Supplementary resource for the book *Deliver Better Results*. More at DeliverBetterResultsBook.com

Technical Cost of Change Self-Assessment

Strategy 4c in *Deliver Better Results* is to reduce **the technical cost of change**. The goal: the system can properly implement reasonable product changes in a reasonable amount of time.

This 10-question self-assessment is designed to provide a quick, qualitative idea of the current cost. This information may help you start important conversations, secure cross-system commitment to reducing the cost, and see change over time.

In the questionnaire, "the team" refers to the technical staff that contributes to the product/solution. If they are organized in multiple teams/pods/squads having separate areas of responsibility, consider conducting this assessment separately for each one. Ideally, have many of the members answer the questionnaire (independently, to maximize objectivity). For each question, average their responses or look for a consensus one.

A. How long does it take to determine confidently that a build can be safely deployed to production?

(**4** = So short the team barely thinks about it; **3** = Short enough that the team deploys to production as frequently as necessary; **2** = Long enough that it's an explicitly managed task (e.g., it might take more than a day); **1** = So long that the team tries to minimize doing it (perhaps by batching features together))

B. How confident do team members feel about refactoring existing working code?

(**4** = Highly confident; **3** = Somewhat confident in most areas; **2** = Largely hesitant; **1** = Not confident at all, avoiding it if possible)

C. How suitable is the architecture to the product's actual needs and likely evolutionary path?

(**4** = It's just right; **3** = It's over- or under-engineered, but generally suitable; **2** = Less suitable that it needs to be, and that hampers development; **1** = It's not, and that's a constant problem)

D. If the integration between two important components is changed, how quickly can the team verify that the components interact correctly?

(**4** = Less than 30 minutes; **3** = Less than half a day; **2** = More than half a day; **1** = More than two days)

E. If the team needs to update existing code as part of new development, how easy is that experience?

(**4** = Very easy; **3** = Generally okay but requires care; **2** = Hard; **1** = Painful)

F. Think of existing components that the team still enhances every now and then. They were designed and implemented some time ago based on knowledge and technologies available then. If they were now magically rewritten overnight using current knowledge and technology, how much effort would that save going forward?

(**4** = Very little; **3** = Some; **2** = A lot; **1** = A ton)

G. With respect to the team's area of the product, how and where can the members contribute?

(**4** = Everyone can contribute competently, even if not perfectly, everywhere; **3** = Most members can contribute competently in multiple areas, but several other areas are effectively owned by specific individuals; **2** = Only one or two people can contribute in several areas, whereas everyone else has defined areas of ownership and specialization; **1** = Every member owns certain areas and can work only there)

H. How challenging is it to understand the flow of code execution?

(**4** = Most members can follow most paths with little effort; **3** = It's not all easy, but the team has most paths under control; **2** = In addition to reading the code, the team must rely on documentation and testing to figure out what happens and how; **1** = Understanding some of the important paths requires too much assuming and guessing)

I. When the team contemplates the next few months of development work, what's the dominant sentiment?

(**4** = Bring it on! **3** = There's going to be work, and some areas will be challenging, but it's fine; **2** = We're not looking forward to it; **1** = High anxiety)

J. In your recent product release, *development time* (iterations/sprints) was followed by some *hardening period* for stabilization and release-readiness. Divide the length of the hardening period by the development time. What ratio do you get?

(**4** = 0–5%; **3** = 6–15%; **2** = 16–35%; **1** = 36% or higher)

Your score

Sum up your answers to product the total score (between 10 and 40). Where do you stand with respect to the following qualitative benchmarks?

- A score above 35 is great. It's rare though, generally reflecting single teams developing smaller products on rapid-cycle, modern platforms.
- A score around 30 indicates a reasonable cost of change. It may be enough for progressing from Fitness Level 4 to 5.
- With a score under 25, expect cycle times to increase gradually and the code to need rewrites in a couple of years.
- Under 20 means trouble, especially if you aspire to agility in the product development process.

Comparison

Now take a few minutes to recall the situation exactly a year ago, then score the team and their product again as if done at that time. Are you seeing a substantial improvement? Or a bounce-back?