The Software Life Cycle Support Environment (SLCSE) A Computer Based Framework for Developing Software Systems

Tom Strelich

General Research Corporation
5383 Hollister Avenue
P.O. Box 6770
Santa Barbara, CA 93111

1 Abstract

The Software Life Cycle Support Environment (SLCSE) is a VAX/VMS-based software development environment framework which presents a common and consistent user interface accessing a comprehensive set of software development tools supporting the full spectrum of DOD-STD-2167A software life cycle activities from Requirements Analysis to Maintenance. These tools utilize a Project Database which maintains information relevant not only to the software under development (e.g., requirements allocation, software interfaces, etc.), but also information relating to the project as a whole (e.g., schedules, milestones, Quality Assurance, Configuration Management, etc.). The Project Database supports the DOD-STD-2167A life cycle model and associated Data Item Descriptions (DIDs). SLCSE's framework approach supports the integration of new tools into the environment and permits the SLCSE to evolve over time and adapt to advances in software engineering technology.

2 Background

Software development environments (SDEs) have been categorized into a taxonomy[1] consisting of four types:

- Language-centered environments provide integrated tools supporting a specific language (e.g., Rational[2] for Ada)
- Structure-oriented environments support direct manipulation of program structures in a language-independent manner and different views of program structures at different levels of abstraction (e.g., the Gandalf[3] Environment)
- Toolkit environments consist of a collection of smaller tools which may or may not share information with one another (e.g., Unix Programmer's Work Bench[4])
- Method-based environments support specific software development methodologies via computer-aided software engineering tools (e.g., Excelerator[5])

The SLCSE* constitutes a fifth type of environment, an environment framework that can create a variety of environments, each tailored to the needs of a particular software development project. Through its framework, the SLCSE supports many of the capabilities provided by the four previously described environment types:

SLCSE as a Language-centered Environment. While not a Language-centered environment†, the SLCSE satisfies both language-specific and lifecycle-specific environment requirements through its Ada, JOVIAL J73, FORTRAN, and COBOL tools and its DOD-STD-2167A documentation support.

SLCSE as a Structure-oriented Environment. The SLCSE provides language-independent access to a comprehensive data model supporting the DOD-STD-2167A lifecycle and Data Item Descriptions at various levels of abstraction.

SLCSE as a Tool-kit Environment. The SLCSE provides an extensive set of tools supporting the complete spectrum of software life cycle activities and provides the mechanisms allowing these tools to communicate and share data with one another (i.e., a repository mechanism).

SLCSE as a Method-based Environment. The SLCSE supports definition and application of software development methodologies to a particular project.

The environment framework is a new concept that, at the time of this writing, is found only in the SLCSE, Macintosh Hypercard., and Atherton Technology's Software BackPlane™.

The following sections describe the SLCSE operational concept, its main features, and a technical description.

* This work was performed under Rome Air Development Center Contract F30602-86-0206

† An important SLCSE requirement is to support other DOD approved programming languages, not just Ada.

Recommended by: *Donald J. Reifer*

3 SLCSE Operational Concept

Figure 1 illustrates the SLCSE operational concept. The following paragraphs describe this figure and discuss its implications.

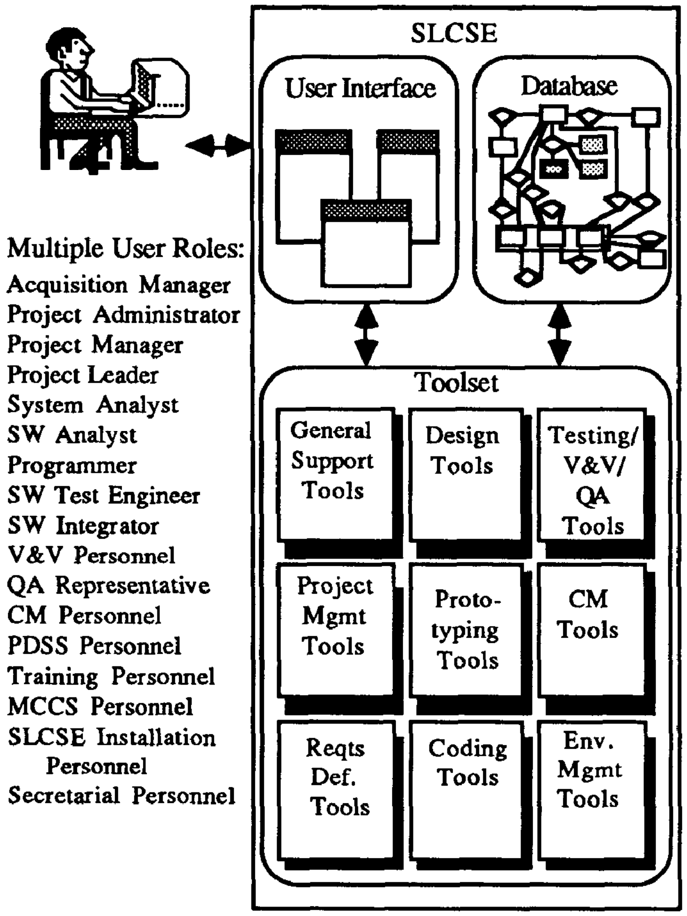


Figure 1. SLCSE Operational Concept.

The User. The SLCSE operational concept starts with the user and recognizes the fact that, for any software development project,a user sits down at a terminal with a job to do. The activities associated with a job, or task, (e.g., requirements definition, software design, programming, etc.) define the user's role. A user's role can change over the duration of a project (e.g., changing from a Designer to a Programmer role as Critical Design Review is completed and the Implementation phase begins). In some instances, a user may also play two or more roles simultaneously (e.g., a Project Leader that also performs Requirements Analysis). The user roles supported by the SLCSE are shown in Figure 1 and are derived from the STARS Operational Concept Document* .The user accesses software tools (applicable to their role) through the window-based interactive user interface which provides common and consistent operation for all users in all roles (i.e., screen layout, keywords, keypad bindings, etc. are consistent from tool to tool and role to role).

