Obey the (Software) Laws

The modern enterprise is software dependent. Whether you develop software in house, commission custom software, or purchase and install commercial software products, software projects are an important cost component and must be well planned and executed. But top-tier business leaders are rarely involved in the day-to-day management of software projects. Their job is to make decisions that affect a firm's strategic direction, policies and profitability. Business leaders can, however, establish procedures and practices that help projects succeed. In this new series, we'll explore how.

Does the term "laws of software development" seem contrived? It shouldn't. In the 40 years since Frederick Brooks wrote *The Mythical Man-Month*, the industry has amassed a wealth of knowledge about how software projects behave. We know what works and what does not. But as Brooks once observed, *knowing* what works and *doing* it are very different things. Failure to acknowledge and obey these laws, whether deliberately or through ignorance, is the principle cause of software project failure.

Law 1. *Every software project has a minimum development time*

Savvy developers and consumers understand that wanting a project to complete in five months doesn't mean a five-month schedule is achievable. If the minimum development time is six months, rationalization and wishful thinking will only postpone the day of reckoning. Back in the 1970s Brooks identified unrealistic schedules as the leading cause of project failure. That is still true today. The solution is to plan for enough time to complete the work. We will address how to determine both the minimal and optimal development times presently.

Law 2. *Schedule and cost do not trade off evenly*

Each development project has a range of *feasible* schedules and budgets. In general, aggressive schedules result in higher development costs and poor quality.

Why is this? Schedule compression is normally achieved by adding staff. But newly added staff take time to come up to speed, increase communication complexity within the project, and decrease the productivity of existing staff who, in addition to their current work, must now train and coordinate with new team members. When large teams are employed from the beginning in order to reduce the schedule, projects have more staff than they need to complete their current workload. In general, the closer project schedules are to the minimum development time, the more they cost. To make matters worse, quality suffers, too.

Using larger than needed teams increases the number of communication (or miscommunication) paths. This leads to more defects that must be found, fixed, and retested. As projects scramble to meet unrealistic deadlines, testing finds itself on the chopping block. And a project that meets cost and schedule constraints but shortchanges quality only creates problems that will be with you for a long time.

