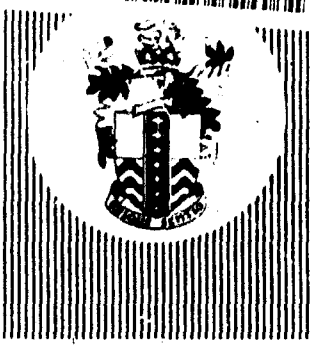
UNLIMITED



Report No. 91016

Report No. 91016

ROYAL SIGNALS AND RADAR ESTABLISHMENT, MALVERN

TRACEABILITY AND CONFORMANCE IN SECURE SYSTEMS

Authors: G P Randell & C T Sennett

PROCUREMENT EXECUTIVE, MINISTRY OF DEFENCE
RSRE
Malvern, Worcestershire.

August 1991

CONDITIONS OF RELEASE

0107605 304210

MR PAUL A ROBEY
DTIC
Attn:DTIC-FDAC
Cameron Station-Bldg 5
Alexandria
VA 22304 6145
USA

********** DRIC U

********** DRIC Y

Reports quoted are not necessarily available to members of the public or to commercial organisations.

ROYAL SIGNALS AND RADAR ESTABLISHMENT

Report 91016

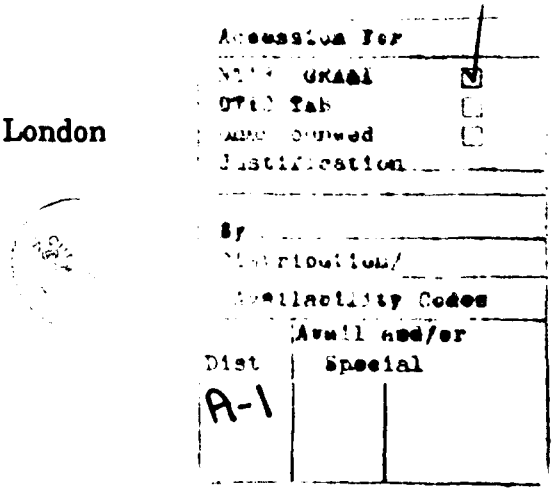
Title: Traceability and conformance in secure systems

Authors: G P Randell and C T Sennett

Date: 7th August 1991

ABSTRACT

Traceability in a software intensive system is the ability to link statements of requirement with the implementation objects which satisfy them and the means used to demonstrate conformance. This report discusses the problems of maintaining traceability when developing large secure systems and the ways in which technology may be used to support it.



1. Introduction

This report on traceability and conformance is concerned with the specification and implementation of software intensive systems, with particular emphasis on security aspects. Traceability and conformance are terms used loosely and it is the purpose of this report to discuss how they might apply to practical software development. The current level of practice might be summarised as traceability by cross references and conformance by testing. This level is not satisfactory but modest improvements in methods would radically transform the situation: future technological change will also help and the report discusses possibilities in this area.

1.1 The meaning of Traceability and Conformance

A software intensive system often starts life as a concept for meeting some military requirement, although in at least as many cases the system will be seen as an enhancement or replacement for an existing system. During the course of development from concept to implementation the system will be associated with or be represented by many documents or items of software. As the development proceeds, the documents become more and more detailed and explicit, ultimately ending up with delivered software and representations of the hardware configuration.

Conformance is the property that an implementation or specification produced during this development satisfies the higher level specification or requirement. Satisfaction conveys the idea of a particular instance of a general property. The requirement is written in general terms and the implementation must be shown to be a particular instance of it. The test for satisfaction depends upon the requirement. For example, response time might be a requirement and this would be satisfied by any system with a response time less than that specified. On the other hand, if the requirement concerned time between failures a satisfactory system would be one having a greater value than the one specified.

The conformance of an implemented system with a requirement could in principle be established by checking it against the requirement when it is actually delivered but this would be extremely unwise and almost certainly infeasible. It would be unwise because it is highly likely that the delivered system would not meet the requirements and be too costly to correct the deficiencies. It is likely to be infeasible because end-system testing and evaluation may not be able to demonstrate conformance adequately. This is particularly the case for security properties where end-system testing, no matter how desirable it is to do, is insufficient on its own to establish trustworthiness. Consequently it is desirable to establish conformance at every stage of the development process.

Requirements are extensive and need to be broken down into smaller items or statements of requirement (SOR). The system as a whole conforms when the implementation satisfies all SORs. For each individual SOR, it may be that most of the implementation is irrelevant. To carry out the conformance check, particularly when this is being

done by informally by inspection, it is necessary to identify the part of the implementation which is relevant. Traceability is the ability to do this, namely to relate an item in a specification or requirement to the item or items in the lower level specification or implementation which satisfy it, and vice-versa. Thus traceability relates an SOR or specification with a specification or implementation via a conformance check. This can be illustrated with the model in figure 1. A software development has high traceability if all the SORs in the requirement can be related to objects at every stage in the development with the conformance report. Note the inclusion of the conformance report which provides the reason for the trace relation as well as supporting the main use of traceability in demonstrating conformance. Note that all of these relations may be many - many: a given requirement may need many implementation objects in order to be satisifed and an implementation object may be associated with the satisfaction of many requirements. Many different kinds of conformance check may be employed.

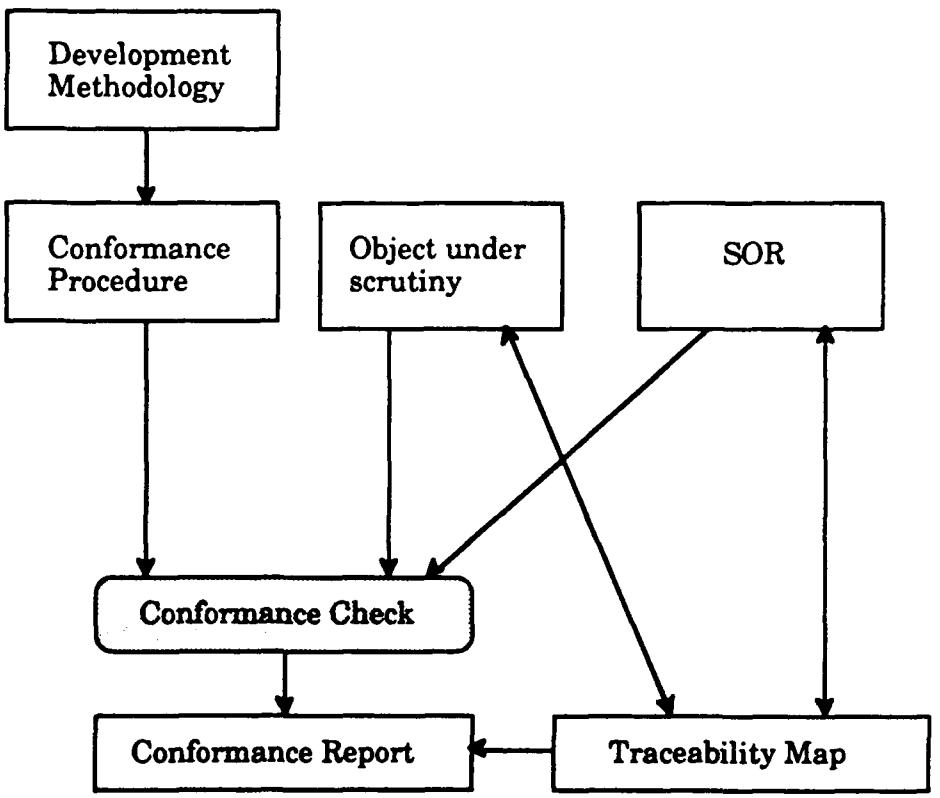


Figure 1 - A Model of Traceability and Conformance

This simple model of traceability needs to be supplemented for two reasons. Firstly, for reasons of quality and assurance, it is necessary to place requirements not only on the delivered system, but also on the system development process itself. A common example is the need to undertake configuration management. The procedures in this case are designed to ensure that a configuration can be identified and that alterations are regulated, a process which supports traceability of the other requirements. A development requirement is satisfied by the set of

procedures used in the development. Traceability is concerned with demonstrating that they have actually been obeyed during the construction of the system under development. Conformance is simply a check that audit records are present for all the objects and events which require it.

