17. The Design, Build, Integrate and Test States

At this point in the System Development Process (SDP), the design state is over and the system starts to come together. The role of the systems engineer undergoes a significant change. Until now, the systems engineers led the technical design activity. Now their role starts to become that of a catalyst. At the completion of the System Requirements Review (SRR) the structure of the system has been designed[1]. These states begin with the design and the preliminary planning of the testing, field installation, and training programs. The design is mostly carried out by hardware and software engineers who produce the detailed schematic diagrams and software listings (Section 3.2.4).

In an effective systems development organization, some of the systems engineers move from the design organization to the Anticipatory Testing department once the Preliminary Design Review (PDR) is over. Others work with the program manager to optimize the activities on a total project basis. This chapter discusses some of their roles during these states of the SDP.

17.1. The role of the Anticipatory Testing department

The Anticipatory Testing department performs the sensitization training and in-process tests and inspections during the design and build states (Section 10.2.2). Later, the systems engineers lead the integration tests to verify that the components of the system work together. The assumption is the units or components have been tested on a stand-alone basis and work correctly. When a full requirement is met and is validated during the integration test, the section of the test procedure may be reused during the later system level acceptance test. The toughest part of their role is not to ensure that what is being done is correct, but to figure out what is not being done and ought to.

17.2. Acting as communicators to locate and resolve problems across interfaces

Effective interface management is critical to the timely completion of these states. When problems show up, the systems engineer takes the lead in analyzing the symptoms to determine the root cause, which might include:

- A unit (hardware or software) does not work as specified.
- The units work correctly on a stand-alone basis, but there is a problem at the interface in one or both units.

When the systems integration state begins, all units are supposed to have been tested and shown to be working. During integration a unit may not work because of an:

- ***Interface problem*** - which was not found by analysis during the design and build states. This situation tends to occur when systems engineering ends at SRR.
- ***Internal unit defect*** - which was not found during unit testing. In this situation, there may be more than one problem. Someone in the organization reported to the customer that the units were working and were ready for integration. The integration process has just shown that one or more units themselves are not working. Whoever reported to the customer that the units were working lied to the customer. If the customer is not told right away that the problems are within the units, this situation may constitute a false claim (Section 19.5.1).

[1] In the 'A' paradigm

17.3. Facilitating communications between the different specialized departments

The systems engineer ensures that the barriers between the different groups are down. For example, at LuZ, the development group needed to exercise the control console software under development. The team wanted to take time out and develop a field simulator that would emulate a number of Local Controllers (LOC) connected to a cable. They estimated the task would have taken about two weeks. At the same time, the manufacturing department needed to burn-in each LOC for a week before shipping them from Jerusalem to California. Their first thoughts were to leave each LOC powered up for a week, then re-run the unit test when the week was up. All LOCs passing the test were to be shipped; failures would then be treated as failures during unit tests.

After some discussion, an agreement was reached between the two departments to burn-in the LOCs concurrent to the Control Console Test. Two control consoles were set up in the development department and connected to a simple serial data switch. A long cable was stretched between the two departments, and the LOCs being burned-in were connected to the Control Console via the switch. One control console exercised the LOCs most of the time using software that had been validated. When the development department needed to access the LOC's, they threw the switch and had access. The LOCs didn't care who was accessing them. When a LOC failed during the burn-in, the development department notified the manufacturing department almost instantaneously, and data was available about the time of failure. This arrangement:

- Provided a win-win situation.
- Avoided the need to develop the field simulator.
- Avoided a schedule slippage of at least two weeks
- Avoided salary costs of at least sixteen man days.

The role of the systems engineer is to identify these types of situations and recommend appropriate actions.

17.4. The cataract approach to build planning

Implementation and delivery of systems are often performed in partial deliveries, commonly called "Builds". Each successive build provides additional capabilities. Planning builds requires allocating system level requirements to builds and documenting the allocation in a Build Plan. MIL-STD-2167A (Section 4.4) provides guidance on build planning for software systems, and may be used as a guideline for systems as a whole. The cataract approach to Build planning may be likened to a Rapid Prototyping scenario in which the requirements for each Build are frozen at the start of the Build.

