Software Development Plan

Superconducting Super Collider Laboratory
Software Development Plan

D. Gurd

Superconducting Super Collider Laboratory*
2550 Beckleymeade Ave.
Dallas, TX 75237, USA

June 1994

1. Introduction. ................................................. 3
  1.1. Project Overview ......................................... 3
     1.1.1. The EPICS System ................................ 4
     1.1.2. SSC Specific Applications ...................... 4
  1.2. Project Deliverables .................................... 5
     1.2.1. System Deliverables ............................... 5
     1.2.2. Application Deliverables ...................... 7
  1.3. Evolution of the Software Development Plan .......... 7
  1.4. Reference Materials ................................... 8
  1.5. Definitions and Acronyms ................................ 8
2. Project Organization. ...................................... 9
  2.1. Process Model ......................................... 9
  2.2. Organizational Structure ................................ 10
3. Managerial Process. ...................................... 12
  3.1. Management Objectives and Priorities ................ 12
  3.2. Risk Management ...................................... 13
  3.3. Monitoring and Controlling the Development Process .... 15
     3.3.1. Application Software Reviews .................. 15
     3.3.2. System Software Reviews ........................ 16
  3.4. Staffing Plan ......................................... 17
4. Technical Process. ...................................... 17
  4.1. Methods, Tools and Techniques ....................... 17
     4.1.1. Development Tools ................................ 18
     4.1.2. Software Repository .............................. 18
     4.1.3. System Development Area ....................... 19
     4.1.4. Application Development Area ................ 23
     4.1.5. Software Test Area .............................. 25
  4.2. Software Documentation ................................ 25
     4.2.1. Software Requirements Specification ........ 26
     4.2.2. Software Test Plans .............................. 26
  4.3. Project Support Functions ............................ 27
     4.3.1. Development System Administration ............ 27
     4.3.2. Configuration Management ...................... 27
5. Work Packages, Schedule and Budget ............... 30

Appendix A System Modification Request Form ............ 30
The Superconducting Super Collider Laboratory is operated by the Universities Research Association, Inc., for the United States Department of Energy under Contract Number DE-AC02-89ER40486.

This Document was printed August 11, 1993; It was created July 9, 1993 and was last modified August 11, 1993.

It contains 32 Pages.

### Revision Log:

<table>
<thead>
<tr>
<th>Revised by</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doug Murray</td>
<td>Aug 1993</td>
</tr>
</tbody>
</table>

---

-2-
1. Introduction

This Software Development Plan (SDP) describes the software development process for Control System Software at the Superconducting Super Collider Laboratory (SSCL).

The Control System Software will be developed using EPICS, the Experimental Physics and Industrial Control System, which contains a set of software "tools" for building control systems.

Accordingly, two separate development efforts will take place; one to extend EPICS, ensuring that it can be kept generic and useful for any control system development, and second to develop applications using EPICS, which will be specific to the SSC1. Developers will build a control system application using EPICS in much the same way that an accountant builds an "application" using a spreadsheet program.

1.1. Project Overview

This project includes software to control equipment in the Linear Accelerator (LINAC), the Low, Medium and High Energy Boosters (LEB, MEB and HEB), the Collider itself, and various test beams.

The project can be considered as having two parts, as mentioned above. Specification documents are organized into a specification tree, over which these two development efforts are mapped.

The following figure shows the organization of documents at Level 3B and Level 4. For our purposes, Level 3B documents describe a global control system, having features common to all accelerator control systems (EPICS); this document refers to the highlighted areas. Control systems at Level 4 deal with problems which are specific to a particular accelerator (the applications), or with changes or extensions to generic system software.

![Figure 1. Specification Levels for SSC Control Systems](image)

1. The term application will hereafter refer to the SSC specific software developed using EPICS. This includes many aspects of a specific development, including control or monitoring algorithms, sequences to accommodate different modes of operation, and the display screens through which operators interact with them.
Software Development Plan

Level 5 (not shown) deals with specific components within each subsystem.

Additional SDP documents may be written as the project progresses. Such documents would address situations which are unique to a particular accelerator or subsystem, which are not addressed in this document. Additional plans would refer back to this document for information common to all development efforts.

The review process for Control System software in general is described under Monitoring and Controlling the Development Process on page 15.

1.1.1. The EPICS System

EPICS was developed at Los Alamos and Argonne National Laboratories\(^2\), and was intended for use as a tool set to build Control Systems. Since it already exists, our goal involves extending EPICS to meet our needs, and enhancing it, making it easier to develop control system applications. The schedule for this work is described under Work Packages, Schedule and Budget on page 30, and is in some part dependent upon the schedule for application development.

Requirements for these tools are documented in the Global Accelerator Control System Requirements document, listed under Reference Materials on page 8.

EPICS was designed to address several requirements which are common to accelerator control systems. A more complete description of EPICS can be found in “Experimental Physics and Industrial Control System (EPICS) Technical Note”, listed under Reference Materials on page 8.

1.1.2. SSC Specific Applications

Applications are built using tools which EPICS provides. Ideally, these tools should be useful to technicians, engineers and non-programmers in general, and should not require an intimate knowledge of EPICS. The tools are themselves considered to be deliverables from EPICS development, and are described under System Deliverables on page 5.

At least two broad categories of applications can be identified:

1. Operator applications, used in a control room setting, which presents a system or component at a high level;

---

2. Engineering applications used in a Laboratory environment, or in the field during periods of maintenance. They offer access to diagnostic features of the system and allow more elaborate testing.

In addition to EPICS tools, some application developers will make use of other pieces of software as the project progresses. This might include, but is not limited to, software available for the X Window System, available commercially or freely distributed, as well as development aids, such as language sensitive editors, debuggers and various other CASE tools. The VxWorks real-time kernel is the platform upon which certain parts of applications will operate.

Details of application schedules will be presented in the subsequent system specification documents, as accelerators and subsystems are designed.

1.2. Project Deliverables

Deliverables also fall into one of two categories, system or application. System deliverables includes software required to develop applications. Application deliverables refer to applications developed for specific accelerators or subsystems.

1.2.1. System Deliverables

Several tools in the EPICS system shall be delivered to application developers at the outset of the project. These tools currently exist and are in use.

First, there exist tools which do not require the developer to develop code in the traditional programming sense. Refer to Reference Materials on page 8 for further documentation:

**dct** the Database Configuration Tool is used to define most of the functionality of the application, by allowing the developer to describe both data and processing in terms of records in a simple database. Further functionality can be added with snc, CA and the C programming language, as detailed below.

**edd** a tool used to build the display screens, through which users interact with the accelerator or subsystem. It is an *editor* in the sense that developers can create or modify the operator's view of the equipment. The developer selects graphic items from a palette which are then placed at various locations on the screen, which might include representations of meters, gauges or dials which would be familiar to the user community.

**dm** a display manager program, which displays the screens built with edd, and ensures that screen items are connected to actual measurements or controls values in the field.

