Using Organizational Engineering to Build Defect Free Systems, On Schedule and Within Budget

Joseph E. Kasser, D.Sc., C.Eng., CM.
University of Maryland University College, University Boulevard and Adelphi Road,
College Park, MD. 20742-1614

Abstract - Today's software and systems development life cycle paradigm is characterized by large cost overruns, schedule slips, and dramatic performance deficiencies in weapon, C4I, and automated information systems. This paper describes an alternative paradigm that can produce defect free systems on schedule and within budget.

I. INTRODUCTION

According to the U.S. Department of Defense, today's software and systems development life cycle (SDLC) paradigm is characterized by big cost overruns, schedule slips, and dramatic performance deficiencies in weapon, C4I, and automated information systems [1].

Most people do not realize **there is an alternative** SDLC paradigm. The use of elements of this alternative paradigm enabled the design, development, production, and installation of a network of approximately 600 microprocessors controlling the solar collector section of the world's first commercial Solar Electrical Power Generating Station (SEGS-1) on schedule and within budget, half way around the world from the development location. The early phases of the SDLC took place in Jerusalem, Israel, the installation was near Barstow, California. The system worked first time on initial installation with only a single Discrepancy Report (DR)[1] in spite of geographic, cultural and language difficulties. In addition, the control system was optimized for a cost saving of at least $300,000.

The alternative SDLC paradigm uses an *integrated product-process* **and management** approach. Since systems engineering techniques are used on the organization as well as on the process and the product, the paradigm is called *organizational engineering* and:

- May be used with conventional or object oriented design methodologies.
- Applies to large and small systems.

The elements of the *organizational engineering* paradigm discussed in this paper are:

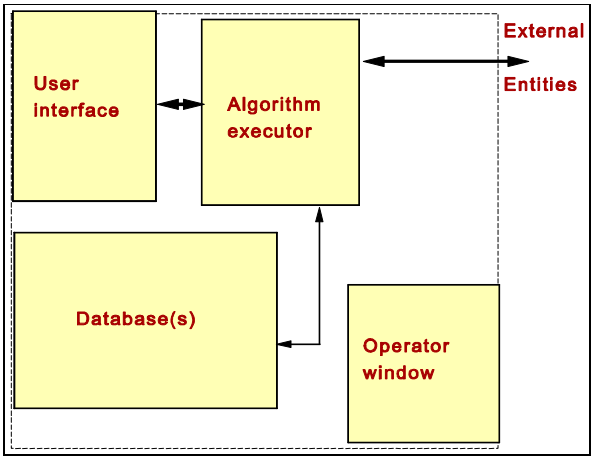
- Partitioning the system.
- Partitioning the implementation.
- Proactive progress management.

II. PARTITIONING THE SYSTEM

For any system or subsystem, the observer determines its boundaries for the sake of simplifying the analysis and design activities. The first level of decomposition of the system is always as shown in Figure 1. Systems may contain up to five top level subsystems as appropriate, namely, the:

- User Interface
- Algorithm executor
- Databases(s)
- Operator window
- External interfaces

Figure 1: Top level system decomposition



A. User interface

The data display and entry device(s) which interact with the users. A single group develops the user interface to ensure consistency.

[1] For a bad type of cable connector.

B. Algorithm executor

The top-level subsystem that carries out the work the system is built to perform.

C. Database(s)

The database components of the system.

D. Operator Window

A window into the system. It can display all status, alarm, error and event states, and the contents of buffers. It should be used as a diagnostic during both the development and the maintenance and operations phase of the system life. It must be designed to display data in a hierarchical manner, with the top level being a simple display of the status of the system. For example, in condition:

- *"green"*. Everything is operating according to specifications.
- *"yellow"*. Minor failures are present, but the system is operational. For example, a failure has occurred, but the redundant component has been activated, or an interface has failed and part of the system is not operational.
- *"red"*. The system is non-operational.

The Operator Window also serves as a major troubleshooting tool both during system commissioning and operational troubleshooting. This feature reduces the need to develop custom tools to test portions of the system, hence reducing the cost of the system.

E. External interfaces

The interfaces to the external elements to the system.

III. RULES FOR SUBSYSTEM DECOMPOSITION

The rules for decomposing the major subsystems are as follows:

- Minimize coupling and maximize the cohesion.
- Consider the operator as part of the system.
- Use self-regulating subsystems.
- Use railroad buffers for signal passing.

Consider each of them.