Tools. The software tools associated with a given role are accessed through the user interface. As shown in Figure 1, these tools are grouped into 9 broad categories based on the activities they support:

- General Support Tools address activities that span all roles such as text editing, document formatting and printing, electronic mail, etc.
- Project Management Tools support project planning, tracking, and reporting
- Requirements Definition Tools support system and software requirements definition, modification, display reporting and consistency checking
- Design Tools support allocation of requirements to design objects (i.e., CSCI, CSCs, and CSUs) and the description of interfaces, data, and processing algorithms
- Prototyping Tools support the rapid construction of Interactive User Interfaces (e.g., window and menu prototype generators) which can be used not only to evaluate the look and feel of a developing software system, but also to serve as the actual User Interface in the final software product
- Coding Tools support creation and modification of the compilable source code implementing the software design (e.g., language sensitive editors, compilers, linkers, and debuggers) Certain aspects of SPR processing/tracking are under the domain of Project management and Configuration Management.
- Testing/V&V/QA Tools support test case definition, organization, and evaluation (e.g., program execution coverage analysis and reporting tools); Verification and Validation (V&V) tools support static analysis of software, collection and evaluation of software quality assurance (QA) metrics, programming standards, consistency, traceability, change impact analysis, and Software Problem Reports (SPRs) processing/tracking*.
- Configuration Management Tools support definition, modification, control, and analysis of a software configuration
- Environment Management Tools support SLCSE environment definition, modification, tailoring (e.g., defining users, roles, tools, etc.), and performance evaluation for a specific site.

* STARS - Software Engineering Environment (SEE) Operational Concept Document (OCD), Proposed Version 001.0. STARS Joint Program Office, October 2, 1985.

*Certain aspects of SPR processing/tracking are under the domain of Project management and Configuration Management.

Database. The SLCSE provides an Entity/Relationship (ER) database (DB) which models the DOD-STD-2167A Life Cycle. It serves as both a repository for system software and project information and as a medium for inter-tool information exchange. Through its organization and access mechanisms the SLCSE DB presents the user with clear and consistent views of the activities and products associated with (but not limited to) the DOD-STD-2167A Life Cycle. The SLCSE database supports the automated generation of DOD-STD-2167A documents and makes software-specific and project-specific information available to tools within the SLCSE environment (e.g., software metric information for quality metric tools, schedule information for project management tools, etc.).

4 SLCSE Features

Multiple User Roles. The SLCSE allows users to assume one or more roles over the lifetime of a project. Upon assuming a role, a user is given access to a set of role-specific tools (i.e., tools applicable to the activities associated with the role). For example, the Project Management Tools associated with Project Management activities would include Spreadsheet Tools for establishing and tracking the project budget, Scheduling Tools for analyzing schedules, milestones, critical paths, etc., and general purpose tools for word-processing etc.

For some smaller projects, selected users may be authorized (by the Project Manager) to play several roles simultaneously†. Changing roles results in changing toolsets, i.e., when a user assumes a new role, a new set of role-specific tools replace the previous toolset. By providing these toolset associations, the SLCSE ensures that users explicitly change roles (i.e., hats) before performing another role's activities. This has several important implications:

- It requires users to recognize the distinctions between software development phases and activities (i.e., programming vs design vs management, etc.)
- It requires users to acknowledge their current role and its associated activities
- It establishes a work context, or frame of reference, for the role and its activities, and the tools available to perform those activities
- It discourages the ad hoc, undisciplined use of tools and the violation of role/activity boundaries

†Generally, smaller projects find single users assuming multiple roles over the duration of an effort (e.g., manager-analyzer, programmer-integrator, etc) in contrast to larger projects which generally find users assuming a single role throughout the development effort (e.g, manager, designer, programmer, etc.).

- It supports the application of a more structured, disciplined approach to the software development project, with the tools and their order of use defining the development methodology

These role-specific toolsets are tailorable and can be modified (by project management) to accommodate the needs of different software development projects. For example, tools can be deleted from a toolset, moved to another role's toolset, and new tools integrated in order to support a given project's development methodology.

Multiple Environments. Since many (if not most) software projects are developed on computer networks, the SLCSE supports definition of project environments over a network of computing resources (or a network subset).

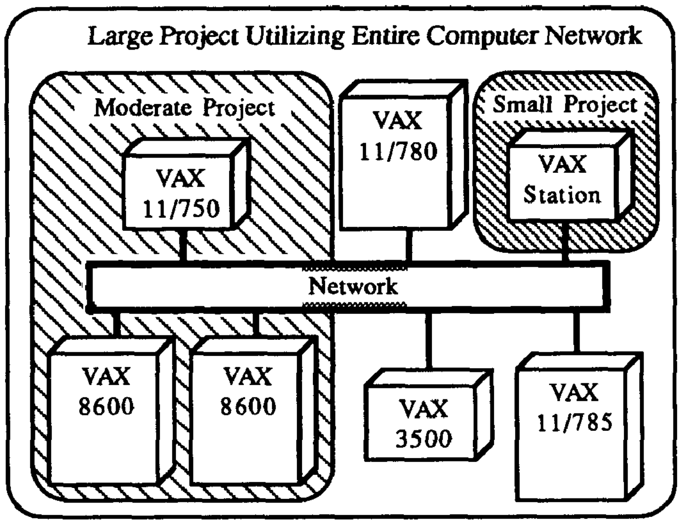


Figure 2. Support for Multiple Environments.

