Software Construction, Renovation,
& Maintenance

Scope, Budget, Schedule + 2

Jim Ladd

September 8, 2024

https://www.linkedin.com/pulse/scope-budget-schedule-2-jim-ladd-lkr3e/

There are very few “absolutes” in this universe. *Speed of light in a vacuum?* Probably. *Death and taxes*? Definitely. *N ÷ 0 != 0?* Sorry kids but “Error” is not the new zero.

There are even fewer absolutes in the realm of software development. To translate an idea into code is simple. To do it well is extremely difficult. Many years ago, three key dimensions were adapted from other industries to assist with managing this process: scope, budget, and schedule.

The **scope** defines what the project will deliver and outlines the specific requirements and features to be developed. It serves as a blueprint for what needs to be achieved and includes:

- *Requirements Gathering*: Identifying and documenting the needs and expectations of stakeholders.

- *Scope Creep Management*: Preventing unauthorized changes or additions to the project requirements that could lead to increased costs and extended timelines.

- *Deliverable Definition*: Clearly specifying what will be delivered at each phase of the project to ensure alignment with stakeholders' expectations.

The **budget** is the financial backbone of any software development project. It encompasses all costs associated with the project, including personnel, technology, tools, and overheads. Establishing a realistic budget involves:

- *Estimating Costs*: Accurately forecasting expenses related to development, testing, deployment, and maintenance.

- *Resource Allocation*: Ensuring funds are allocated efficiently across different phases of the project and among team members.

- *Contingency Planning*: Setting aside a portion of the budget for unforeseen expenses or risks that may arise during development.

The **schedule** outlines the timeline for the project's completion. It includes deadlines for each phase of development, from initial planning to final delivery. Key aspects of managing the schedule involve:

- *Defining Milestones*: Breaking the project into manageable phases with specific deliverables and deadlines.

- *Time Estimation*: Accurately estimating the time required for each task and incorporating buffer periods for unexpected delays.

- *Tracking Progress*: Regularly monitoring progress against the schedule to ensure that the project remains on track and adjusting timelines as necessary.

The often quoted absolute with scope, budget, and schedule is that only two of the three dimensions may be fixed. For example, a project may have a defined scope and budget, but the schedule remains flexible. Conversely, the schedule and scope might be set, leaving the budget adjustable. The theory (and widespread practice) is that by carefully managing these dimensions with the 2 of 3 constraint, project managers can successfully navigate the complexities of software development.

In my experience there are two other, little known, dimensions available for guiding software projects: productivity and quality.

Productivity refers to the efficiency and effectiveness with which software is produced. The following are examples of increasing productivity used in real-world projects:

- *Improve Meeting Efficiencies* – The first and most straightforward step is to eliminate or suspend meetings with low return on investment (ROI). Beyond that, employing basic meeting management techniques can enhance efficiency: set clear objectives, ensure that only the essential participants attend, document discussions and decisions, and follow up as needed. The most crucial aspect, however, is to continually evaluate and refine the meeting process to ensure ongoing improvement.

- *Increase Team Collaboration* – This subject may seem straightforward in principle but can be challenging to implement, especially with technical specialists. I've found that bypassing conventional "team building" gimmicks and focusing on leading by example is more effective. When someone in the enterprise reaches out with a request or question, I try to response as quickly as possible. I also conduct design sessions with various subsets of the team and make a point of holding regular one-on-one meetings with each team member. True collaboration hinges on trust, and building that trust takes time.

- *Broaden the Tool Pool* – While acquiring more or better tools may initially seem like a budgetary concern, these investments should be viewed as long-term assets that benefit multiple projects. I am consistently amazed by how my small consulting company often has more advanced tools than many of our much larger clients.

  There are also valuable opportunities to develop internal tools that can significantly boost team productivity. For instance, one client created a testing framework with record/playback capabilities for their ERP's proprietary API. This successful concept was later extended to HTTPS and JDBC protocols. Another example is an in-house program designed to generate test data by systematically guiding orders through their entire lifecycle.

- *Deepen the Tool Pool* – Once the team has access to a comprehensive set of tools, it's crucial to encourage members to explore these tools in depth and use them effectively. Over the years, I've been surprised to see how some developers underutilize the capabilities of tools such as Integrated Development Environments (IDEs) or API clients like Postman. To address this, more experienced developers can be leveraged to mentor those who are less familiar or skilled, helping to ensure that everyone fully benefits from the available resources.

- *Self-Service and Automation* – There's nothing quite like being abruptly pulled out of a coding flow state to address a trouble ticket. The impact extends beyond just the time required to resolve the issue; it also includes the time and effort needed to regain that productive state. To minimize such disruptions, I've assisted my clients in developing self-service and automation tools that empower non-technical staff to resolve issues independently. Instead of going through the standard trouble ticket resolution process, a simple application—often just a single web page—enables the business team to handle tasks like reloading data or correcting a payload and resubmitting the request.

- *Embrace Consistency* – A key driver of increased productivity is consistency. Consistency leads to greater speed, and with speed comes the potential for automation, further enhancing efficiency. Implementing design patterns, providing code examples, and creating opportunities for CPE (copy-paste-edit) can significantly boost a team's velocity. These practices streamline workflows and reduce the time spent on repetitive tasks, leading to more efficient and effective development processes.

- *Refactor the Org Chart* – Most reorganizational efforts are imposed from the top down, often driven by changes in upper management or company acquisitions. Ideally, the organizational structure should be designed to support ongoing projects. While I've observed temporary teams being formed to achieve short-term goals, I rarely see this approach adopted for the long term. Creating a flexible and supportive structure that aligns with project needs could lead to more sustained and effective organizational performance.

Quality refers to the efficiency and effectiveness with which software is produced. This dimension allows different and dynamic levels of quality to be applied in the project. This does not mean throwing out all quality assurances but deploying them in a more optimal manner for the project. Here are a few areas that should be analyzed:

- *Architecture Red Tape*: Patterns, guidelines, and rules that can be applied across a system are valuable artifacts. However, applying these mandates rigidly, even with the intention of ensuring quality, can result in complex and costly solutions that are challenging to renovate and maintain. Therefore, I recommend critically evaluating these rules to identify opportunities for optimization and to ensure they enhance rather than hinder system effectiveness.

- *Code Overuse:* The antithesis of code reuse is code overuse—prioritizing reuse to the point where it defies common sense. Before promoting objects to the enterprise's "common" repository, evaluate whether they might be better placed within specific domains that align with similar subject matter. Additionally, consider adopting a service-oriented architecture (SOA), which minimizes runtime dependencies by focusing on higher-level services rather than code sharing at build time. This approach can enhance modularity and reduce the risks associated with excessive code reuse.

- *Inflexible Quality Assurances*: Just as different industries require varying levels of quality; different areas of a system should also have tailored quality standards. For instance, on Federal Drug Administration (FDA) regulated projects, my team conducted risk analyses on the code base. We applied formal code inspections to high-risk areas while performing code reviews for other parts. The key is to implement dynamic quality assurance measures based on the specific needs and risks of each area.

Software development is challenging; software project management is even more so. Don't make it more difficult by blindly accepting absolutes like the 2 of 3 rule.