

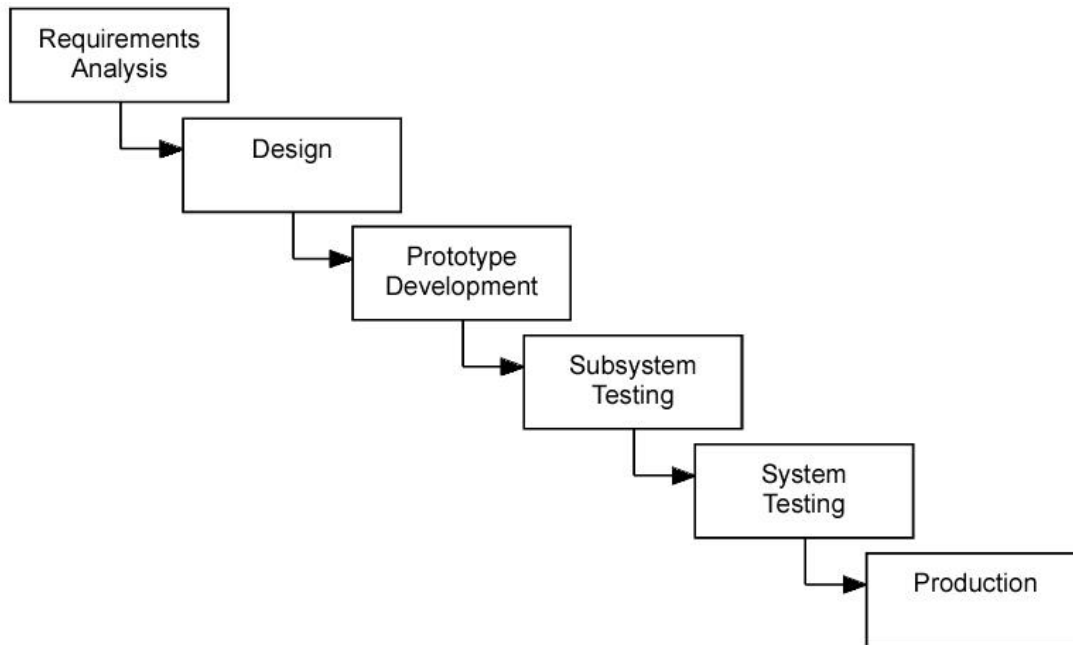
# Simulation Planning

by Jim Ledin, PE  
Copyright © 2000 Ledin Engineering

Before developing a simulation, a plan must be created identifying what is to be simulated, the degree of simulation fidelity required, and how the resulting output data will be used. Many additional details must be addressed as well. There must be a simulation software development process that deals with issues such as configuration management, release criteria, and problem reporting and resolution. There must also be a manageable process that identifies how the simulation is to be operated, including the definition of input data sets, and how simulation output data is analyzed and used as input for decision making.

To achieve maximum value from the simulation effort, the simulation plan must be thoroughly integrated into the larger project plan. The structure of the project development plan must be created in a way that maximizes the benefits from the use of simulation. There are a number of different types of project development approaches, but we will examine just two of them: the waterfall approach and the iterative approach.

The waterfall development approach (shown in Figure 1) has been used in the past for many large development projects. It presents the primary tasks in the design and development process as a set of logical, orderly steps. In practice, development projects that used this approach have frequently encountered problems that resulted in significant delays and cost overruns, as well as occasional complete failures.



*Figure 1 - Waterfall Development Model*

The problems with the waterfall approach stem from the rigidity of the sequence of steps. It assumes that all requirements can be completely specified during the requirements analysis phase. Once that box is exited, no further changes to requirements are anticipated. In practice, changes to requirements occur frequently after they have been supposedly "frozen". One reason these changes occur is because the people who identified the requirements could only imagine what how the final system would behave. They did not have any way to test the assumptions that went into their decisions, or that the final requirements were

