DESIGNING BUDGET-TOLERANT SYSTEMS

David W. R. Denzler
Computer Sciences Corporation
7700 Hubble Drive
Lanham-Seabrook, MD 20706

Joe Kasser
The Anticipatory Testing Corporation
P.O. Box 3419
Silver Spring, MD 20918

ABSTRACT

In today's systems engineering environment, budgets are decreasing while needs are remaining constant or even increasing. This paper discusses the concept of designing systems 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. Instead, the least-important requirements are easily eliminated.

INTRODUCTION

Budget changes lead to changes in performance and vice versa. These factors are two sides of the same coin, yet this very simple linkage does not seem to have been made to date. As a matter of fact, the traditional development philosophy tends to keep the cost information isolated from the people who set requirements. One purpose of systems engineering is risk mitigation, yet mitigating the risks introduced by a budget decrease tends to be ignored in the system development life cycle (SDLC).

In today's environment of *business process reengineering, enterprise engineering*, and *reinventing government*, we really need to modify our methodology to be able to optimally mitigate the effect of a budget decrease. Before considering this modification, consider the effect of a budget decrease with no accompanying methodology to minimize its effect.

EFFECT OF A BUDGET DECREASE

The effect of a budget decrease is change. Some functionality will have to be given up, i.e., requirements will have to be deleted. The change may take one of two forms: (1) cut a certain amount of money from the program, which directly affects the system engineering process within the organization with ramifications on the staffing and schedule or (2) cut some requirements from the system under development, which has a direct impact on the product and an indirect impact on staffing and schedule.

The impact of a change affects requirements, documents, work breakdown structure, builds and deliveries, and cost and schedule, depending on the point in the SDLC in which the change occurred.

- Changes in high-level *requirements* will affect lower level requirements and may affect implementation requirements.
- *Documents* affected could include management plans, operations concepts, manuals, test plans, and procedures.
- *Work breakdown structure* elements include all SDLC activities.
- For *builds and deliveries*, the implementation sequence may be changed to the point where a build does not add any value to the system, so the cost of testing, releasing, and delivering the build may no longer be economical.
- The effects of the changes will show up as a variance in the *cost and schedule*.

THE PRIME DIRECTIVE

The prime directive is to always make a decision about any change to the system with the full knowledge of its impact on the system. This directive may be thought of as a top-level requirement for project management. The program manager must be concerned with the effect of the change in the three dimensions of systems engineering [Kasser, 1995a]:

- **The Product**. There will be an impact on the performance.
- **The Process**. There will be an impact on the schedule.
- **The Organization**. There will be an impact on the staffing.

Decisions often have to be made in a short time. If incomplete information is available, the program manager quickly evaluates it and makes a less-than-optimal decision, which results in subsequent waste further into the project. This waste results from the lack of information or from not producing a document that contains that information [Kasser, 1995b].

Consider what it takes to make an optimal decision. When a budget cut occurs, everyone agrees that effective change management will minimize the impact of the change and the loss of functionality. This effective change management can only be achieved if, when the systems engineers develop a complete set of requirements for the proposed system, they also develop the following additional information for each requirement:

- Priority of the requirement.
- Dependencies with other requirements.
- Cost to implement the requirement.

Consider the changes to the current methodology to obtain and make use of this additional information.

THE CURRENT METHODOLOGY

During the initial phase of the project, often called the concept definition phase, the system concept document is developed. This contains statements explaining why the system is necessary and what it must do. The document provides a set of goals and objectives for the system, including a brief rationale of why it should be achievable. This first system-level needs document may also be called a systems and operations concept document or a concept feasibility report.

The project manager also develops a system engineering management plan that describes

- The resources available to solve the need.
- How the resources will be allocated.
- The system engineering processes to be used (implementation plan).
- The constraints on the SDLC (e.g., schedule, budget).

Once these activities are complete, a system requirements specification is developed during the requirements definition phase that contains the requirements (i.e., actual requirements for the system to be built) and evaluation criteria (i.e., criteria to be used to select the optimal system from the alternatives to be developed from the requirements).

The system requirements specification is then reviewed with all stakeholders in the proposed system at a formal system requirements review. In progressive projects, these involved stakeholders include

- The customer.
- System product users (not always the customer).
- Requirements developers, usually systems engineers.
- System operators.
- System developers (software and hardware engineers).