As suggested by the example scenarios shown in Figure 2, a Small Project has been defined for a network subset consisting of a single VAXStation (such a project environment would be suited to a development effort requiring from 2 to 8 team members), a moderate project has been defined for a network subset consisting of two VAX 11/780's and an 11/750, and a Large Project has been defined utilizing the entire network. It is important to note that all three Project Environments shown in the example scenario (Small, Moderate, and Large) can exist simultaneously on the network.

Multiple Languages. SLCSE is a multi-language environment. It supports several commonly used DOD approved programming languages: Ada , JOVIAL J73 , FORTRAN, and COBOL.

The Ada and JOVIAL J73 (and in some instances FORTRAN) languages are commonly used for embedded Mission Critical Computer System (MCCS) applications.

Additional languages can be incorporated into the SLCSE by integration of the appropriate compiler, linker, Language-Sensitive Editor (when available), and supporting lifecycle tools (e.g., language-based PDLs, object-oriented design tools, graphic design tools etc.).

Common User Interface. SLCSE provides an interactive user interface supporting windowing capabilities on DEC VT100-type terminals as shown in Figures 3 for a user in a "Programmer" role (indicated in the upper right hand corner of the screen). Since VT100s don't support "mouse" input, the Keypad is used to provide equivalent screen navigation and selection capabilities as shown in Figure 4. Screen layout and navigation, keyword syntax and semantics, keypad bindings, etc. are consistent from tool to tool (with certain limitations) and role to role. In this way the SLCSE user interface provides common and consistent operation regardless of the user's role or the current tool in use. The SLCSE user interface permits both Menu and Keyword operation supporting novices (menu mode) to experts (keyword mode) and multi-level online help facilities.

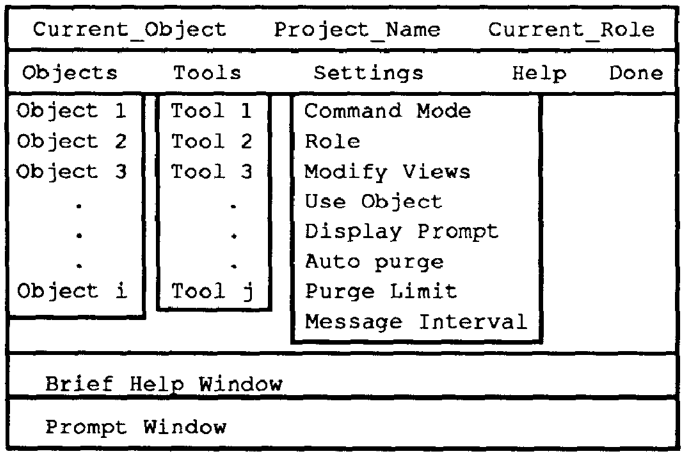
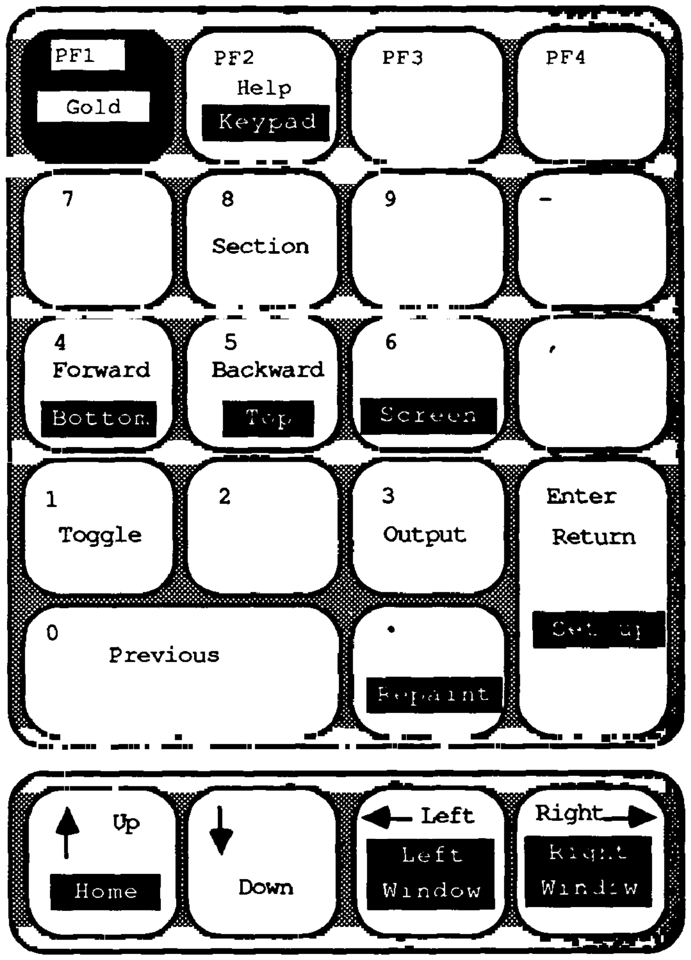


Figure 3. SLCSE User Interface

Knowledge-Based Methodology Support. SLCSE establishes a preliminary framework for application of Knowledge-Based (KB) techniques to the software development process. The SLCSE allows project managers to define sets of rules describing the methods and procedures to be applied to the development effort (e.g., document consistency rules, etc.). Facts about user activities (i.e., tool operation, database access, etc.) are monitored during SLCSE operation and are applied to these rule sets, and when a rule is violated, the user is informed. Knowledge-Based methodology support provides incremental control over user activities (i.e., allowing a user to invoke a tool or exercise some other SLCSE capability only after first verifying that the methodology is being followed). SLCSE provides the basic framework for this capability, with only a preliminary rule base defined; however, in the future this rule base is expected to grow in size and complexity as heuristics about the software development process are formalized.



