The Standard Waterfall Model for Systems Development

The standard waterfall model for systems development is an approach that goes through the following steps:

1. Document System Concept
2. Identify System Requirements and Analyze Them
3. Break the System into Pieces (Architectural Design)
4. Design Each Piece (Detailed Design)
5. Code the System Components and Test Them Individually (Coding, Debugging, and Unit Testing)
6. Integrate the Pieces and Test the System (System Testing)
7. Deploy the System and Operate It

This model is widely used on large government systems, particularly by the Department of Defense (DOD).

As part of this standard approach, the party responsible for contracting out the system development (ESDIS for the ECS Contract) can call on a number of tools to help plan and document the system. ECS followed this planning approach, which means that early in the system development, ESDIS set up a standard set of documents for the contractor to supply, as well as a contractual schedule for the major pieces. The development process provided a number of design reviews, notably

- Conceptual Design Review
- Requirements Review
- Preliminary Design Review (PDR)
- Critical Design Review (CDR)

Until these reviews were completed, there would be little code developed. After the CDR, the contractor would code to the design.

The standard reference for estimating the cost of the system is the COnstructive COst MOdel (COCOMO) developed by Dr. Barry Boehm while he was at TRW [Boehm, B., 1981: *Software Engineering Economics*, Prentice-Hall]. This model relates the development time and workforce [man-months] to the "Source Lines of Code" (SLOC). Roughly, for an ECS type of system ECS, the workforce (and therefore cost) scales as the cube of the development time. There are simple versions of the model and much more complex ones. Generally, all of the relationships used to predict these relationships are statistical in nature: Dr. Boehm and other workers in software project cost estimation build a database of project schedules and costs and then regress those against SLOC estimates. The most recent version of Dr. Boehm's work is provided in [Boehm, B., et al., 2000: *Software Cost Estimation with COCOMO II*, Prentice-Hall.].

There have been a number of criticisms of the standard waterfall model, including

- Problems are not discovered until system testing.
- Requirements must be fixed before the system is designed - requirements evolution makes the development method unstable.
- Design and code work often turn up requirements inconsistencies, missing system components, and unexpected development needs.
- System performance cannot be tested until the system is almost coded; undercapacity may be difficult to correct.

The standard waterfall model is associated with the failure or cancellation of a number of large systems. It can also be very expensive. As a result, the software development community has experimented with a number of alternative approaches, including

- Spiral Design (Go through waterfalls, starting with a very rough notion of the system and becoming more detailed over time)
- Modified Waterfalls (Waterfalls with Overlapping Phases; Waterfall with Subprojects)
- Evolutionary Prototyping (Start with initial concept, design and implement an initial prototype, iterate as needed through

prototype refinement until acceptable, complete and release the acceptable prototype)

- Staged Delivery (Go through Concept, Requirements Analysis, and Architectural Design - then implement the pieces, showing them to the customer as the components are completed - and go back to the previous steps if needed)
- Evolutionary Delivery (a cross between Evolutionary Prototyping and Staged Delivery)

These are discussed in considerable detail in [McConnell, S., 1996: *Rapid Development, Taming Wild Software Schedules*, Microsoft Press]. Commercial software projects often reduce the formality of the full waterfall model. In the last few years, a paradigm known as eXtreme Programming has emerged that emphasizes reducing the cost of software changes, developing test cases before coding, developing code using pairs of programmers, and putting most of the documentation into the code [Beck, K., 2000: *Extreme Programming Explained, Embrace Change*, Addison-Wesley].