she may determine his or her degree of confidence in the ES’s results.

These four application types—transaction, query, DSS, and ES—will be referenced throughout the text to tie topics together and to discuss the usefulness of methodologies, languages and approaches to testing, quality assurance, and maintenance for each.

Embedded Systems

**Embedded systems** are applications that are part of a larger system. For example, a missile guidance application works in conjunction with sensors, explosives, and other equipment within a single missile unit. The application, by itself, is minor; its complexity derives from its analog interfaces, need for complete accuracy, and real-time properties within the missile’s limited life span once it is released. Embedded applications development has been the province of computer science educated developers rather than information systems (IS) educated developers.

As business deploys ever more complex equipment in the context of computing environments, the need for embedded skills will increase. This implies that IS education must also address real-time, embedded system requirements, and that computer scientists will continue to move into business for application development.

Applications in Business

Applications are most successful when they match the organizations’ needs for information. Most information in organizations is generated to allow the managers to control the activities of the organization to reach the company’s goals. Goals may be short-term or long-term. Control of activities implies information evaluation and decision making. There are three levels of organizational decision making: operational, managerial, and strategic. Each level has different information needs and, therefore, different application needs.

At the operational level, the organization requires information about the conduct of its business. Decisions deal with daily operations. For instance, the operational level in a retail organization is concerned with sales of products. The main operational level applications would be order processing, inventory control, and accounts receivable. In a manufacturing business, the operational level is concerned with sales and manufacturing. The main operational level applications would be manufacturing planning, manufacturing control, inventory management, order processing, and shipping.

The information at the operational level is current, accurate, detailed, available as generated, and relates to the business of the organization. Operational information is critical to the organization remaining in business. As a critical resource, the data requires careful management and maintenance. The types of applications that support operational level decisions and information are transaction processing applications (see Figure 1-17). Query applications for current operational data are other applications that support operational level decisions.

The information needs for managerial control are mostly internal information, can be detailed or summary, and should be accurate. Decisions made for managerial control concentrate on improving the existing ways of doing business, finding and solving problems, and take a medium-range (e.g., quarter or year) view of the company’s business. The types of issues dealt with concern reduction of

- costs by comparing suppliers’ prices
- the time to process a single order
- the errors in a process

![FIGURE 1-17  Application Types and Decision Types](http://globaltext.terry.uga.edu/booklist?cat=Computing)
Sequential Project Life Cycle

You should remember from systems analysis that a **sequential project life cycle (SPLC)** starts when a software product is conceived and ends when the product is no longer in use. Phases in a SPLC include

- initiation
- problem definition
- feasibility
- requirements analysis
- conceptual design
- design
- code/unit test
- testing
- installation and checkout
- operations and maintenance
- retirement

These SPLC phases are more appropriate to business than to military/government applications because, in the government, the first four phases (initiation, definition, feasibility, and functional requirements definition) are usually completed by a different organization than that of the implementers. Government projects are subject to congressional review, approval, and budgeting. So, a government project requiring congressional appropriation is usually defined as beginning at the conceptual design phase and ending with deployment of the system with operational status according to Department of Defense standard #2167a [DOD, 1985]. In contrast, business IS are typically initiated by a user department requesting that a system be built by an MIS department. The need for an IS is typically motivated by some business situation: a change in the method of business, in the legal environment, in the staffing/support environment, or in a strategic goal such as improving market competitiveness.

We call these SPLC phases a ‘Waterfall’ approach to applications because the output of each phase feeds into the next phase, while phases are modified via feedback produced during the verification and validation processes\(^8\) (see Figure 1-18).

---

7 Future developments in life cycles are discussed in Chapter 18.

Phases in the waterfall definition are defined as discrete even though, in practice, the information is obtained in a nonlinear manner and the phase beginnings and endings are difficult to distinguish. To identify discrete beginnings and endings, most companies use the completion of the major product (i.e., program or document) produced during each phase as signaling the phase end. So, completion of a feasibility report, for instance, identifies the end of the feasibility analysis phase. In the following subsections, each phase of the project life cycle (SPLC) is defined, with the main activities and documents identified.