GOLD - used in combination with other keys for alternate command request (e.g., Gold PF2 = Keypad).
BACKWARD - set direction to Backward for Section Key.
BOTTOM - go to the last option in the window.
FORWARD - set direction to Forward for Section Key.
HELP - brief help on the item highlighted by the cursor.
HOME - go to the command button line.
KEYPAD - view SLCSE keypad.
LEFT WINDOW - move to the window on the left.
OUTPUT - view output of tool invocation.
PREVIOUS - go back to previous window.
REPAINT - repaint the screen.
RETURN - enter data/select option.
RIGHT WINDOW - move to the window on the right.
SCREEN - write screen image to a file.
SECTION - move cursor one window section in the direction specified by the Forward or Backward key.
SET-UP - define default options for an object, tool, or procedure.
TAB KEY - same function as Previous key.
TOGGLE - toggle enumeration set.
TOP - go to the first option in the window.

Figure 4. SLCSE Keypad

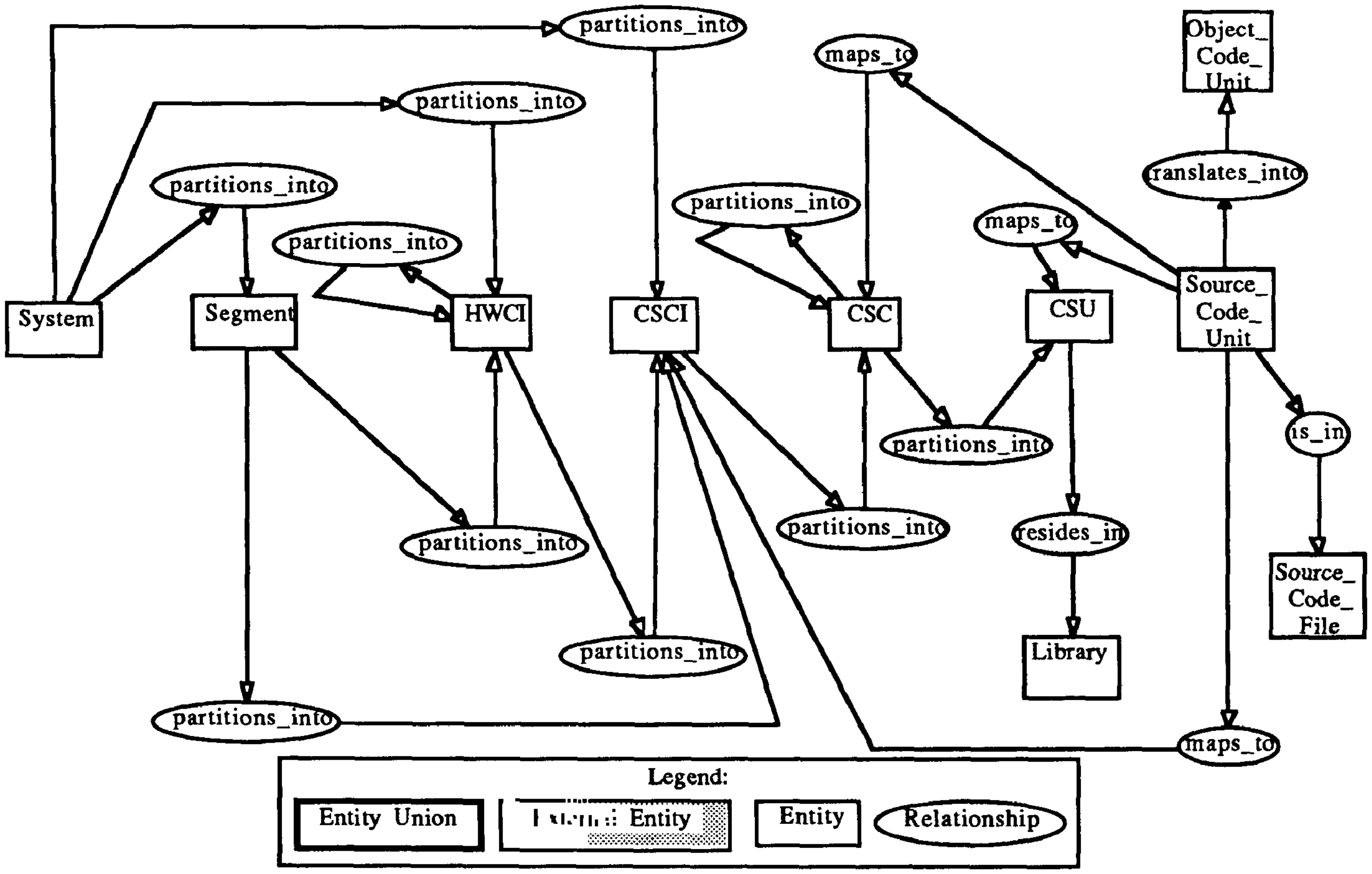


Figure 5. Design Subschema, Vertical Relationships View.

