from the reliability data, the requirements will doubtless overlap, so that a single statement of data requirements is preferable. For this reason, the remainder of this section will address issues wider than reliability alone.

There is an immediate need for an investigation to be carried out to identify data which should be collected. This should include data collected throughout all phases of the software life-cycle.

4.2.2.2 Data Sources
In any data collection activity, it is necessary to reassure providers of data that the data will not be used in ways which would be prejudicial to their interests. The purposes for which data is being collected must therefore be identified to all potential data sources, and an undertaking given that data will not be used for other purposes without seeking specific permission. Without assurances of this nature, commercial considerations may influence the availability of data.

It is anticipated that software developers participating in the Alvey programme will be required to provide a certain basic amount of data, which may be collected automatically or semi-automatically as part of an IPSE development. It is hoped that data will also be available on developments using more traditional methods. Data collected in these ways will be valuable for the assessment of effectiveness of software engineering development methodologies. Substantial economic benefits are likely to come from knowledge of the degree to which suggested development approaches succeed in producing reliable software.

In addition to the above 'passive' data collection activities, more active approaches will be necessary in specific instances. An example is provided by the data needs of the reliability prediction area, where it may be necessary for random testing to be carried out on certain projects (i.e. testing which simulates or uses an actual user environment). There are other areas where controlled statistical experiments may be feasible. Examples include investigation of independence of behaviour in software replication for fault tolerance (4.2.3.3.2), economic trade-offs between single programs and multiple version fault-tolerance (4.2.3.3.3), and structural issues concerning reusable software modules (4.2.3.3.1).

4.2.2.3 Data Organisation
The implication of the requirements for data and the wide range of potential data sources is that a large database system will be needed to receive and store the incoming data. Although such a system will probably incorporate a number of databases for specific purposes, it is important that it should be visible to the user as a single entity, capable of providing information on a wide range of software matters. The facility should include data analysis procedures and should, in the longer term, aim to be accessible on line—subject only to the considerations of data confidentiality mentioned earlier.

An immediate need is for a study aimed at producing a specification for the database system, in order that the database system itself will commence functioning at the earliest possible date.

4.2.3 Reliability Models and Tools
The objective of a research programme into this topic is the development of mathematical and probabilistic models that are powerful in explanation, prediction and control of reliability. Some of these models will find their main application in furthering understanding of the *process* of software development. Others will be primarily intended to provide metrics for a particular software *product* using, for example, failure data on that product obtained during testing. This distinction between process and product, although important, will occasionally become blurred.

Indeed, several of the modelling techniques in the following sections will find application in both areas.

Since software is always a part of a wider overall system, the models and resulting reliability measures must be compatible with this wider context. This may be achieved by regarding reliability as a probabilistic/stochastic quantity. Examples of formal probabilistic reliability measures which can be obtained from this stochastic approach are: rate of occurrence of failures (ROCOF), probability of failure-free working for a user-specified length of time, mean time spent out of action over a specified period, etc. The choice of a particular reliability measure will depend upon the context in which it is to be used. The important point is that the choice be made from the set of formal probabilistic measures which arise from the programme of research outlined here. It should be emphasised, though, that these are designed to be useful as well as formally and mathematically acceptable. The main reason for their being used by a manager, after all, is that he can collect (relatively) simple data, feed it into the model, and obtain meaningful predictions and measures about the reliability of the product.

The previous paragraphs emphasis how formal reliability measures should be prescribed for users. The models from which these measures emanate, on the other hand, must stand or fall on their ability to be seen by the user as genuinely explicative and predictive. That is, a user faced with alternate reliability predictions from two different models will correctly judge the models on the degree to which the predictions are verified in the light of actual behaviour. This kind of judgement is already taking place within software engineering in fairly informal ways (see, for example, the waning fortunes of 'Software Science'). Users require more formal techniques for assessing the degree of trust they can place in the predictions they make about the actual project in hand.

4.2.3.1 Reliability Growth Modelling
The most successful techniques for measurement and prediction of software reliability currently available are reliability growth models. The reliability of software grows as a result of the process of fault identification and correction known as debugging. These models use failure data (for example, execution times between successive failures) to estimate current reliability and predict future failure behaviour (for example, time to achieve a specified target reliability). It should be noted that these models place stringent requirements upon the testing strategy which generates the raw failure data; this issue is discussed in more detail in 4.2.3.5.