The system requirements review formalizes the agreement that the requirements are correct. When the system requirements review is complete, the candidate architectures are developed. Each alternative system is then evaluated against the evaluation criteria and the optimal system is chosen. The optimal system=s preliminary design is presented in a formal preliminary design review to an audience of stakeholders.

Once designed, the system is ready to be built. Implementation and delivery of systems often are 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 provides guidance on build planning for software systems and may be used as a guideline for systems as a whole.

BUDGET-TOLERANT SYSTEM DEVELOPMENT METHODOLOGY

The budget-tolerant system development methodology is based on the traditional waterfall SDLC model, but with significant enhancements. These enhancements require consideration of costs and the importance of requirements as necessary elements in the analysis and design processes.

The methodology consists of 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 effective change management techniques

The following paragraphs describe these steps.

Determine the Feasibility of a Requirement. In the traditional SDLC approach, the tacit assumption is made that we know what we want and what is possible when we write the system concept. However, a comparison of any project's system concept with the final system would probably reflect that we seldom have the promethium powers of predicting the future to accurately state exactly what really is needed, and possible.

The Applied Physics Laboratory of the Johns Hopkins University has for many years successfully used a technique for new system feasibility and needs analysis [Denzler, 1973]. This technique involves building proof-of-principle systems that give a 'fuzzy' representation of a system that the U.S. Navy thinks it might need. The first test of this fuzzy requirement is whether the technology really exists to actually build the critical components of the system. If it does not exist, then spending a lot of money chasing a full military-standard SDLC technological Awindmill@ is avoided. Note that the proof-of-principle system usually provides only a partial functional representation. These proof-of-principle systems are then fielded on real Navy ships (with duct tape and bailing wire) with real sailors operating them.

After the field tests, the users of the system are extensively interviewed. What worked and what was useful becomes a requirement, what did not work indicates a new requirement must be written, and what was not used should be thrown out. The important point about this approach is that *failures* are as important as *successes* in learning what is really required within the realm of the possible. An important corollary to this point is that the degree of innovation is directly related to the degree of freedom to fail; at this time, the penalty for failure is minimized because it is detected during this phase. Field tests of proof-of-principal systems are an excellent mechanism for detecting failed and undesirable requirements before spending the money to implement them.

Develop a Complete Set of Requirements. In the landmark study of the impact of missing and incorrect requirements [Brooks, 1972] points out that the earlier in the development cycle that an error is made that is not detected until the system is built, the more costly it is to correct by orders of magnitude. Therefore, an important technique to avoid spending money building defects is to try to validate that the requirements in the system requirements specification are complete and consistent with each other. Because requirements are typically written as abstracted 'shall' statements grouped functionally, it is difficult to detect if any are missing or that their implementations will interact correctly.

A technique often used in space mission development [Denzler and Mackey, 1994] is to generate operations scenarios from the system requirements specification and try to determine if the desired operations will result. An operations concept document is often required as a companion document to the system requirements specification at the system requirements review. Presentations use the operations scenarios to illustrate the high-level validity of the system requirements specification. This graphical technique for developing and displaying operations scenarios [Denzler and Mackey, 1994] has been in use for more than 10 years. [Denzler and Vallone, 1995] shows their use throughout the SDLC.

Although operations scenarios may be used to detect missing requirements during the generation of the system requirements specification, prototyping may be used during this time to refine those requirements associated with the system's interaction with operators and product users.

This concept of prototyping accepts the fact that in the real world, what the customer really wants probably is not in the system requirements specification. In this activity, the prototype reproduces, in an interactive simulation, the look and feel of the product that the customer or operator will receive. This gives the customer a chance to indicate where the mistakes are in both requirements and implementation before the system is actually built. In this sense, the prototype becomes a virtual system, as described in [Andrews and Goeddel, 1994].

Prioritize the Requirements. Prioritize the requirements as early as possible in the SDLC, and confirm the priorities at the system requirements review. For example, if the needs are for a ship, then offensive and defensive capabilities must be prioritized. There is no need to rank every requirement against every other; grouping them in several categories is enough.