<u>Project Database</u>. The SLCSE DB serves as both an information repository and as a medium for inter-tool information exchange. The database supports the DOD-STD-2167A System Software Life Cycle model* and its associated documentation standards. It is implemented as an Entity-Relationship model and is logically partitioned into 9 subschemas:

- The Contract Subschema
- The System Requirements Subschema
- The Software Requirements Subschema
- The Design Subschema
- The Test Subschema
- The Project Management Subschema
- The Configuration Management Subschema
- The Product Evaluation Subschema
- The Environment Subschema

A portion of the Design Subschema is illustrated in Figure 5 and shows a subset of the relationships existing between Design subschema entities. The figure represents only a small fraction of the complete SLCSE database which consists of approximately 200 entity types and 300 relationships. Subschema entities, relationships, and attributes are accessible to tools within the SLCSE Toolset. These integrated tools are used both to populate the database and to process the information it contains: for example, the Design Tool* is used to populate the Design Subschema shown in Figure 5, and the Document Generator is used to process the data contained in the Design Subschema and produce the Software Design Document.

End-user acceptance of any software tool or environment is dependent on timely response to user interactions -- if the environment is slow, it won't be used regardless of its functionality. For maximum performance, the SLCSE Database is constructed on top of a commercial relational database: SMARTSTAR/IDM is a hardware implementation of a commercial relational database using the Britton-Lee IDM-500 Database Machine as the underlying database engine. SMARTSTAR/RDB is a look-alike product that provides a software implementation using DEC RDB as its underlying database engine. The SLCSE database can use either of these underlying implementations depending on the needs of a particular project and the availability of the database hardware.

* But not to the exclusion of other life cycle models.

* Both the Design Tool and Document Generator are described in the next section.

Extensibility. The SLCSE provides an extensible software development environment framework. This extensibility is achieved through:

- Customization of the User Interface -- The SLCSE User Interface can be tailored for individual users by resizing windows, changing menu item order, or redefining the terminal keypad.

- Customization of the Database -- The SLCSE Database, which supports the DOD-STD-2167A life cycle model as a default, can be extended through the modification of existing subschemas and through the definition of additional database subschemas (i.e., entities and relationships) to support project-specific life cycle models, methodologies, documentation standards, and tools.

- Integration of existing and new software tools -- The SLCSE Toolset can be extended to include existing off-the-shelf tools (such as compilers, simulators, etc.), tool upgrades (i.e., new versions of existing integrated tools), and new tools (e.g., Ada tools, Software Quality Metric tools, etc.).

5 Technical Description

The following sections provide technical overviews, address tool integration issues, and discuss future directions for each of the three major SLCSE subsystems: The User Interface, Database, and Toolset.

5.1 User Interface

Technical Overview. The interactive user interface is implemented using two GRC-developed products:

- Winnie, a window-management package providing window definition and display capabilities for character-oriented terminals (i.e., VT100s and compatibles)
- MOO (Menu Operations Organizer) which provides window control capabilities (i.e., it provides the mechanisms and utilities for defining the menu structure and subsequent run-time traversal of the menu structure based on user inputs)

A primary design objective of the User Interface was to provide window-oriented capabilities without creating special hardware requirements. While bit-mapped displays with mice would have been desirable, not everyone has them, and, at the time the SLCSE was being designed, bit-mapped workstations had not proliferated to the extent they are today. The decision to use Winnie and MOO to provide the window-oriented user interface on character-oriented terminals has three advantages:

- It provides a full-blown window-oriented user interface including pull-down menus, pop-up menus, horizontal and vertical scrolling, window tiling and overlapping, window repositioning and resizing, item selection and toggling, etc.
- It allows the customer (the Air Force) to exploit the window-oriented user interface with existing terminal hardware
- It provides an open evolutionary path to the eventual use of bit-mapped displays since both Winnie and MOO are accessed via Ada packages whose low-level character-oriented screen manipulation mechanisms are hidden from the calling programs and can be modified to use low-level bit-mapped display screen manipulation mechanisms (e.g., X-Windows)

Tool Integration Issues. In order for the SLCSE to provide a consistent user interface, there must be visual and operational consistency between the SLCSE and the integrated tools that it provides. The Macintosh is an excellent example of a consistent environment/tool interface, where both the operating system and applications have visual and operational consistency (i.e., the user interacts with the operating system the same way they interact with the application programs).

In designing the SLCSE, it was important to provide mechanisms supporting construction of new tools that are Conformant with the SLCSE User Interface (i.e., screen format, menu bar position, navigation, selection, etc.). It was equally important to allow existing tools (i.e., Commercial-Off-The-Shelf and Government Furnished Equipment) with their own user interfaces to be integrated into the SLCSE. These two competing design requirements were reconciled as follows:

- For New Tools -- Winnie and MOO are available to tool developers for the construction of user interfaces. In this way, new tool user interfaces can be implemented using the same interactive screen management utilities used for the SLCSE User Interface, thus supporting visual and operational consistency between the SLCSE and its tools.
- For Existing Tools -- A Setup Window capability is provided with the SLCSE which allows tool integrators to define tool invocation windows containing tool execution parameters (e.g., input/output files, runtime qualifiers, command options, flags, etc.). These tools can then be invoked (as batch jobs) from their Setup Windows. However, Setup Windows provide only a partial solution since some existing tools are inherently interactive (e.g., EDT) and there is simply no way to hide their Non-conformant User Interfaces

Developmental Software tools (i.e., software tools developed under the SLCSE project) are, by definition, Conformant. Non-developmental Software tools (i.e., COTS and GFE software) are generally Non-conformant since they provide their own user interfaces. However, when a tool's functionality can be fully exercised through its Setup Windows (in effect hiding the tool's user interface from the user) then it can still be considered Conformant.

Future Directions. Winnie and MOO are currently being modified to support bit-mapped displays. This work is being performed in conjunction with a rehosting of SLCSE to SUN and Apollo systems.

