(Source: *Managing and Leading Software Projects*, Richard Fairley, John Wiley & Sons Publication, 2009.)

In a general sense, all organizational projects are similar:

- objectives must be specified
- a schedule of activities must be planned
- resources allocated
- responsibilities assigned
- work activities coordinated
- progress monitored
- communication maintained
- risk factors identified and confronted
- corrective actions applied as necessary

The general project management procedures need to be tailored for different types of projects (e.g., construction, manufacturing, software development, etc.) based on the nature of the product being developed. Fred Brooks (managed the development of IBM's System/360 and OS/360 wrote about the process in the seminal book *The Mythical Man-Month*, 1975/1995) observed essential properties of software differentiate it from other kinds of engineering artifacts and make software projects difficult:

1. *complexity* - large number of unique (i.e., each must be written), interacting (e.g., serial and concurrent invocations, state transitions, data coupling, and interfaces to databases and external systems) parts (functions, subroutines, objects). Thus, a seeming "small" change in a requirements may cause a large amount of rework. Size and complexity may hide defects that may require unplanned rework later.
2. *software conformity* - software must conform to exacting specifications in representation of each part, in the interfaces to other internal parts, and in the connections to the environment in which it operates. Manufactured/physical parts can interface correctly if they are within their tolerances, but software parts must interface exactly with respect to number and type of parameters and "kind of couplings."
3. *software changebility* - because software is the most easily changed element (i.e., the most malleable) in a software-intensive project, it is the most frequently changed element, particularly in the late stages of a project. Changes may occur because customers change specifications, competing products change, mission objectives change, laws change, etc.
4. *software invisibility* - software has no visible properties (cannot not be seen, touched, hear, etc.) only indirect representations by work produces: requirement specifications, design documents, source code, etc. Well-known "90% complete syndrome" software reported as nearly done with no objective evidence to support or refute claim.
5. *team-oriented, intellect-intensive endeavor* - not like assembly-line work where work is arranged so each person can perform a task with a high degree of autonomy and a small amount of interaction with others.

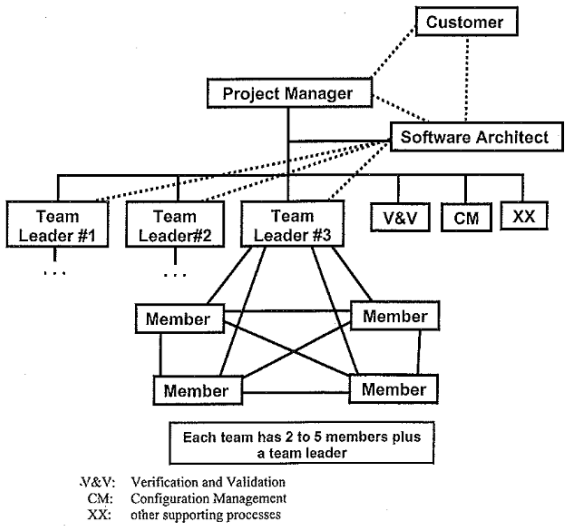

FIGURE 1.7 An organizational model for software projects

Brook's law: "Adding manpower to a late software project makes it later."

What three factors make *Brook's law* true?

The primary objective of software project is to develop and deliver one or more acceptable work products within the constraints of required features, quality attributes, project scope, budget, resources, completion date, technology, etc. The work products to be delivered (e.g., object code, training materials, installation instructions) result from the flow of intermediate work products that are produced by the flow through the work processes (requirements, design, source code, and test scenarios). The general model of project workflow is given below:

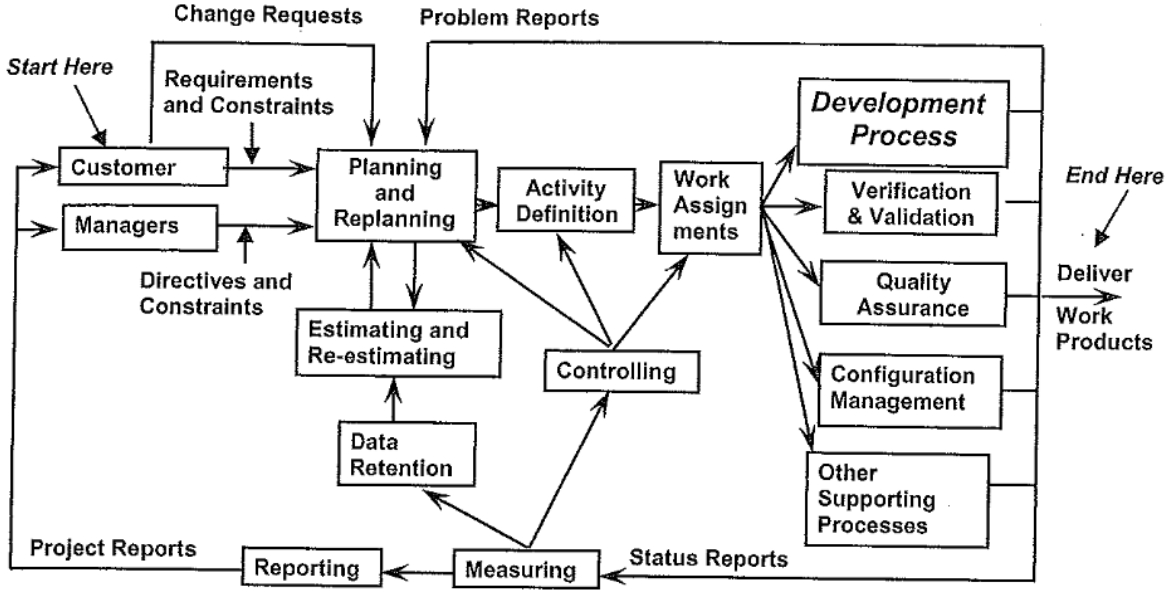

FIGURE 2.1A A workflow model for managing software projects.

Expanding the box in the upper right to for a software-intensive system (with hardware component too):

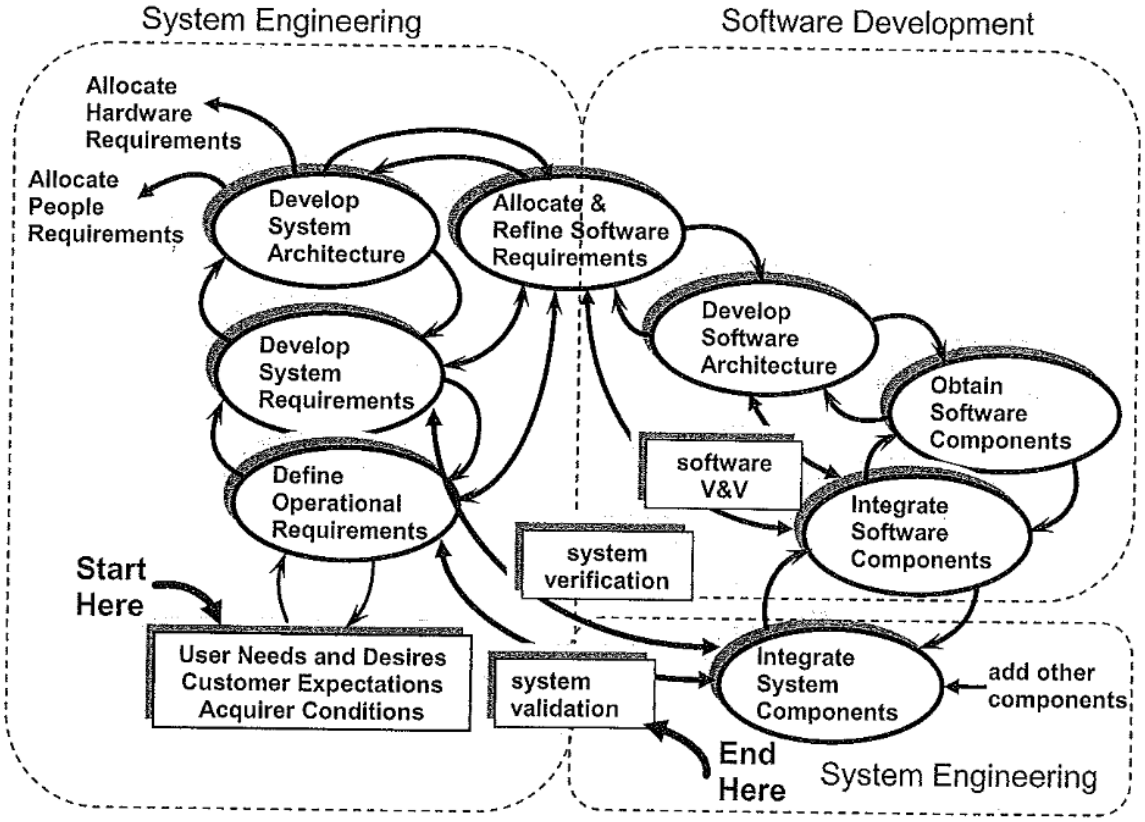

FIGURE 2.1B A process framework for developing software-intensive systems

For a software-only project:

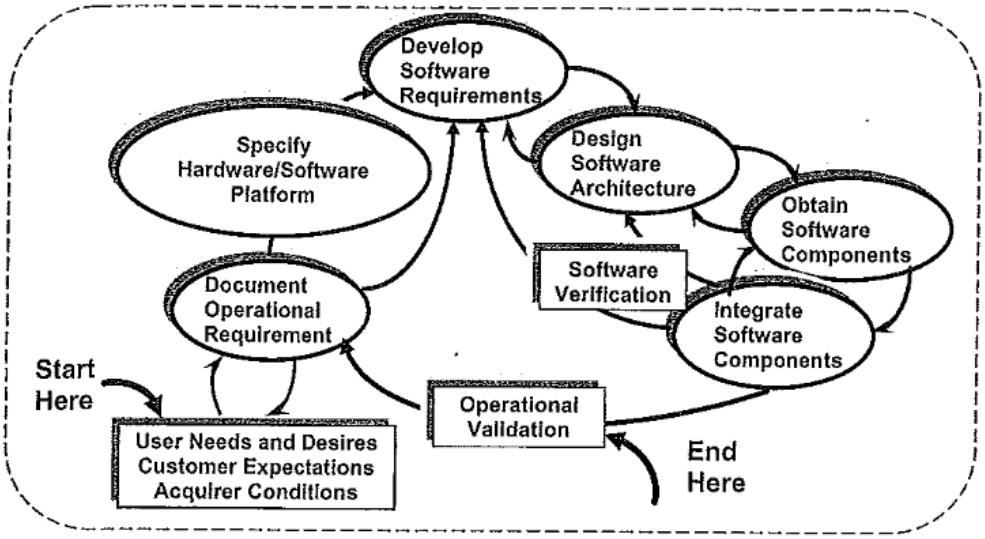

FIGURE 2.5 A development framework for software-only projects

Software development process models:

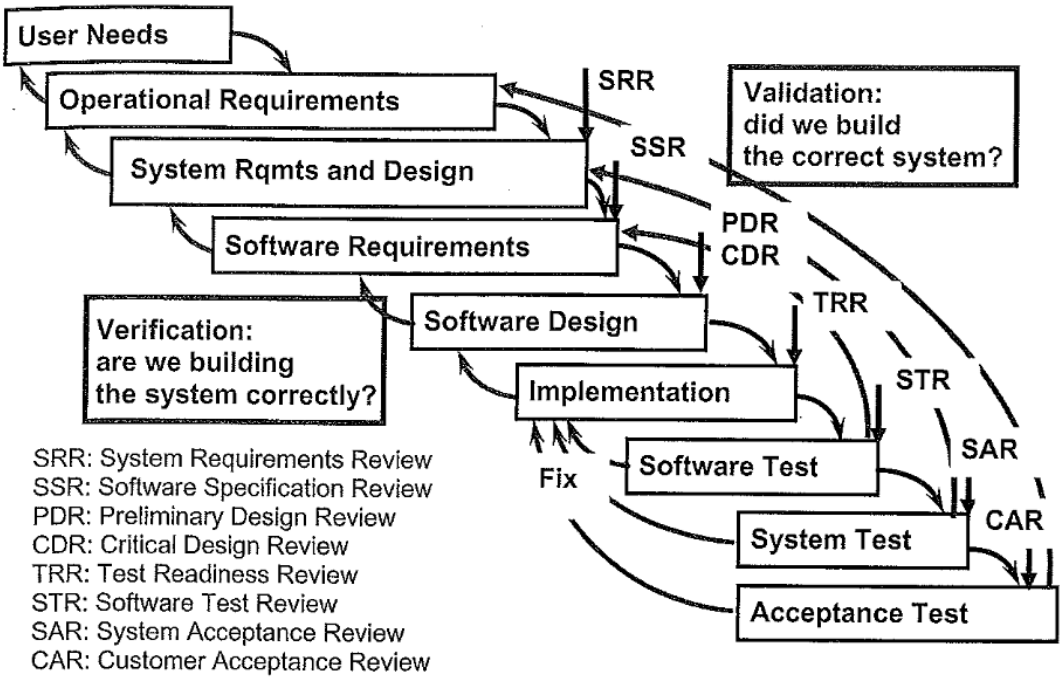

FIGURE 2.8 Traditional depiction of the Waterfall model with milestones

What problem(s) are associated with the Waterfall model?