Recent research shows that the performance of these techniques is variable. For example, comparisons of reliability predictions with subsequent program failure data are consistently poor for some models. Other models sometimes perform well, but will occasionally give misleading predictions for reasons which are not presently understood. This means that a potential user cannot select a model a priori and be certain that the reliability predictions it produces for his/her software will be accurate. On the other hand, it does seem to be the case that at least one of the available models will perform well in a particular situation. Faced with this, a user urgently needs tools which will assist in identifying the reliability predictions which are trustworthy for *the particular software under study.*

It is important that this work, and the resulting tools, are responsive to the needs of the user. Although formally the problem is the statistical one of examining 'predictive quality', it is clear that users place different emphasis upon different aspects of this concept. For example, recent results show that it is easier to obtain a good estimate of the *current reliability* of a program then to predict when a target reliability will be achieved. The first of these is a useful achievement: it would be a necessary prerequisite for formal reliability acceptance testing. The second would be invaluable for software developers involved in estimating target dates.

4.2.3.2 Criteria for Judging Reliability Model Performance
When tools are available for judging the quality of reliability predictions (see 4.2.3.1), it will be possible to investigate in detail the reasons for differing model performance. The objective will be to improve and develop these relatively simple reliability growth models.

It should be noted that this work will have important implications not only for software but also for hardware reliability growth, where traditional approaches are extremely naive. In particular, software reliability growth approaches will prove relevant to the modelling of VLSI design fault occurrence.

The development of adaptive reliability models should be investigated. Given a suitable measure of the agreement between predicted and actual failure behaviour, it may be possible to refine future predictions in the light of past predictive achievements.

4.2.3.3 Structural Reliability Modelling
The most important criticism which can be levelled at the simple reliability growth models of 4.2.3.1 and 4.2.3.2 is that they essentially treat software as a 'black box'. No account is taken of internal structure or other known properties of the program under study.

There is an urgent need for models which can exploit the large amount of structural information which is usually available. Hardware reliability theory provides an interesting parallel: one of the great achievements of this theory is the ability to combine information about component reliability with structural information about the design of the overall system. Unfortunately, software structure tends to be very much more complex than hardware structure. Also, the simple component/design dichotomy is less obviously applicable to software, which can be viewed as solely *levels of design* embedded in higher levels of design.

Work on structural models like these can be seen as a way of improving the very general reliability growth models which already exist. The following are areas where specific structural issues will shortly become important.

4.2.3.3.1 Reusable Software
Reusable software is an obvious candidate for the economic achievement of high reliability (see 4.1.2.2 (C)). The use of software modules, of known performance, within a novel skeletal structure must be a sensible way of lessening uncertainty about the performance of the overall product. However, it is not currently known how the 'module' and 'structure' reliabilities should be combined to yield a reliability metric for the overall system. Indeed, it is not at all

obvious how to assess the reliability of the novel skeletal structure into which the modules are to be embedded. It may be possible to execute such structures, with 'tied-off' modules, but this needs further study.

A more important problem arises from the observation that, in general, a module does not simply have 'a' reliability figure. In fact the reliability of a module (indeed of any program: see 4.2.3.4.2) will depend upon the type of use to which it is put. Specifically, it will depend upon the characteristics of the skeletal software structure in which it is to be used. It is important, then, that a database of reusable modules should contain sufficient information for a potential user to be able to obtain a reliability metric for a module operating in his own environment. The ability to do this depends upon the success of the programme of research into matching reliability performance to operational context which is described in 4.2.3.4.2.

4.2.3.3.2 Fault-tolerant Structures: Ultra-reliability
Important structural questions are associated with the achievement of extremely high reliabilities for safety-related systems. It is already obvious, but not necessarily generally accepted, that heroic debugging of single programs is not a feasible approach to the achievement of these reliabilities. The feasibility and cost-effectiveness of mathematical verification for practical systems is as yet highly questionable. Special fault-tolerant architectures may be successful in achieving high reliabilities, but it is equally important that methods are developed for *assuring* the achievement of a very high reliability for a *particular* system. It is clearly not possible to receive such assurance simply from test data. The use of test data within a structural model may be sufficient, but more work is needed on this. Since achievement of very high reliability in these architectures exploits independence of failure behaviour between disparate replicates, empirical investigation is urgently needed to determine what degree of independence is achievable.

4.2.3.3.3 Fault-tolerance: Economic Optimality Issues
Fault-tolerant structures are usually discussed in the context of very high reliability requirements. It may be the case, however, that fault-tolerance is a more cost effective way of achieving the lower reliability targets of more mundane applications. Equally, it is only when life-cycle costs or unreliability can be estimated that rational discussions can be made about reliability targets (see 4.2.4). The success of recent designs of fault-tolerant *hardware* in quite ordinary applications suggests that many users may be willing to pay for guaranteed high software reliability. It should be noted, though, that these fault-tolerant hardware designs only buy protection from physical-cause failures. Protection from hardware *design* faults is a similar problem to that of software fault-tolerance, and should benefit from this research.