The cataract approach however, is more than just grouping requirements in some logical sequence and charging ahead. Build plans must be optimized on the product, process and organization axes to:

- Make use of the fact that typically, 20 percent of the application will deliver 80 percent of the capability (Arthur, 1992) page 15) by providing that 20 percent in the early builds.
- Allow the waterfall approach (Section 3.2) to be used to plan the Build. This approach is tried and true over a short time frame on a small project.
- Produce a Build with some degree of functionality that also can be used by the customer in a productive manner. For example, the first Build should as a minimum, provide the user interface and the shell to the remainder of the functions. This follows the rule of

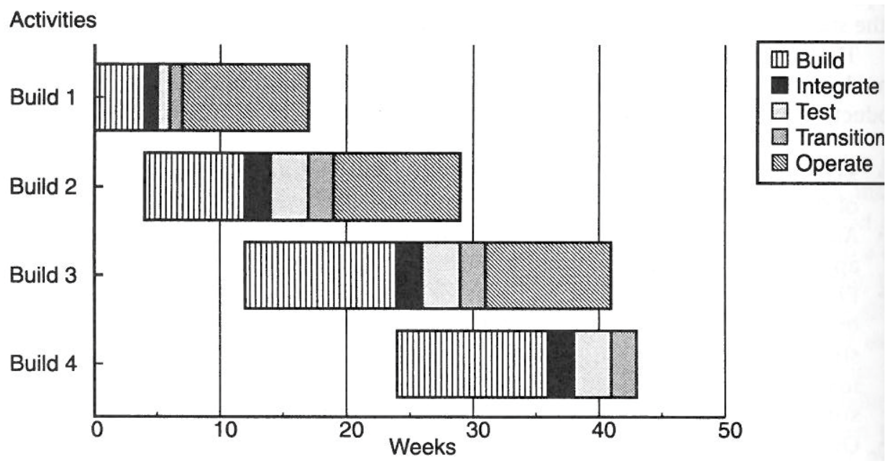

Figure 17.1 Coordinated task activities

designing the system in a structured manner and performing a piecemeal implementation.

- Optimize the amount of functionality in a Build (features v. development time).
- Allow a factor for the element of change.
- Maximize cohesion and minimize coupling of units. This minimizes changes to the units and tends to avoid the need to regression test the whole previous Build.
- Minimize the cost of producing the Build.
- Balance the number of personnel available to implement the Build (development, test and systems engineers) over the SDP to minimize staffing problems during the SDP.

These activities will produce a time line that is as smooth as the one shown in Figure 17.1. Each system level requirement will take a different amount of time to implement, accurate estimates of time to implement requirements are important. Task planning is based on objectives and results (Section 9.4.9), rather than activities will help provide this data.

In a typical project, the states in Figure 17.1 based on the Waterfall view break down as follows:

- ***Design and build*** - A parallel effort in which:
    - ***Software*** - Turns the requirements into code, and evaluates and perhaps incorporates Commercial-off-the-shelf (COTS) software.
    - ***Hardware*** - May be working with disk drives or other storage elements, networks, workstations and custom hardware elements.
    - ***Test*** - Developing test plans.
    - ***Systems engineering*** - are coordinating, technical performance analysis and measurement, change assessment, risk management, anticipatory testing (Chapter 10) and documentation; namely all the functions described in this chapter.
- ***Integrate*** - The hardware and software units are integrated and their working together is verified.
- ***Test*** - The integrated system is tested prior to acceptance by the customer.
- ***Transition*** - The system is turned over to the customer.
- ***Operate*** - The system is operated by the customer.

During the first Build, the activities are straight forward. Once the design of a Build is over, and the system turned over for integration, the design team assists with the integration, while their main effort starts to work on the design of the next Build.

Problems tend to show up during the integration and test states. When a problem is noticed, a Discrepancy Report (DR) is issued. This DR may be resolved by fixing the defect in the current Build before delivery, or assigning it to be fixed in a subsequent Build. In either event, the amount of work performed is increased (the cost of rework). When looking at the work load from this perspective, the importance of fixing defects early in the process (Section 10.6) is substantiated.