A. Minimize coupling and maximize the cohesion

Minimize coupling and maximize the cohesion of the functions performed by lower level subsystem elements. This approach is based on the Ward and Mellor Methodology [2] and fits nicely into both object oriented and conventional design approaches.

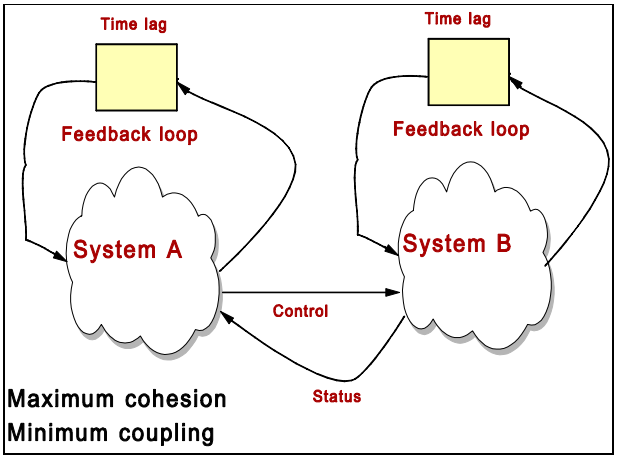
B. Consider the operator as part of the system

This approach allows early builds of a system to perform functions manually, and then provides automated capabilities in subsequent builds as more is learned about the system's behavior. It also allows systems to be coupled in a well-defined manner. For example, one system may act as "the operator" for a second system.

C. Use self-regulating subsystems

Subsystems are designed to perform their tasks in a self-regulating manner. The rules for (minimizing) coupling and (maximizing) cohesion must be observed. The subsystem transmits status information about itself, and receives command instructions from other subsystems. This approach, shown in Figure 2 has many variations. For example, consider the following examples:

Figure 2: Self-regulating systems



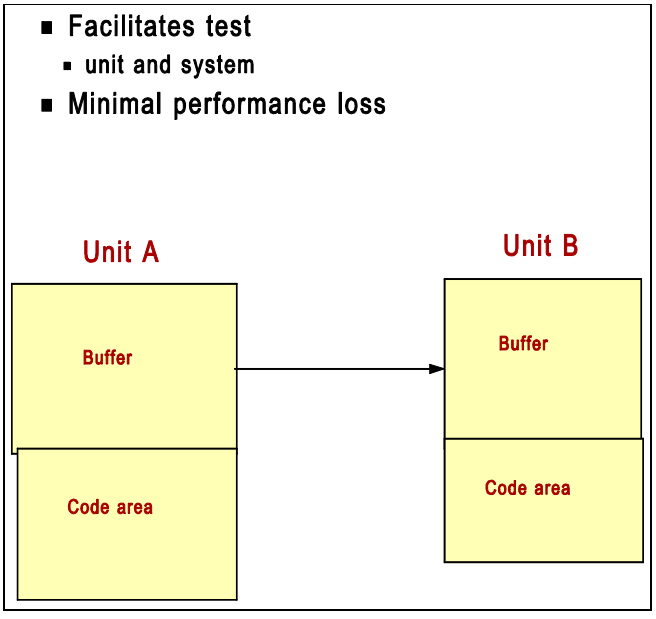
1) Spacecraft or missile control. In an early implementation of a family of spacecraft for communications or observations, System A is on the spacecraft and System B is on the ground. System B performs complex control and monitoring functions that System A cannot. Some System B functions may be even be performed by human operators and analysts. As technology matures, or new technology becomes available,

some System B functions are migrated into System A. The advantages of this approach include:

- Most of the requirements for later generations are known, algorithms have been developed and tested code and requirements may be reused for replacement and later generation spacecraft.
- Faster control responses.
- In case of an onboard malfunction of the migrated functions, System B is still available on the ground to take over.

2) SEGS 1 Sun tracking. Each of the SEGS 1 collectors had to be positioned within "0.1 degree of the Sun. The sun sensor that detected when the array was pointed at the Sun was mounted on the collector. There was no specification for the vibration of the collector due to wind or internal mechanical causes. Each sensor contained a pair of photo diodes and a shield. There was no specification on the sun-sensor other than an uncalibrated output curve showing relative change of output with sun angle as the Sun passed across a typical prototype sensor. The computed pointing angle for each collector was a function of the mounting accuracy of the sun sensor, the latitude and longitude of the site and the alignment of the collector array with respect to North. In addition, there was no specification for the accuracy of the measurement of these parameters. The principle of self-regulation was applied to develop a successful positioning algorithm that allowed for large tolerances on all these parameters.