Methods of deciding in optimal ways between single program and different fault-tolerant architectures are required. It would be valuable for a developer to have guidance on whether a particular reliability target can be achieved with least effort by debugging a single design, or by using a particular kind of design diversity. There is currently little theory to assist in this kind of decision making, and virtually no empirical data. These optimality issues can be stated in different ways. For example, instead of reliability a user might be more interested in the effect of failures on total life-cycle costs (see 4.2.4). Then it would be of interest to know which architecture would achieve acceptable life-cycle failure costs for minimum development costs.

Dependent upon the results of this work, it is possible that fault-tolerant architectures will find widest use because of their economic advantages, rather than because they are the only possible avenues to the achievement of ultra-reliability.

4.2.3.4 Reliability for Process and Environment
This is the area which is likely to provide most immediately recognisable benefits to managers involved in practical software development. What they require is objective guidance as to the relative efficacy of different techniques at the various stages of software development. What they get currently is a large amount of anecdotal evidence and special pleading.

In order to be able to quantify the effectiveness of software engineering methodologies, software models must be able to incorporate variables (explanatory variables in statistical terminology) which characterise these methodologies. In fact these new explanatory variable models will have other important uses. They should be able to characterise the type of use of the software: matching the reliability metrics to the environment in which the software operates. This will be particularly important for reusability to be effective (see 4.2.3.3.1). They should also be able to characterise the nature of the program itself (complexity, size, etc.).

4.2.3.4.1 Characterisation of User Environment
Present models may allow 'the' reliability of a program to be measured and predicted. However, in most cases the actual failure behaviour in use depends upon the type of environment in which the program runs. General purpose operating systems, for example, show different reliability in scientific environments than they show in business data processing contexts. It will be valuable to be able to predict *user perceived* reliabilities from an analysis of test data plus information characterising the user profiles. Success of these new models is vital to the programme for reusability of standard modules.

4.2.3.4.2 Characterisation of Program Properties
Reliability modelling which takes into account properties of the program and its development forms one of the most important software engineering issues. The intention is to obtain quantitative measures of the effect software engineering methodology has upon ultimate product reliability. Present knowledge in this area is largely anecdotal. These new models will be used in investigations of real software development (see 4.2.2) in order to obtain quantitative measures of the effect of different methodologies and program properties.

This work will allow system developers to evaluate the cost-effectiveness of different development approaches as revealed by their past effectiveness. It should not be seen as an alternative to reliability measurement and prediction during program development, but complementary to this. The purpose of this work is not to predict reliability before system design begins. In fact it seems unlikely that this could ever be done with adequate accuracy. It ought to be possible, however, to provide guidelines on how given reliability targets can be achieved.

Real-time programs have particular problems of reliability resulting from timing requirements. Stochastic modelling in this important and neglected area is urgently needed (see 4.3.2).

4.2.3.5 Testing Strategies
The role of testing is two-fold. On the one hand it is seen as a method of finding faults present in the program. The objective here is the *achievement* of reliability, and this view is by far the most common. On the other hand, testing produces the raw data which forms the input to current reliability models: i.e. it is concerned with the *measurement and prediction* of reliability.

Unfortunately, there is currently a conflict of interest between these two roles. For testing to be meaningful in the measurement role it is important that the method of test case generation should accurately emulate the operational environment envisaged for the program under test. Such testing has come to be called random testing, and is normally reviled by software developers because of its inefficiency in capturing faults.

However recent work in the US suggests that this view of the inefficiency of random testing may be unnecessarily pessimistic. More careful quantitative studies are needed to resolve this issue. In addition, the possibility of devising 'accelerated' random testing strategies should be considered. Such a strategy would retain the main advantage of random testing (that faults are discovered during test with probabilities which are proportional to their chances of appearing in use) whilst increasing efficiency.

There should be investigation of testing strategies which are concerned with wider issues than reliability. For example, since the *consequences* of failures are often as important as the *frequencies* of failures, it would be useful to have a testing strategy which matched fault discovery probabilities with their total life-cycle consequences.

4.2.4 Cost/Resource Models
The engineering of systems, and in particular of software, has to take place within economic constraints, and against a background of (often tacit) social and environmental needs. Failure to meet such constraints results in costs which may or may not be measured in monetary terms, and possibly may be incommensurable. Thus it is important that these costs be accounted for, and, if possible, understood. This, in principle, would allow reasonable decisions to be made by project management regarding the optimum deployment of available and often scant resources.