5.2 Database

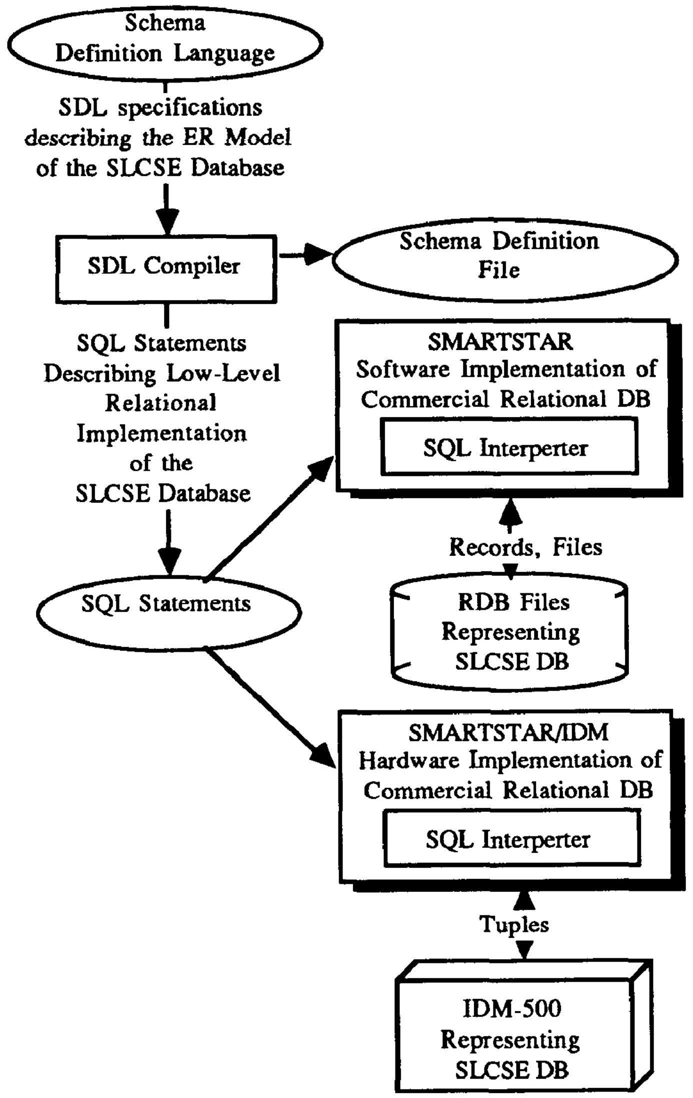
Technical Overview. In addition to the performance issue, two additional reasons drove the design decision to construct the SLCSE database on top of SMARTSTAR:

- A commercial relational database management system would automatically provide many features required by the SLCSE database such as transaction processing and archiving
- Using SQL (Structured Query Language) as the underlying query language for the SLCSE database would make the SLCSE database portable to any host that supported Ada and SQL

Recognizing that the database subschemas will change over time (in response to project-specific needs, revision of DOD-STD-2167A, integration of new tools with unique data representation requirements, etc.), a Schema Definition Language (SDL) was defined. SDL is used to describe the ER model of the SLCSE Project Database. SDL is an Ada-like language which allows subschemas, entity types, relationships, and attributes to be defined. It also supports predefined and user-defined attribute types. The SDL Compiler translates formally specified subschemas into Structured Query Language (SQL) statements, which are, in turn processed by SMARTSTAR.

The SLCSE Database has two distinct phases: 1) creation, and 2) access. Figure 6 illustrates database creation: SDL is processed by the SDL Compiler to generate a set of SQL statements that (when interpreted by SMARTSTAR) create the relational database tables implementing the ER model of the SLCSE Database. The SDL compiler also generates the Schema Definition File (SDF), a runtime symbol table which is used by the ERIF (described in the next section) to allow directly and indirectly-coupled tools to reference database entities, relationships, and attributes by name.

Figure 7 illustrates database access: Directly-coupled tools access SLCSE database entities, relationships, and attributes via the ERIF which interprets tool directives, translates directives into SQL statements, forwards the SQL to SMARTSTAR, and processes and returns the results to the calling tool. Indirectly-coupled tools must utilize the DCL Interface in order to access the database. The DCL Interface processes intermediate files containing database information in a standardized format which is translated into appropriate ERIF operations



Schema Definition Language
SDL specifications describing the ER Model of the SLCSE Database
SDL Compiler
Schema Definition File
SQL Statements Describing Low-Level Relational Implementation of the SLCSE Database
SMARTSTAR Software Implementation of Commercial Relational DB
SQL Interperter
Records, Files
SQL Statements
RDB Files Representing SLCSE DB
SMARTSTAR/IDM Hardware Implementation of Commercial Relational DB
SQL Interperter
Tuples
IDM-500 Representing SLCSE DB

Figure 6. SLCSE Database Creation.

Tool Integration Issues. The SLCSE provides two interfaces to its database one for new tools, and one for existing tools:

- ERIF (Entity-Relationship Interface) - a set of Ada packages providing direct access to subschema entities, relationships, and attributes from Ada applications (i.e., tools). Tools using this interface are termed, Directly-coupled.
- DCL Interface - a set of database access utilities callable from the DCL (Digital Command Language) level. These utilities are stand-alone executable images that, in turn, use the ERIF to manipulate the database). Tools using this interface are termed, Indirectly-coupled.