The model for this form of traceability is illustrated in figure 2. An example of a development step is the entry of a module of code into configuration control. The trace record relates the module being entered to the time of entry and the individual responsible, to ensure that the step is properly authorised. In development traceability, no intermediate representations are present and the audit trail represented by the trace records can be simply related to the requirement which gave rise to it.

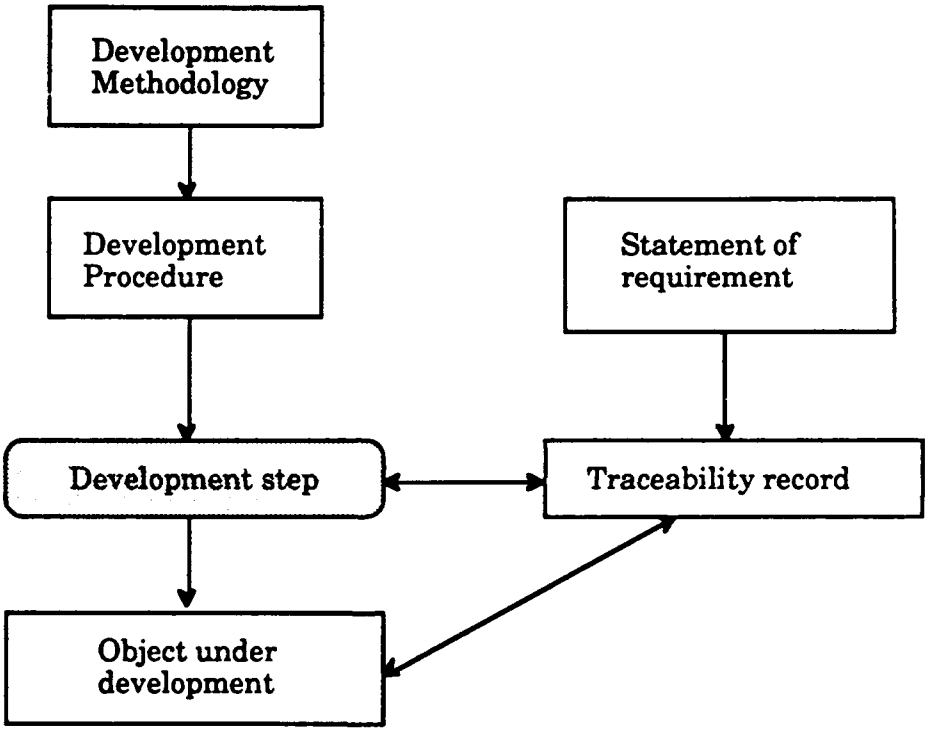


Figure 2. Traceability of methodological requirements

The second reason for supplementing the model of traceability is that at some point in the development tools can be trusted to guarantee the satisfaction of a requirement by an implementation. The most obvious example of this concerns the use of a compiler which is trusted to generate object code from the source text of the implementation language. Unless very high assurance is being sought, this stage of the development process is not usually checked apart from module or final acceptance testing. Analysis or formal verification is usually confined to the source text. Other examples consist of the system building and maintenance tools which are used to generate new issues of a system. Traceability for such unchecked steps requires evidence that only the appropriate tools have been used in the construction of the system. In

this case the evidence would consist of a demonstration that the operational procedures or software controls within the development environment are sufficient to maintain the integrity of construction. This type of traceability will be called constructional traceability.

To summarise, traceability in practice occurs in three different ways:

1. Standard traceability of a requirement to the implemented system, as illustrated by figure 1.

2. Traceability of development requirements, as illustrated by figure 2.

3. Traceability through representations generated mechanically, that is, constructional traceability.

1.2 The benefits of traceability

Traceability is a necessary part of demonstrating conformance. The decomposition of a requirement into units allows the conformance check to be broken down into manageable components. Traceability is necessary to demonstrate that the composition of the individual checks satisfies the total requirement. Traceability helps to show that all of the requirements are met and demonstrates *how* an implementation meets a requirement. Both of these add to the assurance of the implementation. Traceability supports the evaluation process by guiding the evaluators over the implementation.

Apart from these benefits which are directed at high integrity software, traceability helps with the general management and control of the development. The decomposition of the requirement helps in giving structure to the implementation but the traceability is necessary as the structure of the implementation will not directly correspond to that of the requirement. The structure involved also supports project control, costing and maintenance: the traceability relation gives the ability to analyse the impact of changes in the requirement. This applies in both directions. Traceability allows the part of an implementation which would be affected by a change in requirements to be identified. It also allows the conformance checking which would be necessary as a result of a change in the implementation to be identified. This aspect is particularly important during the rating maintenance phase of a project. Once a system has been evaluated the aim must be to identify whether a proposed change is security relevant or not. Traceability helps determine whether re-evaluation is necessary and the extent and scope of it if it is.

1.3 Techniques to support traceability and conformance

Traceability has traditionally been achieved informally. The original requirement is stated informally, the design is described informally, the conformance test is an assessment of a rationale document and the traceability is achieved by cross-references and the consistent use of names. Each of these things would be realised in some form of

documentation. Formal methods add precision to this process and also allow mechanical conformance tests to be applied, for example proof and analysis.

As traceability is concerned with relations it is natural to think in terms of a database to hold the various documents and maintain the traceability relations between them. Documents in the database may be edited by a word processor; this may provide support for cross references or for detailed document structure. Those aspects of traceability concerned with quality and integrity of construction can also be supported by the development environment which can be used to control access to the database of documents and the tools used to check conformance. There are technical developments in this area. There is much interest, both in the UK and the US, in the use of computers to support the acquisition process. The model for procurement envisages an information base from which documents relevant to the procurement are constructed, essentially as views on the database. The model does not explicitly deal with traceability, but the ability to structure information is a necessary prerequisite.

This report will review the extent to which current technology supports traceability and will argue for an integrated approach in which the information base contains the various tools used to demonstrate conformance, as well as text, and is structured to support traceability. Supported by a development environment providing adequate access controls, the whole would considerably reduce the costs of evaluation and simplify the construction of secure systems.

1.4 Current problems with Traceability and Conformance

Demonstrating traceability and conformance is not easy. The greatest problem associated with current projects is one of scale and complexity. A typical requirement for a command and control system, for example, may extend over a 1,000 pages. A typical approach to traceability would be to record all occurrences of the word "shall" as indicating a SOR. At an average of, say, 5 per page, this would amount to a cross-reference matrix of 5,000 entries and many of these entries would be multiple. Current practice does *not* associate the traceability with the conformance test, so the rationale for the reference is often lacking. A cross-reference from a requirement to a procurement specification may be traced, often only ending up with text which does not seem to bear on the requirement. This is often because the conformance check is lacking, which would explain why the reference is there, but more often simply due to errors in the matrix. Establishing the cross-reference matrix in the first instance is a major undertaking, but maintaining it is an immense task, mainly because of lack of adequate support from tools. The lack of association with conformance makes it impossible to tell when a requirement is completely satisfied.

Conformance has many aspects, usually associated with the roles of the persons having an interest in the system. Thus security is mainly of concern to accreditors and evaluators, but it has impacts on the potential users and, for control of the procurement, on the project management. Each of these different persons has some say in the acceptance of the system and consequently has some point of view about conformance tests. Traceability structure does not currently map on to these roles, which tends to lead to problems.

Traceability should be maintained throughout the lifecycle of the project, but tends to be applied only to the development phase. Loss of traceability during project definition and concept formation stages may result in inappropriate requirements being implemented as it may not be possible to say why a system is being built with a particular feature. Loss of traceability during the maintenance phases makes evolution impossible and even minor enhancements difficult. To support maintenance, two way traceability is required, that is, from the implementation back to the requirement and from the requirement forward to the implementation. The former is fairly easy to provide as the requirement can be copied or referred to from the implementation. The latter however, requires a new version of the requirement to be developed which contains the appropriate forward references.

The requirement for life-cycle traceability is particularly hard to meet when the development crosses a contractual boundary, for example, from project definition to implementation. The project definition results in the technical requirement specification which may then be issued for competitive tender. This often results in major loss of traceability from the earlier stages of the project. Another problem related to contracts concerns the conduct of evaluation. Because this is done as a result of an independent contract, this needs its own baseline document. The evaluation baseline will, however, have features in common with other points of view: for example the users will be interested in the security checks they must conform to and this should be described in the user functionality. Thus the evaluation baseline has to be extracted from the overall requirement and in the process traceability can be lost.