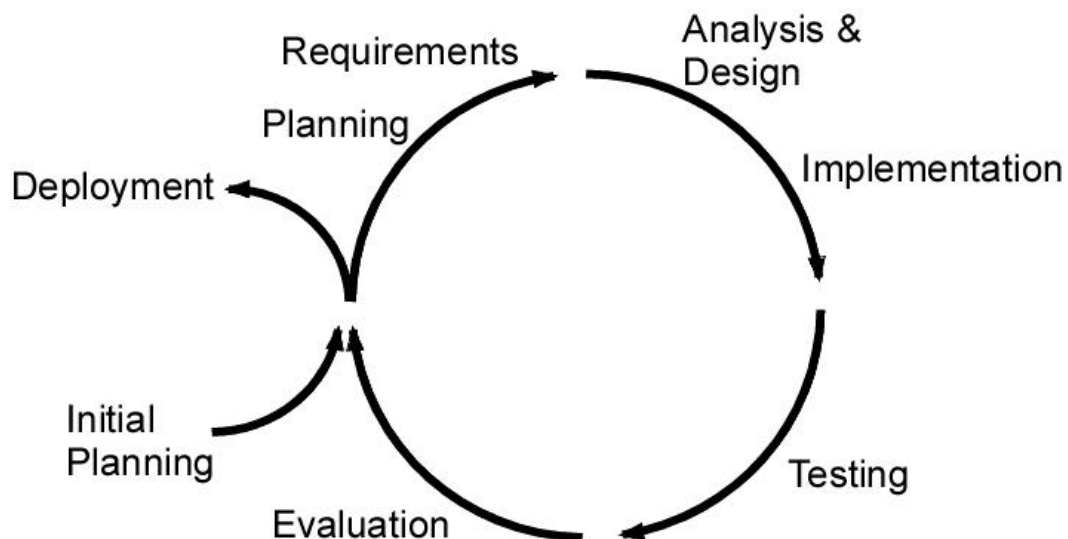
consistent and adequate to for the intended tasks. Requirement changes can come about for other reasons, such as advances in technology or new information that changes the goals for the system under development.

In the waterfall development approach, changes to requirements can result in major disruptions because once each phase is completed the structure used in that phase may be disassembled and scattered. For example, if a requirement is changed significantly after the design has been completed, it may be difficult to resurrect the design group to implement the change because its members may have moved on to other projects. Bringing the designers back to implement unexpected requirements changes may in turn produce unacceptable delays in the other projects they had been working on.

Another problem with the waterfall approach is that project risk is not significantly reduced until late in the development cycle. In other words, significant testing does not occur until the requirements and design are frozen and the prototype is in development. If a significant design flaw is not detected until this late stage, the time and cost required to change the design, implement it, and test it can be horrendous.

In summary, the waterfall development model assumes that the results of each step in the process are correct and complete. In a relatively small development project this may not be a bad assumption, and if there are problems at some stage it may be easy to go back and fix things and then continue on. In the development of a complex system, however, this assumption may be very risky.

An alternative to the waterfall model is the iterative development model as shown in Figure 2. The iterative model applies primarily to the software portion of a development project, and so it is of limited use relative to hardware development in the early project phases. This model is represented as a cycle that may be completed several times during a development project. The key idea is that during each cycle a working prototype of the software is produced and made available for testing and user interaction. The lack of prototype hardware may present problems with testing during these phases, but as we shall see, simulation can help with this.



*Figure 2 - Iterative Development Model*

The results of each cycle are then used as input to planning the next cycle, which includes possible requirement modifications as well as the design and implementation of a new subset of system

functionality. In the early cycles through the iterative loop the software will necessarily have limited functionality. The ideal approach to minimize project risk is to implement complex, high-risk capabilities during early iterations of the loop and test them thoroughly and repeatedly.

In the iterative development model, new requirements can be integrated as part of the normal development cycles. The new requirements will have incremental costs associated with them, but they will not force drastic, disruptive changes to the development process. After enough cycles through the iterative loop have been performed to implement all the software requirements, the system will be ready to exit the loop and be deployed. Using the iterative model, the most complex and riskiest parts of the software will have been thoroughly tested several times as ancillary capabilities are added at each iteration.

Simulation is a natural complement to the iterative development model. Simulations can be developed during the initial planning phase to assist in the requirements definition phases. The people specifying system requirements can observe the simulated system's performance and use that information to assist in determining the requirements.

During the early iterations of the software development cycle, prototype hardware may not be available for testing software prototypes. It may be possible to use a simulation as a testbed for evaluating these early prototype builds. In this scenario, part or all of the system software is implemented as a module in the system simulation. This software receives inputs from the simulation, performs its functions, and produces outputs that are then used to drive the simulation. This technique allows thorough testing of early software builds in a simulated operational environment. Since this is a non-real time simulation, full use can be made of debugging and execution analysis tools with no degradation of test capability.

Later, as prototype hardware becomes available, it can be integrated into a real time HIL simulation for thorough, repeatable testing in a realistic environment. The HIL simulation can be used to speed the software/hardware integration process while making it relatively easy to identify and fix problems as they occur.

The impressive benefits from the use of simulation occur because it reduces uncertainty and risk at each development phase where it is used. The initial set of system requirements will be more certain if the developers have made effective use of a system simulation as part of requirements definition. At each phase where prototype software and hardware become available they can immediately be placed in a simulated environment and tested under realistic conditions. By the time a full system prototype is ready for testing, a high level of confidence can be achieved that no serious problems will be found. The reduction of project risk is the payoff from effective use of simulation from the beginning.