Unreliability is a major cost driver, both by requiring the expenditure of large quantities of resource in reliability achievement and the incurred costs of failure in service. Existing reliability models essentially account for failures, without regard to severity of failure, and usually produce estimates of failure rate and properties of interfailure times, etc. These current techniques can be applied also to individual subclasses of failures (e.g. safety-critical, catastrophic, etc.). It is possible that the hardware techniques of FMECA (Failure Modes, Effects and Criticality Analysis) and FTA (Fault-Tree Analysis) could be extended here, since they both account for different classes of failure and link them to their corresponding effects. Thus if it were possible to cost the consequential effects commensurably, and estimate their rate of occurrence, we would be in a position to quantify the support costs.

It is important to distinguish between cost and value, each of which needs to be measured in commensurable terms in order to take proper decisions, and much work is needed into both topics. Project management and the optimum deployment of resources can only be achieved after the above measures of cost and value are resolved. For example, project management should be able to determine the best cessation time for development/testing and this requires the provision of realistic tools for Life Cycle Costing. This in turn neccesitates the proper accounting for foreseen hazardous failures, out-of-service costs, maintenance costs, etc., against a systems background. The development of models which attempt to take account of the severity of failure in a quantitative manner is likely to be restricted by the paucity of data on costs of failure. It thus seems likely that research in this area will be constrained by questions of data availability rather than theoretical modelling constraints.

Furthermore the economics of software is also likely to be radically different from that of other 'materials', which generally get consumed or degrade with time. The software stock represents a non-diminishing asset and should be treated differently from other assets in accounting for its value. This is especially true if designs are included as being software. How should the value of the FFT (Fast Fourier Transform) Algorithm be evaluated for example? How much resource should be expended on developing particular reusable software? Presently accepted accounting principles, such as NPV (Net Present Value), lead to the conclusion that the value of reusable software is infinite. Such a result implies that it would be economically sound to spend any finite sum of money (however large!) on developing reusable software. It also implies that any relevant reusable software should be utilised almost regardless of cost! Certainly the concept of reusable software seems to be economically and practically attractive even when the administrative and housekeeping burdens are considered. However it is apparent that further research is required to produce new accounting principles which will enable relative value judgements to be made.

The absurdity of this analysis identifies the need for future research into the following specific areas:

—Cost models of failure processes/maintenance processes, etc.
—Value models of software.
—Compatible cost models for catastrophic failure.
—Investigation into the possibility of extending FMECA and FTA techniques.
—Establishment of database including failure-cost data.

4.2.5 Quality Engineering
Quality and productivity cannot be considered in isolation of one another. In one sense they identify the opposing requirements of the software developer and the software user but since they represent an important area of trade-off between customer and vendor, it is essential that software engineers recognise and understand the relationship between them.

In terms of the reliability programme, research topics in this area should be primarily concerned with the identification of improved metrics and the investigation of relationships between those metrics.

Quantitative measures of product quality are needed for all the different representations of the software product found at all stages of software development. Such quality measures should include both inherent product metrics such as 'complexity' and indirect measures of the result of various development processes such as test coverage statistics, and error counts and classifications.

It will also be important to investigate the relationship between quality measures relating to one representation of the software product and measures relating to other representations, e.g. relationships between design complexity and subsequent code complexity.

Research along these lines will permit customers to specify quality requirements explicitly and will ensure that software producers have the means to monitor their achieved quality throughout the development process. Quality metrics will also provide additional visibility of the development process which will improve the management process.

In addition, quality metrics relating to the software product are necessary if quality control and quality assurance are to progress beyond ensuring that certain processes are undertaken, towards certifying the quality of the software itself.

Productivity is the ratio of the 'size' of a software product to the effort required to produce it. Although this appears to be a simple concept, there are a number of problems involved in identifying a measure of size. The metric often used is lines of code. However, this is not a reliable metric because it is affected by programmer style and different programming languages.

Research in the area of productivity should be directed towards:

(i) identifying style and language independent measures of product size for each representation of the software during the development process;

(ii) identifying the relationships between the different size metrics;

(iii) identifying the relationships between production effort (costs) and the product size metrics;

(iv) identifying the relationships between the quality metrics, the size metrics and the effort measures.

This will provide:

(i) objective and comparable measures of productivity;

(ii) the information needed to develop cost models incorporating quality and size considerations;

(iii) quantitative identification of the trade-off between quality and productivity.

Work in this area is related to other parts of the reliability programme in the following ways:

(i) Metrics of product quality are needed to provide some of the explanatory variables needed for the next generation of reliability models (see 4.2.3.4).

(ii) Quality and productivity metrics and information about the relationship between them are essential to the development of micro-models for cost and resource estimation (see 4.2.4).

