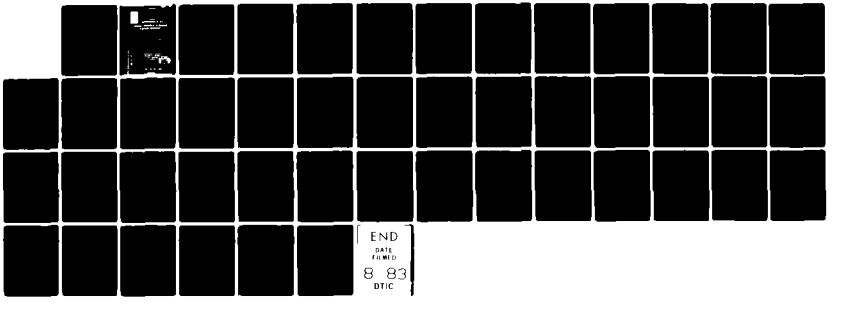
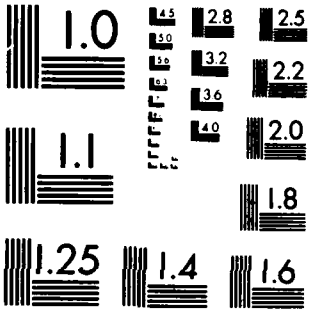
AD-A130 109 SOFTWARE CONFIGURATION MANAGEMENT IN AN INTEGRATED PROGRAMMING SUPPORT EN..(U) ROYAL SIGNALS AND RADAR ESTABLISHMENT MALVERN (ENGLAND) M STANLEY APR 83 1/1

UNCLASSIFIED RSRE-MEMO-3578 DRIC-BR-88136 F/G 5/1 NL



END
DATE FILMED
8 83
DTIC



ROYAL SIGNALS AND RADAR ESTABLISHMENT

Memorandum 3578

TITLE: SOFTWARE CONFIGURATION MANAGEMENT IN AN INTEGRATED PROGRAMMING SUPPORT ENVIRONMENT

AUTHOR: Margaret Stanley

DATE: April 1983

SUMMARY

(PSE)

Configuration management is defined as the management of change. This paper is about the use of an integrated Programming Support Environment in Software Configuration Management. It discusses the current methods of software configuration management, and how the control of change in evolving (PSE) with an adequately design database and an integrated set of software tools. The discussion is based on the facilities to be provided in the database and tools of the UK CHAPSE (CHILL Ada Programming Support Environment).

SOFTWARE CONFIGURATION MANAGEMENT IN AN INTEGRATED PROGRAMMING SUPPORT ENVIRONMENT.

CONTENTS

1.0 INTRODUCTION.

A Programming Support Environment (PSE) is intended to support the development and maintenance of applications software throughout the life cycle. Stoneman (ref. 1) sets out the requirements for an Ada PSE (APSE) with particular emphasis on software for embedded applications. It provides guidelines indicating the general requirements for such a support system, and a propos software configuration management, it says:

1. "the APSE must enable configuration control to be maintained: for any configuration of software, it is necessary to be able to determine the origin and purpose of each component of the configuration and to control the process of further development and maintenance of the configuration";

2. "PROJECT TEAM SUPPORT: An APSE shall support all functions required by a project team: these functions include project management control, documentation and recording, and long-term configuration and release control".

It is an essential characteristic of software that it is created and maintained in various forms and embodiments during its life cycle. For example initially the software will exist in the form of a requirements specification. At a later stage in the life cycle the software will include design documents and source code. When the software development is complete the software will exist as a set of compiled units and as an executable unit, plus a set of acceptance tests and test results.

Configuration management has been defined by Dunn and Ullman (ref. 3) as the management of change. During the life cycle of a software product the software undergoes perpetual change. That change needs to be controlled if the product is to perform as required.

Software configuration management techniques have been developed piecemeal in response to a need rather that as an integrated set of procedures. This gives rise to a number of problems including:

1. difficulty with software regeneration, which is often risky or even impossible because of insufficient data on how the original software was put together;

2. difficulty in determining the product status at any time;

3. difficulty in tracing the effects of making changes to the system;

4. inadequate or non-existent derivation histories;

5. confusion of the original structure of the software when evolution of a software system leads to unplanned variants.

2.0 WHAT IS A SOFTWARE CONFIGURATION?

Any part of the software may be embodied in documents on paper as well as in computer storage as a file or set of files. Each of the forms of the software (requirements specification to executable code and all the intermediate forms) and the embodiments (for example source code listings and files in computer store containing the source code) are aspects of the same thing and proper records must be kept showing how they relate, how they may be transformed to create other forms and what changes have been made to them, if the software development and evolution is to be controlled effectively. The various forms and embodiments of the software that exist at a given time are called the software configuration at that time. Thus the software configuration varies with time as specification, design and implementation progress. The different forms and embodiments of the different parts of the software are called software configuration items (SCIs). The current software configuration is the current and controlled definition of the software and consists of all SCIs that are sufficiently stable to be under configuration management.

The components of a software configuration will include:

1. requirements documents;

2. design documents;

3. program source text;

4. compiled code, linked compiled code and executable code;

5. user documents and instructions for starting and recovery after failure;

6. compilation, link and load commands for each version of the executable programs, including versions of the compiler etc. used;

7. changes requested (whether approved or not), with history of authorisations and action taken;

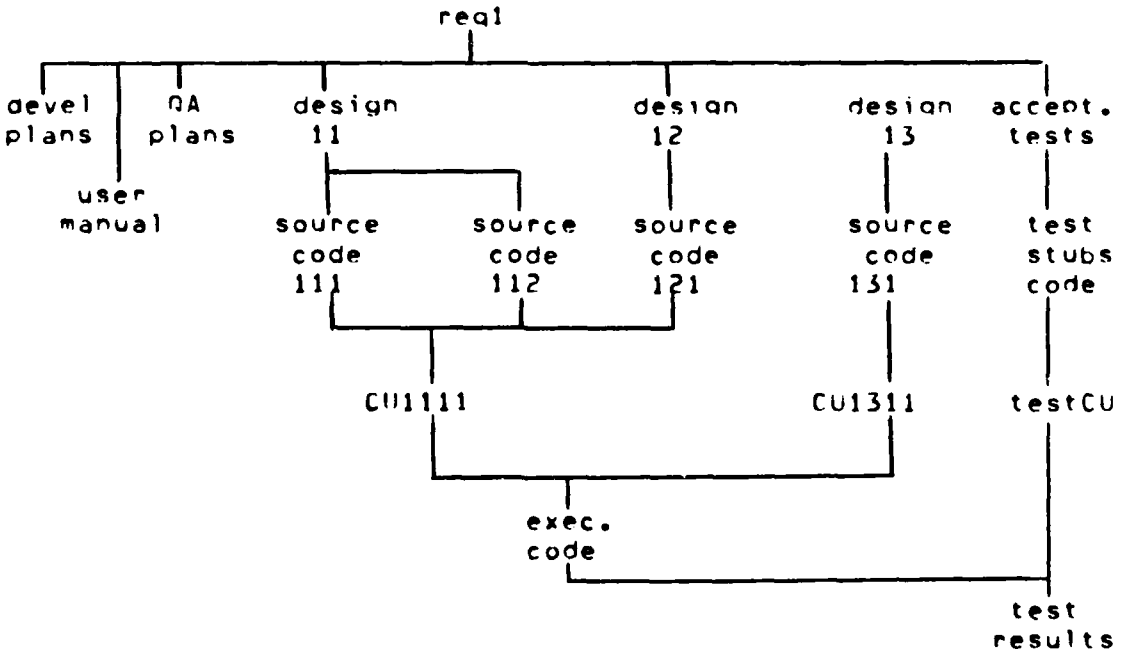
8. running environment (hardware and software) required for each executable program;

9. sets of test data and expected results (all versions);

10. record of actual test results (all versions).

2.1 Software Configuration Item (SCI) Network.

Software configuration items (except the overall system requirement specification) usually depend on some item or items already in the software configuration. For example a design document is a design intended to satisfy a particular requirement, and a source code file is intended as the implementation of some design or an executable program has been created by linking several items of compiled code, which in turn are derived from source code items. Thus the configuration items are connected in a directed network, which indicates dependencies between the different items. Frequently the network is a tree.



CU=Compilation Unit

Note the difficulty in nomenclature shown by CU1111 which is dependent not only on source code 111 but also on source code 112 and on source code 121.

2.2 Baselines.

A baseline is an approved list of SCIs. The development plan for a software product will define a succession of planned baselines coincident with the reference points or milestones in the life cycle. As the development proceeds the definitions of the baselines are refined until the relevant milestone is reached, when the baseline is said to be established.