9 This definition is adapted from work conducted during The Assessment and Development of Software Engineering Tools project sponsored by the U.S. Army Institute for Research in Management Information, Communications, and Computer Sciences (AIRMICS), contract DAKF11-89-C-0014.
SPLC Phases

INITIATION. Project **initiation** is the period of time during which the need for an application is identified and the problem is sufficiently defined to assemble a team to begin problem evaluation. The people and organizations affected by the application, that is, the **stakeholders**, are identified. Participants from each stakeholder organization for the development team are solicited. The outcome of initiation is a memo or formal document requesting automation support and defining the problem and participants.

FEASIBILITY. **Feasibility** is the analysis of risks, costs and benefits relating to economics, technology, and user organizations. The problem to be automated is analyzed in sufficient detail to ensure that all aspects of feasibility are evaluated.

**Economic feasibility** analysis elaborates costs of special hardware, software, personnel, office space, and so forth for each implementation alternative.

**In technical feasibility** analysis, alternatives for hardware, software, and general design approach are determined to be available, appropriate, and functional. The benefits and risks of alternatives are identified.

**Organizational feasibility** is an analysis of both the developing and using organizations' readiness for the application. Particular emphasis is placed on skills and training needed in both groups to ensure successful development and use of the application. The decision whether or not to use consultants and the type of role they would play during development is made during organizational feasibility analysis. Organizational decisions include effectiveness of the organization structure and definition of roles of individual jobs in the organization as they will be with the new application.

The feasibility report summarizes

- the problem
- the economic, technical and organizational feasibility
- risks and contingency plans related to the application
- preferred concept for the software product and an explanation of its superiority to alternative concepts
- training needs and tentative schedules
- estimates of project staffing by phase and level of expertise

After feasibility is established, the **Software Development Life Cycle (SDLC)**, a subcycle of the SPLC, begins. This subcycle typically includes phases for analysis, conceptual design, design, implementation, testing, and installation and checkout. SDLC end is signaled by delivery of an operational application.

ANALYSIS. The **analysis** phase has many synonyms: Functional Analysis, Requirements Definition, and Software Requirements Analysis. All of these names represent the time during which the business requirements for a software product are defined and documented. Analysis activities define

1. Functional requirements—"what" the system is supposed to do. The format of the functional requirements definitions depends on the methodology followed during the analysis phase.
2. Performance requirements—terminal, message, or network response time, input/output volumes, process timing requirements (e.g., reports must be available by 10 A.M.).
3. Interface(s) requirements—what data come from and go to other using applications and organizations. The definition includes timing, media, and format of exchanged data.
4. Design requirements—information learned during analysis that may impact design activities. Examples of design requirements are data storage, hardware, testing constraints, conversion requirements, and human-machine interaction requirements (e.g., the application must use pull-down menus).
5. Development standards—the form, format, timing, and general contents of documentation to be produced during the development. Development standards include rules about allowable graphical representations,
documentation, tools, techniques, and aids such as computer-aided software engineering (CASE) tools, or project management scheduling software. Format information includes the content of a data dictionary/repository for design objects, project report contents, and other standards to be followed by the project team when reporting project accomplishments, problems, status and design.

6. The plan for application development is refined.

Analysis documentation summarizes the current method of work, details the proposed system, and how it meets the needs of the required functions. Requirements from the work activities are described in graphics, text, tables, structured English, or some other representation form prescribed by the methodology being used.