The Incremental-build model is a build-test-demonstrate model of iterative developmemt where frequent demonstrations of progress and verification and validation of work to date are emphasized. A design partitioning phase allocates partitions the software-architecture into a prioritized sequence of builds with each build adding new capabilities and features to incrementally grow the product.

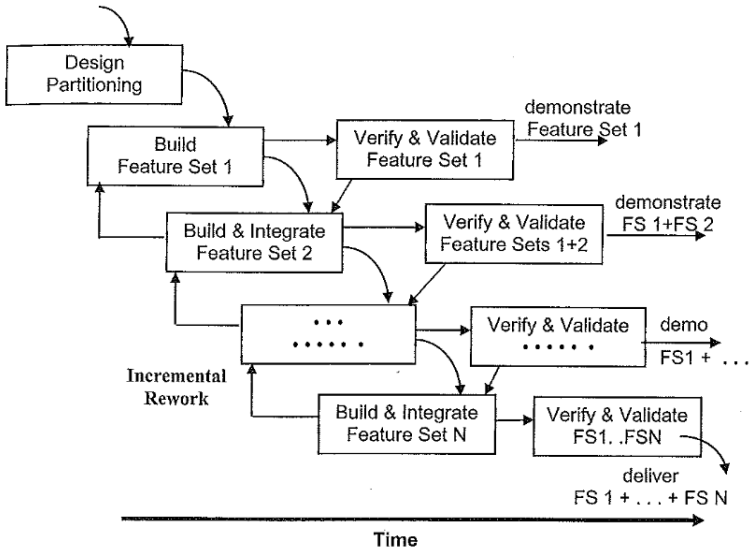

FIGURE 2.11 Incremental build–verify–validate–demonstrate cycles

The Evolutionary Model is a development process to systematically evolve the requirements, design, and code.

Why is the Evolutionary model appropriate when the system concept is unclear or the feasibility of the project is in question?

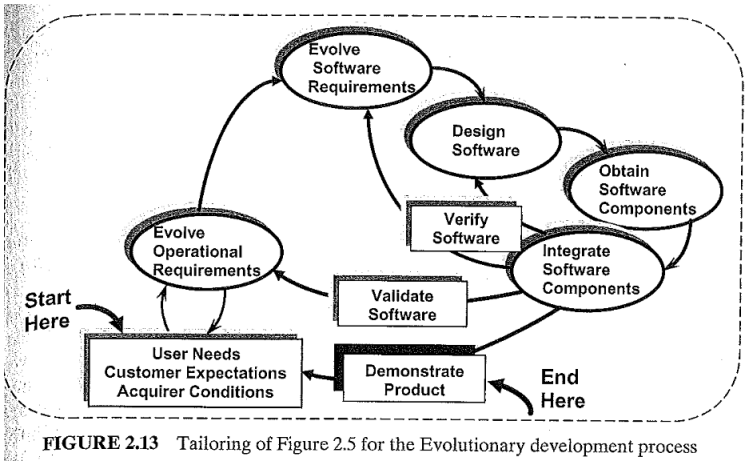

FIGURE 2.13 Tailoring of Figure 2.5 for the Evolutionary development process

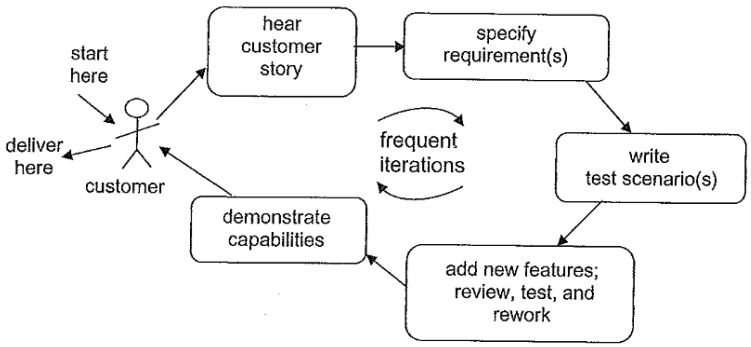

FIGURE 2.15 An Agile development process

Why is agile development best suited to small application projects that are conducted in the presence of a knowledgeable customer/user who has a clear understanding of the needs to be satisfied by the system being built?