Because several different implementations of a product need to coexist, there may be several versions of each baseline. For example, during product maintenance there will be established baselines corresponding to the current realisation of the product in addition to baselines for the revised realisation that is under development. The SCIs in these baselines will probably overlap to a considerable extent, because parts of the system will not need to be changed.

A single software configuration network can cover all the different versions of the product. There will be a set of baselines for each version of the product.

The current software configuration is usually defined by reference to the current baselines (i.e. the most recently established baselines) plus a list of increments to those baselines. Alternatively, it may sometimes be useful to define the current software configuration by reference to the next planned baseline plus a list of deviations from the planned baseline, showing the particulars in which the software configuration at that time deviates from the planned baseline (i.e. those SCIs in the planned baseline that have not yet come under configuration management, and those SCIs that have come under configuration management that were not included in the planned baseline although they are a part of the configuration for that version of the product).

Both baselines and the software configuration are controlled by software configuration management.

The following baselines are commonly used (refs 2 and 5):

1. functional baseline, established prior to software development,
   (may include program development plan; quality assurance plan; configuration management plan and the system requirement specification);

2. allocated baseline, established when the functions to be performed have been allocated to specific hardware and software,
   (may include the same configuration items as the functional baseline, plus the software requirements specification and acceptance test plans);

3. design baseline, established when the top level design of the software is complete,
(may include the same items as the preceding baseline, plus documents giving the functional breakdown of the requirement; the structure of the software; descriptions of the data and other high level design documents; the integration test plan and test procedures);

4. detailed design baseline, established when the design of the software is complete,
(may include the same items as the preceding baseline plus detailed design descriptions of each software module and system user manuals);

5. product baseline, established when the software is ready for delivery to the customer,
(may include the same items as in the preceding baseline plus source code; compiled source code; compiler and loader reports; maps showing the allocation of the executable code to computer store and to firmware; executable program and test reports, but excluding the integration test plan and the acceptance test plan);

6. operational baseline, established on delivery of the software product to the customer,
(may include the same items as in the preceding baseline except for the program development plan and the quality assurance plan).

3.0 WHAT IS SOFTWARE CONFIGURATION MANAGEMENT?

Software configuration management (SCM) is the discipline of systematically controlling changes to the configuration to maintain the integrity and traceability of the configuration throughout the software system life cycle (Ref. 2). It is a part of software product management rather than software project management, in that it is concerned with controlling the product rather than the organisation that develops the product.

A configuration management plan identifies all interested parties and all documents and computer files that are to be controlled, indicates when each item will come under control, who will control the item, and how changes are to be made and controlled. A software configuration item is said to be registered when it has come under the control of software configuration management.

In some organisations (ref. 12) there are two levels of control applied to SCIs: registration and bonding. Bonded SCIs are subject to stricter and more formal control of changes than is applied to SCIs that have been registered but not bonded. The control of bonded SCIs involves approval of bodies external to

the product development organisation. Changes to SCIs that are registered but not bonded must still be properly authorised, but the body required to give the authorisation may be internal to the product development organisation.

Configuration management has four complementary functions: identification; control; auditing; and status accounting.

Configuration Identification involves:

1. labelling each configuration item and showing how the various items are combined to produce the different versions of the executable programs;

2. defining and updating the baselines and the SCI network;

3. maintaining the dependencies in the SCI network;

4. listing the current baselines;

5. listing those SCIs belonging to each baseline, and whether they are bonded in that baseline;

6. listing when each registered SCI came under configuration control and when each bonded SCI was bonded;

7. listing the current software configuration, indicating deviations from the baselines (i.e. the SCIs that have yet to be registered in the baseline; the SCIs that should have been bonded , that have been registered but not yet bonded; and the SCIs that have been registered or bonded although not in the baseline).

Configuration control involves:

1. approving and monitoring the registration of objects as SCIs;

2. registering approved SCIs (i.e. creating and maintaining the library of SCIs);

3. approving and monitoring the bonding of registered SCIs;

4. bonding registered SCIs;

5. approving, monitoring and controlling changes to registered or bonded SCIs, (i.e. making the implications of a proposed change visible prior to change approval and ensuring that all actions required when changing the software configuration have been done,

such as approval by the appropriate body, completion of appropriate tests and related changes to all dependent configuration items.).

Configuration Auditing involves:

1. sanctioning a new baseline by verification and validation;

2. listing the ways in which the new baseline differs from the stated needs as given in the requ ements specification and design documents.

(verification is defined as checking that the new baseline is a correct interpretation of the preceding base e and validation is defined as checking that the pr as defined in the new baseline meets the customers requ ents (ref. 2)).

Configuration status accounting involves recording and reporting all significant events in the product life cycle related to the software configuration, including:

1. recording the establishment and attainment of each baseline;

2. recording any changes to the configuration, with reasons for the changes;

3. recording, controlling and actioning all defect reports or change requests, (changes to registered or bonded configuration items are usually controlled by requiring the submission and approval of defect reports or change requests, for every change however trivial).

4.0 HOW IS CONFIGURATION MANAGEMENT TRADITIONALLY ACHIEVED?

Configuration management today is a largely manual process, involving collecting each configuration item into a central library, indexing the library and managing the procedures for system development and change.

HOW IS CONFIGURATION MANAGEMENT TRADITIONALLY ACHIEVED?

4.1 Development Of A Configuration Identification System.

The development of a software configuration identification system, with labelling conventions reflecting the dependencies of the different items, and matching the software design, is usually a joint responsibility of software configuration management, software quality assurance and product management. The network of dependencies will develop as the design develops.

4.2 Definition Of The Baselines.

Baselines and the associated milestones are defined by project management supported by quality assurance and configuration management when the software development plan is produced. The detailed definition of the content of a baseline will develop in step with the software design.

4.3 Registration.

Programmers and software designers work on objects outside the purview of software configuration management. When the programmer or designer is satisfied that the item is stable and meets its objectives it will be submitted for registration as a software configuration item. The first step is the completion of a design review to check that the item meets its objectives. If the item meets its objectives it can be registered, together with associated SCIs (for example a source code item will probably be registered at the same time as the compilation unit derived from it, the executable code in which it was tested and the test results).

4.4 bonding.

A registered SCI is bonded when it is t ought to be unlikely to need further change unless the change is required because of change to an associated item or change to the user requirement.

4.5 Project Librarian.

A project librarian is responsible for recording, storing, indexing, controlling access to and issuing reference copies of all registered SCIs. He is also responsible for issuing change information to affected personnel (e.g. programmers responsible for SCIs dependent on a modified SCI) whenever an SCI is modified. He produces reports, for use by project management and software quality assurance, on the current planned baselines, the

current software configuration, what has been changed, and what authorised changes have not yet been implemented. The librarian is also responsible for archiving, and making copies of all SCIs to ensure the integrity of the configuration.

4.6 Change Mechanism.

During development and maintenance of software it is inevitable that changes will need to be made to the software configuration, either because of changes in the requirement or the target hardware, or because defects are discovered in the design or implementation. A mechanism is therefore required for controlling changes and ensuring that none are overlooked or acted upon without proper authorisation. A formal method for proposing changes, and reporting defects is usual, involving a set of forms, and required authorisations. The process entails:

1. proposal of a change (possibly by the user or the implementer);

2. incorporation of the change proposal into a document (whether or not the proposed change is accepted);

3. review of the proposed change by an appropriate authority;

4. alteration to the software configuration or planned baseline to incorporate the change if it is approved.

For example, (refs 2 and 5), the following system may be used:

1. when it is believed that the software may be deficient, either because of an incident when running a program or as a result of a design review or other cause, a Software Defect Report is completed identifying the suspected defect and giving reasons;

2. Software Defect Reports are reviewed and marked as 'no action required', 'deficiency needs remedy' or 'change in requirement needed', (each marking needs endorsement by an appropriate authority);

3. if 'deficiency needs remedy' then a Software Change Notice is completed, that may authorise a change in documentation, in design or in source code and dependent SCIs. The Software Change Notice will list the affected SCIs;

4. if 'change in requirement needed' then a Software Change Request is completed. Such change requests may also be generated without a preceding defect report. Change requests must be submitted, with supporting information, for formal authorisation by a body such as the Configuration Control Board;

5. if changes in requirement are authorised then a Software Change Notice may be completed, or another form called an Engineering Change Proposal may be required;

6. all defect reports and software change requests are archived, for configuration status accounting. It is the responsibility of the project librarian to progress any authorised changes and to notify affected personnel.

It is essential that versions of an SCI that exist prior to an authorised change are retained, partly to support any subsequent audit, and partly so that, for example, the requirement specification for an existing product remains available despite the evolution of the specification into a later baseline.