There are a number of technical problems too. During the course of development, the system will be described in many notations. Even the simplest systems will be described in English and in the implementation language, but there will usually exist other notations associated with the design method. Tools to support the differing notations rarely interact. The simplest possible traceability requirement would be standard use of identifiers, but because of lack of communication between tools this requirement is not well supported. There are no facilities, for example, to look up flow names on a dataflow diagram while in the middle of word processing for a design document. As a result inconsistencies arise and traceability is lost. At the moment there are no facilities to check that a formal specification in, say, Z, is compatible with a source text in an implementation language. This loss of traceability considerably weakens the chain of assurance in a high integrity development.

Traceability to hardware is also a problem, because the final target is neither software nor a document, and hence cannot be stored on a computer. The trace from a requirement to a hardware item which satisfies it must involve a manual procedure.

Security gives rise to another technical problem. Because security is essentially concerned with *forbidding* unauthorised information flows, the security requirement cannot be captured at one level of specification. As the development proceeds, adding further implementation detail, the security requirement may need to be re-established, to ensure that the implementation has not introduced vulnerabilities. Quite apart from traditional covert channel analysis there is a necessity to give a rationale for the conformance to the overall

security requirement in terms of the security properties of the constituent components.

Finally, for requirements on the development process, there is little support in standard development environments for controls which would ensure that unauthorised alteration of the information base was forbidden. There are two problems: one is that the granularity of protection afforded by standard operating systems is too large and the other is that the control afforded is of the wrong nature, namely discretionary access control rather than that required for release and modification of configurable items.

1.5 Structure of the Report

The remainder of this report is structured as follows. Section 2 discusses requirements, in particular the nature of security requirements. Section 3 discusses the fundamentals of conformance testing including formal and informal methods, testing and analysis. Traceability aspects of the technologies to support conformance are discussed in the next three sections, followed by a section on development traceability and the technology to support it. The report concludes with a summary of the recommendations.

2. Requirements

Conformance and traceability are affected by the nature of the requirement. A requirement such as limiting the total heat dissipation for a system to a certain amount, requires only tracing through equipment lists and the conformance check carried out with, say, a spread sheet tool. Conformance with a functional requirement can be established by software testing or by formal methods. As discussed previously, requirements are either system requirements which can be found in the delivered system, such as functionality or availability; or they are methodological requirements which may be found in the processes used for construction, such as the use of project control procedures; or they are implementation requirements which may be found in the implementation, such as the need for implementation in Ada.

Security requirements appear in all three categories. First of all there are the simple functional ones, such as the mechanisms for identification and authentication, the mandatory and discretionary access controls, audit, labelling and security officer functions. For security, the requirement is not simply that these functions should be present, but rather that they should demonstrably not be capable of being by-passed. This follows from the higher level security requirement that under no circumstances should information flow from a classified source to a more lowly classified sink. This non-functional requirement is quite difficult to test and may be given informally as requiring an absence of covert channels, a property established by evaluation In any case all security functions need to be evaluated to check for absence of by-pass and so any secure system will have requirements to ensure that the system is capable of being independently evaluated. The most important of these is an implementation requirement that the system software should be capable of being divided into trusted and untrusted software. The trusted software constrains the remaining untrusted software to

obey the overall security requirement of regulating information flows. Given an adequate regulating mechanism, only the trusted software will need evaluation. Traceability is particularly important here as it is necessary to exhibit the structure of the trusted software to show how the flow regulation is actually brought about. This will usually involve breaking trusted properties down into simpler ones.

This implementation requirement may be supplemented by others, which might be called defensive measures. Typical examples are the requirement for object re-use or the requirement to irreversibly encrypt passwords. These requirements are somewhat controversial inasmuch as they are implementation requirements and therefore solution-oriented, but they are nevertheless frequent in common practice. The traceability problem particularly affects the object-reuse requirement inasmuch as the granularity of objects chosen to satisfy the criteria needs to be justified.

The evaluatability requirement is further expressed in a system requirement that there shall be evaluation deliverables, which provide the evidence on which the evaluation proceeds. The traceability problem here is that the evidence required depends upon the implementation and so the requirement can only be specified in general terms.

Finally, the evaluatability requirement is completed by methodological requirements guaranteeing access to the evaluators for the purpose of inspection, configuration management controls, documentation standards and so on.

In summary, security requirements consist of security functionality, the requirement to exhibit the trust structure of the implementation and requirements for defensive measures, evaluation evidence and for trusted methodology. Each of these will need to be traced using the appropriate model.

3. Fundamentals of Conformance

Conformance may be demonstrated informally, by testing, by analysis, by proof or by construction. Conformance requires a specification, which may be given formally, or informally. By "informal" is intended not only natural language descriptions, but also the diagrammatic notations whose meaning is established by convention. This section will briefly describe the technical aspects of conformance, mainly with a view to establishing the terminology.

3.1 Formal Specifications

Most work on formal specifications has used a model based approach. The model of the system consists of a formal description of a state machine. An operation is a transition of the machine and so is specified in terms of the changes to the state. An operation is specified in terms of its precondition and its postcondition. The precondition gives the possible states under which the operation may be invoked, while the postcondition gives the new state achieved (possibly in terms of the initial state). The system specification is given by defining the state and the set of operations required. This is particularly useful for functional properties, but has the drawback that the explicit mention of the state in the specification may constrain implementations.

A rather more abstract approach is provided by the algebraic methods. In these the operations are given simply in terms of the types of the parameters and the expected results. A semantics is provided by giving equalities which are satisfied by expressions formed within the algebra. For example, within an algebra of numbers with addition as the operation it is possible to define zero using the equation x + zero = x, and so on. Algebraic specifications are particularly suited to relatively low level properties of an implementation, and may be used for compiler construction, or as the basis for analysis of implementations.

A requirement may specify behaviour rather than function: that is, it may constrain the order of events rather than the nature of them (which would be provided by a functional specification). For example it may be a requirement that one event may only take place after another. This is rather cumbersome to specify in a model based approach, so the usual solution adopted is to use a process algebra, such as CSP or CCS [Hoare 1985, Milner 1980].

Finally, it is worth recalling that a formal specification may not be part of the original requirement, but arise as a result of implementing that requirement. This particularly applies to process oriented specifications which may arise from the need to implement a requirement on a distributed system.

3.2 Refinement

Refinement is the technical term used to indicate all the means by which an implementation can be shown to satisfy a formal specification. There are several forms of refinement, depending on the nature of the formal specification and the type of conformance required. For simple functional specifications there are two varieties of refinement called operational and data refinement.

3.2.1 Operational refinement

Operational refinement is the process by which an operation specified at an abstract level is transformed into a program to carry out that operation, thus adding algorithmic content or other implementation detail. There are two fundamental rules for carrying out a refinement of an operation: strengthening the postcondition and weakening the precondition. The former means that more is known about the final state of the refined operation than about the abstract one, and the second means that the refined operation is applicable at least whenever the abstract one is. In other words operational refinement guarantees the functional conformance that the implemented operation works whenever it was required to and does at least as much. These fundamental rules may be supplemented by others which allow for the introduction of implementation language structures, such as conditional and loop statements, or the introduction of procedures and declarations.

The set of rules is called a refinement calculus, an example being that given in [Morgan 1990]. A program which has been developed by transformations from a specification according to the rules of the refinement calculus is guaranteed to conform to the function defined by

the specification. Each refinement step can be regarded as giving traceability of an element of the implementation to the design specification which gave rise to it. The conformance check is that the rules have been correctly applied, a process akin to proof.

3.2.2 Data refinement

Data refinement is rather more subtle than operational refinement, although there is only one rule to follow. It recognises the fact that a specification may be written in abstract terms, such as sets and functions, whereas the implementation must be expressed in concrete terms, such as arrays and linked lists. Data refinement is the process by which a specification, expressed in terms of operations on an abstract state machine, is translated into an equivalent specification expressed in terms of operations on a more realistic state machine. The new specification is the design for the implementation. Thus the abstract functional specification defines state variables and a set of operations which bring about changes in the state variables. These variables will be expressed in terms appropriate to the level of abstraction and so may use sets or functions without being concerned about how they may be represented in the machine. The design specification uses different state variables and different operations, which are specifications for what will actually be implemented. These variables will be defined in terms of implementation entities, such as arrays or linked lists.