3. VxWorks is a trademark of Wind River Systems, Inc.
alh a tool for display, confirmation, acknowledgment and recording of alarm conditions, as configured by the developer of the application.

ar the Archiver is used to record measurements at various intervals, as configured by the developer of the application. This tool is only used for capturing and storing data, so its name might be misleading.

arr a tool to report on data which has been recorded by the archiver. The developer can configure the tool to display data in a variety of graphical formats.

Other tools will be delivered to application developers, which require coding in the more traditional sense of programming:

snc is the State Notation Compiler which allows developers to program state transitions which are required in their applications. Often used for implementing various modes of equipment or systems, this tool understands SNL, a state notation language which can interact with simple database records developed with dct (see above).

CA refers to Channel Access, which is a communications mechanism within EPICS. Programmers using the C programming language can make use of the channel access subroutine library to retrieve or set values in the accelerator or subsystem. This is typically used only for specialized applications which cannot be handled by the generic tools edd, dm, or snc.

Other tools will be developed for the applications developer, to present EPICS as a more integrated environment. Although not critical, these tools address self-imposed requirements concerning ease of use, automatic document generation (design drawings, for instance) and configuration management. These tools include, but are not limited to:

fbed a function block editor, which provides the developer with a graphic tool with which to visualize the processing required in their application. It can take the place of dct.

statem a state machine editor, which provides the developer with a graphic tool with which to visualize the states and transitions between them, required in their applications. It can take the place of snc and the SNL which it understands.

alconf a configuration tool for the alarm handler which allows a developer to determine graphically how alarms will be handled (if acknowledgment is required for certain alarms, etc.).

arconf a configuration tool for the archiving mechanism, which allows the developer to specify details of data capture and recording.

4. Refer to Management Objectives and Priorities on page 12 for more information.
arview a graphic tool through which previously acquired measurements can be viewed. While very similar to arr, this tool understands a more elaborate data management scheme using a commercial database. The developer would make use of a configuration mode to relieve the operator of understanding details.

epics a graphic window based program through which the developer gains access to the EPICS tools and their development environment. It is also used to start applications, and navigate through application specific directories on control system computers.

Enhancements to EPICS will be needed. These include:

CA changing the Channel Access mechanism will be required for EPICS to accommodate a Control System of the scale required for the SSC. In fact, the communication scheme which EPICS uses will require a substantial change of design.

alh changing the alarm handling mechanism so that a daemon is constantly running, waiting for alarm conditions to change.

data handling the mechanism through which EPICS saves measured data, and through which it configures front-end computers must be extended to accommodate the configuration management strategy (in the latter case)

1.2.2. Application Deliverables

Applications for each accelerator or subsystem are considered deliverables, for which descriptions and schedules will be documented in subsequent specification documents (refer to Project Overview on page 3).

1.3. Evolution of the Software Development Plan

The complete plan for developing software ultimately includes this (so called Level 3B) document and any accelerator specific SDPs (so called Level 4; refer to Project Overview on page 3) which might be produced in the future.

With this in mind, any additional Level 4 SDP documents would be developed as each set of applications is specified. With respect to EPICS itself, our development strategy may change as experience is gained with application development. This is considered a minimal risk, considering that the core of EPICS already exists. Refer to Risk Management on page 13.

Suggestions for changes to this plan shall be submitted to the Software Group Leader (SGL), who is responsible for updates and distribution of the SDP to the EPICS software engineers, and application
developers. Suggestions for changes to subsequent (Level 4) SDPs shall be submitted to the application engineer responsible for the corresponding project.

1.4. Reference Materials

The following documents are related to this SDP, and are available from the documentation manager in the SSC Controls Department:

- Engineering Management Plan (EMP) SSC Document E10-000029
- Global Accelerator Control System Requirements (GACS)
- Software Requirements Specifications (SRS)
- Software Test Plans (STP)
- Software Style Manual
- Experimental Physics and Industrial Control System (EPICS) Technical Note AT-8:SYS:89-005
- Getting Started with (EPICS) DCT (April 1991)
- EPICS IOC Address Specification Technical Note AT-8:IOC:88-02
- Object Oriented Requirements Analysis and Logical Design, Donald G. Firesmith
- SCCS User’s Guide
- Make User’s Guide

1.5. Definitions and Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANL</td>
<td>Argonne National Laboratory</td>
</tr>
<tr>
<td>ARR</td>
<td>Accelerator Readiness Review</td>
</tr>
<tr>
<td>ATP</td>
<td>Acceptance Test Plan</td>
</tr>
<tr>
<td>ATPR</td>
<td>Acceptance Test Plan Review</td>
</tr>
<tr>
<td>CA</td>
<td>Configuration Administrator</td>
</tr>
<tr>
<td>CDR</td>
<td>Critical Design Review</td>
</tr>
<tr>
<td>CM</td>
<td>Configuration Management</td>
</tr>
<tr>
<td>CSPM</td>
<td>Control System Programming Manual</td>
</tr>
<tr>
<td>EMP</td>
<td>Engineering Management Plan</td>
</tr>
<tr>
<td>EMU</td>
<td>Emittance Measurement Unit</td>
</tr>
<tr>
<td>GACS</td>
<td>Global Accelerator Control System</td>
</tr>
<tr>
<td>IOC</td>
<td>Input Output Controller</td>
</tr>
<tr>
<td>LANL</td>
<td>Los Alamos National Laboratory</td>
</tr>
<tr>
<td>LBL</td>
<td>Lawrence Berkeley Laboratory</td>
</tr>
<tr>
<td>MPO</td>
<td>Memorandum Purchase Order</td>
</tr>
<tr>
<td>OPI</td>
<td>Operator Interface</td>
</tr>
<tr>
<td>PDR</td>
<td>Preliminary Design Review</td>
</tr>
</tbody>
</table>
2. Project Organization

The Accelerator Systems Division consists of several Departments involved in the Control System development effort. This involvement ranges from development of device specific software within the system, to specific engineering applications (see SSC Specific Applications on page 4).

2.1. Process Model

The process by which development will proceed is dictated by two factors, specifically how requirements for applications are generated and met, and then how requirements for EPICS are generated and met.

EPICS already exists, and was designed to meet common requirements for generic control systems, gained through years of experience with different accelerators and process control systems (refer to EPICS Technical Note, listed under Reference Materials on page 8). EPICS shall change (or be extended) if the tools it provides are not adequate for developing the required applications (refer to Risk Management on page 13). To some extent, further requirements shall be generated from application developers as they learn what is required for their applications. Test plans and quality assurance issues for EPICS will be addressed separately from those of the applications. Refer to Monitoring and Controlling the Development Process on page 15 for more information.

Applications: The Project Management Office (PMO) will be the primary source of requirements for the accelerators and subsystems. Requirements for control system applications will follow from these, and the design and implementation of these applications will be the responsibility of specific development teams, as mentioned under Organizational Structure on page 10.
Applications shall be developed by these teams, according to guidelines discussed under the section Application Development Area on page 23. Reviews shall be held as work progresses; refer to Application Software Reviews on page 15 for more information.

Testing will be performed on components as development proceeds, and is outlined under Software Test Plans on page 26.