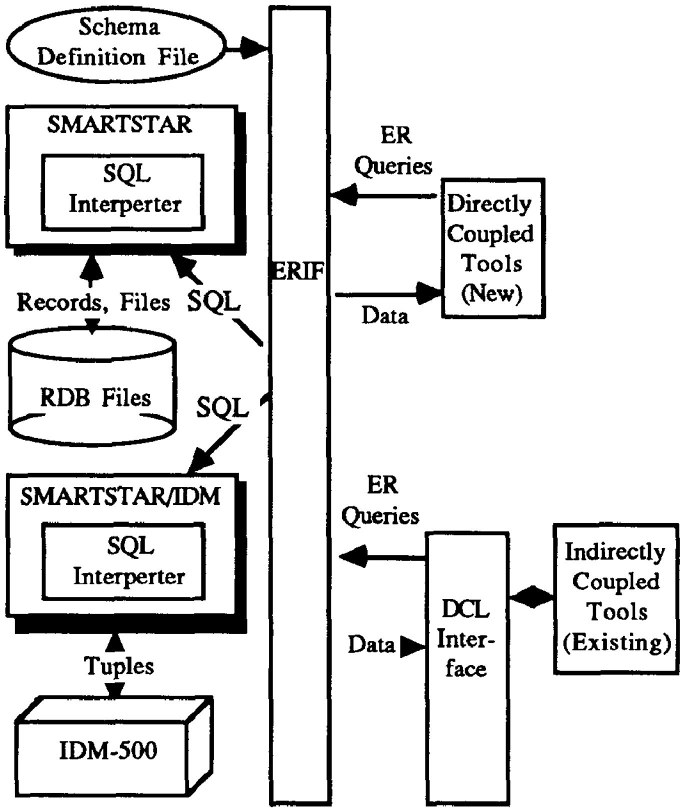


Figure 7. SLCSE Database Access

The advantage of providing both application-level and DCL-level access to the database is that it allows the SLCSE to integrate both new and existing tools into its toolset: new tools (Ada applications) accessing the database directly at the application-level, and existing tools accessing the database indirectly at the DCL-level from .COM files.

Future Directions. Refinement of the database subschemas, measurment and improvement of database performance, and enhancing ERIF functionality are the principle future directions for the database.

5.3 Toolset

Design Overview. The tools included in the SLCSE are listed in Table 1. These tools provide a representative, but by no means exhaustive, collection of capabilities supporting the full spectrum of software life cycle activities. Most of the tools were obtained as either Commercial-Off-The-Shelf (COTS) or Government Furnished Equipment (GFE). However, several new tools were developed as part of the SLCSE project. These directly-coupled tools were constructed to populate and extensively utilize the SLCSE database. Selected (i.e., lesser known) tools are briefly described in the following paragraphs.

	Non-developmental	Developmental
General Support Tools	EVE EDT LQP MAIL	Modify-ER DOCGEN
PM Tools	MicroPlanner	Plan Importer
Reqt's Tools		Requirements Tool Consistency Checker DOCGEN
Design Tools	ADL SDDL	Design Tool
Prototyping Tools	Winnie MOO SDL Compiler Convert-ER Analyze-ER SQuery/Design/Form Pigmy	
Coding Tools	LSE Ada Compiler/ACS Jovial J73 Compiler FORTRAN Compiler COBOL Compiler MACRO Assembler Linker Debugger	
Testing/ V&V/ QA Tools	ATVS J73AVS RXVP80 CAVS AMS	Test Manager PCR Processor Consistency Checker
CM Tools	CMS/MMS ALICIA	Baseliner
Environment Management Tools	SQL/SMARTSTAR Winnie & MOO Analyze-ER Convert-ER	SEM SDL Compiler Perf. Eval. Tools

Table 1. SLCSE Toolset.

LQP is a general purpose text formatting tool.

DOCGEN generates formal documentation based on the contents of the SLCSE Project Database. The tool retrieves entity, relationship, and attribute information from the SLCSE Project Database and produces DOD-STD-2167A documents (e.g., SRS, SDD, SDP, etc.).

Modify-ER allows authorized users to graphically view the structure of the SLCSE ER Database, traverse the Database, and modify the contents of the database (e.g., insert entities and relationships, enter and modify attribute information, etc.) It supports population of SLCSE database subschemas for which there are no available or adequate tools.

MicroPlanner/Plan Importer are companion tools which allow project managers and planners to do their work on a Macintosh (using it as a Project Management Workstation) and upload the project plan information to the host VAX for incorporation into the SLCSE Database.

The Requirements Tool populates the System_ Requirements and Software_Requirements Subschemas of the SLCSE Project Database. Through the use of interactive forms, the user can create and modify Requirements entities and establish relationships allocating requirements entities to software elements (i.e., CSCIs, CSCs, etc.).

The Design Tool allows users to populate the Design Subschema of the SLCSE Project Database. Through the use of interactive forms, the user can create and modify Design entities (including descriptions of interfaces, data, and processing algorithms) and establish relationships partitioning System elements into software elements, nesting entities within entities, etc..

ADL supports text-oriented descriptions of software design utilizing a formal, Ada-like, specification language, and generation of reports based on these specifications. ADL is geared toward the development of Ada software.

The WINNIE Prototyper is an interactive tool supporting rapid construction of window-oriented user interfaces for VT100-compatible terminals.

MOO supports definition of an interactive application's operational structure: it serves as a unifying mechanism connecting the interactive windows (constructed via WINNIE) with the user responses directing operation of the interactive application.

Convert-ER/Analyze-ER are companion tools that convert Schema Definition Language statements into PROLOG and then analyze the PROLOG form of the schemas to check for consistency. Convert-ER translates SDL into PROLOG, and Analyze-ER processes this PROLOG and checks for consistency within the schema definitions (e.g., relationships have both domains and ranges).

SQUERY/SDESIGN/SFORM are companion tools are provided with SMARTSTAR and support the prototyping of relational database queries, relational table designs, and relational database data entry/display forms.

Pigmy is a general purpose graphics utility.