5.0 AIDS TO CONFIGURATION MANAGEMENT IN THE UK MCHAPSE.

An Ada Programming Support Environment (APSE), as defined in the Stoneman Requirements (Ref. 1), is intended to include facilities to assist in configuration management. An integrated programming support environment (PSE) includes a computer controlled database and a set of software tools (programs) for manipulating objects in the computer controlled database. The database not only acts as a repository for all information associated with a project throughout the project life cycle, but also relates the different objects to one another. In fact, the main difference between a PSE and currently available collections of software tools, provided for building and controlling the production of software systems, is the existence of an underlying computer controlled database relating different objects associated with the development and providing facilities for controlling relationships and dependencies, together with an integrated set of software tools for operating on objects in the database and creating new database objects.

5.1 UK MCHAPSE.

The MCHAPSE (Minimal CHILL Ada Programming Support Environment) is being developed in the UK to satisfy the Stoneman requirements. Refs 10 and 11 describe the MCHAPSE as it is currently envisaged.

The MCHAPSE will include a PSE database although its software toolset is the minimum set of tools needed to support the development of Ada and CHILL applications for embedded software. It is the intention that further tools be added to transform the MCHAPSE into a full PSE in due course. It is also intended that these tools shall all operate on the PSE database

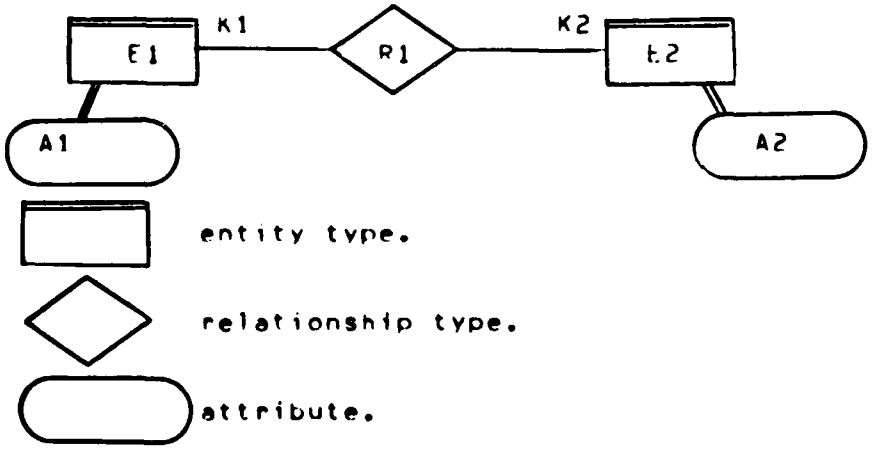
and communicate with the users through a common set of interfaces. The tools to be developed will include tools for configuration management. The facilities required for software configuration management are independent of the language in which the software is implemented.

5.1.1 Database Terminology.

Objects in the MCHAPSE database (see ref. 10) are known as entities and relationships. Entities are grouped into disjoint sets of objects having common characteristics. The disjoint sets are called entity types. A connection between two entities is called a relationship. Relationships also belong to disjoint sets, called relationship types, that connect two entity types. Both entities and relationships have attributes that represent the properties of the objects. An entity type is said to have a role in a relationship type. A relationship type may be identified by the name of one of the two roles.

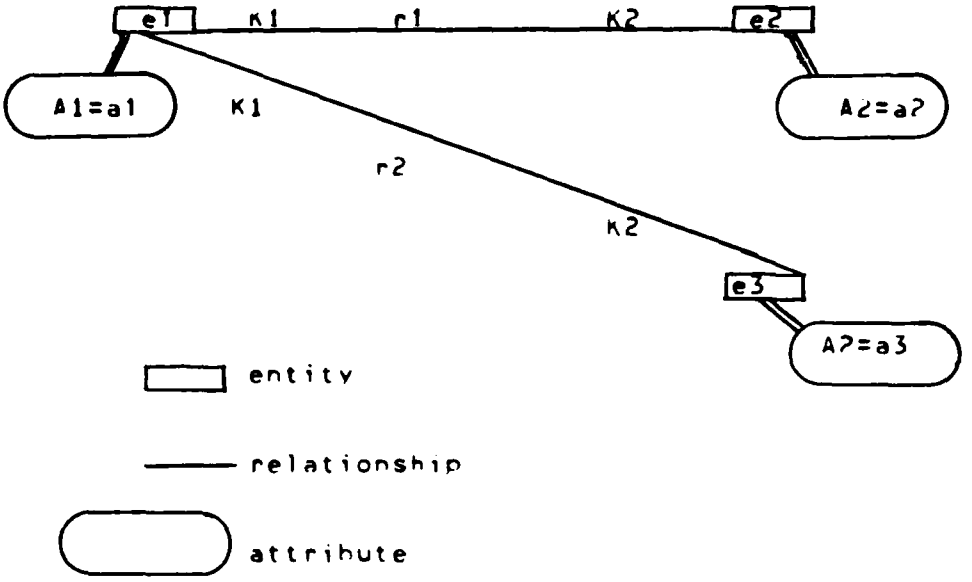
A database schema is used to define the structure of a particular database, showing the types of objects that may exist in the database, the ways in which the objects may be related, the attributes they may have and the properties of the attributes, relationship types and entity types.

Diagram of a database schema, showing two entity types connected by a relationship type.



Entities of type E1 may be connected to entities of type E2 by relationships of type R1;
The role of entities of type E1 in relationship type R1 is K1;
The role of entities of type E2 in relationship type R1 is K2;
Entities of type E1 have attribute A1;
Entities of type E2 have attribute A2.

Instances of database objects connected as in the above schema:



e1 is an entity of type E1, with value a1 for attribute A1;
e2 is an entity of type E2, with value a2 for attribute A2;
e3 is an entity of type E2, with value a3 for attribute A2;
e1 is connected to both e2 and e3 by relationships, r1 and r2, of type R1.

The tools that provide access to the database will ensure that the structure and properties of the database, as set out in the schema, are maintained. They also check that the access protection specified for objects in the database is not violated and they provide facilities to users for maintaining the integrity and consistency of the database. In this paper these priveleged tools will be called the database tools.

The core schema is that part of the database schema specified for the MCHAPSE (ref. 11). It is the mimimum schema needed to provide a database to support the MCHAPSE tools, in particular the compilation and linking system. Any schema for a database to support a full PSE based on the MCHAPSE will need to include the core schema as a subset. Extension to the core schema will be needed to support software configuration management.

The PSE database will include objects that are not under SCM as well as objects that are under SCM. The part of the database schema that defines objects under SCM will be called the SCM schema. The part of the database holding objects under SCM will be called the software configuration database.

5.1.2 Information Associated With SCIs.

The attributes of an entity contain the information closely associated with the entity such as creation date, name, body (of a text file for example) or the history of the entity. Information less closely related to the entity (such as the name of the person responsible for the entity) may be held in the attributes of related entities or of relationships.

5.1.2.1 Mandatory Attributes.

A mandatory attribute is an attribute that every member of a specified entity type or relationship type must have. There is a set of predefined attributes that are mandatory for all entity types in the CHAPSE database, and a set of predefined mandatory attributes for all relationship types in the CHAPSE database. The list of mandatory attributes of any entity type or relationship type can be extended to include mandatory attributes which are not in the predefined list.

Some of the predefined mandatory entity attributes hold information needed for software configuration management, for example:

1. name of the object;
2. version (successor and variant) of the object;
3. creation date;
4. name of person (author) who created the object;
5. derivation of the object;
6. access controls.

Some additional information must be associated with every SCI. For example, every SCI must have:

1. SCI identification of the object;
2. reason for creation of the current version of the object.

In addition, specific kinds of SCI may need to to have other information associated with them. For example, a change request must have an attribute to hold the status (accepted, rejected, pending) of the proposed change.

The database schema specifies the attributes of an entity or of a relationship that are mandatory and the database tools ensure that entities and relationships are not formed with their mandatory attributes unset.

A tool for registering an object as an SCI could prompt for values for the mandatory attributes and could check that the values were acceptable.

5.1.2.2 Mandatory Roles.

Some information that must be associated with an SCI may be better catered for by forcing the SCI to have a given type of relationship to another entity type than by the imposition of mandatory attributes. For example, every SCI must be the responsibility of one person. The details of the person, such as qualifications, department and telephone number may be held as attributes of the person entity. The SCI therefore needs a relationship to one person entity, indicating responsibility.

The database schema indicates the relationships an entity must have by specifying the mandatory roles. The property of having mandatory roles in a relationship will be useful for configuration management. They can ensure that some of the dependencies in the SCI network are enforced in the database. For example, an SCI must have a relationship to the person responsible for the SCI.

In addition, particular kinds of SCI will need other mandatory roles. For example, any Engineering Change Proposal must have a relationship to at least one Software Change Request, to at least one responsible person, and to at least one SCI.