Figure 3: Railroad buffers for signal passing



D. Railroad buffers for signal passing

This is a key element to the paradigm. All signals are passed between processes via buffers at both ends of the interface as shown in Figure 3. Modules are not allowed to build and transmit messages on the fly, or react to messages as they are received. The term “railroad buffers” is used because the interface area of a system may look like a freight yard at a railroad station. This element allows modules to be tested in both static (standalone) and a dynamic manner. The interface is tested by placing known data in a transmitter buffer and ensuring the data appearing in the corresponding receiver buffer is correct after the necessary event which initiates the transfer takes place. The modules are tested by placing data in the receiver buffer, and initiating the processing task. The data in the output buffer or the state of the module is then checked to see it meets the specifications for the processing task. This element has much in common with client-server techniques, but may cause a small loss in performance. ***These buffers may also be considered as a software equivalent of hardware test points***.

IV. PARTITIONING THE IMPLEMENTATION

The implementation of the system takes place according to the following approach:

- Design and implement the structure of the system.
- Flesh out in subsequent builds.
- Build a little test a lot.
- Anticipate changes.
- Use a budget tolerant methodology.
- Use the Cataract Approach.
- Prevent defects.

Consider each of these elements.

A. Design and implement the structure of the system.

The initial Build provides the structure of the system, the user interface and the operator window.

B. Flesh out in subsequent builds

Once the structure is in place, elements of the subsystems may be added in an incremental manner. The actions of the system may be observed via the user interfaces, and the operation verified by means of the operator’s window. The subsystems builds must be synchronized so that the user interface and operator window subsystems are updated in ad-

vance of the implementation of the algorithms accessed by the interfaces and operator window.

C. Build a little test a lot

With a working user interface, and operator window, elements of the system may be built and tested in an incremental manner using all the best practices. Conventional SDLC Builds take a minimum of six months. This approach allows for shorter Builds, of the order of weeks or even days. Software systems developers can ship sample copies to selected users for comments well before the release of the complete product.

D. Anticipate changes

Changes are continuous for various reasons which include changes:

- In the mission definition.
- Due to user reaction to early builds.
- Due to changes in the project budget.

Since changes are common to all projects, the SDLC must incorporate a change tolerant methodology to be successful. *Success is defined as a completed project, which meets its requirements, and is completed on schedule and within budget.*

E. Use a budget tolerant methodology

In today's systems engineering environment, budgets are decreasing while needs are remaining constant or even increasing. Consequently, systems must be designed so that in the event of budget reductions, there is no need to cancel the project and restart the development of a system with lower capability [3]. The budget-tolerant system development methodology is based on the traditional waterfall SDLC model with enhancements that require the consideration of the costs and the importance of the requirements as necessary elements in the analysis and design processes. The methodology consists of the following seven steps:

1. Determine the feasibility of a requirement.
2. Develop a complete set of requirements.
3. Prioritize the requirements.
4. Cost each requirement.
5. Establish a baseline.
6. Use the cataract approach to build planning.
7. Use proactive progress management.

Steps 1 – 3 are traditional and need not be discussed here. Step 4 is new and self-evident. Step 5 is also traditional. Consider the last two steps in the sequence.

V. THE CATARACT APPROACH

The Cataract Approach plans the system implementation in a series of Builds wherein each Build contains a full waterfall or mini SDLC [4]. *This approach allows changes to occur under configuration control*. 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. This 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 dimensions as follows:

- Use the waterfall approach for each Build.
- Implement the highest priority requirements in the earlier Builds. Then, if budget cuts occur during the implementation phase, the lower priority requirements are the ones that can readily be eliminated because they were to be implemented last.
- Make use of the fact that, typically, 20 percent of the application will deliver 80 percent of the capability by providing that 20 percent in the early Builds [5].
- Produce each Build with some extra degree of functionality that it can be used by the user (customer) in a productive manner. This follows the rule of designing the system in a structured manner and performing a piecemeal implementation.
- Allow a factor for the element of change. Optimize the amount of functionality in a Build (features versus development time).
- Minimize the cost of producing the Build. Balance the number of personnel available to implement the build (development, test, and systems engineers) over the SDLC to minimize staffing problems during the SDLC.
- Prevent defects - Traditional Quality Assurance and testing functions are independent from the Development effort and act after the fact. Consequently, errors are first made and then corrected. This means the elements in the Work Breakdown Structure are planned and budgeted as one-time efforts, yet are in fact performed twice, resulting in overruns and delays. The anticipatory testing approach combines prevention with in-process testing in a synergistic manner to eliminate defects in two ways [4], namely, by:

    - *Testing* - The earlier the testing can be performed in the SDLC, the greater the reduction in the penalty costs of not doing it right the first time, so do the

testing at well established checkpoints in the SDLC. These check points include:

- Concept reviews.
- Implementation (Management) Plans.
- Requirements reviews.
- Design reviews.
- Code walkthroughs.
- Code inspections.
- Test Plan reviews.
- Test Procedure reviews.