Ada Test and Verification System (ATVS) supports static analysis (e.g., global symbol cross-reference, compilation order, exception propagation, etc.) and dynamic analysis (e.g., program coverage, timing, and tasking characteristics) of Ada programs. JOVIAL J73 Automated Verification System (J73AVS) supports static and dynamic analysis of JOVIAL J73 programs, RXVP80 provides static and dynamic analysis of FORTRAN programs, and the COBOL Automated Verification System (CAVS) provides static and dynamic analysis of COBOL programs.

The Test Manager supports population of the Test Subschema of the SLCSE Project Database. Through the use of interactive forms, the user can create and modify Test subschema entities (e.g., Test, Test_Result, Test_Case, etc.) and establish relationships between entities.

The Automated Measurement System (AMS) supports the definition, collection, and reporting of software quality metric information based on the RADC Software Quality Framework. AMS allows users to interactively enter raw data pertaining to software quality metric information. AMS also analyzes Ada and FORTRAN source code to generate software quality metric information.

The Problem Change Report (PCR) Processor allows users to create, process, and track Software Problem Reports (SPRs) through the use of an interactive SPR form. The tool populates the Configuration_Management Subschema of the SLCSE Project Database (primarily the Problem entity and its associated entities and relationships).

The Consistency Checker performs consistency checks on various SLCSE Project Database subschemas, and produces reports identifying any inconsistencies.

The Baselining Tool allows users to interactively define configurations, include both entity types and entity instances in configurations, define configuration baselines, edit configuration entities, and generate reports describing the contents of a configuration.

Automated Life Cycle Impact Analysis (ALICIA) allows users to interactively browse the SLCSE Project Database, identify entities for a proposed change, and review the estimated impact of the change on SLCSE Project Database entities and relationships.

The SLCSE Environment Manager (SEM) allows authorized users to define, modify, and tailor an environment for a particular software development project. It supports definition and subsequent modification of a site/company/organization environment: that is, the computing facility providing resources common to all software development projects using the environment (e.g., the computer network, tools, roles, and users). It supports definition and subsequent modification of one or more software development projects active within the larger environment utilizing a subset of the environment's resources (i.e., the subset of computer nodes, tools, roles, and users for a specific software development project).

The Schema Definition Language (SDL) Compiler translates SDL source code (a specialized Ada-like language for specifying subschemas and the entities, relationships, and attributes that comprise them) into Structured Query Language (SQL) statements which are interpreted by SMARTSTAR (a commercial relational database supporting SQL) to create a low-level relational implementation of the ER SLCSE Project Database.

The Performance Evaluation Tools are used to measure the performance characteristics of the SLCSE (e.g., system load, database size, ERIF initialization, ERIF calls, Command Executive speed, etc.). Based on this information the SEM can be used to create an optimal environment for a particular project or site. The information can also be used to identify SLCSE performance bottlenecks for future modification and optimization.

Tool Selection Criteria. The SLCSE development effort concentrated on the design and implementation of the SLCSE itself, it was not a tool development effort. Neither was it an exhaustive tool evaluation and selection effort. As a result tool selection was based on several basic criteria:

- Did the tool effectively support software lifecycle activities such as project planning, requirements definition, coding, testing, etc. (i.e. did it fall into one of the nine tool categories
- Was the tool available at the customer (RADC) site or was it otherwise obtainable as public domain software, GFE, or COTS
- Could it be integrated with the SLCSE User Interface and Database

With few exceptions, most of the Non-developmental (i.e., COTS and GFE) Software tools included in the Toolset do not make extensively use the SLCSE database. As a consequence, in order to effectively evaluate the user interface and database model, several tools had to be developed from scratch as part of the SLCSE project. These tools were developed to be Conformant (i.e., maintain visual and operational consistency with the SLCSE User Interface) and Directly-coupled (i.e., access the SLCSE database directly via the ERIF). These tools (indicated in Table 1 and described in the preceding paragraphs) were specifically developed to populate and extensively utilize the various database subschemas.

Future Directions. The primary focus for the toolset is the addition of new tools. However, more important, and more technically challenging, is the comprehensive integration of these tools with the Database. Simple integration of a tool can be accomplished very quickly, but comprehensive integration requires analysis of the database subschemas, a working knowledge of the ERIF or DCL interface, and the data requirements for the integrated tool.

7 Status and Conclusion

GRC and its subcontractors, Intermetrics and Software Productivity Solutions began development of the SLCSE in August 1986 and will conduct final acceptance test in November 1988. Currently GRC is finalizing Beta-testing arrangements for the SLCSE at various Government and Government contractor installations.

RADC plans to use the SLCSE as the unifying foundation for the integration of advanced software technologies: both for ongoing projects such as the Ada Test and Verification System (ATVS) and the QUality Evaluation System (QUES), and for proposed projects such as the Requirements Workstation and the SLCSE Project Management System. GRC plans to commercialize and rehost the SLCSE on a variety of hardware architectures.

8 References

1. "Software Development Environments", Susan A. Dart, Robert J. Ellison, Peter H. Feiler, and A. Nico Habermann, Computer, November 1987

2. "Rational's Experience Using Ada for Very Large Systems," J.E. Archer, Jr., and M.T. Devlin, Proc. First Int'l Conf. Ada Programming Language Applications for the NASA Space Station, NASA, June, 1986

3. "Gandalf: Software Development Environments," A.N. Habermann and D. Notkin, IEEE Trans. Software Engineering, December 1986

4. "Unix Time-sharing System: The Programmer's Workbench," T.A. Dolotta, R.C. Haight, and J.R. Mashey, Interactive Programming Environments, McGraw-Hill, New York, 1984

5. "Excelerator," Index Technology Corp., Proc. Computer-aided Software Engineering Symposium., Digital Consulting Inc., Andover, Mass., June 1987