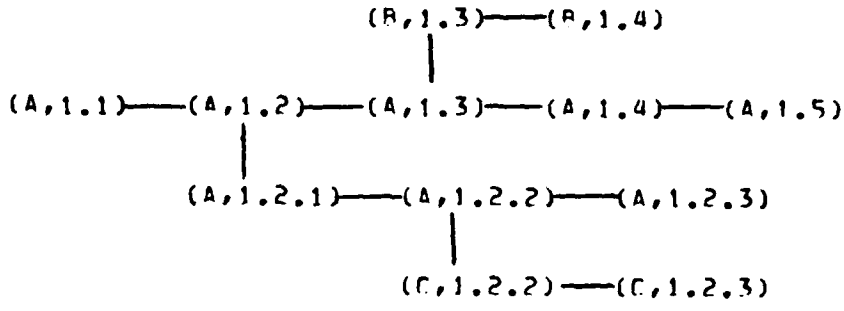
5.1.3 Versions.

Since software is constantly being changed, any SCI may exist in a number of different versions, all of which need to be retained. In addition to the different versions related in a predecessor/successor fashion, there will be different versions of the same thing arising, for example, because of variations in the implementation of a design, or minor differences between SCIs to cater for different target hardware configurations.

To assist in dealing with this problem the CHAPSE database will provide two forms of versions, called successors and variants, for objects that have the same name in the database. The variants represent different interpretations of an object, such as different implementations of the body of an Ada package,

both satisfying the same Ada package specification. The successor versions of an object describe an ordering of the object representations. For example, the evolution of a piece of text may be represented as a set of successor versions of a database object. The successor representations provided for the CHAPSE form a hierarchy, as shown below:

Diagram showing a set of successors and variants of a single database object with three variants, A,B and C



Successors have numeric identifiers, with hierarchic structure as illustrated for variant A.

Variants: B is a revision of (A,1.3); C is a revision of (A,1.2.2)

The entity has several "current" versions, viz. (B,1.4); (A,1.5); (A,1.2.3); (C,1.2.3).

It is recommended that only a single chain of successors be used for registered SCIs to avoid confusion as to which is the "current" version of each variant.

Adequate precautions will have to be taken to ensure that the correct version is used in any software configuration. It will probably be necessary to name successor and variant explicitly in any reference, rather than using the default successor and variant. The database tools allow for a preferred variant and a preferred successor to be used as defaults if no version identification is given.

5.1.4 Modification Of Registered SCIs.

Ideally, once an SCI has been registered it should not be modified. However, if a modification is authorised, it should result in the creation of a new version of the SCI, rather than a change in the attributes or relationships associated with the existing version of the SCI. Even though it will be possible to

change the attributes or relationships of an entity in the database without creating a new (version of the) entity this is not recommended.

The CHAPSE database provides a facility whereby an object or an attribute of an object can be bound (for details see ref. 11 and ref. 16). A bound object or attribute cannot be deleted or modified. Changes to the bound object can only be effected by creating a new object or a new version of the object. Similarly, relationships can be bound so that it is impossible to destroy the relationship between two entities as long as both entities exist. The property of binding can be used to prevent modification to or deletion of registered SCIs. Binding is imposed on an entity or its attributes or its relationships by the creation of a relationship, defined in the database schema to have the binding property, between the entity and some other entity.

In order to impose the binding required on SCIs it will be necessary to represent an SCI by a network of entities and relationships, rather than by a single entity in the database.

5.1.5 Copying SCIs And Creating New Versions Of SCIs.

When SCIs are to be issued for amendment, or when new versions are to be created, all the required attributes and relationships must be carried forward into the new entity. Since SCIs may be represented in the database by a network of entities and relationships, rather than by a single entity, the preservation of the information associated with the SCI may be complicated.

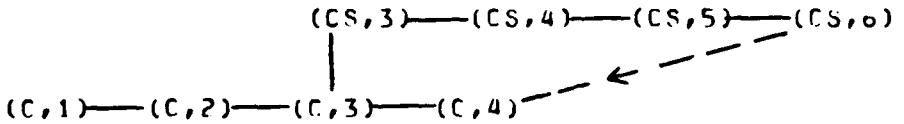
The CHAPSE provides facilities for creating a new version of an entity (called revision) or for creating a separate copy of the entity, with a new name in the database (called copying). Certain attributes and relationships of the entity are automatically inherited without change. Other attributes are either given the value "unset" or their new values are specified by the user when the entity is revised or copied. The database schema specifies the attributes and relationship types that are inherited by members of an entity type. A tool will be needed to issue copies of SCIs for amendment, with the necessary attributes and relationships preserved but with some of the binding relationships deleted, and with changed access protection.

The facilities in the CHAPSE database only allow a new version of an existing entity to be added to the database by revision from an existing version of the entity. Since an amended SCI entity should be added to the software configuration database as a successor or variant of the original SCI entity, the tool that issues an SCI entity for modification, and the

re-registration tool, must revise the SCI to be issued rather than copying to a new name. The issued entity may be a new variant of the original SCI.

When a registered SCI is issued so that an authorised amendment can be made, the copy will probably go through several versions before the amended SCI is ready for re-registration. It will be advisable to divorce the SCI versioning from the versioning of the issued copy to avoid unnecessary confusion.

Diagram showing a set of successors and variants of a an SCI.



The SCI has two variants, C for the controlled variant and CS for the issued variant which is not under configuration management.

Successors have numeric identifiers.

Variants: CS is a revision of (C,3), that goes through several successors before it is ready for re-registration. (C,4) is a revision of (CS,6), created when (CS,6) is re-registered as a successor of (C,3).

When (CS,6) has been re-registered as (C,4), the successor chain for the CS variant can be deleted, since it is not under configuration management.

5.1.6 Uniqueness Of Entities.

Entities in the database are usually distinguished by naming the entity type, entity name, successor and variant. However, since many independent users may share an entity type, each with their own entity naming conventions, it is possible that more than one entity in a single entity type will share the same name, successor and variant. Ambiguity caused by sharing a name is normally avoided because entities are found by searching (see ref. 16) via other entities and relationships, from a known entity or set of entities. If different users have chosen the same name for different entities in the same entity type, the entities and relationships through which they can be reached will usually be different. Entities with the same name can also be distinguished from each other by the values of other attributes.

Since finding a set of entities when only one entity was expected could lead to confusion, there is a facility to specify in the database schema that an entity performing a specified role in a given relationship has the 'unique naming property'. This means that when searching via that relationship type, from a unique entity, the values of entity type, name, successor and variant will be sufficient to identify at most one entity.

This property may be important when setting up a database schema for configuration management, to prevent confusion caused by accessing a set of entities when only a single entity was required.

5.1.7 History.

The history of every SCI (i.e. all information relevant to the creation of the object) is needed for configuration status accounting. Every database object must have a mandatory history attribute that "shall contain sufficient information to provide a basis for comprehesive configuration control. Any necessary constraints shall be imposed on database operations so that the validity and consistency of history attributes is ensured." (Stoneman). The history attribute is called derivation in the CHAPSE.

In addition to derivation it is recommended that all SCIs have a mandatory attribute to hold the reason for the creation of that version of the SCI, with references to relevant change requests if appropriate. Registration tools could prompt for the reason when an SCI is registered.

It is intended that all PSE tools shall record derivations of any object that they create or amend.

5.1.8 Reconstruction Of SCIs.

It is necessary for configuration management that mechanisms be provided whereby all database objects needed to recreate a specified object are maintained in the database as long as the specified object itself remains in the database. This means that it will not be possible to delete from the database any SCI necessary to the recreation of another SCI. It is expected that, with the exception of text files modified by the Editor, it will always be possible to recreate an entity from the derivation information. Derivations will probably include reference to the version of the tools used to create or amend the entity (e.g. the compiler and linker that were used in the creation of compiled objects and linked compilations from source files), to command files and to input data files needed for the recreation

of the entity. In addition to derivation certain objects in the database will have a steering file attribute that may hold the commands used when the object was created.

Given a program or partial link it will be possible, using derivations and steering file attributes, to discover the names and versions of any intermediate components, and source entities, and names and versions of the tools used to create or to modify them.

The binding characteristic may be used to prevent deletion of or amendment of objects needed for reconstruction. The transmission of binding through a chain of objects each of which is necessary to the creation of the next object in the chain is important in this context.

For example, the existence of a binding relationship between a linked program and its constituent parts can prevent change to or deletion of the compiled parts. The relationship between each compiled part and the source code from which it derived can in turn bind the source code. The relationship between the source code and the design specification of which it is an implementation can bind the design specification.

When source code entities are compiled in the MCHAPSE, the source code entities and the body attribute of the entities (that holds the actual source text) are bound until the compiled entity is deleted.

When an SCI is registered the registration tool will need to create the network of binding relationships needed to bind the SCI and the objects on which it depends, such as those needed for its reconstruction.