When successfully tested, the application, including display screens, processing (database) records and any other components shall be registered into the Configuration Management mechanism, as outlined under Configuration Management on page 27. Subsequent modification or extension of the applications require a System Modification Request.

Systems: Changes or enhancements to EPICS shall be solicited through a System Modification Request, as described under Configuration Management on page 27. Once approved, the change will be implemented, with appropriate design, and tested.

Different groups participate in the Development Process. The Application Developers and EPICS Programmers, as depicted in Figure 3., correspond to the Applications and Systems Groups, respectively, as shown in Figure 4.

Systems Engineering and Quality Assurance Groups within the SSC shall be involved in requirements analysis and reviews as the project proceeds. Two separate sets of requirements and reviews, corresponding to Applications and Systems work, are shown in Figure 3.

2.2. Organizational Structure

The Software Group is responsible for the EPICS system software and many of the applications. This group is contained within the Controls Department, and is shown in Figure 4. Tasks are assigned to teams under Systems or Applications.
The Controls Department is responsible for integrating the efforts of other Departments within the Accelerator Systems Division, as shown in Figure 5. The Systems Engineering and Quality Assurance functions are performed by separate groups within ASD. Another separate group within ASD is responsible for administration of development computers and software, to the specifications of the developers.

The Controls Department is also responsible for integration efforts from other Divisions within the SSC, including the Machine Simulation Group and various detector groups. Note that the Global Machine Safety Systems Group within ASD is responsible for a completely separate and distinct system, but the state of that system must be available to the Control System for display and interlock purposes.
Software Development Plan

Figure 5. Organization of the Accelerator Systems Division.

Software efforts are also engaged from other Laboratories, through memorandum purchase orders (MPO). Initiation of such agreements, formal statements of work (SOW) and monthly progress reporting shall be carried out according to existing SSC policies.

3. Managerial Process

The Control System for the SSC represents a major, complex software project. Priorities have been established, and risks have been identified.

3.1. Management Objectives and Priorities

The Control System Software Group must be both system developers with a research and system design attitude, but also a service oriented organization, which must ensure that accelerator operators have all of the applications available for controlling the SSC.

Given these objectives, priority shall be given to those aspects of development related to serving the accelerator operations staff. This includes reliable software for accelerator specific functions, accurate and available documentation and well defined procedures for problem resolution.

Beyond that, highest priority shall be given to those development efforts which are directly tied to accelerator schedules. Specifically, applications developed with EPICS tools take priority over enhancements to EPICS itself, unless those enhancements are deemed critical to the application operation by the SGL.
3.2. Risk Management

Applications: Development of applications will have risks, based upon issues relating to the accelerator or subsystem in which they will be used. A Safety Analysis Report (SAR), prepared for each accelerator system or subsystem, shall identify areas of risk for various applications.

Test plans shall be outlined in Acceptance Test Plan Reviews (ATPR), which will address all of the problems associated with testing applications for specific accelerator systems. Refer to Software Test Area on page 25 for more information.

Schedules for these applications will contain critical path items, which will be determined and presented as part of regular Controls Group reviews. Work-around alternatives will be determined and scheduled to be presented as backup plans.

Configuration: Proper Configuration Management (CM) is essential to the success of this project; hundreds of people could potentially have access to application source files (display templates, configuration database templates, etc.), and to the EPICS source code as well. This issue represents great risk if the implementation is not adhered to; the implementation is described under Configuration Management on page 27.

EPICS problems: EPICS was developed with very general requirements in mind, based primarily on previous experience of the engineers. There is a risk that their work will not scale to meet the needs of the SSC, or that application developers may have requirements which the existing EPICS tools cannot meet.

The physical distances between components in the SSC, and their numbers present problems which are unique. The communications system must accommodate much greater amounts of data and larger distances than did previous systems. Accordingly, EPICS must be changed to make effective use of that system. The risks presented by such a change are minimized however, because of adherence to communications standards in hardware and software; ideally, changes to a layer of communications software should be transparent to EPICS, so that no changes will be required. If changes to the system are indeed required, they will be localized and minimal.

Changes to EPICS: Enhancements to EPICS will be required, and risks are greater than those associated with other software projects. It is often much more difficult to modify code developed by other engineers, using different development plans and style guidelines, than to develop the sys-
tem in a consistent way from the beginning. A strong configuration management and quality assurance scheme is essential to minimize these risks.

EPICS collaboration: Several Laboratories in the United States are working together to develop extensions to EPICS. This includes Los Alamos National Laboratory (LANL), Argonne National Laboratory (ANL), Lawrence Berkeley Laboratories, and the SSC. Benefits from this collaboration are great, but there is a risk that changes to existing software, or even the direction of EPICS in the future, may not meet ongoing requirements at the SSC. We have the option of leaving the collaboration at any time, should our situation warrant it.

EPICS dependencies: EPICS imposes certain restrictions on the types of hardware which can be used. Currently, computers from SUN Microsystems must be used, with their version of the UNIX operating system. VxWorks is required as a real-time kernel, which imposes further restrictions upon processor architectures and types. There is a risk that future development will be dependent upon actions of these vendors, and that advances in technology from other vendors will not be available to us.

Steps are being taken to overcome these risks; plans have been proposed to produce software, or modify existing software, to make it POSIX compliant5. We are making use of other standards, such as Motif on the X-window system for user interface displays and graphics, so that as many vendors as possible can be considered.

Outdated technologies: Technology used in all aspects of development are susceptible to becoming outdated. There is no risk in being outdated, other than the technical support and enhancements to that technology might fall off in the future. While attempting to develop software based on standards, and putting effort into ensuring portability across platforms, it is in our best interests to consider new technologies as they become available. EPICS was developed using currently accepted tools and languages such as C and yacc for example, but new developments could make use of C++ or other object oriented technologies, which are likely to be much more popular (better supported and more widely used) in the future.

SDP Changes: Changes to the SDP will occur over the lifetime of the project; we do not associate a high level of risk with these changes, because requirements analysis and development details will be determined as the project proceeds. This SDP document is the cornerstone of a complete development plan, and the development of accelerator specific applications will be documented as their requirements are made known. Refer to Evolution of the Software Development Plan on page 7.

5. when details of the POSIX standard, including real-time extensions, are well defined.
3.3. Monitoring and Controlling the Development Process

The EPICS software will be extended as application dependent requirements become known. The review process is outlined below. Applications software will be reviewed when the associated accelerator or subsystem is reviewed.

3.3.1. Application Software Reviews

**Formal Reviews:** Reviews for application software shall be integrated with reviews for their corresponding subsystems or accelerators, which are described in the SSC Engineering Management Plan (EMP). To summarize, they include:

- Preliminary Design Requirements Review (PDRR)
- Preliminary Design Review (PDR)
- Critical Design Review (CDR)
- Acceptance Test Plan Review (ATPR)
- Accelerator Readiness Review (ARR)

Systems Engineering shall support all activities associated with these applications reviews including preparing documents, taking minutes and recording action item assignments during the meeting. Systems Engineering distributes the minutes within ten days following the meeting, and tracks action items.