For the design specification to be a valid refinement of the abstract specification, there must be a corresponding operation in the design for every operation in the abstract specification; the design and abstract state must be related so that it is possible to tell what abstract state is being represented by a given design state; and the design operations must have a corresponding effect on the design state as the abstract operations have on the abstract state. The relation between the abstract state and the design state is a very important element of data refinement: it is called the abstraction invariant.

The justification for the data refinement process is that the design is a model of the specification and does everything required by the latter. This is often represented by means of a commuting diagram, as shown in figure 3 below.

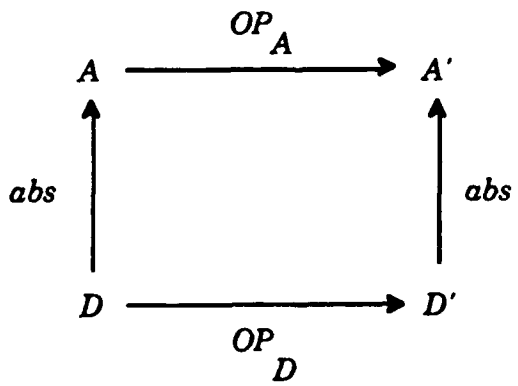


Figure 3. Data refinement

In this diagram, the abstraction invariant is represented by the function *abs*, which gives the abstract state *A* corresponding to the design state *D*. An operation OP_D takes *D* to *D'* which must correspond to the abstract operation OP_A taking *abs(D)* to *abs(D')* (that is, *A* to *A'*).

In order to demonstrate convincingly that the design specification is a valid refinement of the abstract specification, that is, that a correct implementation will be produced, it is necessary to prove the refinement theorems. The first is to show that the initial design state corresponds to the initial abstract state through the abstraction invariant, that is, that $abs(INIT_D) = INIT_A$. Then, for each operation, there are two theorems to be proved: the readiness and correctness theorems. The former demonstrates that the precondition of the abstract operation implies the precondition of the corresponding design operation, and the latter that the design operation produces the correct result.

Thus data refinement provides both traceability and conformance. The abstraction invariant, whether formal or an informal English description, describes precisely what concrete representation is used to represent an abstract representation. This gives traceability. Carrying out the refinement proofs demonstrates conformance.

3.2.3 Algebraic refinement and analysis

An algebra is implemented by providing a set of operations which satisfy the type constraints and equations of the algebra. The relation between two implementations of the algebra is called a homomorphism: a compiler for a language defined algebraically can be thought of as a homomorphism because it outputs an expression in object code for every expression input. This provides one approach to trusted compilation, which is part of development traceability. However the more usual way in which algebraic specifications are used is in terms of analysis. The algebra is used as a model of computation and required properties such as information flow properties may be expressed in terms of it as an implementation or interpretation of the algebra. This is illustrated in figure 4.

An example of this model is the use of MALPAS [RTP 1987] for information flow analysis. The computational model is the flow algebra underlying the MALPAS IL. For a given module implementing a functional requirement one can derive the MALPAS translation which forms the model of the implementation object. Non-functional requirements such as the absence of illegal information flows between security relevant implementation objects can be translated into the model, thus expressing such requirements as "the current security level is not altered by an untrusted function". With this scenario there is a two stage traceability: from the general non-functional requirement to the analysis procedure and from the analysis procedure to the particular implementation object being analysed.

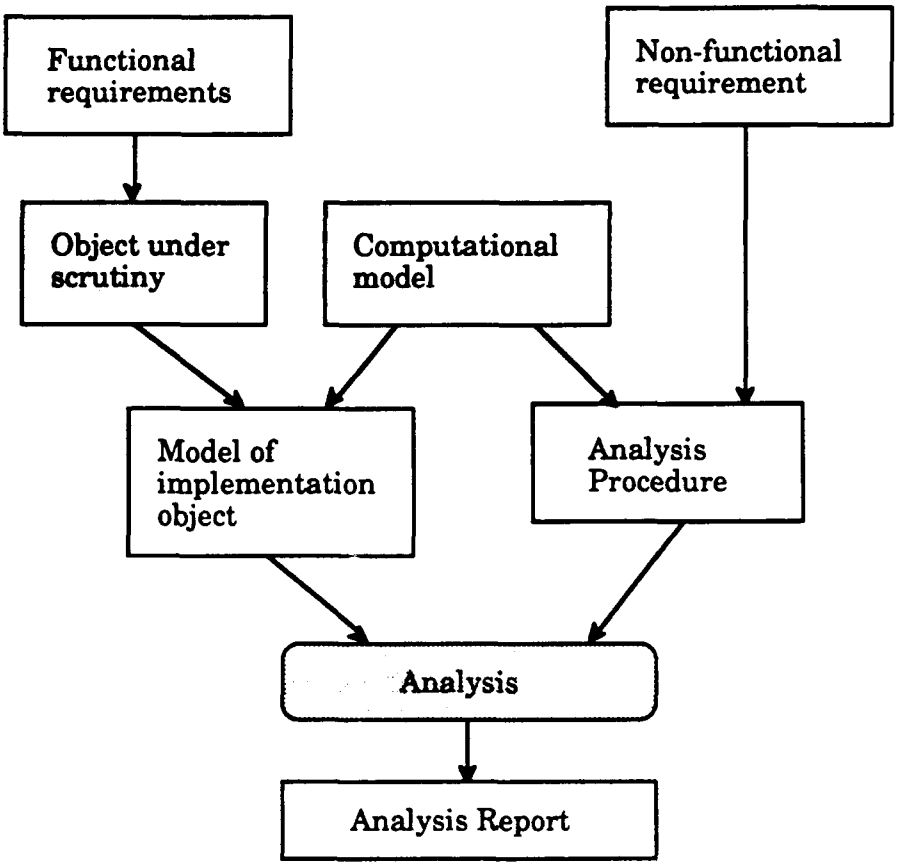


Figure 4. Model of algebraic forms of requirement

3.2.4 Behavioural Refinement

A behavioural specification consists of a set of allowable behaviours: a conforming implementation would exhibit one of these on execution. Consequently behavioural refinement is concerned with demonstrating that the set of possible behaviours of an implementation is a subset of that given by the specification. This is called safety refinement.

Thus behaviour consists of sets of objects: the nature of these objects is determined by what is meant by "allowable". If the property required is not affected by deadlock, behaviour can be represented by simple sequences of events, called traces. If deadlock is not allowable, behaviour must be more complex and contain the sets of events a process is prepared to engage in at any one time, rather than a single event. More complex interpretations of "allowable" require more complicated models of behaviour.

3.2.5 Promotion

Promotion is a refinement method used when a state based specification is to be refined into an implementation involving parallel processes, as for example when implementing a distributed system. It is a form of data refinement, where, in the design specification, the state data are distributed across more than one component, such as, for example, occurs with a central database computer and distributed workstations. As a result of distributing the state data, the operations are decomposed into parts, each part occurring in a different component. The refined operation is constructed in stages. The first is to decide how to split the operation into its central and local parts. The local part describes the effect of the operation at a local component, that is, one of the surrounding components (e.g. a workstation). A promotion schema is then used to "scale up" (or promote) that local operation to determine its effect on all of the local components. Often, this may say simply that no other local component is affected by the operation. The promoted local operation is then combined with the part occurring centrally to give the complete refined operation.

As with any data refinement, traceability is given by the abstraction invariant which describes how to retrieve the abstract specification from the distributed design specification. Conformance is then shown by proving the refinement theorems as before.

3.2.6 Formal construction and type theory

Type checking may be used to show that functions are only applied to values within a certain set and also to specify the set of values delivered. Thought of in this way, a typing for a set of functions is a specification for them: the domain of the function is the pre-condition and the range is the post condition. Depending on the type system employed this may imply either a strong or weak specification. The usefulness of type checking probably lies as a mechanism for imposing a weak specification, as the type checking may then be established automatically. Typically, types are used to establish certain integrity properties, for example integrity of array access, unforgeability of references, non-interference between procedures and so on.