- *Prevention* - According to Philip Crosby, prevention includes [6]:

  - Sensitization to probable defects (training).
  - Improving the process
  - Risk management

VI. PROACTIVE PROGRESS MANAGEMENT

Many of the measurements made in current projects enable suppliers to improve their processes and products. These may be used for proactive progress management in several ways, including i**mprovement of the Quality Index** - These measurements may be used to lower the cost of doing work by improving the *Quality-index* of the organization [7] which is a three dimensional measure of the:

- Effectiveness of the production process.
- Degree of conformance of the product to its requirements.
- Effectiveness of the organization in which the process takes place.

VII. CATEGORIZED REQUIREMENTS IN PROCESS

The budget tolerant methodology categorized requirements by cost (to implement) and priority [3]. Tracking the implementation of the categorized requirements has led to a measurement approach that has **the potential of providing a measurement of completeness of the product at any of the milestones in the SDLC**. This approach is called *categorized requirements in process* (CRIP) [8]. The four step CRIP approach is:

- Categorize the requirements.
- Quantify each category into ranges.
- Place each requirement into a range.
- Monitor the **changes** in the state of each of the requirements **between** the SDLC reporting milestones.

The first part of the approach avoids the problem of comparing requirements of different complexities. **The last step is the key element in the CRIP approach.**

A. Categories

Typical categories are:

- **Priority** of the requirement to the customer.
- **Complexity** of the requirement., i.e. the difficulty of implementing the requirement.
- **Cost to implement** the requirement by the supplier.

B. Ranges

Each category is split into no more than ten ranges. Thus, for:

- **Priority** - requirements may be allocated priorities between one and 10.
- **Complexity** - requirements may be allocated estimated complexities between “A” and “J”.
- **Cost to implement** - requirements may be allocated estimated costs to implement values between “A” and “J”.

The ranges are relative, not absolute. Any of the several techniques for sorting numbers of requirements into relative ranges may be used. The act of categorizing the requirements into relative ranges is in itself a beneficial act. For example if a low priority requirement has a high cost to implement, the need for the requirement in the system should be reevaluated. Cost may not always be the same as complexity. For example, the use of Commercial-off-the-shelf (COTS) products may lower the cost, but not change the complexity of a requirement.

The buyer and supplier may determine the range limits in each category. A requirement may be moved into a different range as more is learned about its effect on the development during the implementation phase. Thus, the priority of a requirement or the cost to implement may change between SDLC reporting milestones. However, **the rules for setting the range limits must not change during the SDLC.**

C. States

The state of implementation of each product requirement varies during the SDLC as follows:

- **Identified** - A requirement has been identified, documented and approved.

- **Working** - The supplier has begun work to implement the requirement.
- **Completed** - The supplier has completed work on the requirement.
- **In test** - The supplier has started to test the requirement.
- **Accepted** - The buyer has accepted delivery of part of the system (a Build) containing the implementation of the requirement.

The categories, ranges, and states of each of the requirements are presented in tabular format (a CRIP chart) at reporting milestones (major reviews or monthly progress meetings) as shown in Table 1 where each cell in a range shown in the Table contains the following three elements:

- **Expected** - The number of requirements planned to be in the Implementation State, based on the previous reporting milestone.
- **Actual** - The number of requirements in the implementation state.
- **Planned** - The number of requirements planned to be in the Implementation State in the following reporting milestone.