Components of applications shall be tested separately, to as great an extent as possible. There shall not be an independent acceptance test plan (ATP) for application software, since the applications are so closely tied to hardware components. The ATPs shall test software and hardware in an integrated fashion, and ATPRs shall be scheduled to review the acceptability of both, working together.

Certain applications might be developed which are generic in nature, and can be "copied" as similar subsystems or hardware components are installed. Smaller, informal test reviews of these specific applications may be justified, to ensure that they will indeed be suitable for similar subsystems.

**Informal Reviews:** The Software Group will also hold regular informal reviews for all operator applications. These are referred to as in-process reviews (IPRs) and will be scheduled by the responsible applications engineer, on a monthly basis. Such reviews shall deal with one or more of the following topics, having information specific to that application, and gleaned from the engineering notebooks (refer to Software Requirements Specification on page 26):

6. the application might be, more appropriately, multiply instantiated.
7. Reviews will be held only for those Engineering Applications which will be used in a Control Room. Refer to SSC Specific Applications on page 4.
Software Development Plan

- **Operator Interactions**: a review of operator's display screens, and how the operators interact with them.
- **Hardware Channels**: a review of known channels which come from, or go to, hardware modules, and a list of all known modules.
- **Data and Processing**: a review of EPICS database records, how they are linked and the process variable names.
- **Modes and Transitions**: a review of system states and transitions using SNL.
- **Component Tests**: a review of major tests performed on the software, both independent of, and integrated with, hardware components.

These informal reviews will provide the basis for the formal reviews mentioned above, and shall be attended by applications developers, members of the quality assurance organization and system engineers. Representatives from PMO will be asked to attend, so that change requests can be approved.

Weekly status reports from application development teams will be submitted to the SGL, who will make them publicly accessible.

### 3.3.2. System Software Reviews

Review of EPICS related work will be less frequent because the system already exists. Future developments, including enhancements to existing code, shall be reviewed according to the format described in *Object Oriented Requirements Analysis and Logical Design: A Software Engineering Approach*. Specifically, Chapter 9 of that book is entitled Development Cycles and Major Reviews, and describes how reviews can best accommodate modern development methodologies. It makes reference to DOD-STD-2167A.

Requirements for enhancements to (or extensions of) EPICS often originate with application developers, but may also be generated from within the team of programmers who are responsible for making those changes. In any case, the SGL may decide if a Software Specification Review (SSR) is warranted.

One reason for insisting on requirements before any work begins, is to ensure that all engineers understand that extra functionality is not wanted. Referred to as *creeping functionality*, engineers often implement the required features and decide that additional features are easily added.

---

The extent to which requirements are understood and documented before any work begins shall be determined by the SGL. If requirements are completely understood beforehand, additional functionality or other features shall be documented and submitted to the SGL for approval.

In any case, regular IPRs will be held, and scheduled by the responsible engineer at intervals no longer than 30 working days without approval of the SGL.

Weekly status reports from system development teams will be submitted to the SGL, who will make them publicly accessible.

3.4. Staffing Plan

Each of the teams in the Software Group, depicted in Figure 4, shall have between 3 and 7 members, including a Team Leader, for the duration of the project. Additional application teams may be required as the project continues. Additional teams of technicians will be required for system administration and maintenance functions, as accelerators become operational.

4. Technical Process

Requirements for applications will originate primarily from within the PMO.

Requirements for the tools used to build those applications will come from application developers and members of the Software Group.

An overview of the project organization and how it relates to this process, as well as the Development Process itself is contained in the Section entitled Process Model on page 9.

4.1. Methods, Tools and Techniques

Development of applications software, as well as enhancements or extensions to EPICS software, shall take place on workstations running the UNIX operating system, specifically computers from SUN Microsystems as dictated by the EPICS system. This SDP will change as new processor types are made usable.

Single board computers (IOC computers in the EPICS context), shall include any processor supported by the VxWorks real-time kernel, and authorized by the Software Group Leader (SGL).

4.1.1. Development Tools

The tools used for applications development include all of the deliverables which EPICS will provide; refer to the list under System Deliverables on page 5. Additional tools might be identified as textual editors or documentation aids, described below.

Enhancements to the EPICS system itself will be made with a variety of tools, dependent on what was used initially, to maintain consistency, but also on contemporary standards authorized by the SGL.

Currently, these tools include compilers and debuggers for the "C" and "C++" programming language, the yacc compiler, compiler and lex (a tool for lexical analysis of input tokens to compilers). Assembly language will be used only with approval of the SGL, and then only on real-time systems.

Specific editors will not be required for coding, although all programmers currently use "vi" or "emacs" (or tools based upon them). FrameMaker shall be used to produce documentation (refer to Software Documentation on page 25), and Xmosaic shall be used to assist in on-line documentation. Autoplan shall be used for tracking schedules.

Documentation for this software is available from the SGL.

4.1.2. Software Repository

The software repository (SR) contains the software materials including source code, object code and other software documentation. The SR provides the project's primary means for exercising configuration control over the software and related documentation during development and release, as described under Configuration Management on page 27.

The current versions of documentation, source code and other files are always available in the repository area. Development takes place when programmers or application developers check files out of the system (via the CM mechanism), into their own development areas. After work is complete, including testing, the files are registered back into the hierarchy by the Configuration Administrator (CA).
The SR shall exist on a UNIX file system, and shall be accessible from a single location, as shown in Figure 6. This location shall be specifiable with the same pathname on any development computer. It is assumed that the reader is familiar with the concepts of files and directories used in modern computer systems.

![Diagram of Directory Structure for Control System Software in the Repository Area.](image)

Figure 6

Within the topmost control system directory shall exist subdirectories for applications and the EPICS software. Separate subdirectories shall exist for documentation (doc), to allow easy retrieval of documents, and for schedules (schedule), to allow access to the project schedule with respect to controls.

An important aspect of the UNIX file system is the ability to have "symbolic links" to a file or directory. This is essentially a pointer which itself appears as an entry in the file system; using them, files or directories might seem to exist in several locations, ideally where people would expect to find them. Physically, there is only one copy of the "target", and its actual location might not be important. The documentation subdirectory is an example, in that it contains symbolic links to the documentation files located elsewhere.

This hierarchy shall be writable only by the user account named "controls", as described under Configuration Management on page 27. The documentation (doc) files shall be accessible for reading by any other user account.

The SR also contains a separate integration and build area, which is a protected repository of code and text files that have been released by the developers, but has not been integrated or released to the user community.

As software is released into the SR area, the CA compiles and links them into executable software. The build is then released to the software test area for testing.

4.1.3. System Development Area

System software development will take place under the login account of individual programmers. Routine login procedures requiring a user name and password provide access control to the programmer's own work area. Weekly backups of all files will be maintained in order to prevent loss of data.
Each programmer shall have one or more subdirectories under their own computer account with a structure similar to that of the system's, as shown in Figure 7. EPICS documentation refers to this area as a "shadow directory". Programmers shall "check out" files for editing or perusal, then inform the Configuration Administrator (CA) when work is done and tested, and files need to be integrated back into the system. For more information, refer to the Figure entitled Layout of a Development Area using SCCS on page 29.