3.3 Informal Methods

Conformance using informal methods is concerned with demonstrating the satisfaction of a requirement by the production of rationale documents which are meant for human inspection. Nevertheless the principles of traceability still apply: the document should contain a reference to the SOR, the conformance check to be used, the object under test and the informal rationale for why the object satisfies the test. The first of these may be a reference and the rationale will be a narrative, so the documentation problems only arise with the object under test and the conformance test itself. In many cases, the conformance test could be an informal description of a formal method described above. The formal method used informally in this way gives a clear idea of the conformance requirement without the cost of presenting the specification or design in a formal notation. There will however be many

cases where one does not have a suitable formal method and must make do with a clear statement of the conditions for satisfaction. The object under test can either be some aspect of the design or a module of the implementation. For the latter case, the source text of the module can be incorporated or referred to in the documentation, so the main difficulty in establishing the conformance lies in maintaining it through the design stage.

In current practice, designs are usually expressed using a structured method. The term "structured method" is used to describe software analysis and design methods which are not mathematically based. Rather, they use diagrammatic techniques and offer guidance to control the whole development process. Examples of structured methods are HOOD (Hierarchical Object Oriented Design), SSADM (Structured Systems Analysis and Design Method), and Yourdon.

Structured methods use several diagrammatic notations, among the most common being data flow diagrams, entity life histories and entity relationship diagrams. The more prescriptive methods, such as SSADM, also insist on recording cross-references in documents using a fixed format. This helps to provide traceability. For example, the problem/requirement form used in SSADM to record problems found or additional customer requirements has a column for recording the data flow diagram or other diagram in which that problem is resolved or the requirement added.

The diagrammatic techniques themselves may be used to provide traceability and, to some extent, evidence of conformance. As an example, consider data flow diagrams (DFDs). These are diagrams which show the movement of data within a system and between the system and the outside world. An example data flow diagram, in SSADM style, is shown in figure 5 below. There are four components: external entities, which are sources or recipients of data outside the system; processes, which are activities which transform or manipulate data; data stores, which are repositories of data; and data flows, which show the movement of data, with the direction of flow indicated by an arrowhead.

Data flow diagrams are intended to be simple diagrams, which may be readily understood. As such, they should fit onto a single sheet of A4 paper. Unless a system is trivial, a single DFD will not contain all the necessary detail. To overcome this problem, a set of DFDs, in a hierarchical structure, is used. The top-level of the hierarchy is known as the Level-1 DFD. It establishes the basic characteristics of the system, namely the system boundary, external sources and recipients of data, main system inputs and outputs, and the main system functions. The main system functions are represented by processes on the Level-1 DFD and each of these may be expanded into a lower level DFD. Each process can be thought of as a "window" into a diagram at a lower level in the hierarchy. The lower level diagram contains an expansion of the detail of the higher level in terms of Level-2 processes, data stores, data flows and lower level external entities. These Level-2 processes may then be extended further and the procedure continued down through the hierarchy until a satisfactory level of detail is produced.

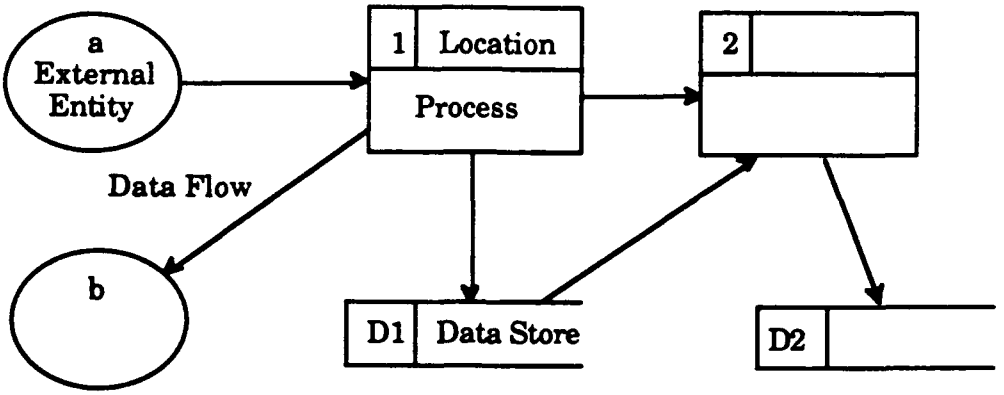


Figure 5. A data flow diagram

Traceability is maintained through the process identifiers. The processes at the top level are processes 1, 2, 3, etc. Each process within the boundary of the lower level DFD is identified by a decimal expansion to the higher level identifier. So, process 1 gives identifiers of 1.1, 1.2, etc. If process 1.1 were then expanded to a Level-3 diagram, the identifiers of the processes at this level would be 1.1.1, 1.1.2, etc. Simple conformance checks are also possible, although not required by SSADM. The only check that is made is that the lower level diagram has the same data flows into and out of it as the process it represents has in the higher level diagram. This check is an implicit one - it is a consequence of the suggested way of drawing the lower level diagrams.

Tools which support SSADM may carry out conformance checks on the diagrams, such as simple checks on the identifiers used, to ensure they conform to the convention described above, checks on data flows to ensure that they are consistent between diagrams in the hierarchy, as previously mentioned, and checks that the decomposition of data flows into more detailed flows are recorded as a form of data refinement in the accompanying data dictionary.

The emphasis in structured methods is to control the development process and document all stages thoroughly. This provides traceability, but, as yet, little emphasis has been placed on providing written evidence of conformance. Rather, walk-throughs and reviews are used.

3.4 Testing

Testing consists of executing software, under controlled conditions, to observe whether or not predicted results actually occur. The test process requires three inputs: the object under test, the input data for the test and the expected results, the latter often being called the test oracle. The input data and the oracle need to be prepared from the specification of the object. If this is all that is used, the testing is called black box testing because no account is taken of the design of the object. A more aggressive form of testing seeks to explore likely weaknesses in the design (such as initial and end cases) and is called white box testing. Thus a model for testing is given below:

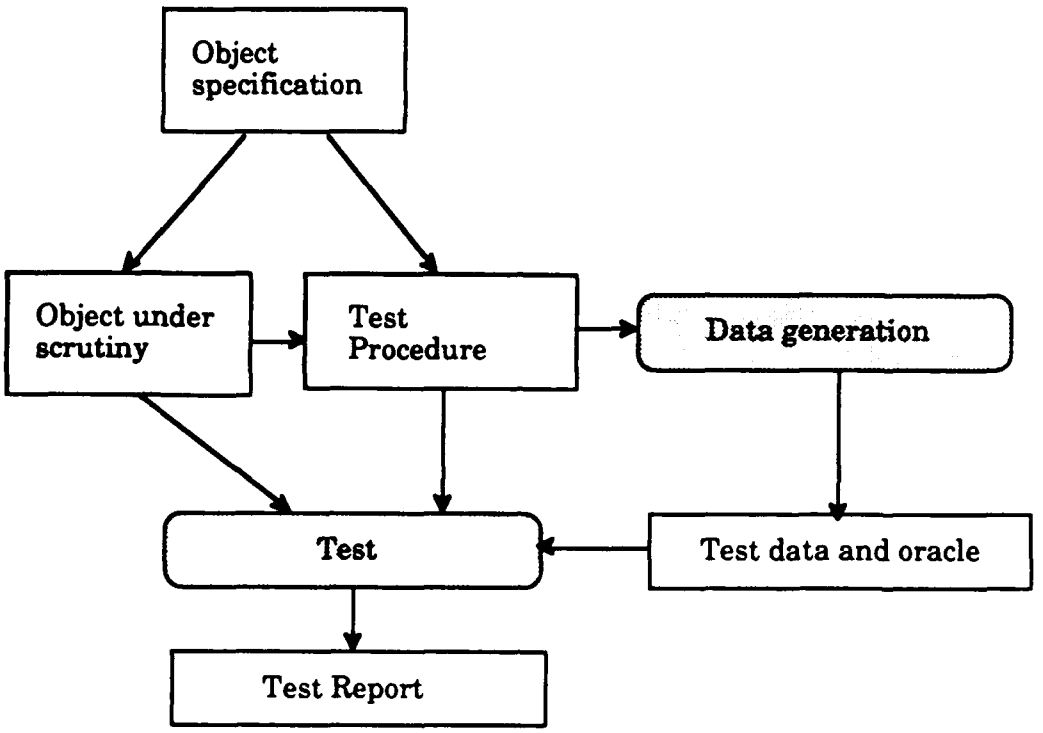


Figure 6. A model of testing

Traceability within this model is to some extent determined by the nature of the test procedure and the weight which is to be put on testing as an element of conformance. In an ideal, fully automated procedure, the test could be characterised by an integer, a seed for random data generation, with the oracle being determined by the specification and the test result being capable of analysis automatically. One then simply needs to record the input seed, the object under test and the result of the test. Usually however, testing does not achieve this level of automation and much more will need to be recorded.