(iii) Quality and productivity metrics are needed to evaluate software development methods and tools (see 4.2.6).

(iv) Quality and productivity metrics are needed to quantify models of the software development process (see 4.2.1.1).

It is related to the wider Software Engineering programme in the following ways:

(i) Quality and productivity metrics will need to be incorporated into management tools.

(ii) Quality metrics and the ability to monitor quality should influence quality assurance and program certification methods.

(iii) The ability to specify quality requirements should be incorporated into formal specification techniques.

(iv) The ability to record quality and productivity metrics should be incorporated into formal specification tools.

4.2.6 Technique and Tool Evaluation

4.2.6.1 Evaluation of Software

The ability to model and measure software quality and productivity should be exploited by means of a programme aimed at automating the software quality control, quality assurance and certification tasks. This programme will require a further research element to progress from the stage of theoretical models to concrete procedures for specific tasks. It should then continue with the production of software tools to assist the tasks. Tool development will encompass:

(i) Automatic and semi-automatic data collection, probably related to a major data collection programme as outlined in section 4.2.2.

(ii) Analysis routines related to each identified task which allow a structured assessment of specific programs.

4.2.6.2 Evaluation of Software Development Methods, Techniques and Tools

An important application of the models and metrics research will be the ability empirically to evaluate software development techniques. Such evaluation will require a large amount of data from projects which were developed using a variety of development techniques, together with measures of the quality and reliability of the resulting software. The evaluation should consider the following questions:

(i) Are any techniques or tools clearly superior/inferior with regard to cost and quality?

(ii) Are there quantitative relationships between development techniques or tools and subsequent software reliability?

(iii) Is it possible to predict a cost/reliability/technique trade-off at an early stage of the development cycle?

This programme should aim to validate the metrics and models previously developed and should result in improved models both of software development and software reliability.

4.3 Systems in Practice

The computing systems which are utilised by commerce and industry (for applications such as process and plant control, or intelligent robots for factory automation) or those deployed for military purposes (for example command and control systems) may pose additional problems for the software reliability engineer. Successful introduction of advanced reliability techniques into these areas of industrial practice may necessitate, on the one hand an integrated approach combining all that is best from R and D advances, and on the other careful consideration of issues relating to specific application domains.

For particular system architectures and certain environments the software characteristics pose special problems, for example the assured provision of high levels of reliability in systems for life-critical applications and in systems with decentralised control. In such situations achieving and assessing required standards of reliability may entail the use of specifically tailored techniques. The programme of research should include provision for activities directed towards improving reliability for systems in the following areas.

4.3.1 Critical Systems

A system may be termed critical if the cost of any failures of the system is extremely high, either in terms of loss of human life or in financial terms. In such systems requirements are often imposed in terms of safety considerations. The required software reliability for such systems is commensurately high—failure rate of 10^{-9}/hour for commercial flight systems provide an example. Current technology can neither provide nor measure reliability standards of this order. Other examples of critical systems are control systems for nuclear power stations, life support systems, and military intelligence systems.

4.3.2 Real-Time Systems

A system may be termed real-time if the value of the service it provides depends crucially on the timing of the delivery of that service. The internal design of such systems often involves the imposition of real-time constraints, concurrent activity and the use of multiple processing elements. Consequently these systems are subject to timing and processing interaction problems in addition to the usual reliability considerations. Areas in need of investigation are: design methodologies which build in real-time aspects from the outset of the design process, integration of hardware and software reliability concerns, techniques for data and event driven systems, recovery and fault tolerance approaches. The opportunity to develop and deploy more sophisticated techniques for building and assessing reliable real-time systems stems from the ever declining cost of hardware.

4.3.3 Embedded Systems

A computing system may be termed embedded when it merely forms a part of a much larger system. In such systems reliability problems arise because of failure interactions between the computing system and the containing system. The computing system must be designed to cope with unusual circumstances due to malfunctions in its environment. Design and assessment of the reliability of the overall system requires a methodology embracing software, hardware and components of the containing system.

4.3.4 Distributed Systems

A system may be termed distributed if it contains multiple autonomous processing elements usually connected via a communications network. Reliability problems in such systems include data inconsistencies, resource contention, distributed control, network configuration, agreement protocols. Of course, the replicated resources inherent in distributed systems provide opportunities for achieving high levels of reliability via strategies for exploiting redundancy, but reliability assessment techniques are required. This is because of the possibilities of reconfiguration and self-configuration. A vulnerability study of such systems may make it possible to model their reliability and integrity and find strategies to optimise these.