TABLE1
THE CRIP CHART

	Identified	Working	Completed	In test	Accepted
A	26-23-4	6-5-9	5-12-45	12-15-1	10-0-20
B	43-2-4	34-6-9	5-12-3	12-5-0	0-0-0
C	12-5-46	2-9-18	4-4-4	0-0-0	0-0-0
D	5-12-5	4-9-12	2-2-2	0-0-0	0-0-0
E	6-12-9	12-9-4	5-9-15	0-0-0	0-0-0
F	5-12-34	6-12-4	8-12-8	0-0-0	0-0-0
G	5-12-5	1-1-1	6-9-12	0-0-0	0-0-0
H	6-9-19	2-9-10	6-9-15	0-0-0	0-0-0
I	12-5-3	6-9-12	5-4-3	0-0-0	0-0-0
J	0-0-0	0-0-0	0-0-0	0-0-0	0-0-0

VIII. EXAMPLES OF USE

The CRIP chart may be used on a standalone basis or in accordance with budget and schedule information. For example, if there is a change in the number of:

- **Identified requirements** and there is no change in the budget or schedule, there is going to be a problem. For example, if the number of requirements goes up and the budget does not, the risk of failure increases. If the number goes down, and the budget does not, there is a financial problem.
- **Requirements being worked on** and there is no change in the number being tested, there is a potential supplier management or technical problem if this situation is at a major milestone review.
- **Requirements being tested** and there is no change in the number accepted, there may be a problem with the supplier's process.
- **Identified requirements** at each reporting milestone, the project is suffering from poor buyer requirements.

IX. CRIP CHARTS AND CONTRACTOR PAST PERFORMANCE

The CRIP chart numbers at major milestones can provide objective evaluation criteria of past performance and force cost effective behavior. Consider the following examples.

A. Requirements are met or they are not.

Waived requirements are not accepted requirements by definition. Hence the process of 'waiving requirements that a supplier cannot meet at the end of a project' shows up in the CRIP chart when the number of accepted requirements at the pre-completion milestone is different from the number of requirements accepted at the completion milestone.

B. Requirements creep.

Requirements creep shows up on the CRIP chart in the number of identified requirements at major milestones. If requirements creep is a negative evaluation criterion in a cost plus contract, then it is in the supplier's interest to:

- Identify a full set of requirements as early in the program as possible. The supplier is now motivated to get it right the first time. The underlying information (reasons for the changes) will not be enclosed with the CRIP chart numbers.
- It may be possible to develop a CRIP rating based on the difference between the number of system requirements identified over the SDLC, and the number accepted at the completion of the project and the total cost of the project as a function of the number of categories and ranges within each category. However, this rating will require 'a CRIP standard' for future contracts.

If the CRIP charts show that all the buyer's requirements are met, yet the subjective past performance rating is poor, there may be a problem with the buyer's project team. This is something the Agency should investigate.

C. Time tells.

If requirements can only be tested over time, such as mission effectiveness requirements and failures, the buyer can update the CRIP chart in the past performance database to reflect the status of the requirements after several months of use.

The CRIP approach has the following advantages, it:

- May be used at any level of system decomposition.
- Provides a simple way to show progress or the lack of it, at any reporting milestone. Just compare the numbers and ask for an explanation of the variances.
- Provides a window into the project for top management (buyer and supplier) to monitor progress.
- Identifies some management and technical problems as they occur, allowing proactive risk containment techniques.
- May be built into requirements management, and other computerized project and design management tools.
- May be built into Government contracts via the SOW. Here falsifying entries in the CRIP chart to show progress is fraud.

X. DISADVANTAGES OF THE CRIP APPROACH

The CRIP approach has the following disadvantages, it:

- Is a different way of viewing project progress.
- Requires categorization of the requirements.
- Requires a process.
- Requires configuration management.

XI. CONCLUSIONS

Defect free systems can be built on schedule and within budget. However, to do so requires integrating the product, process and organization dimensions. In addition, this integrated state is not readily achieved in today's systems and software development organizations.

REFERENCES

[1] Department of Defense, *The Program Manager's Guide to Software Acquisition Best Practices*, Version 1.0, July 1995.

[2] Ward, P.T., Mellor, S.J., *Structured Development for Real-Time Systems*, Yourdon Press Computing Series, 1985.

[3] Denzler, D.W.R., Kasser, J.E., "Designing Budget Tolerant Systems", *The INCOSE 5th International Symposium*, St. Louis, MO, 1995.

[4] Kasser, J.E, *Applying Total Quality Management to Systems Engineering*, Artech House, 1995.

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

[6] Crosby, P.B., *The Art of Getting Your Own Sweet Way*, Second Edition, McGraw-Hill Book Company, 1981, p 131.

[7] Kasser, J.E., "There's No Place For Managers in a Quality Organization", *The 9th Annual Conference on Federal Quality*, Washington DC., 1996.

[8] Kasser, J.E., "What Do You Mean You Can't Tell Me How Much of My Project Has Been Completed?", *The 7th Annual Symposium of the International Council on Systems Engineering*, Los Angeles, 1997.