CONCEPTUAL DESIGN. Once the proposed logical system is understood and agreed to by the user, conceptual design begins. Other names for conceptual design activity include preliminary design, logical design, external design, or software requirements specifications. The major activity of conceptual design is the detailed functional definition of all external elements of the application, including screens, reports, data entry messages, and/or forms. Both contents and layout are included at this level. In addition, the logical data model is transformed into a logical database schema and user views. If distribution or decentralization of the database is anticipated, the analysis and decision are made during conceptual design. The outputs of conceptual design include the detailed definition of the external items described above, plus the normalized and optimized logical database schema.

Not all organizations treat conceptual design separately. Outputs of conceptual design may be in a conceptual design document or might be part of the functional requirements documents developed during analysis. Depending on the project manager’s personal taste and experience, the conceptual design might be partially completed during logical design and fully completed during physical design. In this text, the two phases, design and conceptual design, are treated as one.

DESIGN. Design maps “what” the system is supposed to do into “how” the system will do it in a particular hardware/software configuration. The other terms used to describe design activities include detailed design, physical design, internal design, and/or product design.

During the design phase, the software engineering team creates, documents, and verifies:

1. Software architecture—identifies and defines programs, modules, functions, rules, objects, and their relationships. The exact nature of the software architecture depends on the methodology followed during the design phase.
2. Software components and modules—defines detailed contents and functions of software components, including, but not limited to, inputs, outputs, screens, reports, data, files, constraints, and processes.
3. Interfaces—details contents, timing, responsibilities, and design of data exchanged with other applications or organizations.
4. Testing—defines the strategy, responsibilities, and timing for each type of testing to be performed.
5. Data—physically maps “what” to “how” for data. In database terms, this is the definition of the physical layout of data on the devices used, and of the requirements, timing, and responsibility for distribution, replication, and/or duplication of data.

SUBSYSTEM/PROGRAM DESIGN. Subsystem and/or program designs are sometimes treated as subphases of the design phase. Whether they are separate phases or not, the software engineering team creates, documents, and verifies the following:

1. Application control structure—defines how each program/module is activated and where it returns upon completion.

---

Source URL: http://globaltext.terry.uga.edu/booklist?cat=Computing
Saylor URL: http://www.saylor.org/courses/bus206/#2.1

Attributed to: [Sue A. Conger]
2. Data structure and physical implementation scheme—defines physical data layouts with device mapping and data access methods to be used. In a database environment, this activity may include definition of a centralized library of data definitions, calling routines, and buffer definitions for use with a particular DBMS.

3. Sizing—defines any programs and buffers which are expected to be memory-resident for on-line and/or real-time processes.

4. Key algorithms—specifies mathematically correct notation to allow independent verification of formula accuracy.

5. Program component (routine with approximately 100 source procedure instructions)—identifies, names, and lists assumptions of program component design and usage. Assumptions include expectations of, for instance, resident routines and/or data, other routines/modules to be called in the course of processing this module, size of queues, buffers, and so on required for processing.

CODE AND UNIT TEST. During coding, the low-level program elements of the software product are created from design documentation and debugged. Unit testing is the verification that the program does what it is supposed to do and nothing more. In systems using reusable code, the code is customized for the current application, and checked to ensure that it works accurately in the current environment.

TEST. During testing—sometimes called Computer Software Component (CSC) Integration and Testing—11 the components of a software product are evaluated for correctness of integrated processing. Quality assurance testing may be conducted in the testing phase or may be treated as a separate activity. During quality assurance tests, the software product (i.e., software or documentation) is evaluated by a nonmember of the project team to determine whether or not the analysis requirements are satisfied.

IMPLEMENTATION. Also called Installation and Checkout, implementation is that period of time during which a software product is integrated into its operational environment and is phased into production use. Implementation includes the completion of data conversion, installation, and training.

At this point in the project life cycle, the software development cycle ends, and the maintenance phase begins. Maintenance and operations continue until the project is retired.

