Abstract

This paper presents a framework and hierarchy of techniques for accelerating systems engineering while conserving or increasing its effectiveness. It is organized around ways to shorten the critical path of tasks involved in a project’s systems engineering activity, including eliminating tasks (e.g., via business process reengineering, reusing assets, generating drafts, and prioritizing system scope); reducing time per task (e.g., via tools and automation, increasing parallelism, and Pareto 80-20 based work streamlining); reducing risks of single-point failures; reducing backtracking; activity network streamlining; increasing the effective workweek; improving staffing and incentives; and measurement-based continuous process improvement. It provides descriptions and examples of successful use of the framework and techniques.

Keywords: systems engineering; schedule acceleration; opportunity tree;

1. Introduction

The desire for more complex and sophisticated systems coupled with an urgency to deliver systems quickly with reduced budgets in an era of rapid changes in technology, threats, operational concepts, and workforce characteristics creates significant challenges for the systems engineering and development communities—challenges that are seldom met today. Recent GAO reports [1] indicate average United States Department of Defense (DoD) system delivery overruns of $300 billion/year cost and 21-22 months per project in delivery schedule. Additionally, needs for cyber security, physical self-defense, net-centric system interoperability, and rapid system adaptability for both existing and new systems

* Corresponding author. Tel.: 1-213-740-8163; fax: 1-213-740-4927.
E-mail address: boehm@usc.edu
portend of complex interdependencies between systems, higher costs, and longer schedules. Asymptotic growth in cost and schedule as systems near their performance and assurance limits is a common occurrence. While system interdependencies, cost, and schedule are very much intertwined, this paper focuses on opportunities to improve schedule performance in the development of new system capabilities and systems.

2. Background

The planning for the development of large complex systems (or new capabilities that cross multiple systems or subsystems) typically starts with a development concept and associated estimates for effort, cost, and schedule. However, as engineering progresses, there are often unexpected events or discoveries that lead to rework and backtracking (as illustrated in Figure 1) or tasks taking longer than expected to complete. This model is a bit oversimplified, given that the real world has partial dependencies and more complex constraints, but these do not cause major complications with respect to the use of the model to identify sources of cycle time reduction.

![Activity network with backtracking](image)

Fig. 1. Activity network with backtracking

The causes for rework/backtracking and schedule extension are many. However, identification of specific causes can lead to opportunities to refocus and streamline activities in ways that can shorten schedules. Opportunities for shortening the critical path from inception to fielding can be grouped and organized into a tree structure. This tree, often referred to as a Rapid Application Development (RAD) Opportunity Tree, is a hierarchical taxonomy of sources of cycle time reduction, which can be used as a framework for assessing various mixed strategies for tailoring a rapid application development approach to a given organization’s environment, culture, technology, and constraints. This taxonomy is developed in the context of software development as an activity network of tasks with backtracking and was inspired by a presentation by Motorola [2] on their corporate “10X” initiative to reduce their critical-path time by a factor of 10 (they only achieved factors of 3 to 4, but such factors are a definite improvement).
3. SE Schedule Acceleration Techniques

As mentioned above, there can be many opportunities to accelerate systems engineering as illustrated in Figure 2. Brooks' law [3] shows for software development that adding more people is often not the answer. It can similarly be shown that just adding more systems engineers does not accelerate systems engineering and reduce the schedule. Each major source of cycle time reduction shown in Figure 2 is elaborated below.

3.1. Eliminating tasks

Business process re-engineering can discover and eliminate non value-adding tasks, such as unnecessary purchase approvals or change control boards operating at too low a level. Reusing assets and automated applications generation can eliminate many tasks, but they require up-front investment in domain architecting and product line infrastructure. Design-to-schedule can also be highly effective, but again requires up-front investment in prioritizing features and architecting so that features can be dropped without ripple effects.

3.2. Reducing time per task

Reducing time per task can be addressed through technology or management. Tools and automation may be employed when existing tasks lack appropriate technology support and where such supporting mechanisms (e.g., interactive development environments, automated testing packages) are commercially available. When tasks involve getting people together to review or agree on system features or capabilities, but the people involved are not co-located, then the use of collaboration technology, groupware or wide-area workflow management tools may be helpful. On the management side, Pareto
80-20 analysis can be effective for work streamlining. For example, if 20% of the tasks cause 80% of the time delays, then task streamlining should be focused onto that 20%.

3.3. Avoiding single-point task failures

System development projects are sometimes prone to single point failures, which can negatively impact completion schedules. For example, hardware platforms or components can go down or fail at untimely moments (a.k.a., “Murphy’s Law”). Similarly, key project personnel such as lead system architects may leave the company or be pulled off to save another project. The basic way to mitigate single point failures is to provide some form of back-up or parallel capability. However, what is key is detecting or anticipating where these point failures may occur, and taking preventive measures, such as scheduling back-up hardware or providing an apprentice or “stand-in” to key project personnel. Hardware design and software inspections, primarily considered as a defect-detection activity, are also excellent for spreading key product knowledge across the project team.

3.4. Reducing backtracking

Rework is perhaps the most common form of time-sink that system development projects experience. Generally, rework does not add value. Thus, the challenge is how to minimize its occurrence. The following summarizes common causes of rework and actions that can be taken to reduce rework.