4.3.5 Future Systems

With the continuing burgeoning cost of software production and the concurrent diminishing in cost of computer hardware, it is anticipated that the architectures of future computer systems will depend more intimately upon the needs of cost-effective software engineering. It seems possible therefore that research in the area of software reliability will need to recognise the impact of these new machines. Similarly it is possible that software development processes will reflect in an intimate way the results of software reliability research, and that this research will therefore lead to new system architectures.

It is also anticipated that new 'language dependent' architectures will appear in the future, and that these languages, and hence architectures, may be influenced by reliability research.

System design methodologies should also change radically to reflect the reliability achievement problems of highly complex systems: it seems likely that various forms of defensive design methodology will become important, and the effect of these techniques on structural and functional reliability models will need to be assessed.

4.3.6 Support Systems

Computing systems which are used to support and assist software development must themselves be very reliable. Obviously, a software fault in, for example, a compiler or run-time support environment, could undermine the best efforts to construct highly reliable software. Ensuring the reliability of the software embodied in an IPSE is certain to be a major undertaking, but one which must be tackled.

4.4 Results of the Programme

(i) Short term (2½ years)

Within the first few years of the programme much of the fundamental research work will be done. The expected results in this time period are:

—NQCC using improved reliability models (such as adaptive models) for certification
—mathematical formulation of new classes of reliability models incorporating additional information (project characteristics, development techniques and processes, and customer profile)
—handbooks for developers outlining best of current practice in producing reliable software
—guidelines for project managers based on new models of the software development process incorporating cost and quality considerations
—data collection processes established and working.

(ii) Medium term (5 years)

During this time period the theoretical models developed at the start of the time period will be evaluated and refined using data from a variety of software projects. In addition work will continue on developing reliability models tailored to special types of system (real time, embedded, distributed, etc.). The expected results by the end of the programme will therefore be:

—NQCC using new classes of reliability models incorporating data collected from ISF's for product certification
—risk models of reliability identifying product characteristics and development methods most likely to achieve various levels of reliability (in principle these will be similar to medical risk models which identify the characteristics of groups at risk from various diseases).
—mathematical formulation of reliability models for special systems (validation may not be possible within these time scales)
—improved software engineering techniques to enable the construction or more reliable software
—new handbooks incorporating new information resulting from the entire software engineering programme and validation exercises performed on new development techniques
—cost and quality models incorporating and reconciling macro-level (strategic) and micro-level (project management) decision making
—automatic collection of project data for certification and industry standardisation by ISFs.

(iii) Long term (5 to 10 years)

After the completion of the current Alvey programme, work will still need to continue to incorporate the new methods of software development and novel system architectures developed by Alvey collaborators into reliability modelling and certification procedures.

It can be anticipated that models of system and software reliability will need to be adapted and validated as new concepts of software engineering continue to emerge.

5. Education, Technology Transfer and Technology Assessment

The buzz-words 'technology transfer' have come to be standard for the process of educating a technical community in new results. There are, in fact, two types of educative processes which need to be considered here. The first process, which involves keeping the community (generally of researchers, but also 'real' software engineers) abreast of new developments, we shall continue to call technology transfer. The second process, in many ways more important, is the general education of a wider community. This latter topic will now be discussed.

5.1 Education

It has been argued that, before large sums are spent on developing new tools for software development, software developers should be persuaded to use the ones which are already available. Clearly, these strategies are not mutually exclusive. However, the point is well taken: many techniques which are widely recognised to be useful are used less often than they ought to be. It is important that this poor take-up by potential users is not a feature of the Alvey programme.

The problem is one of general education, and positive selling of what is available. Of course, parts of the research programme described here will have an important function in providing concrete evidence for the real-world effectiveness (or lack of it) of various approaches. If managers can see the cost-effectiveness of a technique, they will use it. Such results must be disseminated to a wide audience.

The Alvey SE directorate will support a programme of workshops and courses aimed at practitioners. Particular workshops and courses will be repeated at different times and venues so that potential attendees can have some choice.

The Alvey SE directorate will sponsor a series of handbooks, regularly up-dated, dealing with specific tools and techniques.

Both these activities will involve the research community. Although researchers may see this educative role as a distraction from their primary occupation, the benefits from contact with practitioners, and thinking about the real problems of practitioners, are immense.

This programme should start as soon as possible. In the early stages it will be reporting on work which pre-dates the Alvey programme, and it is only later that results from this programme will become available.

The software reliability education programme will be run as a part of the wider SE programme. In fact, integration of the reliability aspects (the visible results) with the process aspects (the detailed description of the tools and techniques) will be a powerful persuader to potential users.

It is important that the country's educators are kept abreast of developments in the area of software reliability. Lecturers in universities and polytechnics will be encouraged to participate in workshops and courses sponsored by the Alvey directorate. In addition, particular attention will be paid to this audience in the dissemination of research results.