5.1.9 Dependencies Between Ada Compilation Units.

The Ada separate compilation system (see Ref. 15) allows several separately compiled units to be combined in an executable program. Package and task specifications may be compiled separately from the corresponding bodies. With the separate compilation facility, several separately compiled entities may be interdependent. It will not be possible to delete a compiled unit from the database while other units exist in the database that depend on the unit to be deleted.

The Ada language definition (ref. 17) requires that if any compiled unit is changed, all the units dependent on the compiled unit must be marked as invalid so that they may be recompiled. Invalid compiled units must be recompiled before they can be linked to the new version of the compilation unit. The user of the MCHAPSE will be offered assistance such as a listing of units

that have been invalidated by a compilation, and help with recompiling.

The following dependencies are relevant:

1. a specification or body is dependent on a library unit specification if the source of the compilation unit quotes a 'with' clause (in this context a library unit is another unit that exists in the database;

2. the body of a separately compiled in-line subprogram is dependent on the body of the compilation unit in which the subprogram is defined;

3. a body depends on its specification;

4. subunit bodies depend on their parent bodies;

5. a body or specification that instantiates a separately compiled generic body is dependent on the generic body, which may be supplied in several separately compiled parts;

6. a partial link is dependent on all its constituent parts;

7. a program is dependent on all its constituent parts.

This requirement goes a long way to ensuring the integrity of executable programs, but it is not sufficient for total control of dependencies, since dependencies such as a source code file being dependent on documentation must also be controlled.

5.1.10 Sharing Compiled Code Between Products.

It may sometimes be desirable to share compiled code between different products using the same database, or between different programs within a project. This will be the subject of another paper.

5.1.11 Acquisition Of Entities From Another Database.

If software is to be reused, libraries are to be shared or a software project is to be undertaken in physically different locations, it will be necessary to acquire software from external sources. The acquired software will need to be incorporated into the database and the necessary entities and relationships will need to be set up. This will be the subject of another paper.

5.2 PSE Tools.

A PSE includes an extensible set of software tools (programs) for operating on objects in the database.

5.2.1 MCHAPSE Tools.

The MCHAPSE tools will provide the minimum set of machine independent facilities that enable a programmer to develop Ada and CHILL software conveniently. All access to objects in the database will be made via the privileged MCHAPSE tools that enforce the database characteristics described above, such as binding, mandatory attributes etc.. They also provide the facilities such as checks to prevent violation of access controls, and transaction management to assist in maintaining the integrity of the database (see below). These tools can be used by any tool in the PSE.

The MCHAPSE toolset will include the tools necessary for creating entities such as text objects, for compiling and linking programs and parts of programs written in Ada or CHILL, with the characteristics described above for keeping control of interdependencies, for running and testing programs and for developing new tools.

5.2.2 Facilities Provided By The Database Tools.

5.2.2.1 Access Controls.

Access to entities, relationships and attributes will be controlled, in the CHAPSE, to allow only authorized access. The access allowed to database objects is indicated by the access attributes of each object and can be set individually for each entity, for each relationship and for every attribute. Access is restricted according to the identity of the user requesting access, and in certain cases according to the tool being used to obtain access. The details of the access controls are discussed in Ref 16.

The complex system of access controls will aid configuration management by enforcing the use of appropriate tools to access registered SCIs. It may also be useful to provide the project librarian with special permits giving the librarian greater access to registered SCIs than accorded to other users.

5.2.2.2 Transaction Management.

Transaction management is a facility whereby a user can specify that a certain sequence of operations on the database shall be performed either completely or not at all. The user defines the sequence of operations to be so treated as a transaction. In effect he is then asserting that if the database was consistent at the start of the transaction it will still be consistent on completion, even if it is inconsistent at some stage during the transaction. The database tools will ensure that if a transaction fails to complete, the database is returned to the state it was in before the transaction had commenced.

It will be essential to use transaction management when updating the software configuration database, to prevent partial entry into configuration control of configuration items that consist of more than one entity or relationship, and to prevent creation of a relationship that binds an attribute until the attribute has had the required value assigned to it.

5.2.2.3 Archiving.

Facilities will be provided in the MCHAPSE for long term archiving of database entities and their relationships. This is a vital function for configuration status accounting, which will be needed by the project librarian.

5.2.3 Other PSE Tools For Configuration Management.

A data dictionary, which is a database holding descriptions of the entity types and relationship types in the main database will be available for the development of tools for the production of software configuration management reports. It can include useful aliasing information and cross referencing information as well as descriptions of the meaning of the different kinds of database objects, attributes and structures. It can also include cross reference information showing, for example, the tools that use each part of the database and the people that require the different reports.

A query language for the PSE database might use the data dictionary and would be very useful for the production of reports.

6.0 DESIGNING THE SCM SCHEMA.

It is important that a database schema be devised that takes full advantage of the available database properties. It will need to be an extension of the core database schema proposed for the MCHAPSE which includes the schema for the compilation and linking system. The core database schema clearly provides an excellent starting point for an SCM schema.

6.1 Choice Of Entity Types And Relationship Types.

The decision as to whether different kinds of object having common characteristics should be represented by a single entity type, or by a number of different entity types, is a difficult one. For example, there are many different kinds of object whose most important attribute is a text file. These objects will have various different functions (e.g. Ada source text, command files, different kinds of documentation). Potentially this could be reflected by having several different entity types for the different kinds of text object. The benefits would include:

1. tools that only accept input from appropriate entity types;

2. the ability of database tools to prevent illegal relationships between different text entity types;

3. the facility for a user to determine the classification of a text entity without examining its contents;

4. the possibility of defining different attributes for different text entity types.

The disadvantages would include:

1. no direct facility for copying between entity types in the database, (it would, however, be possible to write a tool to copy the attributes of an entity from an entity of one entity type to an entity of another entity type. It would be necessary to check that the appropriate relationships and bindings were established for the new entity.);

2. the need to modify the database schema to accomodate any new text entity type, with all the concomitant problems;

3. the potential proliferation of text entity types adding to the complexity of the database schema;

4. the use of a particular text entity for more than one purpose (e.g. a design document may contain some Ada source text or extracts from documents of one kind might be used within

documents of another kind).

In view of the disadvantages listed above, the core database schema proposed for the MCHAPSE (Ref. 11) caters for only one entity type for text objects, TEXT, with a single attribute holding the body of the text. It is not, however, necessary that the SCM schema restrict itself to a single entity type for text objects, provided that objects such as source text that are used by MCHAPSE tools are retained in the entity type defined for them in the core database schema.

The differences in the different kinds of TEXT entity all belonging to the same entity type can be identified by the role the entity plays in various relationships with other entities in the database. For example, if TEXT entity S is a source file that realises a design documented in TEXT entity D, then a relationship may exist between S and D in which S plays the role "realises" and D plays the role "design".

6.2 Choice Of Attributes And Relationship Types For A Given Entity Type.

Information can be associated with an entity either directly by being contained in an attribute of the entity, or indirectly by being contained in an attribute either of a relationship of the entity or of another entity connected to the given entity by a relationship. The decision as to how the information will be associated with an entity will depend to some extent on the strength of the association, and the extent to which the information has meaning independent of the entity

6.3 Dependencies And Design Of The Database Schema.

Entities in the software development system will depend on each other in a variety of different ways. For example, there are the dependencies (already catered for in the MCHAPSE database schema for the domain) of compilation units on each other and on source code entities. There are also dependencies of designs on requirements, of test results on test data and on the entity under test, of software entities on personnel entities and the dependencies implicit in the SCI network. The SCM schema needs to be designed to enforce certain dependencies by using mandatory relationship roles, to prevent the accidental deletion or change

of entities on which other items depend by the use of binding, and to provide for the tracing of dependencies by definition of appropriate relationship types.

6.4 Content Of The SCM Schema.

This example is intended to illustrated the features of a software configuration management database. It is not intended to be complete.