The System Development area refers to this hierarchy of directories beneath any given programmer's account, usually on their own development workstation computer. Accordingly, only a subset of the system directories may exist there at any given time, depending on the development work taking place. Typically, a programmer will have a directory "skeleton" having a structure similar to that of the SR area, but only those files related to his current work will exist there. Many of those files will exist as symbolic links, since actual copies are required only when editing is being performed on them.

These subdirectories have well defined contents:

- **epics/include** - a directory containing all of the "include" files used by parts of the software. Include files contain common definitions used in different pieces of source code, and are named with a trailing "h" extension. Specifically, include files which must be included by more than a single library or program must appear here; all include files used *only* in a single program or library can be located in the corresponding source code directory. Include files at this level are therefore used primarily within EPICS source code.

- **epics/src** - a directory containing all source code under separate subdirectories. The constituent files should not exist in this directory, but in the subdirectories described below it. Each subdirectory shall have a Makefile, containing required targets, and this directory itself will contain a Makefile which will descend to the appropriate subdirectories and "make" each of them in turn.
epics/src/lib a directory containing source code which constitutes a library (not shown in Figure 9.). Ideally, the directory will be named for specific functions, so several libraries might exist in a single project. For example, one might be called "src/libca", referring to a library for EPICS Channel Access.

epics/src/lib/Test a directory (not shown in Figure 9.) which contains test programs for libraries. As an example, src/libca/Test would contain programs which make use of the channel access routines in the channel access library. The Makefiles found in these test directories shall link their programs with libraries in the directory immediately above, and not the normally installed libraries for the project. There shall also exist a file named README which describes the (informal) test procedures.

epics/src/prog a directory containing source code for all EPICS programs. The word prog refers to some specific EPICS tool, such as alh, arr, and others (refer to System Deliverables on page 5). Note that include files (ending in "h") may appear in these directories only if it's contents are completely localized, never to be used by other software. Source code for UNIX and VxWorks platforms shall be contained in subdirectories to this one, when required.

epics/src/prog/Test This is a directory (not shown in Figure 9.) containing test routines for programs contained in the directory immediately above it (prog in this example). The Test directory shall include any files used for testing the corresponding software in its' parent directory. For example, test data generators might be used for device driver testing until hardware is available and tested, or EPICS database records and alarm configuration files might be stored there, with which the Alarm Handler (alh) would be tested. Makefiles shall be present in each of these Test subdirectories, along with a file named README which describes the (informal) test procedures.

epics/etc a directory containing administrative files, such as generic startup files, which programs or libraries would make use of. This includes such things as lists of trusted users or hosts, default options to use on program or library startup, such as "rc" files which take effect if no such files appear in login or current directories.

It may also contain executable files (or links to them for concerns of different architectures), used for administrative purposes, or for programs which normal system users would not typically use.

epics/lib a directory containing the library (archive, or "a") files. This might actually be a symbolic link, pointing to a directory containing the appropriate libraries for a particular machine architecture.

epics/bin a directory containing the executable files which developers would typically use. The directory might actually be a symbolic link, pointing to a directory containing the appropriate executable images for a particular machine architecture.
epics/man a directory containing UNIX style manual pages, for each program or library routine (or set of related routines) in the src directories. These are meant to be concise reference pages, used by other programmers. There shall be a subdirectory for each section of the manual which has been written for this project; a subdirectory named man1 will appear for user-initiated commands. One named man3 would exist for subroutines, and so forth. See the standard UNIX manuals for a more detailed description, or look under /usr/man on the UNIX system for examples.

epics/doc a directory containing any other documentation, with appropriate suffixes indicating the tool used; a "".fm"" indicates a FrameMaker format document. User manuals for EPICS tools will be here, along with requirements documents and certain hardware documentation.

Each subdirectory under the src directory shall be ""self contained", in that it can exist independently. Containing symbolic links to files, or actual copies of files checked out for editing, each of these subdirectories shall contain a Makefile, as required by the UNIX ""make"" utility; make is a program which understands how to build software, based on the contents of the configuration file (Makefile), and is specific to a given piece of software. The Makefile itself is always stored with the source code or documentation, and is kept under version control, as described under Configuration Management on page 27.

Makefile Setup Each Makefile must be named ""Makefile", and must have the following targets and functions defined in addition to the targets for each program or library:

- all causes each program (or test program) to be compiled, or each library to be constructed.
- install causes each program to be compiled, then installed in the bin directory, or each library to be compiled and constructed, then installed in the lib directory.
- clean causes extraneous files to be removed, such as core files, temporary output or debugging message files, or test output files.
- cleanclean causes all generated (usually non-source code) files to be removed. It might also remove source code which is generated by an intermediate step in the build process. This includes all object (.o) files, lint (.ln) files, archive (.a) files and executable.
- lint causes lint to be run on ""C"" source code.
- print causes a listing of source code to be printed.
- depend causes a new Makefile to be generated, which will accommodate changes to all ""include"" files, and recompile or relink code accordingly.

10. For more information on make, refer to the Make User's Manual, as distributed with UNIX.
4.1.4. *Application Development Area*

Development of applications shall take place under the login account of individual developers. As with system programmers, routine login procedures requiring a user name and password provide access control to the developer's own work area.

The application developer's work area will be set up in the same way a system programmer's development area, as described under *System Development Area* on page 19.

One important difference is obvious when one considers the number of accelerator-specific applications required for the entire project. Specifically, a developer will *not* have an empty "skeleton" of the entire application hierarchy, in which applications are developed. Typically, a developer will have a single directory called "applications", in which current projects are contained. The complete hierarchy will exist in the SR area, in a format shown in Figure 8.

![Figure 8. Software Repository showing Structure of Application Development Area](image)

Note that each subdirectory beneath the applications level shall contain applications software for each accelerator. The generic subdirectory is an exception, and will be described below. Each accelerator directory shall have a subtree beneath it, broken down by sector, then system and so forth. In fact, there might be up to six levels in each subtree, as each accelerator is divided into smaller separate pieces. Conceptually, every level in each subtree could be named, as shown in Figure 9.

At any given level of this hierarchy, two types of subdirectories shall exist: directories for the next lower level of applications, and directories for the applications themselves, which belong at this level.

As an example, consider an emittance measurement unit (EMU) which contains a scanning wire monitor. A stepping motor is used to position the wires. Several applications could exist for this EMU.

At the lowest level of detail, an application could exist which allows a technician to test all aspects of the stepping motor as an individual piece of equipment. The files used in this application would exist at the component level of the hierarchy, with a separate subdirectory labelled doc, containing all of the relevant documentation.
Another application could exist at the directory above it, which allows integration of the motor with other pieces of equipment within the EMU. Such an application would allow an operator to interact with the EMU as a single device, and might not allow access to the motor directly. For instance, a start button would send the motor(s) through some pre-defined paces, transparent to the operator. Again, a subdirectory titled doc would contain associated information about that application.

