Steering software projects more predictably

Measuring design quality provides breakthrough insight for steering.

Software delivery outcomes have proven to be inherently unpredictable, especially at large scale. Figure 1 illustrates project profiles for two similarly complex DoD systems. Why do similar projects exhibit such a wide variance in delivering their product? Complexity. Process complexity and product complexity translate directly into management uncertainty and unpredictability.

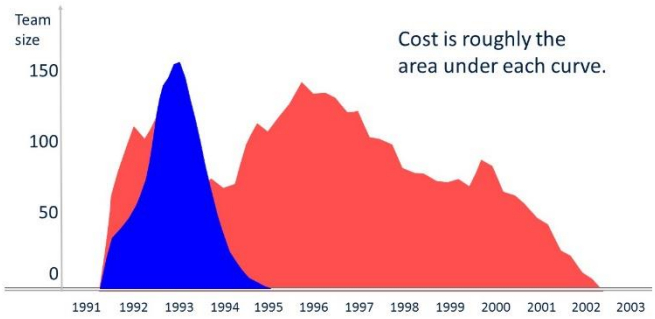

Figure 1: The outcomes of similar projects can vary widely.

Software delivery must manage highly uncertain outcomes.
Software outcomes are highly dependent on continuous negotiations, accurate predictions, value judgments, innovations, team collaboration, technical debt, and various economic tradeoffs. Success is much less dependent on contractual requirements, Gantt charts, laws of physics, properties of materials, mature building codes, and certified engineers. In short, steering software projects is more a discipline of economics than it is of engineering. Unlike most engineering disciplines, software delivery has more complexity, hence more uncertainty.

How do we simplify software projects and reduce uncertainty? Better measurement is a good place to start. Scientists define measurement as *an observation that reduces uncertainty, where the result is expressed as a quantity.* We confront significant uncertainty in specifying the scope, the design, and the plan, where we are bounded only by the limits of human imagination. Quantifying the uncertainty in plans and scope is considered in many projects but is typically lacking in practice. Quantifying uncertainty in design quality, however, has been largely unaddressed, even though it is equally important.

Steering through uncertainty requires a change in mindset.
A steering mindset is well-articulated in *The Lean Startup* by Eric Ries (2011). Using this approach, a business case is captured as a prediction and then tested through a sequence of experiments that validate a business strategy. Progress is measured not by how much stuff has been developed but rather by validated learning. The best way to quantify validated learning is a reduction in the uncertainty remaining in the plans and design. It makes sense to first test the riskiest assumptions because these tests result in the largest reductions in uncertainty, or the most validated learning.

A steering mindset demands a more honest style of leadership, driven by measurably attacking what you don't know rather than posting early demonstrations of what you do know. Project targets should be represented as probability distributions of possible outcomes. More honest predictions are achieved by discussing the variance of the distributions rather than the expected value or the mean. Performance and predictability are improved by continuously negotiating and steering toward a moving target.

How do we quantify validated learning?
As illustrated in Figure 2, software designs and plans must be treated as a sequence of predictions with explicit uncertainty. These predicted outcomes are measured against evolving evaluation criteria, not against contracts with the implied certainty of requirements. Needs and designs emerge over time from a coarse vision with a wide variance (more uncertainty), to more precise, testable specifications with narrowing variance (less uncertainty). Nearly everything is negotiable early in the life cycle, and the feature set and operational characteristics remain negotiable as tradeoffs evolve from speculative debates to objective decisions. Project teams need to plan their activities and early releases to drive integration testing targets to closure before unit testing targets. This integration-first spirit is the crux of "shift left" thinking: Testing the design (and collaborative teamwork) is more important than testing the coded units (and individuals). What must shift left is the validated learning of design quality.

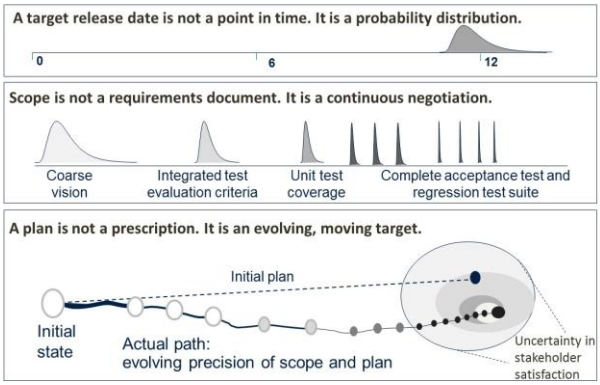

Figure 2: Transforming to probabilistic targets for steering

Traditional project management measures focus on the efficiency and agility of the process. But these process measures provide only half of the insight you need. Design quality and code quality measures are the other half, the more important product measures that teams need to steer software outcomes more predictably. *Business agility is as much a function of product design as it is of process*. Better steering involves tradeoffs among competing dimensions. Efficiency (progress and process agility) and effectiveness (design quality and product agility) must be communicated transparently in objective measures to realize more reliable steering decisions.