The database will include:

1. a skeleton network of SCN entities representing the planned SCI network, with relationships representing the connections in the network;

2. baseline entities, with relationships to every SCN in the baseline, perhaps using successors and variants to represent the different versions of successive baselines;

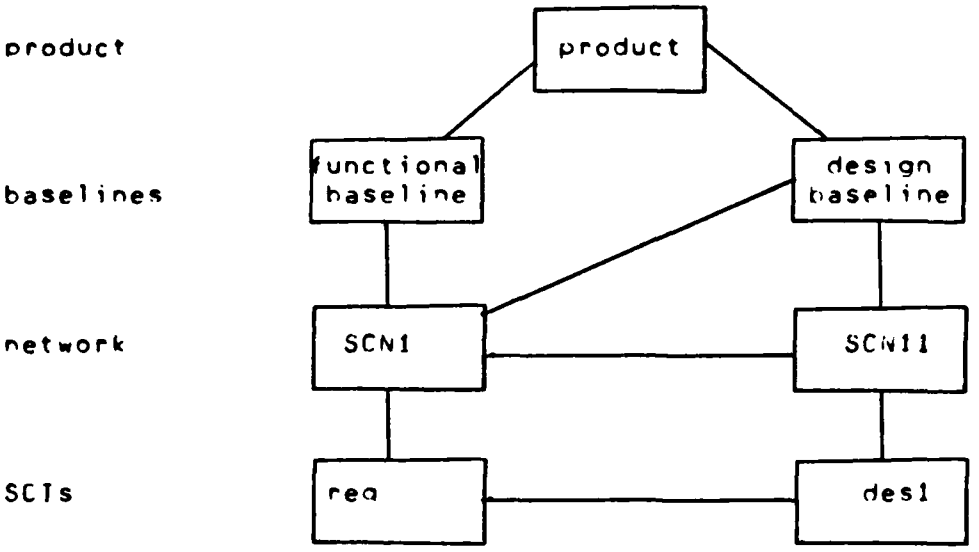
3. a product entity related to every baseline entity associated with the product;

4. entities holding the body of each SCI object such as specifications, plans, source code etc.., with relationships giving the connections between them;

5. relationships connecting the network of SCN entities to the entities holding the body of each registered SCI, (these relationships will be created on registration of the SCI, and will bind the body of the SCI).

SCN entities are connected in the skeleton SCI network described above. The actual objects to be registered as part of the software configuration are called SCIs. On registration of an SCI a binding relationship will be created to precisely one SCN entity.

The database facilities provided in the MCHAPSE do not permit the searching for all entities in a given configuration unless they have some other attribute in common. It will therefore be useful to provide a product entity connected to all baseline entities associated with the product.



6.4.1 Objects Under SCM.

Many kinds of object will need to be catered for in any software configuration database. Some will be catered for by entities and some by relationships. It may be that several different kinds of object can most usefully be catered for by a single entity type.

It does not seem likely that the database schema will use different entity types for registered and non registered SCIs, because of the difficulty of granting appropriate access to tools for compilation and linking.

The following kinds of object need to be considered:

1. Software documentation objects:

    1. requirements documents;

    2. design documents with relationships to the supporting requirements;

    3. user documents;

    4. instructions for starting and recovery after failure;

    5. running environment (hardware and software) required for each executable program.;

2. Compilation associated objects:

    1. program source text with relationships to supporting design documents;

    2. compiled code;

    3. linked compiled code;

    4. executable code;

    5. compiler and loader reports;

    6. maps showing the allocation of the executable code to computer store and to firmware;

    7. compilation, link and load commands for each version of the executable programs, including versions of the compiler etc. used.;

3. Test associated objects:

    1. database of test data and expected results;

    2. test procedures;

    3. record of actual test results.;

4. Management objects:

    1. product entity;

    2. SCI network (of SCN entities);

    3. baselines for each milestone, and each version of the product;

    4. product development plan;

    5. quality assurance plan;

    6. configuration management plan;

    7. acceptance test plans;

    8. integration test plan;

    9. personnel associated with a project;

    10. bodies able to authorise registration and bonding of SCIs with relationships to members of the bodies.;

5. Change control objects:

    1. Software Defect Report, with substance of the report, author, status, and relationships to person responsible for responding to the report and to other defect reports covering the same fault;

    2. Software Change Request, with substance of the change, author, status, relationships to bodies who need to authorise the request and relationships to affected SCIs;

    3. Software Change Notice, with status, relationships to affected SCIs, relationships to Software Change Requests and Software Defect Reports if appropriate;

    4. Engineering Change Proposal, with relationship to Software Change Request..

The successor and variant system of versioning will need to be worked out to match the version labelling of SCIs and SCNs.

6.4.2 Characteristics Of Objects Under SCM.

The different characteristics of entities in the database can be catered for by attributes or by relationships to other entities.

The mandatory predefined entity attributes that give derivation, date of creation and last amendment and name of user who created the entity and who made the last amendment will provide for some of the needs for configuration status accounting, although it is likely that more of the history of a project will need to be kept than will appear in these attributes. A reason attribute to hold the reason for any change made to the configuration could also be of value. Tools could be required to write to a Configuration Status Accounting entity or attribute every time anything is done to any item associated with the configuration of the software.

The SCN entities will need the following characteristics:

1. SCI identifier;

2. status of the SCI (registered or bonded) and date on which it achieved that status, held as attributes of relationships between the SCN and the SCI;

3. relationships to the bodies by whom authorisation for registration/bonding can be granted (the relationships can have the status and date attributes to show whether and when registration and bonding have been approved);

4. relationships to each baseline entity if the SCI is a member of the baseline (the relationship can have an attribute to show whether the SCI is bonded or registered in that baseline).

The registered SCIs will need to have the following characteristics, (in addition to the mandatory predefined entity attributes):

1. kind of object (where an entity type caters for more that one kind of object);

2. relationships binding all objects and attributes on which the SCI depends (this dependence may be of the form 'A' is needed to recreate 'B', or a more general form of dependence whereby a change in 'A' implies a possible need for a change in 'B'.);

3. a relationship to the SCN entity, binding the significant attributes and relationships of the SCI to prevent amendment of the attributes or deletion of the relationships;

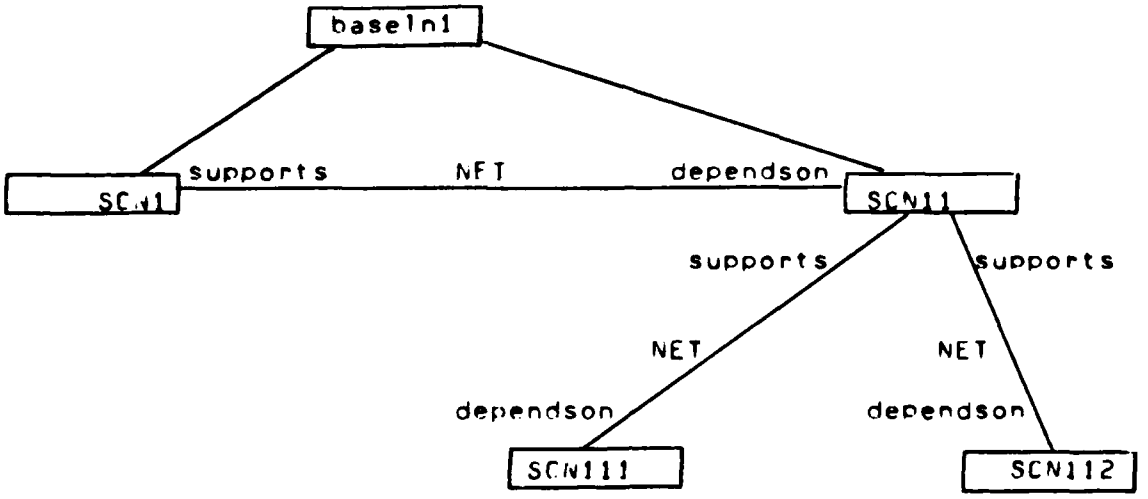
4. mandatory relationship to the database object representing the person responsible for the SCI.

6.5 Example Of Part Of An SCM Database.

This example is intended to illustrated the use of binding and of the skeleton SCI network.

Let there be an entity type containing the SCNs and another entity type containing the baselines and let the SCI network connections be represented by relationships (NET), with roles "supports" and "dependson".

A database for an SCI network is shown below:



Suppose that TEXT is the entity type containing objects that have text attributes, including requirement specifications, design specifications and source text, and suppose that the following relationships can connect TEXT entities to each other:

1. relationship type RR connecting requirement specifications (role "req") to designs (role "meets");

2. relationship type DD connecting source code (role "impl") to designs (role "des");

Suppose also that the role "meets" is a binding role, and binds the entity and the body attribute of the entity performing the "req" role, and the role "impl" is a binding role, and binds the entity and the body attribute of the entity performing the "des" role plus all relationships of type RR in which the bound entity performs the "meets" role.