Quality function deployment is an excellent technique for prioritizing requirements. It originated in 1972 at Mitsubishi's Kobe shipyard [Hauser and Clausing, 1988]. Using this approach, Toyota Autobody reduced preproduction costs by more than 60 percent between 1977 and 1984 [Hauser and Clausing, 1988]. Quality function deployment:

- Is an effective technique for capturing, communicating, and understanding the customer=s requirements.
- Is a structured methodology to increase the probability that products will be designed to reflect customer's desires and tastes.
- Facilitates teamwork and a concurrent engineering approach, namely allowing marketing people, design engineers, and manufacturing staff to work closely together from the time the product is first conceived.
- May be used to force the customer to think about the real need for each requirement by allowing attributes to be assigned to each requirement, including:
    - A priority weighting
    - An estimate of the degree of technical difficulty of implementation (risk factor)
    - An estimate of the cost of meeting the requirement

As these attributes are identified, the customer can make an informed decision as to the real need for a particular requirement in light of budgetary or schedule constraints. Systems engineers take the lead using quality function deployment to translate customer requirements into technical requirements for each stage of product development [LaSala, 1994].

Cost Each Requirement. When the system requirements review is finished, several alternative architectures or candidate systems are identified. Each candidate system must be designed in sufficient detail to ensure that it is feasible within the constraints of available resources and capable of being realistically evaluated against the evaluation criteria.

An alternative architecture analysis is then performed. This process analyzes each candidate system to see how well it meets the system requirements and provides a rough order-of-magnitude estimate of the resources needed to build the candidate system; i.e., cost and schedule.

Establish a Baseline. The design is then baselined and presented at a preliminary design review. This particular review differs from the traditional preliminary design review in that life-cycle cost estimates and requirements priorities are included in the design trade studies. A detailed design-to-cost development is then initiated, where the highest priority requirements are selected until the sum of their costs to implement is within the appropriate margin of the total allowed cost. In this design exercise, the

- Cost of all selected requirements is computed for the entire life cycle of the system
- Most necessary requirements are those selected for implementation
- Builds are organized so that the most critical requirements are implemented first

Use the Cataract Approach to Build Planning. The major consequences of failing to control changes are moving baselines and confusion leading to cost escalation and schedule delays [Kasser, 1994]. Various approaches have been used to make up for the deficiencies of the waterfall approach in an environment of changes. These approaches include Rapid Evolutionary Development [Arthur, 1992] and Structured Rapid Prototyping [Connell and Shafer, 1989]. Both of these approaches use the system being delivered as the tool for communications between the customers and the developers and start with a deliverable prototype and build onto it.

The cataract approach [Kasser, 1995a] is an alternative approach. It involves planning the system implementation in a series of builds wherein each build contains a full waterfall or mini SDLC. This approach allows changes to occur, but in a controlled manner. The goal of the cataract approach is to optimize the factors involved to ensure as smooth an implementation path as possible.

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 axis.

- Implement the highest priority requirements in the earlier builds. Then, if budget cuts occur during the implementation phase, the lower priority portions 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 [Arthur, 1992] by providing that 20 percent in the early builds.
- Allow the waterfall approach to be used for each build. This tried-and-true approach works on a small project over a short timeframe.
- 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, at a minimum, provide the user interface and shell to the remainder of the functions. 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.

Use Effective Change Management Techniques. Once we start building the system, change becomes more complex because the impact of a change can obviate portions already built, as well as cause redesign of yet-to-be-implemented requirements. When a change request is made, the systems engineer performs a impact assessment what-if scenario. The priorities of the requirements and the major cost drivers are known, so change management is simply as follows:

- Budgetary changes: Identify lowest level requirements. Assess impact of deleting them. Sometimes work already completed may change absolute costs. Delete the lowest priority requirement(s) consistent with the budget reduction.
- Requirements changes: Assess cost and schedule impact of change. Assess priority of additional requirements.