How can project leadership better reason about uncertainty?
Suppose you need a new software product to be delivered in 12 months. Your leadership team (project manager, architect, development, and test) analyzes the project scope and constraints to estimate the resources. You use empirical models to forecast that the project should take 11 months. Excellent! A traditional project manager would lay out a detailed plan for 12 months, nail down the requirements more precisely, and plan on conducting an early design review to demonstrate quick progress. The team would feel confident with an extra month in the schedule.

A more enlightened team understands that the schedule estimate is the mean of a more complex random variable. They ask to see the range of possible outcomes, as in the top diagram of Figure 3.

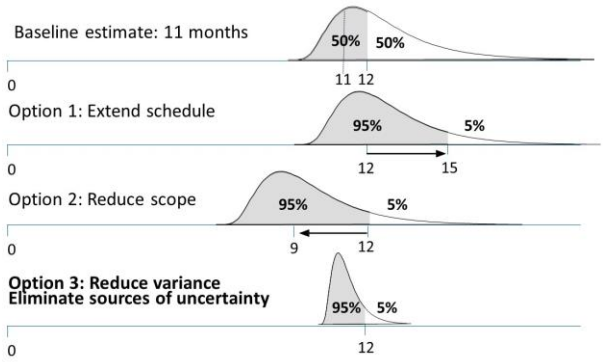

Figure 3: Modern reasoning of delivery targets

The team wants to go into the project with a 95% expectation of delivering on time. The baseline distribution exposes that about half of the outcomes will take longer than 12 months, with only about a 50-50 chance of delivering on time. The reason for the broad range of outcomes is the significant uncertainty in the various input parameters, reflecting the team's lack of knowledge about the scope, the design, and the plan. The input parameters to the estimation models are also predictions (random variables) with some significant uncertainties. Consequently, the variance of the distribution of outcomes is wide.

As Figure 4 shows, there are three ways to move forward:

- Option 1: Move the delivery date out to 15 months to ensure that 95% of the outcomes complete within the target date.
- Option 2: Rescope the work, eliminating some of the features or backing off on quality targets, so that the target estimate moves up to 9 months and 95% of the outcomes complete in 12 months.
- Option 3: Explicitly reduce the uncertainties in the scope, the design, the plans, the team, the platform, or the process.

The first two options are usually unacceptable to external stakeholders, leaving the third option as the commonly preferred alternative. The leadership team lays out a sequence of demonstrable capabilities, starting with the architectural foundation and resolving the largest sources of uncertainty first. Each intermediate milestone results in a measurable checkpoint from which the variance in the forecasts can be reduced and the predictability of delivering on time improved.

A conventional mindset ignores the big uncertainties in the design, postponing resolution until early project momentum is built up by tackling the straightforward tasks. More enlightened software leadership attacks the bigger uncertainty sources like design tradeoffs first, perhaps showing less early progress, but increasing the probability of long-term success. This stark comparison illustrates the difference in mindset between the plan-and-track mentality of conventional engineering governance and the predict-measure-steer-and-adjust mentality of steering leadership culture.

Deterministic planning kills trust because everyone knows it doesn't match the reality of the software development world and its uncertainties. Communicating plans and targets probabilistically and explicitly quantifying the uncertainty in design, scope, and planning builds trust because this approach more honestly portrays our understanding of where we are and how to resolve uncertainty and reason about the future objectively.

Complexity and design quality can be quantified.
After 15 years of research across thousands of diverse software systems, Silverthread has pioneered the use of design structure matrices (DSMs) to visualize design quality and architectural complexity. Figure 4 illustrates examples of DSMs on opposite ends of the quality spectrum.

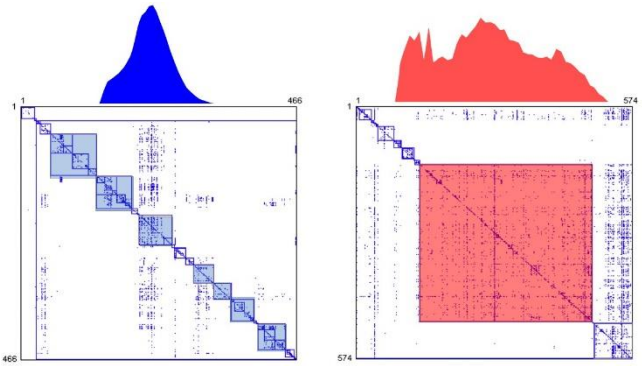

Figure 4: Design complexity is a dominant source of uncertainty.

The modular structure on the left exhibits locally tight coupling within components but relatively loose coupling among components. The alarming structure on the right exhibits tight coupling across a large, dominant core component with more loosely coupled peripheral components.

The most important characteristic of software is that it is "soft." The easier software is to change, the easier it is to achieve any of its other required characteristics. Understanding and quantifying design quality translates into less uncertainty, more benign changes, and more predictable project steering.

Contact Us

Silverthread's mission is to advance the state of software measurement practice by quantifying complexity and design quality. Our measurement know-how can establish a more trustworthy foundation for improving software economics.
http://silverthreadinc.com