In entity type TEXT there are the following entities:

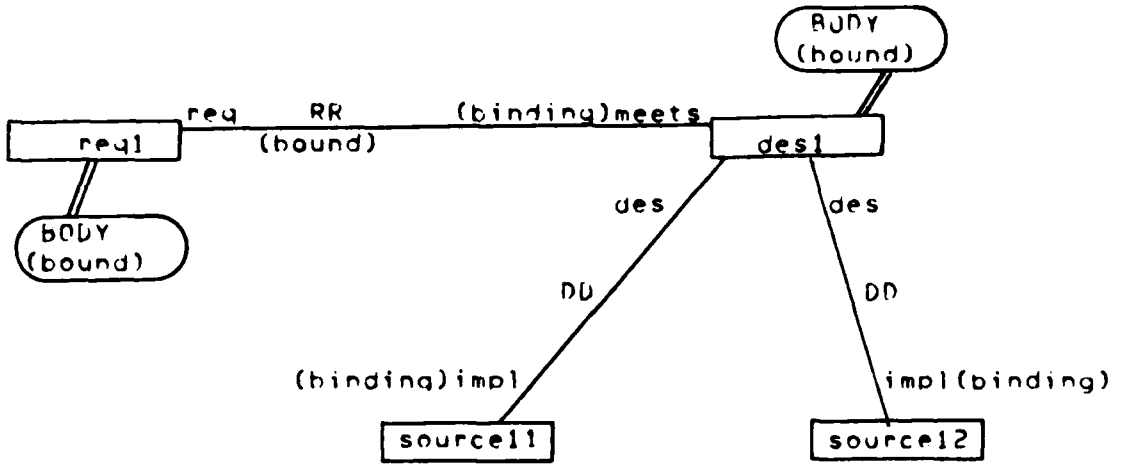
1. req1=requirement spec (SCI1);

2. des1=design (SCI11) to satisfy req1;

3. source11=source code (SCI111) for one part of des1;

4. source12=source code (SCI112) for another part of des1.

The database for the text entities is as shown below:



The existence of the relationship between des1 and source11 binds des1 and the body attribute of des1, and the relationship between des1 and req1.

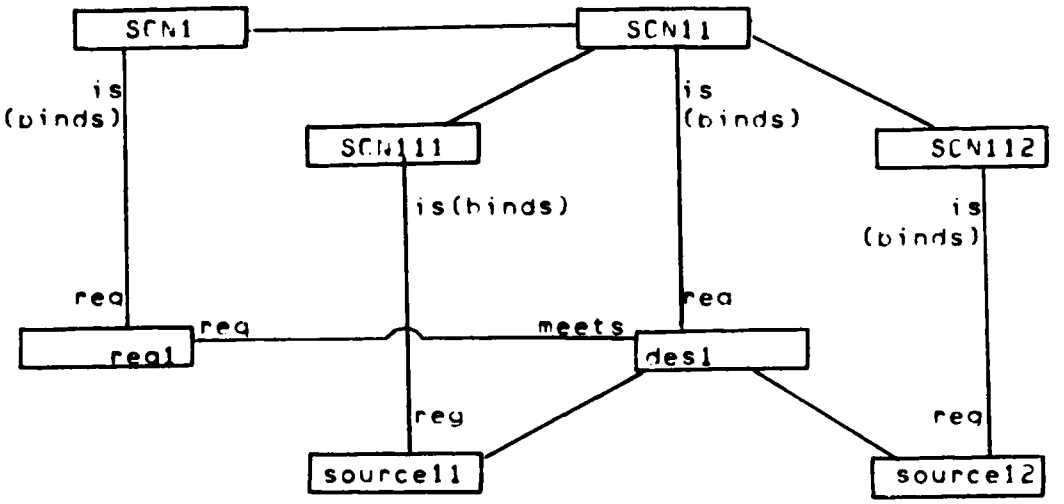
Similarly, existence of the relationship between des1 and source12 binds des1 and the body attribute of des1, and the relationship between des1 and req1.

The existence of the relationship between des1 and req1 binds req1 and the body attribute of req1.

Any attempt to delete req1 or to change the body attribute of req1 will fail because of the relationship to des1 that in turn is bound by both the relationships between des1 and the two source entities.

Let the relationship connecting an SCN to its registered SCI entity have the roles "is" and "reg", where "is" binds the entity performing the "reg" role.

As each entity in the text entity type shown above is registered the connection between it and the corresponding SCN entity will be created, thus binding the registered SCI as shown below:



req1 is bound by the relationship to SCN1 (role "is");
des1 is bound by the relationship to SCN11 (role "is");
source11 is bound by the relationship to SCN111 (role "is");
source12 is bound by the relationship to SCN112 (role "is").

7.0 SOFTWARE TOOLS FOR CONFIGURATION MANAGEMENT.

The database tools will check that the access controls are enforced and will provide archiving and recovery facilities. The software configuration management tools should always ensure that all necessary authorities have been obtained, and all the requisite information has been supplied and is acceptable, before modifying the software configuration database or issuing registered SCIs. They should use transaction management where appropriate to preserve the integrity of the software configuration database and should always write the history information needed for configuration status acccounting into the configuration database. They should prompt for reasons for any changes, to be stored in the reason attribute.

The tools for listing and cataloguing the content of the software configuration database may be based on a standard database query language and associated data dictionary, if such a language is developed for the PSE database.

7.1 Access Controls And The Project Librarian.

The project librarian will normally be responsible for the creation and maintenance of the software configuration database. The librarian must be the only person having access to registered SCIs. When an SCI is registered, the registration tool may change the access permits of the SCI possibly restricting access to special tools. The librarian will then need a special set of tools that have access authority to operate on registered SCIs with restricted access rights.

7.2 Configuration Identification.

Tools are needed to:

1. enter and update the configuration management plan, including the SCI network, definition of the baselines and the authorities who control changes to the software configuration database;
2. print the SCI network;
3. list a baseline, with details of when the items came under control and whether or not the items are bonded;
4. list the items in a baseline that have not yet been registered;
5. show how the various items combine to produce different versions of the executable software;
6. catalogue in various ways the registered SCIs, their dependencies, and the associated entities and relationships (e.g. associated personnel).

7.2.1 The SCI Network Tool.

The SCI network tool may:

1. set up the network of SCNs;
2. connect the baseline entities to the SCNs;
3. connect the product entity to the baselines;
4. set up the entities representing the bodies needed for the different kinds of authorisation with relationships to the person entities and to the SCNs;

5. provide facilities for extending and modifying the network of SCNs as the project progresses.

7.3 Configuration Control.

Tools are needed to:

1. register new SCIs, checking that the necessary authorities have been obtained;

2. bond SCIs, checking that the necessary authorities have been obtained;

3. list authorities required for registration or bonding of an SCI, or for modification;

4. list all items dependent on an SCI with details of the personnel who need to be informed;

5. provide a mailbox facility to inform personnel of actions required by them;

6. progress defect reports and change requests through the system;

7. issue reference copies of registered (bonded) SCIs as authorised;

8. re-register amended SCIs;

9. archive all registered and bonded SCIs.

7.3.1 The Registration Tool.

The registration tool may:

1. check that all authorities required for registration have been granted;

2. check that all the required tests have been successfully performed;

3. change the access rights to the registered SCI;

4. create a binding relationship between the SCN and the SCI;

5. scan the derivation attribute of the SCI in order to discover the registered SCIs needing to be related to the new registered SCI, and to set up any new relationships that are needed;

6. write the history information necessary for configuration status accounting.

7.3.2 The Bonding Tool.

The bonding tool will be similar to the registration tool except that instead of creating the binding relationship between SCI and SCN and scanning the derivation attribute it will need to modify the level of control attribute of the relationship between SCN and SCI.

7.3.3 Change Control Tools.

The change control tools will include three groups of tools:

1. tools for progressing a change through its various stages;

2. tools for the issue of SCIs so that changes can be made;

3. tools for re-registration, including any necessary modification to steering files to cater for new versions and variants.

The tools for progressing changes through the different stages may:

1. build up a network of relationships between change documents and the SCIs to be changed, as the change progresses;

2. modify the baseline entities and the network of SCNs to accomodate authorised changes;

3. list the authorisations required for a given change, the SCIs affected, the people responsible and the status of the change;

4. provide reports on changes to a configuration, such as listing outstanding changes, changes implemented for a particular baseline etc.;

5. write the history information necessary for configuration status accounting.

The tool for issue of SCIs may:

1. refuse to permit issue without the necessary authority;

2. revise the SCI to be modified, creating a new version with a relationship to the parent SCI;

3. change the access controls to the issued SCI;

4. delete the binding relationships so that the modifications can be made.

The tool for re-registration may:

1. prompt for the reason for making changes, to store in the reason attribute;

2. modify steering files as appropriate to cater for changes in version and variant of the SCI;

3. invoke the registration tool to register the amended SCI as a new version of the SCI which has been amended.

A useful tool has been developed by Bell Laboratories, (ref. 9) called 'Modification Request Control System (MRCS), that provides several of the facilities needed for change control. MRCS tracks and reports project change requests through interactive input and extraction of change request reports from computer files.

7.3.3.1 Text Object Version Control.

There is a tool developed by Bell Laboratories that provides some very useful facilities for version control of text objects. Called 'Source Code Control System (SCCS) (refs. 4 and 13), it provides facilities for storing, updating and retrieving all versions of text objects, for controlling updating privileges and for recording who made each software change, and when, where and why the change was made. The control is implemented by keeping a master copy plus a set of sequenced deltas (list of changes) to the preceding version, which uses less space than keeping full copies of each version. SCCS reconstructs the requested version from the master and the deltas. If a similar tool is to be used in the PSE the re-registration tool could be required to create a delta between a new version of an SCI and its predecessor.