Testing can be regarded either as an element of acceptance or as a necessary part of quality in high integrity software development. If the former, it becomes particularly important to trace the tests to the specifications and SORs it is supposed to satisfy. However, as an element of quality, it may be necessary only to demonstrate that the system has been thoroughly tested, in which case the traceability is to the methodology and one simply needs to demonstrate that agreed procedures and standards for testing have been complied with.

It is of course well understood that testing is limited in its ability to demonstrate conformity, because the test only applies to the particular situation represented by the input data and there are many more possibilities for input than can be tested in any reasonable length of time. There are however some measures of the extent to which the test has exercised the software: it may not be possible to exercise all possible input values, but it is possible to ensure that all instructions have been obeyed. The degree to which the software has been exercised is called a

coverage measure and there are several possible metrics. It is desirable to work in terms of an agreed measure, which can be established by analysis, and incorporate this into the test procedures.

Testing can be undertaken at three stages within the development:

1. Module tests - dynamic testing of the basic individual modules of the system in isolation, performed against the corresponding parts of the design specification.

2. Integration tests - dynamic testing of progressively larger combinations of modules, performed against the design specification.

3. System tests - dynamic testing of the developed system, performed against the functional specification.

Module and integration testing is usually done as part of the development methodology, while acceptance testing is usually done as a system test.

4 Traceability aspects of documentation

Documentation is easily the most important component in traceability and conformance. Requirements originate as informal descriptions and can only be given as a document. Many requirements can only be satisfied informally and there will always be limits to what can be formally specified, verified or tested. From our model of traceability, when the conformance is informal, one is required to identify within the documentation the original SOR, the object which (partially) satisfies it and the conformance report. The maintenance of the document must involve the maintenance of the relation between these entities and, in addition, the informal conformance test must be a rationale document which is available for human inspection. This latter requirement does not exclude structured documents, but does mean they must be suitable for human perusal. In practice this may well require the production of a paper document.

There is a lack of commercial products to support traceability. In the absence of specific tools, off-the-shelf word processors are often used and indeed, their use has revolutionized the production of requirements documents. The main problem with word processing is that it does not support the structure which is a necessary part of the requirement. At the very least a better identification of a SOR than the use of the word "shall" is required, but for the purposes of traceability it is necessary to have associated with the SOR the checks which will be used to demonstrate conformance to it and some means of referring forwards to the implementation or design which satisfies it. The check backwards from the implementation to the requirement is more easily provided as the requirement is more stable: it can either be referred to or quoted. However, even in this case there remains the problem of ensuring the requirement is not inadvertently altered.

A word processor does not naturally support different viewpoints of the information which is required to control a project. An example

from security is that the evaluation contractor will need a baseline document describing the trusted functionality to be assessed. However the impact of the security mechanisms on the functioning of the system is also of keen interest to the users of the system. To meet this problem the evaluation baseline is usually extracted from the requirement specification, leading to copying and the possibility of inconsistency. Finally, there is an immense problem in maintaining references, particularly when the system design changes representations. In this case there is a need to be able to communicate identifiers between, say, a diagram editor and a word processor, to achieve at least some commonality of naming conventions.

One can summarise the deficiencies of current technology as lack of means to impose structure on the documentary information, together with a lack of integration between tools. There are however a number of promising future developments. From the point of view of structure, there is the availability of database technology. The problem itself is more oriented towards an entity relationship database rather than a relational one and it may well be that the technology will be provided by advances in development environments rather than databases as such. The PCTE development [Lyons and Tedd 1987] provides an example of an environment which would support the information base approach. Using such an information base it is possible to identify SORs more easily; to classify them according to their relevance to a point of view such as security; and to associate them with conformance checks and the implementation elements which satisfy them, using simple database manipulations. This use of structure for the purpose of traceability must fit in with other uses of requirements information. Consequently work is also needed to establish the structures which would support the human process of procurement.

Apart from databases, structure can be provided by hypertext and by processing based on tags. The hypertext concept supports the ability to navigate around a document and so provides part of the solution for cross-references and structure. It does not support the concept of viewpoints and the current technology is primitive, particularly when concerning maintenance and alteration of the structure.

An important example of processing based on tags is given by Knuth's WEB program [Knuth 1984]. Here the structure is used to present a program in a "literate" way, that is, in a form suitable for human perusal, not the form suitable for the compiler. The programming language text is presented in parts interleaved with informal commentary, providing a natural and elegant solution to the traceability problem at the level of the detailed program text. The programmer works on the literate text, thus having as a main aim the production of the evaluation deliverable which relates the program to the requirements it is to satisfy. The WEB program extracts the program text from the literate text using the tags as a guide. This is a simple and very cost effective approach which should be adopted whenever feasible.

Integration of tools and support for cross references between representations has two problems, one technical and the other practical. The technical issue concerns the capabilities of tool interfaces. At the moment the concept of a tool interface is of a passive object, essentially not much more than a file of unstructured data. Integration of tools will require much more capability than this, in particular the support for

procedures as first class objects and the support for persistent, structured values. Few environments provide this. The practical problem is concerned with the integration of existing tools. It is unlikely that it will be possible to give up commercial word processing and so there will be a requirement to deal with texts provided outside the information base. Clearly there will be a need to transfer data from commercial word processors into the information base and vice-versa. The transfer process, by supporting conventions and checking cross-references, could support the integration of tools, although in a somewhat cumbersome fashion.

The need for import and export is a requirement for any procurement process based on electronic document interchange, such as that proposed for the US Department of Defense computer acquisition and logistics support (CALS) strategy, or the equivalent UK initiative. Both of these are aimed at transforming procurement information into electronic form over the next decade. The document interchange standard proposed uses the ISO standard generalized markup language (SGML) [ISO 1986]. SGML provides a system-independent means of marking up the text of a document, describing its content (e.g. chapter, heading, paragraph, list item) rather than its typographical features (e.g. new page, 14pt Times bold, italic). The SGML standard does not describe a language in which to encode documents. Rather, it gives a metalanguage - an abstract syntax for describing the character set and syntax of encoded documents, together with a default concrete syntax in which to code such descriptions. SGML is designed for full multi-media database publishing. It addresses the requirement for a straightforward means of passing coded documents unambiguously between systems for the raw text of documents, independent of page layout, hyphenation, etc.

The origin of SGML is with its use for typography, but because it is a metalanguage it supports any sort of structure and a variety of usages. In particular it provides a way of defining an interface between the information base and the commercial tools used with it and for structuring documents into requirements, traceability references and so on.

5 Traceability aspects of formal methods

5.1 Security and refinement in practice

Formal methods may be used to specify simple security functions, such as a security officer function, or more global information flow requirements. In either case, security considerations complicate the issue. For information flow requirements the preferred methodology is given in CESG Computer Security Manual F [CESG 1991]. In this methodology, the security properties of interest are expressed in terms of a security model, a finite state automaton in which the security elements of the state are made visible as subjects and objects and the security properties are asserted in terms of this automaton. At this level of description, information flow *constraints* can be expressed in terms of non-interference assertions [Goguen and Meseguer 1982], very general input-output properties which constrain all information flows supported by the implementation to obey the security policy. A security model in this form can express many different security properties, according to the security policy required.

For a formal development, to level 6 of the ITSEC criteria [ITSEC 1990] for example, it is required to develop a formal specification of the design as a state machine. Subjects and objects are instantiated in the model by the design entities and the design state machine is interpreted as an automaton required by the model. The security properties of this interpreted automaton may then be verified. From the traceability point of view, the model and its associated security properties form the requirement, the design specification satisfies it and traceability is provided by the interpretation function. Conformance is given by proof that the property required is present in the interpreted model. For developments at a lower level of assurance the formal method can sensibly be used as a basis for an informal description.