Project personnel move from one build to the next. Ideally the Builds are sequential with no wasted time between them; the development team moves from one build to the next, as does the anticipatory testing team. Each Build may pass through the elements of the waterfall process. The customers tend to get more and more involved with the system during successive Builds.

The costs of the later Builds may be estimated more accurately than those of earlier builds if product or results based cost measurements (Section 9.4.10) are taken during the earlier Builds. Consequently the planned and actual costs of the later Builds will tend to converge. The project may still go over budget, but at least there will be fewer surprises.

17.5. Assessing the impact of changes

The only thing constant in life is change. Design is under way and any change will affect future work, and may affect present and past work. Changes in requirements during the SDP (Section 5.8) constitute a major risk to a project and tend to cause, and hence are related to, cost and schedule impacts. As the development proceeds down the SDP, the cost of making changes increase since parts of the system have already been built and may have to be scrapped.

The task of assessing the impact of changes also becomes more complex. This is where money “saved” up-schedule tends to start to hurt the project. Money is usually not spent up-schedule on adequate documentation or on adequate Computer Enhanced Systems Engineering (CESE) tools, which makes change management difficult at this time, and hence expensive.

The role of the systems engineer is to work with the engineering specialists, the configuration control department and program management to assess the effect of a change and provide a cost and schedule impact assessment report (Section 5.8).

Effective implementation of this role will require the use of computer software, namely:

- ***CESE Tools*** - may be used to assess the impact of changes in requirements.
- ***Project management software*** - can be used to compare the differences between the pessimistic and optimistic times in performing each of the revised project tasks. When a task has a risk of overrunning its allocated time, the software can play a role in assessing some “what if” scenarios. Different resource allocations can be suggested and evaluated in a very short time to try to shorten the time to perform the task. If that doesn’t become possible, then other tasks may be changeable to ensure that the overall schedule is maintained.
- ***Spreadsheets*** - may be used to assess cost impacts and test potential implementation scenarios.

17.6. Performing hardware-software trade-offs

By virtue of knowledge of both hardware and software the systems engineer can make optimal trade-offs between hardware and software as appropriate. For example, a high speed data processing function may perhaps be implemented by:

- ***Build*** - Designing a printed circuit card containing custom high speed digital circuitry.
- ***Buy*** - Purchasing a COTS product containing a Digital Signal Processing Integrated Circuit (IC) and circuitry and using a mixture of COTS and custom software.

The role of the systems engineer is to analyze the effectiveness of each approach for the organization at a particular time and with specific available resources, and recommend the optimal approach.

17.7. Test planning

The role of the systems engineer is to assist the test department in developing test plans (Section 12.6) for:

- The system.
- Each individual sub-system.
- Coordinating the development and use of test tools.

When the system requirements are written to comply with the requirements for writing requirements (Section 12.4.1) the task of building a requirements-test matrix is simple, since a test will verify compliance of a whole requirement. When, as in the real world, the requirements are defective, the task of ensuring that the delivered system meets the customer's needs is more difficult. Tests have to be designed for partial requirements, and there is no simple way to effectively ensure all sections of all requirements have been tested.

17.8. Planning the transition to the operations and maintenance state

Systems engineers in the Anticipatory Testing department work with operations personnel to plan the transition of the system after acceptance testing. This may occur in Builds, so the transition may be considered as an upgrade. Should the system be supporting operations, non-interference with operations maybe an additional complexity.

17.9. Summary

In these states of the SDP, systems engineers move from the design organization to the Anticipatory Testing department. Others work with the program manager to optimize the activities on a total project basis. Although the number of systems engineers decreases during these states, they still have an important part to play as the glue that holds the project together. Important aspects of their work include:

- Interface management.
- Measuring technical performance.
- Ensuring the design meets the requirements.
- Facilitating communications between the different groups working on the project.
- Optimizing build planning.
- Managing change
- Performing design trade-offs.

17.10. References

Arthur, L. J., *Rapid Evolutionary Development*, John Wiley & Sons, Inc., 1992.