7.4 Configuration Auditing.

The state of the art in tools for validation and verification in the sense implied for configuration auditing is not currently very advanced. However the database could include attributes detailing the findings of the verification and validation that has been performed. Tools could then be provided to list the ways in which a baseline differs from the needs implied in earlier baselines.

7.5 Configuration Status Accounting.

The necessary history will be stored in the database and tools must be provided for producing the appropriate reports.

7.5.1 Recording Terminal Dialogue.

because the librarian function is so vital in software configuration management, it may be desirable to record all terminal dialogue between the librarian and the system, so that when problems occur it will be possible to trace what has happened.

8.0 SUMMARY.

Any Programming Support Environment needs an integrated set of software configuration management tools that will, combined with an adequately designed database schema, support control of the whole life cycle of a software product.

The design of a consistent and complete software configuration management system has three major facets:

1. judicious design of an underlying database schema to provide the necessary network of entity types and relationship types, each having attributes appropriate for configuration management, using the properties provided such as binding, mandatory attributes and roles etc.;

2. selection of a strategy for the use of access permits, access authorities and access rights to support control of the software configuration database;

3. design of appropriate tools for each aspect of configuration management, using the transaction management facilities to ensure database integrity.

SUMMARY.

The SCM database schema needs to be as independent as possible of the details of the rest of the database schema so that the overall database schema can be extended and, if appropriate, members of the new entity types and relationship types can be brought under SCM without redefining the basic SCM database schema and tools.

8.1 SCM Functions.

The difficulties implicit in developing software with inadequate SCM facilities are reduced when an SCM toolset is integrated with the underlying software configuration database.

8.1.1 Software Regeneration.

The risk of incorrect software regeneration because of inadequate historic information is much reduced by the use of the derivation attribute, together with relationships to steering files and the enforced recording of history by the SCM tools.

8.1.2 Product Status.

The product status at any time will be visible through the baselines and the change control system together with tools for listing and cataloguing the software configuration database.

8.1.3 Effects Of Changes.

The effects of making any proposed changes will be traceable through the relationships in the SCI network and the relationships between registered SCIs. The relationships to the personnel responsible will ease the problem of evaluating the results of changes.

The facilities for two forms of versioning will prove useful when building software systems that must differ only in some small particular. It will be important to ensure that the successor and variant of any entity are given explicitly where any confusion might arise. However, the use of preferred successor and variant may well be useful in command files for the creation of new entities, to remove the need for change to such files when the entities referenced in them are updated.

8.1.4 Control Of Software Evolution.

Since the tools can check for proper authorisation of all changes, it will be easier to prevent the creation of unplanned variants.

SCIs can be forced to have at least one of any attributes and relationships that are deemed necessary, by the use of mandatory attributes and relationships. For example source code entities can be forced to have at least one relationship to a design entity.

Unauthorised modification to or deletion of registered SCIs can be prevented by binding and by access controls. The ability separately to control access to the different parts of the configuration database and to modify the access controls of individual objects when appropriate is a powerful way of preventing unauthorised access to registered SCIs. The fact that access rights may be different for different users is also useful in that the librarian may be given much more general access to registered SCIs than granted to any other user.

8.2 Tools.

It will be necessary to develop a special set of tools for configuration management. The transaction management facilities will be used to maintain the integrity of the software configuration database. The tools will use the relationships in the database to discover dependencies so that the appropriate personnel can be informed when changes are required. They will check that all the necessary authorities have been obtained and all necessary information has been supplied before modifying or issuing items from the software configuration database. They will also ensure that adequate historic information is stored in the configuration database to satisfy the requirements for reconstruction of software and for configuration status accounting.

Tools will be needed for each of the software configuration management functions, including tools for:

1. setting up and updating the configuration management plan including the baselines, the SCI network and the set of authorities involved in configuration management;

2. registering and re-registering SCIs;

3. issuing reference copies of specific registered SCIs to authorised personnel;

4. cataloguing in various ways the registered SCIs, their dependencies, the associated SCIs (e.g. associated personnel);

5. establishing what is in the current baseline and what is missing from a planned baseline;

6. progressing software defect reports and software change notices;

7. reporting on history and current status of the software configuration.

The database tools will provide the archiving facilities necessary for handling the volume of data implicit in configuration management.

9.0 CONCLUSIONS.

The proposed MCHAPSE offers an excellent starting point for the development of an integrated system for software configuration management. The CHAPSE database offers facilities for defining a schema capable of supporting the requirements of configuration management, with the necessary relationships between different kinds of SCI, and binding together with access controls to prevent unauthorised change to the software configuration.

Good schema design, together with the use of a data dictionary, will make it possible to provide the various reports needed for software configuration management and will prevent modification of or deletion of objects on which other objects depend. A standard PSE database query language, if available, will ease the production of reports.

Good tool design will preserve the integrity of the software configuration by the use of transaction management. The tools can prevent registration or issue of SCIs without proper authorisation and can ensure that the history of the evolution of the software is maintained. They can produce reports to assist in proper planning and control of the software evolution and can provide facilities for change control, tracing through all defect reports and change requests and showing the interdependence of SCIs.

The basic SCM schema and tools should be designed, as far as possible, to cope with extensions to the overall database schema, allowing new entity types to be brought under SCM as required without rewrite of the tools or redesign of the SCM schema.

10.0 BIBLIOGRAPHY.

1. US Department of Defense "Requirements for Ada Programming Support Environments 'Stoneman'" Feb 1980;

2. E.H.Bersoff, V.D.Henderson and S.G.Siegal "Software Configuration Management - An Investment in Product Integrity" Prentice Hall 1980;

3. R.Dunn and R.Ullman "Quality Assurance for Computer Software", McGraw Hill 1982;

4. A.L.Glasser "The Evolution of a source code control system" Proceedings of the Software Quality and Assurance Workshop in Software Engineering Notes, Vol 3 No 3 July 1978;

5. E.H.Bersoff, V.D.Henderson and S.G.Siegel "Software Configuration Management: A Tutorial" Computer Jan 1979;

6. K.J.Pulford "Configuration Management and the Ada Programming Support Environment" 1982;

7. Z.Jelinski "Configuration Management and Software Development Techniques" Defense Systems Management Review Vol 2 No. 3 Summer 1979;

8. R.McCarthy "Applying the Technique of Configuration Management to Software" Defense Management Journal Vol 11 No 4, Oct 1975;

9. D.B.Knudsen, A.Barofsky and L.R.Satz "A Modification Request Control System" Proceedings of the 2nd International Conference on Software Engineering, IEEE 1976;

10. RSRE "MCHAPSE Requirements specification Issue 1" Nov 1982;

11. AGL "Final Report of Study into the MCHAPSE" Sept 1982;

12. NCC Course Notes for course on Software Design;

13. M.Rochkind "The Source Code Control System" IEEE Transactions on Software Engineering, Vol SE-1 No. 4, December 1975;

14. E.Sibley, P.G.Scallan and E.K.Clemons "The software configuration management database" NCC AFIPS Conference Proceedings, Vol 50 pp (249-255), 1981;

15. T.A.D.White "Separate Compilation in the MCHAPSE", RSRE Memo. (to be published);

16. S.J.Bevan "The UK KAPSE Data Base", RSRE Memo. (to be published);

17. US DoD "Reference Manual for Ada Programming Language" July 1982;

18. W.F.Tichy "Software Development Control based on Module Interconnection," COMPSAC 1981, Tutorial on Software Development Environments;

19. W.F.Tichy "Design, Implementation and Evaluation of a Revision Control System." Proceedings of 6th International Conference on Software Engineering, 1982;

20. O.Shigo, Y.Wada, Y. Terashima, K.Iwamoto and T.Nisimura, "Configuration Control for Evolutional Software Products", Proceedings of 6th International Conference on Software Engineering, 1982;

21. Draft British Standard "Code of Practice For Configuration Management of Computer based Systems" 1983;

22. W.H.Josephs, "A Mini-computer based library control system", Proceedings of the Software Quality and Assurance Workshop, Performance Evaluation Review, Vol 7 Nov 1978;

23. A.Begley, "The software development environment for a large real-time project" First Annual Conference on Computers and Communications, IEEE 1982;

24. M.L.Flournoy, "Software Workbench automates program modification." Data Communications, Feb 1982;

25. B.Sheekey and N.R.P.Milway "System X: Design and Support, Part 5 Change Control and Documentation Database" PUFEJ Vol 73, July 1980;

26. M.G.Walker "Managing Software Reliability, The Paradigmatic Approach." Elsevier North Holland 1981;

27. P.W.Metzger, "Managing a programming project" Prentice Hall Inc. 1973.