The security requirement is expressed as a property of the state machine at a given level of refinement. The level chosen is to some extent fixed by the need to specify the interface to the trusted computing base (TCB), either as a formal or descriptive top level specification (FTLS or DTLS) according to the assurance level. In the case of a TCB implementing security controls such as file access for example, the controlled operations would consist of things like read and write accesses to files. The conformance of the TCB to the security requirement is given by the functional specification of the operations together with the reason for the controls not being subject to by-pass. This would be related to the protection mechanisms, either hardware or software, which are employed.

The user requirement actually arises at a higher level than this. The basic transactions of interest to the user are made up of many TCB operations. Consequently the traceability problem arises as to how to progress from the user level requirement to the FTLS and how to ensure that the user level requirement is consistent with the security properties. For example a command and control system implementing electronic mail will usually require the notification of mail arrival, even if classified, to a user currently working at an unclassified level. Another example is the security condition for navigating through an entity relationship database. Navigation through a node of the database should not necessarily raise the security level when the node is pointing at a classified object. These problems can be handled by giving very careful consideration to what consitutes an object in the model. This will, of course, have to be related to what threats this would expose the system to.

Unfortunately, this is an implementation rather than a requirement activity. The precise way in which security objects relate to user entities is a matter for implementation, so it is hard to specify security aspects of user functionality without pre-empting the design. The best approach seems to be to give the user requirement in the form of scenarios, which can be used to check the design as it evolves.

Thus the procedure for traceability from the requirement to the DTLS may be given as

1. Establish the security aspects of user requirements in the form of scenarios of allowable behaviour.

2. Express the general information flow constraints in the security model.

3. Establish the boundary of the trusted code using appropriate protection mechanisms (hardware or software).

4. Define the FTLS as state and operations at the TCB interface.

5. Interpret state in terms of security objects and show the model properties are satisfied.

6. Check scenarios to ensure that desired usage of operations is achieved.

Functional security requirements are relatively easily traced and not usually included in the procedure above. What is required is the ability to demonstrate correctness, as with other functional requirements, supplemented by absence of by-pass. This is equivalent to step 3 above.

Security considerations are also needed during subsequent refinement of the design to an implementation. Neither operational nor data refinement is guaranteed to preserve security as either may introduce additional state (intermediate results or implementation data) which give rise to undesirable information flows. There is also a continuing interaction with user requirements. Taking file access as an example again, the operational refinement of the user level transaction will usually involve read and write operations to several objects. A standard problem is that the inability to sanitize the workspace leads to overclassification of written objects and an unusable system. Again this usability aspect is an implementation dependent matter and difficult to put into a requirement. The solution is probably to ensure that scenarios cover this aspect. For example, a scenario for use of a tote display could specify that it should be possible to update one element of the display without the updated element taking the classification of the display as a whole.

On the implementation side, many of these problems may be controlled by hiding the implementation data from the untrusted code: that which cannot be seen cannot give rise to security problems. Formal techniques for addressing this include defining operations in terms of abstract data types and guaranteeing no access to the implementing type. This use of type checking to support security is little used at the moment, although it appears to be a powerful and low-cost mechanism.

The other issue needing to be addressed during implementation is concerned with the integrity of objects. The objects of a security specification are refined into implementation entities such as records and arrays. If it is possible to forge references or access outside array bounds, the implementation will be able to violate security while meeting the requirements of the security model. This traceability requirement can be met by constructional traceability on the compiling system, provided it supports object integrity adequately. If it does not, it may be necessary to analyse the compiled code.

Security implementation requirements also have traceability problems. A requirement like object re-use for example will need to be defined in terms of objects at one level of refinement. Usually this would be applied to backing store objects, but may be extended to refined forms involving buffers and so on. There is also an operational refinement

problem in deciding whether to standardise backing store on release or allocation. Any of these cases could be acceptable, but will need to be traced with a rationale based on a risk analysis demonstrating that the one selected meets the requirement.

5.2 Future developments in formal methods and traceability

The above discussion shows that although the fundamental mathematics supporting security and software development is well understood, technology and methods to support the practice are not well developed. The main problem is notational complexity compounded by a lack of support for structural decomposition. Notational complexity is an almost inevitable consequence of the introduction of any method of precise specification. The counter to it is to choose a level of abstraction appropriate to the properties under consideration together with a decomposition of the problem into components. For example, it would be useful to decompose an overall specification for a CCIS into separate security requirements for database and workstations.

There are a number of technical approaches under consideration at the moment which may help overcome this problem. The first is to consider the expression of security requirements in a behavioural way, rather than using state machines. The state based approaches tend to force implementations, whereas behavioural specifications are more abstract. Furthermore, they support the decomposition into components as a parallel composition of processes. The technical problem with this approach is a lack of understanding of the practical consequences of the approach and a lack of technology to demonstrate compliance with a behavioural specification.

Within the state based approach, structural decomposition is addressed using the promotion technique. This should be fairly straightforwardly applicable to the formal methodology described above, but there are no worked examples of its use.

There is also a rather large gap in technology between the FTLS and the implementation. Having proved consistency between a security model and the FTLS, assurance criteria tend to call only for testing and informal methods. The problem is that operational and data refinement taken down to the code level are not well supported by tools and currently tend to be too resource consuming to be cost effective. At the code level, conformance is usually given by annotations. It is good practice to insert these even if verification conditions are not going to be generated or proved. Unfortunately there is no tool currently available to give a formal check between a Z specification, for example, and an annotated program, so traceability from design specification to implementation has to be given informally.

Functional correctness is only one aspect of security and it is possibly more important to establish lack of by-pass. As discussed above, the technical approach here is to use type theories to express the requirement and type checking for conformance. This approach will be further discussed under the heading of software analysis.

6 Traceability aspects of testing and analysis

Testing and analysis are considered together because they have similar problems from the point of view of traceability. In either case there is a

problem in automating the generation of test data or analysis properties from a specification and equally a problem in checking the output of the test run or analysis procedure for compliance. There are very few cases where an automatic procedure can be applied and generation and compliance analysis nearly always involves human interaction. This makes analysis and test costly to incorporate within the software development process.

Testing and analysis can be considered as aspects of quality during the development process, or as techniques to use during evaluation. The traceability conditions necessary will differ according to the mode of use. Considered as an aid to quality, testing and analysis can be handled using development traceability. The quality consists in demonstrating that a given testing or analysis regime has been followed with evidence in the form of error rates, rather than details of individual errors. During evaluation on the other hand, one is probing the implementation for errors: this is rather akin to acceptance testing, an activity which is done towards the end of the implementation. Traceability forwards from evaluation is required for future maintenance of the evaluation rating, so it is still necessary. However, the situation is not as dynamic as during the actual system development itself, so the lack of automatic aids is not felt quite so strongly.

6.1 Traceability aspects of testing

For development quality it is usual to do black box testing in which the test data is generated from the specification alone. It will usually suffice to have test scripts which can be available for inspection by the evaluators. These can be assessed from the point of view of conformance informally, or with the aid of a test coverage monitor. It is useful if the test history of a particular module can be recorded. For high integrity developments, the trusted code should have few or no errors. For this mode of use, it is important to recognise when a module is submitted for test and distinguish this status from the programmer's normal exploratory testing as part of development. The maintenance of the test history and module status will require enforcing controls within the development environment if it is not done independently of the programmers by the evaluators.

For evaluation, it is important to do white box testing. During evaluation the requirement is to expose flaws in the design, so it is obviously desirable to explore weak points and the concept of test coverage is relatively meaningless as the weak points may correspond to a very low level of coverage. The test data will need to be hidden from the implementor and can probably be generated as a test script for the final acceptance. The details of the script can fairly easily be related to the requirement specification, but it will also need to be traced to the design when this has been used. This traceability forwards is necessary to reduce the costs of future evaluations, incurred as a result of changes in the system.