Finally, the Alvey programme will examine ways in which the best and most up to date techniques can be incorporated into routine teaching practice.

5.2 Technology Transfer and Assessment

The technology transfer problem is well understood, although its solution is not obvious. The phrase 'technology assessment' is used here to emphasise the judgmental role involved in deciding which parts of world research are worth disseminating to our own community.

The technology assessment function is the first stage in a process which would aim to make available to UK industrial and research communities the best of world research results. Thus the end-user of the service will be wider than the Alvey community (although this community will be an important part of the audience). For this reason the judgment and assessment roles are paramount. These can best be fulfilled by individuals who are actively engaged in research.

The technology assessment function requires action in three areas:

—information gathering
—information assessment and evaluation
—information dissemination.

5.2.1 Information Gathering

The information that is needed comes from three sources: articles in technical journals and books; reports of conferences and workshops; personal contact between researchers.

Journals and books present formal descriptions of research results, but may be up to two years out of date. At the other extreme, personal contact provides up-to-date information, but this may be anecdotal and speculative. Conferences and workshops form a middle ground in both formality and timeliness. All three sources should be exploited.

5.2.2 Information Assessment and Evaluation

The most difficult task is the formation of judgments about the worth of research results revealed in the information gathering exercise. Inevitably, this role must be carried out by research workers who are technically competent to make critical judgments about the likely usefulness of research work. Perhaps more importantly, such individuals will be members of informal networks within the international scientific community. This will enable them to keep a finger on the pulse of work in their field around the world. Indeed, this is something that active research workers expect to do as part of their own need to keep up-to-date.

Assessments should be judged according to their firmness. Thus there should be early warning of work which looks likely to bear fruit in the future, as well as dissemination of hard results.

5.2.3 Information Dissemination

This is likely to be a two-way activity. For potential users seeking advice, it is important to have a central facility which can steer requests for information and advice towards sources and expertise. Such an organisation should also initiate other dissemination activities: publications, workshops, seminars, etc.

5.2.4 Infrastructure

The Alvey programme will examine the most appropriate ways of establishing an infrastructure capable of supporting these activities.

6. Conclusion

It is essential that the UK retain and if possible expand its share of the world IT market. In order to do this innovative product development will not be enough. Customers will demand high quality and, in particular, high levels of reliability from software-based systems. A considerable competitive advantage will accrue to software producers who are prepared to provide warranties to their customers backed by independent product certification. If UK Companies are unable to provide this, there is no doubt that the US and Japanese companies will.

Therefore the Alvey Directorate will back a major research programme in software reliability as detailed in this document. The objective of this research programme is to provide the capability to design and build cost-effective reliable systems such that:

—a valid base for independent software certification and warranty is achieved
—software reliability can be measured, predicted and controlled
—evaluation of development methodologies is based on objective measurements
—the development of reliable software can be appropriately engineered and managed.

Acknowledgement
CSR wishes to express its gratitude for the helpful comments and suggestions made by several members of the software engineering community on earlier drafts of this document.

Appendix A

Research Activities and Priorities

A coherent research programme has to have priorities. The priorities for this research programme, which follow, have been determined using the following pragmatic criteria:

(i) importance of research area in terms of the likely contribution which will be made to overall Alvey objectives;

(ii) chances of success within reasonable timescales and likely available budgets.

The priorities which follow have been classified A, B. A represents a programme which is vital and fundamental to the success of the Alvey objectives and will often be supportive of other parts of the programme. Programmes with category A stand a high chance of successful completion.

Category B contains programmes which generally contribute less to the Alvey objectives, and often carry a high risk of being unproductive in the timescale envisaged.

Priority A programmes will normally be funded by the Alvey directorate, whereas priority B programmes will be funded only if resources are still available. Note that priorities do not refer to *amount* of support, or the *time* when support should begin. These are dependent upon the nature of the particular research programme and its relationship to other programmes.

Priorities

Software Management

A. Assessment of existing managerial tools, techniques and procedures, and identification of missing or inappropriate elements (4.1.1.3 (b)).

B. Preparation of management guidelines and procedures (4.1.1.3 (d)).

B. Review and co-ordination of new tools and techniques (4.1.1.3 (c)).

Comment: Management guidelines and procedures will not be the sole concern of the Alvey programme. These have traditionally been the responsibility of other bodies, e.g. MoD, BSI, etc.

Engineering for Reliability

A. Assessment of effectiveness of validation and verification techniques, leading to recommended inclusions in IPSE's (4.1.2.2 A).