Law 3. *Projects grow*

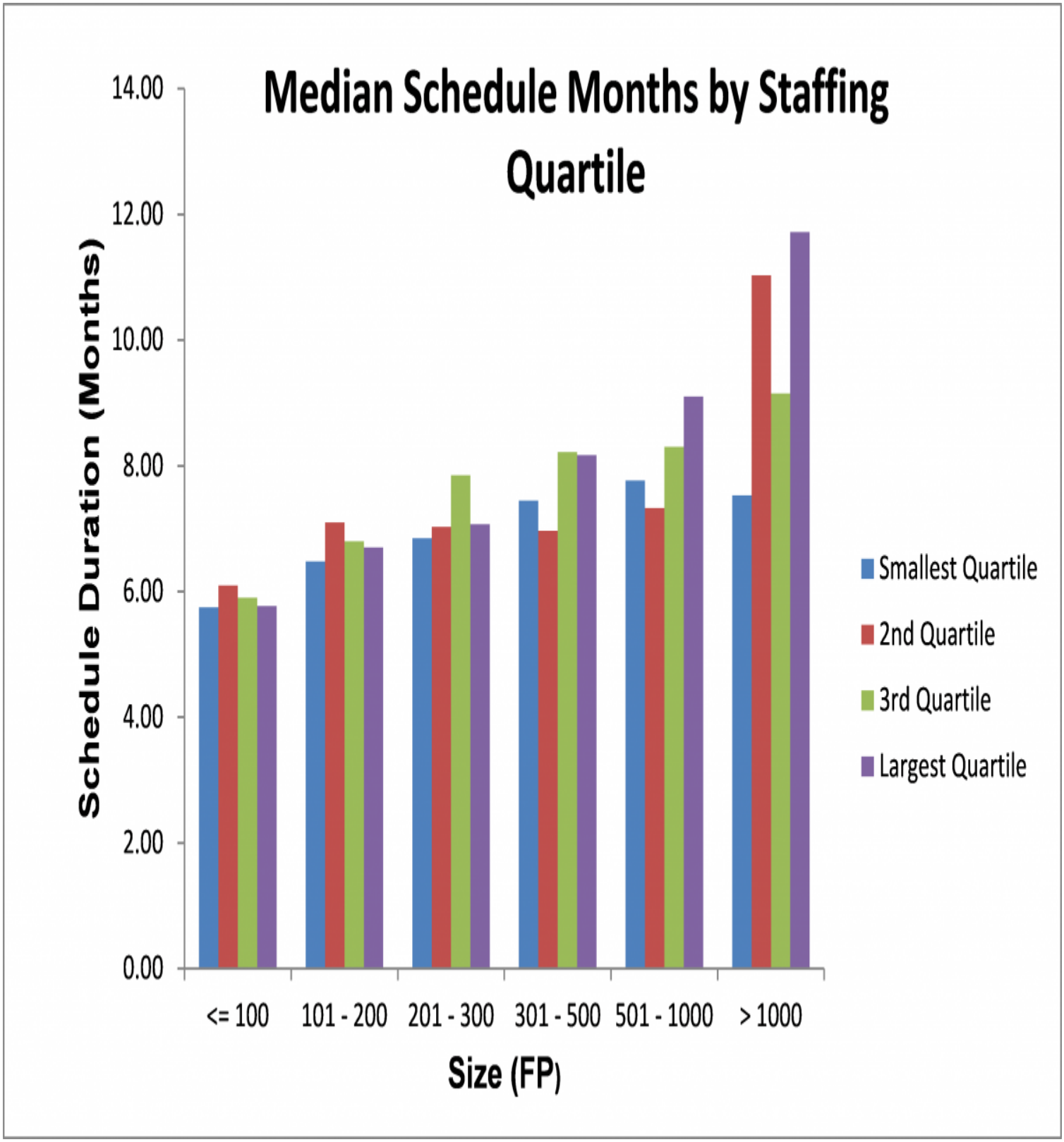
In the early planning stages, software project schedules and budgets reflect only what is known of the requirements at that time. Often, early schedule and budget estimates are overly optimistic. What happens next compounds the problem. Between the time a software project begins and when it is delivered, it almost always grows.

Why is this so? First, at the time the budget is allocated and schedule determined, requirements are both high level and abstract. As requirements are refined and coding begins, details emerge that add to the project's scope. What looks simple and straightforward at a high level may turn out to be much more complex when development work is actually performed. Second, new requirements are often added to a project without adjusting the budget and schedule. Now, a project that already has trouble meeting its schedule and staying within budget has more work to do but no additional time or money. There is a fairly simple solution: build contingency into both schedule and budget while enforcing a change management process that evaluates every request for its impacts and adjusts schedule and budget accordingly.

How much do projects grow? A 2007 study by QSM found that project scope increased on average by 15%, schedule by 8%, and cost/effort by 16%. Simply building these contingencies into the project from the outset will eliminate many issues. This is not to say that you should not strive to contain scope creep; you should. But, allow for the fact that sometimes you won't.

Law 4. *Small teams are better*

Large software projects do not always require large (and expensive) development teams. The analysis is conclusive on this. Smaller teams deliver superior quality software at less cost in about the same time as do large teams (see "Staff That Project"). The following graphic is based on a study of over 2,200 completed software projects. In it, the vertical bars represent the median schedule for the smallest to largest staffing quartiles for various sized projects.



Note that larger teams do not succeed in reducing schedule even though they normally represent an effort to accomplish exactly that. All they succeed in doing is being more expensive since they use many more people to accomplish what a smaller staff could do in the same time. We will address how to determine the appropriate project staff later in a case study.

Law 5. *Look Before You Leap: allow sufficient time and effort for analysis and design*

When projects are faced with “aggressive” schedules, there is an almost irresistible pull to jump right in and start producing code. This is exactly the opposite of what should be done. In the past decade QSM has twice compared projects that expend more than average cost/effort on analysis and design with those who used less. The results leave no room for doubt. The projects that spent greater than average percent of effort (cost) during analysis and design **completed sooner; cost less; were more productive; and had fewer defects.**

Dedicating sufficient time and effort to analysis and design isn’t enough to overcome impossible schedule deadlines (see Laws 1 and 2). However, incorporating at least 20% of planned project effort into analysis and design pays big dividends.