At each higher level, applications would perform more general functions, such as changes to modes of a subsystem, or even “turn on” an accelerator to produce beam with certain characteristics.

**Multiple Instances** One important aspect to this structure is that applications, or just pieces of them, can be reused as equipment is added to the accelerators. This might mean copying existing application files into new subdirectories as equipment is added. Ideally, this would be done with a mechanism like symbolic links, so that bug fixes or enhancements would be done in a single location, and would be effective for all equipment of the same type, regardless of it’s subdirectory. Copying the files outright must still be an option, since it is likely that certain specific pieces of equipment might have slightly different applications from the others of that type. In any case, a repository shall exist for all generic component applications, so that developers can simply copy (or link) files to start their work. This repository is called generic, and is located immediately beneath the applications directory, on the same level as accelerator applications, as shown in Figure 8.

Since this structure exists below the Software Repository, it is under CM control, as described under Configuration Management on page 27. Application developers would build their development area based on this directory structure.
4.1.5. **Software Test Area**

There shall be separate test areas for EPICS system software and applications.

**Application Tests:** The first kind of testing for applications involves basic component testing. This shall be performed from within the development area of the developer's computer account, as described under Application Development Area on page 23. Accelerator specific hardware modules associated with each step of the application shall be available in a laboratory area, in which the developer shall have access to their computer account and all required files. Such testing shall be carried out by the developer.

Further testing of applications shall be performed in a separate Test Stand area, which presents an environment closer to that of an operational control system computer. The developer's own account (the application development area) shall not be accessible; such testing shall be performed after the component testing mentioned above, with access to the same accelerator specific hardware modules. This testing is performed by someone other than the developer, such as a member of the Quality Assurance organization.

Details of both procedures shall be outlined in the Acceptance Test Plan for the corresponding system or subsystem.

**System Tests:** Changes or enhancements to EPICS shall be tested in the development area, within the developer's own working account. A standard set of test applications shall be used, based on the part of the system which was changed. The programmer shall perform this preliminary test independently.

Further testing shall be carried out by someone other than the programmer, in a separate test stand area. A suite of applications which test all aspects of the EPICS software shall be used in the test, which should be attended by a member of the QA organization. At no time shall application tests be performed concurrently with the system tests, using the same system software; the test stand area shall have the system development area directly accessible to it, ensuring that any applications testing is completely separate.

4.2. **Software Documentation**

A User's Manual shall exist for each piece of software accessible to a user; basically, any program which displays information to, or accepts input from, a person shall be documented as to it's use.

A Programmer's Manual shall exist for all libraries which are accessible to programmers; each subroutine, function or method shall be described according to the format of UNIX programmer's manual pages.
4.2.1. **Software Requirements Specification**

Requirements for systems and subsystems at Level 3B and Level 4 (refer to Figure 1.) shall be documented in Software Requirement Specifications (SRS) as outlined in the Engineering Management Plan.

Software at Level 5 shall have an SRS and simple design document combined in an Engineering Notebook, which shall be produced by each developer as work proceeds. The notebook itself will be a document consisting of all of the constituent notebooks from all of the developers, gathered at regular intervals as work proceeds.

Notebooks shall exist as computer files, stored in a format selected by the developer (ascii or FrameMaker, for example), and shall adhere as much as possible to the following outline:

1. General Description
2. Requirements
   2.1. Functional
      2.1.1. Inputs
      2.1.2. Processing
      2.1.3. Outputs
   2.2. Performance
   2.3. Constraints
3. Analysis
   3.1. Design options
   3.2. Risk Assessment
4. Design
   4.1. Computer targets
   4.2. Grafset Charts
   4.3. Record Layouts
   4.4. Custom Code Layouts
5. ICD
   5.1. Data Dictionary
6. Test Plans
7. Notes

Each developer's notebook shall be considered during the informal reviews for that application as described under Application Software Reviews on page 15, or during in-process reviews for EPICS work, as described under System Software Reviews on page 16.

4.2.2. **Software Test Plans**

Control System applications are closely tied to the accelerator systems and components which they control and monitor. Accordingly, most of the applications testing shall occur within Acceptance Test Plans, which also test hardware and other system related issues, such as operating modes. These test plans and their procedures may refer back to this SDP document, making reference to basic test procedures.
Pieces of applications can be tested independently, but all applications shall go through (at least) two test stages; first, to test the application as an independent component from within the development area, and second, to test the application while other developed applications are working concurrently. This would occur in a laboratory test stand, typical of an operating control system environment without access to the development area. Testing of system software changes would be performed in a similar fashion.

Further description of basic test procedures is contained in the section entitled Software Test Area on page 25.

4.3. Project Support Functions

Support functions are carried out by members of various teams within the Controls Software Group. A notable exception is the administration of development computers for the development teams.

4.3.1. Development System Administration

Applications and EPICS system work shall be performed on engineering workstations. Development tools and other utilities shall exist on one or more central file servers, and be accessible to the development computers.

The administration of these workstations include day-to-day tasks such as making backup copies of the work, installing new or upgraded software, and maintaining peripheral devices, such as printers.

The administrative functions for the development workstations shall be carried out by a computer support group within the SSC, which is outside of the Controls Department.

4.3.2. Configuration Management

The configuration management process shall be applied to applications and system development efforts.

For system software, this includes EPICS software source code, system level (technical) documentation, utility control files (Makefiles, for instance), and other documents which the developer deems relevant to the system.

With respect to applications, CM shall apply to EPICS record (database) files, screen layout files, SNL files; basically all files which go into making an application work. Additionally, all technical documents shall be under CM control, as well as SRS documents, such as the files which constitutes the developer's notebook

**Procedure:** The development process, which includes CM activities, is described under Process Model on page 9. The structure of the CM controlled area is described under Software Repository on page 18.

The CM process includes a Configuration Administrator (CA) who shall report to the Software Group Leader. The CA shall maintain a list of programmers, both internal and external to the Software Group, who are allowed to check out copies of files. Separate lists may exist for different files. The commands which will be used to actually retrieve the copies determine if those copies are meant to be edited or not: although copies can be checked out, and the system tracks all actions, only the CA can check the files back into the system and build another release. Layout of these files is discussed below, and in UNIX reference manuals.

**Version Control:** The software used to manage version control shall be based on the Source Code Control System (SCCS), as distributed with the UNIX operating system. Both systems and applications files shall come under SCCS control.

When a system programmer creates a development area (see page 19), he is actually creating links to the files, so they appear to exist in his own development area. This structure is described in more detail under Software Repository on page 18, but basically each development area shall have a subdirectory to it, called SCCS.

Each of the SCCS subdirectories is actually a symbolic link to a corresponding directory entry in a central EPICS location. This allows administrative information to be kept with the master versions of the software, without appearing in the actual project directories. This will include such things as a file containing a list of developers to whom electronic mail will be sent after versions are changed and properly remade. The setup and maintenance of these links and directories is done by a set of programs used by the developers and project administrators.

An example is shown in Figure 10. The Alarm Handler (alh) source code appears to exist in the development area, but is actually just a link to the original unchangeable file. If the developer needs to make changes to that file, the file is actually checked out of SCCS and copied to the development area, where changes are made.