A. Fault tolerance in concurrent and real-time systems (4.1.2.2 B).

A/B. Reliability aspects of reusable software (4.1.2.2 C).

Systems Issues

A/B. Reliability aspects of integration (4.1.3.1)

A/B. Assessment of suitability of tools and techniques to reliability related requirements such as safety, security, integrity, etc. (4.1.3).
(This has links with 4.1.2, priority A, above.)

Comment: The importance of this work would warrant a category A priority. However, it is likely (and probably desirable) that it be supported by bodies other than the Alvey directorate, e.g. MoD, UKAEA.

Understanding Causes of Unreliability

A. Micro and macro levels of understanding (4.2.1.1).

B. MMI reliability issues (4.2.1.3.4).

B. Linking of micro and macro views (4.2.1.1).

B. Cognitive processes (4.2.1.2).

B. Human factors (4.2.1.3.1).

Comment: The category B programmes are important but likely to be integrated with other aspects of the Alvey programme.

Data Issues for Support of Reliability Research (4.2.2)

A. Identification of data requirements, for reliability and other software metrics.

A. Specification of database system (i.e. one or more databases, as appropriate to the task).

A. Creation and management of database.

Reliability Models and Tools

A. Characterisation of program properties (4.2.3.4.2).

A. Model improvements and developments (4.2.3.2).

A. Tools identifying good predictions (4.2.3.1).

A. Characterisation of user environment (4.2.3.4.1).

A. Fault-tolerance: economic optimality issues (4.2.3.3.3).

A. Structural models for fault-tolerant systems (4.2.3.3.1).

A. Fault-tolerance: independence issue (4.2.3.3.2).

A/B. Structural models for reusability (4.2.3.3.1).

B. Testing strategies for consequences of failure (4.2.3.5).

B. Testing strategies: conflict of interest (4.2.3.5).

B. Adaptive models (4.2.3.2).

B. Accelerating testing (4.2.3.5).

Cost/Resource Models (4.2.4)

A. Evaluation of current cost models, identifying inadequacies and areas of potential development.

B. New cost models. Customised cost models for specific environments.

Quality Engineering (4.2.5)

A/B. (The precise category depends upon the extent of research activity in this area under the other parts of the SE programme.)

Technique and Tool Evaluation
A. Evaluation of techniques and tools (4.2.6.2).

B. Automated data collection (4.2.6.1).

A. Validation of metrics and models (4.2.6.2).

Systems in Practice
A/B. Critical systems; ultra-reliable systems (4.3.1).

Comment: There are many important issues in this area. The only reason for it not being a top priority for the Alvey programme is the likelihood that other organisations will be taking responsibility for it.

A. Real-time systems (4.3.2).

A. Embedded systems (4.3.3).

A. Distributed systems (4.3.4).

B. Future systems (4.3.5).

A. Support systems (4.3.6).

Appendix B

Contributors to Report:
Members of the Centre for Software Reliability

Mr. B.G. Anderson,
British Aerospace Dynamics Group,
PB 225,
P.O. Box 19,
Stevenage,
Herts. SG1 2DA.

Dr. T. Anderson (Director),
Centre for Software Reliability,
The Computing Laboratory,
University of Newcastle upon Tyne,
Newcastle upon Tyne NE1 7RU.

Mr. C.J. Dale (Secretary),
National Centre of Systems Reliability,
UKAEA,
Wigshaw Lane,
Culcheth,
Warrington,
Cheshire WA3 4NE.

Mr. B. de Neumann,
GEC Research Laboratories,
Marconi Research Centre,
West Hanningfield Road,
Great Baddow,
Chelmsford,
Essex CM2 8HN.

Ms. G. Frewin,
Standard Telephone Laboratories,
London Road,
Harlow,
Essex CM17 9NA.

Mr. C.H. Gribble,
Ferranti Computer Systems Ltd.,
Cheadle Heath Division,
Bird Hall Lane,
Cheadle Heath,
Stockport,
Cheshire SK3 0XQ.

Mr. L.N. Harris (Chairman),
British Aerospace Dynamics Group,
PB 401,
P.O. Box 19,
Stevenage,
Herts. SG1 2DA.

Dr. B.A. Kitchenham,
International Computers Ltd.,
Westfields,
West Avenue,
Kidsgrove,
Stoke on Trent ST7 1TL.

Dr. B. Littlewood (Director),
Centre for Software Reliability,
The City University,
Northampton Square,
London EC1V 0HB.

Mr. A.A. Wingrove,
Royal Aircraft Establishment,
AW3,
Farnborough,
Hants. GU14 6TD.

Document prepared by:
Ms. G. Palmer,
Centre for Software Reliability,
The City University.