3.4.1. Timely decision making

When key engineering decisions are not made in a timely manner, engineering teams will often proceed at risk by making assumptions about the pending decision. When the final decision does not match the engineering assumptions, significant rework can occur. By going slowly to explore alternatives associated at key decision points then making the key decisions based on analysis results, engineering teams can move out quickly (and often in parallel) through the next phases with minimal risk of rework due to incorrect assumptions.

3.4.2. Rework due to errors

If rework is due to errors of various kinds, it can be reduced by developing and employing a matrix or taxonomy of errors and corresponding repair actions. While repairs may seem obvious to task experts, to task novices they can be of profound time-saving benefit. Even more effective, however, are techniques for error avoidance and prevention, and early error elimination when backtracking is easier. Process anchor points provide a management framework to help determine process goals, objectives (milestones), and progress measures.

3.4.3. Mature processes

Process anchor points and baselining help to establish attainable progress markers and overall project development “velocity.” In turn, project velocity typically increases as development processes mature, stabilize, and get reused. Such maturity happens most rapidly through top management commitment and resource investment to make it happen.

3.4.4. Tightening convergence loops

Rework can be reduced by tightening convergence loops or by articulating where progress disconnects occur. For example, when design reviewers review designs outside the presence of the designers, then some record of their discussions and understanding must be prepared, conveyed, and re-explained to the designers. Instead, it is far more efficient to co-locate design reviewers and designers together, or employ collaboration technology to help capture and fill-in the gap between the reviewers and designers.
3.5. Activity network streamlining

As displayed in Figure 1, project activity networks may reveal many possible paths to project completion. PERT/CPM tools and techniques may help identify critical paths in workflow, resource dependency, or schedule as well as opportunities to work on key tasks in parallel. When projects cross organizational boundaries, then project activities should preferably do so off the critical path. When activity networks get too “bushy” (when certain activities have a high number of input or output paths), then bottlenecks can occur. Decompose and spread out these high fan-in-fan-out nodes, and increase parallelism. One way to increase effective parallelism in developing a number of components is to ensure precise, well-validated component interface specifications. Then, the development effort for each component can proceed in parallel with minimal delays due to interface reconciliation or cross-component ripple effects.

Lastly, as the critical path determines the shortest route to project completion, look for ways to get time-consuming tasks off the critical path. This may be possible through task decomposition and parallelization, or through network reconfiguration. Some strategies, such as pre-positioning facilities, components, tools, experts, or data, may add somewhat to the cost but be worth it in schedule savings. A good example is “overinvestment” in reusable components. Typical component reuse ROI models conclude that you should expect to use a component at least 3 times to achieve a net payoff, but a lower expected number of uses is appropriate if schedule is more important than cost.

3.6. Increasing duration or number of workdays

Getting project staff to work harder is seldom a viable project management strategy. However, it is all too frequently employed. Staff burnout and untimely turnover can result, which in turn can slow progress and project completion. If non-critical development tasks can be outsourced to others (e.g., off-shore providers, or other corporate divisions in global locations), then so-called 24X7 or round-the-clock, round-the-globe development efforts may be possible. However, this usually requires some amount of up-front investment in creating a shared product vision, establishing the ground rules for inter-firm collaboration, and ensuring consistent technical decision-making in order to succeed. Similarly, swing-shift workers or swing-shift automation mechanisms may be employed, ala Microsoft’s nightly builds, developer-tester buddy system, and continuous automated testing. Finally, second or third shift developers, or “weekend warriors” can be employed during project surge or crunch periods, but with varying results and quality outcomes.

3.7. Better people or incentives

Better people usually can get a development effort done with less extraneous effort. However, everyone wants to hire the “best people” and most developers see themselves among the best, when in fact they aren’t. When you have to go with “best available” rather than “best,” the objective is to establish the sustained means for how to get the most from the people on the job. Motivation is key, but motivation backed by personal commitment and job/career incentives is most effective. This is a proven project management technique that is often not employed, since many systems engineering project managers lack management education and experience. The biggest payoffs from incentivizing and getting the best people come because they will be best at selecting and articulating how to employ the other cycle time reduction techniques noted above.

3.8. Transition to learning organizations
The “sixth” level of process maturity is the transition to a learning organization. Learning organizations can do more than optimize and manage their processes. They have instead cultivated a culture of continuous improvement and process redesign as routine activities, rather than as uncommon events. Learning organizations are adaptive. They are less concerned about whether they should adopt some new improved technology, simply because it’s new or it’s supposed to be better. Learning organizations have the resources, staff, and slack to be able to perform and master new development methods. They need not lock themselves into a single tool, technique, paradigm, or fad, since they maximize productivity and minimize cycle time as the normal mode of work.

4. Conclusions

A number of companies have applied combinations of these techniques to achieve success in rapid and effective systems engineering, in such areas as multimedia, aerospace, and supply chain management [4]. The RAD Opportunity Tree has been used as the basis for a different class of software cost-schedule estimation model called the Constructive Rapid Application Development Model (CORADMO) [5]. One feature of CORADMO is that its rating scales for eliminating tasks, reducing time per task, etc. also capture the slowdown effects of imposing non-value-adding tasks, using sequential vs. concurrent engineering, etc. If your customers or upper management say, “we’re here to help,” and try to impose such slowdowns, it is important to convince them that the Opportunity Tree works in both directions. Some further good references on rapid development include [6], [7], [8], [9], and [10].

References