This structure also allows separate backup copies of the current versions and the master code to be kept, and gives a common area for statistics to be collected. It also allows a separate area to be maintained for system distribution, in cases when a set of distribution tapes needs to be built.

---

12. Programmers may exist within other Departments at the SSC, but shall still be responsible for reviews and documentation, as if they were in the Group.
This directory structure also allows developers to maintain a private image of the source code, common definition files and documentation in a local area. Only files which need to be edited will be copied out of the central project directories, and all other files shall be symbolic links.

This mechanism also accommodates compilation of code for different machine architectures. Each such directory need only contain symbolic links back to the appropriate source code, but the object files (".o") and resulting executable files will exist locally.

The directory layout for application code is very similar to the system software layout, although the actual directory structure is more complex because of the numerous applications that will exist. The directory structure is outlined under Application Development Area on page 23.

**Change Requests:** Each extension or enhancement to control system software, whether relating to a problem or an enhancement, shall be described in a System Modification Request form, describing the reason for the work, who is requesting it, and other information. A copy of the SMR form appears under the Appendix System Modification Request Form on page 30.

**Request Tracking:** Records of all SMR forms shall be kept electronically, in a database. Requests shall be submitted electronically or on paper, but will ultimately be entered into the database. Software shall track the progress of the request, and ensure that nothing "falls between the cracks".

The procedure starts when the form is submitted. All SMR forms dealing with issues relating to accelerator controls shall be submitted to the Controls Department.
The SGL shall review each request, even if generated from within the Controls Department, and assign it to an engineering team. There shall be regular weekly reviews of all outstanding SMR requests. The SMR shall be used for hardware, communication and even procedural issues in the control system, as well as software.

An analyst is assigned to the request, and shall suggest workaround solutions to the requestor if warranted, mark a priority for this problem or change, based on their experience with that particular component, and recommend that a particular action be taken.

An engineer may then be assigned to resolve the problem. The original requestor of the change shall be notified when the issues have been resolved, and the change has been made, or if an analyst has decided against making the modification. It is quite possible that the analyst and engineer assigned to resolve the issues are the same person.

5. Work Packages, Schedule and Budget

Schedules and budget details will be done in accordance with the SSC Project Management Cost and Schedule System, and the SSC Integrated Project Schedules.

Internal schedules shall be maintained, which contain details of team goals. Most schedule items dealing with applications shall be tied to specific accelerator milestones. Schedule items dealing with EPICS system related work shall be tied to the applications milestones when appropriate.

Project schedules shall be maintained within the Controls Department using Autoplan software. Team leaders will have access to their own schedules, and all teams members will have the ability to peruse the schedule database.

Appendix A System Modification Request Form

The following pages depict information on the front and back side of an SMR form at the SSC. The front page is filled out by the requestor, while the back side is filled out as the modification is made, or the problem resolved.

13. Recall that the team might actually be in a different department.
## SYSTEM MODIFICATION REQUEST

Please submit as much information as possible, but do not fill in the other side of this page.

<table>
<thead>
<tr>
<th>Modification Name:</th>
<th>Please enter a concise name to identify the modification</th>
<th>Date:</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>SMR Number:</th>
<th>(Filled in by Controls Dept.)</th>
</tr>
</thead>
</table>

### Is modification related to:
- [ ] Hardware
- [ ] System
- [ ] Documentation
- [ ] Software
- [ ] Uncertain

### Type of Change:
- [ ] Improvement
- [ ] Resolution of Problem (discrepancy/correction)
- [ ] Enhancement (new requirement)

<table>
<thead>
<tr>
<th>Requestor:</th>
<th>Phone:</th>
<th>E-mail:</th>
<th>Division/Department:</th>
</tr>
</thead>
</table>

### System:
- LINAC
- LEB
- MEB
- HEB
- Collider
- ASST
- Other

### Location:
Include Sub-system name, Room or area name, rack or console number.

### Component Name:
Command, program name, hardware module. Include version number if applicable.

### Priority:
- [ ] Low
- [ ] High
- [ ] Critical

### Signature:

### Symptoms:
- [ ] documentation missing
- [ ] documentation wrong
- [ ] system crashes
- [ ] data was lost
- [ ] data was incorrect
- [ ] incorrect behavior
- [ ] unfriendly behavior
- [ ] inconsistent behavior
- [ ] message unclear or wrong
- [ ] display is incorrect
- [ ] operation never returns
- [ ] calculation is wrong
- [ ] occurs randomly
- [ ] occurs in certain operating modes
- [ ] system is unresponsive

### Impact:
Describe activities which this problem affects if not corrected.

### Describe options or workaround solutions:
Is there any way to continue until the modification is made?

### Suggested Action:
Can you suggest a solution?
**SSC Laboratory**  
**ACCELERATOR CONTROL SYSTEM**  
**SYSTEM MODIFICATION REQUEST**

*Please fill out other side of this form first*

---

**Analysis**

<table>
<thead>
<tr>
<th>Analyzed by:</th>
<th>Phone:</th>
<th>E-mail:</th>
<th>Department:</th>
<th>Date Received:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Response Priority:**

- Resolve IMMEDIATELY
- Important; put at head of queue
- Place in normal queue
- Low priority; use workaround
- Lowest priority; last thing to fix
- Will not be resolved

- [ ] Recommended Action:

**If possible, give an estimate of number of days required for this modification:**

---

**Resolution**

<table>
<thead>
<tr>
<th>Resolved by:</th>
<th>Phone:</th>
<th>E-mail:</th>
<th>Department:</th>
<th>Date Received:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Briefly describe the action taken:**

- Code change required
- Documentation change required
- Design change required
- No change made (give reasons)
- Duplicate SMR
- Specification change
- User mistake
- Not reproducible

---

**Date Modified:** | **Date Tested:** | **Date Requestor Notified:**
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

SMR Version 1.1-05-05-93

*Work Packages, Schedule and Budget* - 32 -
APPENDIX

Three Level Requirements Specifications
and
Software Notebooks

1.0 Introduction

ASD requires a procedure for documenting software development which will serve the needs of software developers, their managers, and those charged with oversight of the software development process, in this case the ASD QA group. The goal here is to suggest a minimalist approach, having as few documents as possible, attempting to eliminate redundancy, and emphasizing flexibility in the face of evolving requirements. The brief discussion below applies to the software developments of the Controls Department, but should be applicable to the work of other departments as well.

The approach used and the nature of the documents required will relate to the nature of the project itself, and to the level of the specification with which it is associated.

2.0 Software Development Plan.

There is a need for only two software development plans (or one plan with two parts) which should cover all of the software to be developed in the Controls Department:

- The Plan for development of application software under EPICS ("Using EPICS"), and;
- The Plan for developing modifications and enhancements to EPICS ("Changing EPICS").

To first order, all of the software described below falls in one of these two categories, and should be able to refer to the appropriate SDP. Of course, if cases arise where a different or modified plan is required, such a plan would be developed for that project.