OPERATIONS AND MAINTENANCE. Operations and maintenance is the period in the software life cycle during which a software product is employed in its operational environment, monitored for satisfactory performance, and modified as necessary. Three types of maintenance12 are

1. Perfective—to improve the performance of the application (e.g., make all table indexes binary to minimize translations, change an algorithm to make the software run faster, and so on.)
2. Corrective—to remove software defects (i.e., to fix bugs)
3. Adaptive—to incorporate any changes in the business or related laws in the system (e.g., changes for new IRS rules)

Each type of maintenance requires a mini-analysis and mini-design to determine social, technical, and functional aspects of the change. The current operational versions of software and documentation must be managed to allow identification of errors and to ensure that the correct copy of software is run. One aspect of change management specifically addresses configuration management of application programs in support of maintenance activities.

RETIREMENT. Retirement is the period of time in the software life cycle during which support for a software product is terminated. Usually, the functions performed by the product are transferred to a successor system. Another name for this activity is phaseout.

11 This is a term used by DOD standard #2167a, 1985.

12 A detailed discussion of maintenance topics is presented in Lientz and Swanson, 1980.
UNIVERSAL ACTIVITIES. There are two universal activities which are performed during each life-cycle phase: verification and validation, and configuration management.

An integral part of each life-cycle phase is the verification and validation that the phase products satisfy their objectives. **Verification** establishes the correctness of correspondence between a software product and its specification. **Validation** establishes the fitness or quality of a software product for its operational purpose.

For instance, the individual code module specifications from design are verified to ensure that they contain accurate and complete information about the functions they perform. The modules are validated against the analysis phase specification to ensure that all required functions have corresponding designs that accurately reflect the requirements.

**Configuration management** refers to the management of change after an application is operational. A designated **project librarian** maintains the official version of each product. The project librarian is able at any time to provide a definitive version (or *baseline*) of a document or software module. These baselines allow the project manager to control both the software maintenance process and the software products.

History

The sequential life cycle was originally developed and documented in the 1960s to provide defense contractors a life-cycle documentation standard for Department of Defense (DOD) projects. The current DOD Standard #2167a lists all activities and details all documentation required for software development as fulfillment of military contracts. As industry recognized that their own application development projects were out of control, over budget, and unsatisfactory when complete, they modified the standard to eliminate defense/aerospace terminology and replace it with industry specific terminology. Organizations modified the standard to incorporate elements of methodologies, such as structured development, data flow diagrams, and walk-throughs, that were becoming known at the same time. In the late 1960s and early 1970s the waterfall and 2167 documentation standard were used throughout most Fortune 500 companies as cast-in-concrete requirements for building and documenting systems.

Problems

As nonnegotiable documentation requirements, projects frequently produced thousands of pages of documentation that no one except the authors ever read. Information about applications was rarely in any one person’s head and communication overhead became a major problem to completing systems successfully. User/management approval to continue with each phase was not based on their knowledge of what the system would do, but on some other criteria. Published studies showed that the typical written application requirements document contained, on average, one-half to one error per page. The conclusion that paper prose is not a good medium for conveying the complex variety of application requirements led to the development of more graphical representation forms.

Eventually, IS managers realized that the waterfall, when applied too stringently, not only did **not** solve the problems of bad systems, it contributed to a new generation of overdocumented bad systems. The result has been a scaling back on required documentation. Standards have become ‘guidelines’ for experienced project managers to consider and to provide new project managers with review lists of activities whose relevance they should consider. Each project team customizes the documentation and development activities in addition to the tools and techniques they use.

Even with relaxation of required documentation, a sequential life cycle does not recognize the iterative, nonlinear nature of application development, and cannot easily accommodate overlap of phases. Many organizations now use a variant of the waterfall by performing the activities in an overlapped manner, sometimes called the ‘pipeline’ approach. Finally, the waterfall approach does not recognize that the level of detail necessary to adequately document application functions is significantly different with the use of automated tools, use of diagrams (e.g., DFDs) to replace text, and use of high level, fourth-generation languages (e.g., SQL).