6.2 Traceability aspects of analysis

Analysis techniques commonly employed consist of control flow, data use and information flow analysers. Control flow can be of some use

during evaluation as it gives a measure of complexity of an implementation, but for development purposes it is probably better to rely on coding standards, with informal checks for compliance. A coding standard can capture other aspects of well structured software apart from control structure and the control flow analysis does not really give a very objective measure of quality. Data use analysis can be a useful technique in demonstrating compliance with object re-use requirements. The problem with the technique is that it is unselective. Standard data use analysis will generate many data anomalies, which are generally innocuous. What is required is selective use to demonstrate compliance with those quantities which are required to be unalterable by untrusted code, such as security levels. Information flow is another useful technique for demonstrating compliance, and in this case commercial analysers provide some means of automating the compliance check. This is because they provide input/output relations for procedures which can be automatically checked by information flow analysis. The input/output relations ("derives" annotations) are relatively easily traced back to higher level requirements.

In evaluation mode, analysis tools are probably best regarded as tools to assist the evaluator in understanding the implementation. Faults suggested by analysis will normally be verified by testing, and so there is no requirement to trace the analysis process itself.

6.3 Developments in techniques for testing and analysis

The principle problem in testing and analysis comes in automating the procedures and being selective in the properties tested or analysed. The topic of generating test data from a specification is an issue in safety engineering which has received some attention, but the application of the techniques to security engineering has not been investigated, although it is likely that the techniques are applicable. The security properties needed for analysis on the other hand, are well understood. What is required is the ability to demonstrate access to data only via certain procedures, lack of interference between procedures, the ability to hide data, the ability to make certain items unforgeable or unalterable and so on. These particular properties are based on particular applications of data use and information flow, but applied to certain variables. Work at RSRE has shown how these particular properties can be expressed as types: the type system is a development of that available in standard languages and the best way of implementing it would seem to be as an input to an analysis phase which guaranteed that the usage of identifiers in the implementation language also conformed to the usage specified by the secure type system. This would give a way of checking that the security structure required was present in an implementation.

7 Traceability and Conformance in the Development Environment

It should by now be apparent that the requirement for traceability are on the bounds of feasibility for current technology for development environments. For simple informal developments, the problems are maintaining cross references between requirements, rationale and design documents and the source text of the implementation. For formal

developments the problems are similar although complicated by the need to trace between representations needed by differing formal descriptions. In addition, a high assurance formal development will need to verify the conformance between representations using formal proof: thus there are additional interfaces between formal descriptions and proof tools. Traceability requires evidence that the proofs displayed are valid.

In theory all of these checks could be done manually but for a development of any size at all, manual checking consumes unreasonable amounts of effort, both in the maintenance of traceability during implementation and cross checking during evaluation. Tools can assist this process by implementing these checks mechanically: the development environment can support the tools by providing structured data on backing store (object management systems or persistent values) which can provide means for indicating dependencies.

Apart from support for cross references the development environment has a role to play in both development and constructional traceability. For the former one could naturally think in terms of audit trails and these do have a part to play in establishing the history of configuration items. However, in many cases the development traceability can be expressed in a similar fashion to constructional traceability and so use similar techniques. For example the requirement that modules should reach an acceptable testing level before being entered into the configuration management system can be expressed in a constructional way by requiring that the configuration is only made up from tested items. Techniques for establishing constructional traceability again involve the use of types, which can express constraints of this nature: for example, that systems are built from modules which may only be produced by compilers. Types are appearing more and more in specifications for development environments and so one may expect to see some form of typing available for industrial use quite shortly.

Development traceability also includes some form of accountability. It is important to know not only that a procedure has been followed, but also who has invoked it. This involves the usual mechanism for identification and authentication supplemented by audit and discretionary access to tools. This integrity policy will need to be trustworthy, but as the control is over actions rather than information flow it should be easier to implement to a given level of assurance than a security policy.

8 Conclusions and Further Work

There is no doubt that traceability of current secure systems is inadequate: it is one of the causes of costly implementations, even at low levels of assurance. It is a major cause of costly evaluation and the reason for what will no doubt prove to be an almost total inability to maintain trusted systems. Technology to support traceability is clearly deficient, but the problems are not entirely technological. The following recommendations are made for providing traceability within current technology:

Within requirement specifications, SORs should be indicated more positively than by the simple occurrence of the word "shall".

Each SOR should be associated with some conformance mechanism to provide the acceptance test and the means of providing traceability specified.

Each SOR should be categorized so that ones relevant to a particular point of view, security for example, may be extracted.

More attention should be directed at controls in the development environment ensuring development traceability.

SORs should be traced back to the operational concepts and assumptions which gave rise to them.

Following these recommendations will involve giving serious consideration to the structure of requirements. The discussion of this report has been centered on the issues of conformance which naturally arise during procurement and implementation. Although this is a major issue, it is certainly not the most important use of a requirement: other uses are concerned with effectiveness, feasibility, costing, performance and risk analysis generally. Points of view on systems cover physical aspects, communications and human computer interactions as well as software. Development of a structure to support these uses within these points of view needs to be matched to the roles of those participating in the procurement process. The description of a reasonable structure which could be validated against a typical procurement would be useful work and this gives the following recommendation for a study:

The procurement process for large software intensive systems should be studied to establish appropriate information structures.

Technologically, there is much research and development work needed on conformance. For example, the generation of test data from specifications, better methods of specification, refinement and theorem proving and so on. These are somewhat outside the scope of traceability as such, although tools to support these methods need to be constructed with the needs of traceability in mind. The specific technological problem associated with traceability is concerned with the integration of tools and the integrity of the development process. The support for this is provided by tool interface definitions and data repositories. Such technology already exists, but there is no work being done on assembling it into a trustworthy process suitable for the development of secure systems in an industrial context. This leads to our final recommendation

Work need to be undertaken leading to the demonstration of a trusted software development system.

References

CESG (1991). A formal development methodology for high confidence systems. CESG Computer Security Manual F.

Goguen J A and Meseguer, J (1984). Unwinding and inference control. Proc 1984 Symposium on Security and Privacy, Oakland, California, USA. IEEE Computer Society 1984.

Hoare, C A R (1985). Communicating sequential processes. Prentice Hall International series in Computer Science.

ISO (1986). Standard generalized markup language (SGML) for text and office systems. ISO 8879-1986. (Also available as British Standard BS 6868:1987).

ITSEC (1990). Information Technology Security Evaluation Criteria. (Harmonized criteria of France, Germany, the Netherlands and the UK, available from CLEF Certification Body, CESG, Fiddlers Green Lane, CHELTENHAM Glos. GL52 5AJ.)

Knuth, D E (1984). Literate programming. Computer Journal, 27, 2, pp97 - 111.

Lyons, T G L and Tedd, M D (1987). Technical overview of PCTE and CAIS. Ada User, 8(supplement): S73 - S78.

Milner, R (1980). A calculus of communicating systems. Lecture Notes in Computer Science 92, Springer Verlag, New York.

Morgan, C (1990). "Programming from Specifications", Prentice-Hall International Series in Computing Science.

RTP (1987). MALPAS Intermediate Language Manual. Rex, Thompson and Partners, Newhams, West Street, Farnham, Surrey.

REPORT DOCUMENTATION PAGE

DRIC Reference Number (if known) .....

Overall security classification of sheet .....UNCLASSIFIED.....

(As far as possible this sheet should contain only unclassified information. If it is necessary to enter classified information, the field concerned must be marked to indicate the classification eg (R), (C) or (S).

Originators Reference/Report No.	Month	Year
REPORT 90016	AUGUST	1991

Originators Name and Location
RSRE, St Andrews Road
Malvern, Worcs WR14 3PS

Monitoring Agency Name and Location

Title
TRACEABILITY AND CONFORMANCE IN SECURE SYSTEMS

Report Security Classification	Title Classification (U, R, C or S)
UNCLASSIFIED	U

Foreign Language Title (in the case of translations)

Conference Details

Agency Reference	Contract Number and Period
Project Number	Other References

Authors RANDELL, G P; SENNETT, C T

Pagination and Ref
28

Abstract

Traceability in a software intensive system is the ability to link statements of requirement with the implementation objects which satisfy them and the means used to demonstrate conformance. This report discusses the problems of maintaining traceability when developing large secure systems and the ways in which technology may be used to support it.

Abstract Classification (U,R,C or S)
U

Descriptors

Distribution Statement (Enter any limitations on the distribution of the document)
UNLIMITED