3.0 Level 3. PMO Specifications

In accordance with the revised specification tree attached, there will be a Global Accelerator Systems Specification, at Level 3A, and a Global Accelerator Control System Specification at Level 3B. Both of these specifications and their associated reviews are the responsibility of PMO. The Controls Department will be consulted with respect to the GACS 3B spec. The Controls Department may be asked to contribute presentations at the Level 3 reviews where appropriate, but will not be required to submit documentation other than copies of transparencies.

The GACS 3B specification will call out the specifications and reviews to be held at Level 4, which are described below.
4.0 Level 4: ASD Specifications

Just as in the case of the individual Machine Specification Trees, control system specifications and reviews at Level 4 are the responsibility of the ASD Controls Department. As indicated on the accompanying specification tree, there will be two global reviews and specification documents, as well as one specification and series of reviews for each Machine. The two Level 4 Global specifications relate to Process Control and to Beam Control. The machine specifications and reviews trace down from the Machine Level 3 specifications as well as from the Global Level 3B specifications.

4.1 Global Specifications

The Global Level 4 Beam Control Specification will be made up of three parts: a System Specification, a System Software SRS and a Global Application Software SRS. (These will be derived from the already existing draft specification prepared by Rolf and heretofore thought of as a 3B specification).

4.1.1 Global System Specification

The Global System Specification will include specification for all the major system hardware components. This includes:

- Front End Systems and Standards
- Global Interface Definition for
  - Magnet Systems
    - Ramp Magnets
    - Correction Magnets
    - Pulsed Magnets
    - DC Magnets
  - RF Systems
  - BI systems
    - BPMs
    - BLMs
    - others
  - Abort System
  - Timing System
- Communications
- Central and Sector Head facilities
- Control Room
- Global Architecture

At this level, only globally estimated data rates and volumes should be specified. The main requirement at this level is to build in sufficient flexibility and capacity to meet specific requirements as they grow and change.

There is no software component to this specification.
4.1.2 Global Application Software SRS

The Global Application Software SRS will describe those applications which apply to all machines, and which are (or should be) inherent to EPICS. These include:

- Remote Control and Display
- Alarm Manager
- Save and Restore
- Archiver
- Sequencer

We should not write more than we have to. For example, because EPICS already implements many of these applications, references can be made to the EPICS documentation where appropriate and adequate, with emphasis being placed on requirements not met by EPICS, and requirements for necessary changes or additions to EPICS. Necessary changes to any of these global applications will be defined as system software projects, listed in this specification, and supported by a software notebook as described below.

4.1.3 Global System Software SRS

The Global System Software SRS will describe all those changes, improvements, etc to EPICS which are required to adapt EPICS to the SSC environment.

This will include, for instance:

- Nameserver software, name caching software
- Changes to adapt EPICS to T1 communications
- Changes to the API, changes to Channel Access
- MBS (Message Broadcast) and timing support software

The Global System Software SRS will also include off line tools such as for database editing and graphical programming.

All of these projects should be identified and listed in the Level 4 Global System Software SRS, and each of them will be supported by a notebook at Level 5 (See below).

4.2 Machine Specifications

In addition to the Global specifications, there will be a Level 4 specification for each Machine: LINAC (probably too late for this one), LEB, MEB, HEB, Collider, Test Beams. Each of these will include the relevant transfer line (whatever that may be at the time).

In general, the machine specifications will refer to the global system and software specifications. The implementation of the global system specification specific to the individual machine will be described. Here the detailed placement of crates and racks and the identification of their contents will be given. No mention of global architecture need be given, except to identify the interface, and any differences or special features that are required for the particular machine.
In general, no SRS will be required for an individual machine at Level 4. For the
global applications, reference should be made to the global application software SRS.
For example, if the generic archiver specified in the global SRS will do just fine for the
MEB, then it will suffice to say so. However for each machine specifics, such as (for the
archiver) numbers and rates of points to be archived, will be given.

In addition, for each machine a number of machine specific applications will be
required. These will include commissioning programs (such as the 24 which have been
identified for the LEB by Wienands, Bourianoff and Bork) and simpler screens for
display and control of equipment, subsystems, etc. Each of these applications should be
listed in the Level 4 Machine Specification. This list will correspond one-to-one with the
software notebooks to be developed at Level 5, and described below.

5.0 Level 5.

Software Notebooks

This is the level where actual software modules or specific applications will be
defined. Here a "living" software project notebook (or folder) will be maintained. The
model will be the CEBAF approach. For example, the front page is very general - "write
software support routines for the CCL vacuum system." The notebook keeps up a
traceable record as the specifics are discussed, agreed, and modified. It is this notebook
that would be reviewable by QA.

The notebook approach will also be applied for modifications or enhancements to
EPICS. Examples might be "modify Channel Access for Nameserver and cacheing," or
"add graphical database editor," etc.

The notebook will contain at least the following:

• Title Page. Brief (one or two lines) description of the project, as described
  above. Each notebook should refer back to the level 4 SRS in which it is called out.
• Table of Contents. Most notebooks would include most or all of the sections
  listed below.
• Sign-off page. Indicates who is required to sign off on what parts of the
  notebooks, eg who must sign off on the requirements (normally customer, developer,
  developer's supervisor), who on the ICD section, who on the test plans, etc. This page
  should be audited for appropriateness of sign-off authorities identified.)
• SDP Section. Normally a line will be sufficient, referencing the applicable SDP.
  Otherwise any special development practices will be recorded here.
• Requirements Section. Here the requirements are listed in as much detail as
  possible. This section will be recognized as the required SRS. Requirements and
  changes to them during the development and commissioning cycle are signed off as
  required on the sign-off page. One acceptable way of presenting requirements would be
to include sketches of operator screens where appropriate. Requirements could be
  audited for ambiguity or incompleteness or inconsistencies.
• Test Plan Section. As the requirements evolve, a test plan will be developed in
  this section which tests each of the specified requirements. For small systems, a system
test (hardware and software together) will be appropriate. Where suitable, separate
software test plans may be required. Test plans would be audited for traceability to
requirements.
• ICD Section. Here is where one would keep signal lists, interface specs, etc. Reference could be made to applicable hardware module documentation where appropriate. ICDs would be audited for completeness.

• Configuration Section. Here would be kept records of all files, where they are, etc. Configuration Management of these files would be by SCCS or equivalent, as called out in the relevant SDP. This section would be audited for conformance to the SDP software configuration plan.

• Design notes. Here would be kept notes on the evolving design, listings, charts, etc. Auditable for completeness.

• Developer Log. Here the developer would keep a personal record of activities, progress, problems, brilliant ideas, attaboys and ahshits. Normally this would be the only non-auditable part of the notebook.

The notebook can be kept in whatever way seems most convenient to the developer, including electronically. However a paper version must eventually exist so that it can be signed-off as required. I expect a typical quality audit would involve checking the first page to see who is supposed to sign what, and then verifying that the correct signatures have been obtained. Each page or section of a completed notebook would require a QA signature. I imagine the paper version of the notebook being kept in a three ring binder.