An effective way to optimize the project implementation path is to use computer-enhanced systems engineering tools [Kasser, 1995a]. From the perspective of minimizing the effect of change, these tools provide the following capabilities:

- Automate requirements extraction from statements of work or other source documents.
- Detect changes made in the contents of two versions of the same source document.
- Record a complete history of all changes.
- Trace requirements.
- Make use of a common language for specialists from multiple disciplines to communicate while working together on a systems development project.
- Develop conceptual designs as block diagrams.
- Display the conceptual design from multiple perspectives; i.e., hierarchical, functional, documentation, and technical budget.

SUMMARY

In this brave new world of design-to-cost systems with ever-shrinking budgets, paradigms exist that allow us to develop new systems "better, faster, and cheaper." Just as our systems have had to become more integrated, our systems definition, requirements prioritization, and cost paradigms also must become more integrated. However, budget-tolerant systems are possible if we accept that we must reengineer our system development process.

REFERENCES

Andrews, Blake A. and Goeddel, William C. Jr., AUsing Rapid Prototypes for Early Requirements Validation.@ *4th Annual International Sym-posium of the National Council of Systems Engineering (NCOSE),* San Jose, CA, 1994.

Arthur, Lowell Jay, *Rapid Evolutionary Development*. John Wiley & Sons, Inc., 1992.

Brooks, Fred, *The Mythical Man-Month*. Addison-Wesley Publishing Company, 1972.

Connell, John L. and Shafer, Linda, *Structured Rapid Prototyping*, Prentice-Hall, Inc., 1989.

Denzler, David W. R. and Mackey, William, AAn Operations Concept Development Methodology Using a Graphic Process Flow Technique.@ *4th Annual International Symposium of the National Council of Systems Engineering (NCOSE)*, San Jose, CA, 1994.

Denzler, David W. R. and Vallone, Dr. Antonio, AOperations: The Missing Third of Systems Engineering.@ *5th Annual International Symposium of the National Council of Systems Engineering (NCOSE)*, St. Louis, MO, 1995.

Denzler, David W. R., *Evaluation of the TRANSIM Low-Cost Satellite Navigation System*, The Johns Hopkins University Applied Physics Laboratory, APL/JHU CP 027, May 1973.

Hauser, John R. and Clausing, Don, AThe House of Quality.@ *Harvard Business Review*, MayBJune 1988, 63-65.

Kasser, Joe, AGaining the Competitive Edge Through Effective Systems Engineering.@ *4th Annual International Symposium of the National Council of Systems Engineering (NCOSE),* San Jose, CA, 1994.

Kasser, Joe, *Applying Total Quality Management to Systems Engineering*, Artech House, 1995.

Kasser, Joe, AImproving the Systems Engineering Documentation Production Process.@ *5th Annual International Symposium of The National Council of Systems Engineering (NCOSE)*, St. Louis, MO, 1995.

LaSala, Kenneth P., AIdentifying Profiling System Requirements with Quality Function Deployment.@ *4th Annual International Symposium of The National Council of Systems Engineering (NCOSE)*, San Jose, CA, 1994.

BIOGRAPHIES

David W. R. Denzler is a Senior Consulting Engineer with CSC. He has a B.S. degree from New Mexico State University and has worked professionally as a systems engineer for 30 years. The majority of his career has been devoted to developing ground systems to support spacecraft operations and data processing for the Department of Defense, NASA, the European Space Agency, and the Maritime Administration. He has published several reports on satellite navigation in the National Technical Information Service and delivered papers to the Radio Technical Commission for Marine Sciences, Air Traffic Control Association, and NCOSE. He is a member of the Institute of Electrical and Electronics Engineers and the Institute of Navigation. He teaches system requirements generation in CSC's courses on systems engineering.

Joe Kasser is president of the Anticipatory Testing Corporation, an organization he founded to reduce cost and schedule overruns in systems engineering. He has spent the last 20 years applying total quality management to systems engineering, resulting in the achievement of cost-effective implementation of international and domestic aerospace, communications, and solar power systems. He is a recipient of NASA=s Manned Space Flight Awareness (Silver Snoopy) Award for quality and technical excellence. He is also an Institute of Certified Professional Manager's (ICPM's) Certified Manager and a recipient of the ICPM=s 1993 Distinguished Service Award. Parts of this paper were developed for his proposed doctoral dissertation at The George Washington University, as well as from his book, *Applying Total Quality Management to Systems Engineering*. Material from the book is reproduced with permission from Artech House.