You Won’t Change Your Organization Without A System Optimization Goal

A distinguishing feature of Craig Larman’s work (e.g. *Scaling Lean & Agile Development*) is the explicit focus on a system optimization goal for a change initiative. Here’s an example system optimization goal that we consider consistent with Agility:

Increase an organization’s ability to respond to change.

Craig Larman clarifies:

> *tis true that the org design from LeSS is consistent with broad adaptiveness, but... suggest that we coach that adaptiveness is not for its own sake. rather, adaptiveness is in the service of something else: learning towards the discovery and delivery of high value or high impact, in the context of a world in which it’s usually difficult to a priori know for sure what is high value/impact, and a world in which competition does stuff, new technologies emerge, new trends emerge, etc.*

So, less succinctly:

Increase an organization’s ability to discover and deliver the highest value in a world where we don’t know everything, and everything’s changing.

I’m not seeing any consistent system optimization goal in SAFe, Scrum@Scale, the “Spotify model” and the way some people explain basic Scrum.

Why not just “Make Everything Better”?

Better in some ways is *worse* in others. For example, the goal of *increased customer satisfaction* could be inconsistent with *increased stock price this quarter*. Or another example, I heard a project manager turned Scrum trainer say “In my experience, Scrum of Scrums works great!” And I can see how that could be true, if we’re optimizing for the sort of problems project managers are usually asked to solve. But if we’re doing Scrum to increase agility, we’ll want to consider some better alternatives.

What happens without a system optimization goal?

Example A:

I spent a little some time with a company which I initially thought was a perfect candidate for an Agile adoption, a slam dunk. They had less than 100 people in the company, all co-located on the same floor of their hip, modern office. Their management initially seemed quite gung ho. But as the discussions progressed, it became more clear that this management did not *want* to untangle the byzantine organizational structure and the overspecialization they had built up over the past 20 years, didn't think people could learn new skills, and really felt it was best to micromanage employees.

When the company attempted an Overall Retrospective (https://less.works/less/framework/overall-retrospective.html), their actions were to *increase* the organizational complexity that was at the root of their problems! Local optimization bias is so powerful that doing retrospectives blindly can actually make things worse if we are not clear about our optimization goal.

If management cannot express a clear optimization goal and act consistently with it, perhaps we're dealing with too low a level of management.

Example B:

At another similarly-sized company I worked with, the CEO himself came to our mob programming training sessions to see the company's code for himself, and suggest ways of adding automated tests. (This is similar to Ahmad Fahmy's Gemba Sprint (https://www.infoq.com/articles/guide-gemba-sprint/)) This sent everyone a clear message that it's often appropriate to *stop and fix*, rather than continuing to add bugs by churning out crap code.

What's the right system optimization goal?

Example C:

I worked with a multi-team product development group that was living in *hot-fix hell*. Developing new features was impossible because so much energy was spent on fixing and supporting previous releases. Their releases were often so buggy that customers declined to take them, further increasing the support effort as they tried to hot fix multiple versions. To escape the situation, teams had to increase their focus on writing automated tests, increase their focus on reducing code duplication, and increase their focus on collaborating with other teams. But this was *inconsistent* with what they'd been supervised to do in the past – typing lots of crap code – and initially there were complaints that Scrum was "reducing productivity." Fortunately senior management made it clear that the old kind of micro-efficiency and their old ideas about what "productivity" meant were not the reasons for the change initiative.

Eventually the effort paid off, they started getting solid builds, and they were able to release solid products. And then they stopped changing their organization! I was initially disappointed because I saw additional changes they could have made to become more adaptive to customer needs (aka. *Agile*). But they were so pleased their releases no longer sucked that they didn't have an appetite for the additional changes that would have increased their agility.

While the focus of LeSS is increased adaptiveness/Agility, not just *releases that don't suck*, the latter is still consistent with adaptiveness and with LeSS's guidance. You can't adapt to changing customer needs if you're drowning in hot fixes.

The experience taught me to be less attached to my idea of what an organization's system optimization goal should be. There isn't a "right" one. At the same time I've come to believe that senior management should

1. carefully consider what the goal is,
2. express it clearly to everyone, and also
3. express what attachments we're willing to let go of, particularly the difficult ones.

When are goals context sensitive?

People sometimes seek *predictability*. On one hand Scrum tries to make some things as predictable as possible, such as having a shippable product every couple weeks. We want to eliminate *unnecessary unpredictability*. But maybe we do this to increase our ability to cope with *necessary unpredictability* such as our evolving understanding of the requirements, learning new technologies, etc.

Does the change initiative have consistent optimization goals?

Here's a list of other optimization goals[1] that may exist in your organization. They may be explicit or implicit. Some may be consistent with each other in your situation, and others may not:

- increased release frequency
- fewer defects in each release
- predictability
- clarity about who does what

- resource utilization (keeping people busy)
- ideation (creating ideas, as IDEO and Xerox PARC were famous for)
- typing code faster
- secrecy[2]
- comfort
- employee satisfaction/retention (break this down into different categories of employees: coders, line managers, department heads, etc.)
- customer satisfaction
- appearance that the change initiative has succeeded
- increase top-line revenue
- maintaining lead or monopoly
- reduce cost per developer
- product versatility
- broaden/diversify customer base
- preserve status quo
- compliance with rules/regulations
- lead the market by disruption
- increase return on investment
- survival
- conformance to a plan
- rate of developing features immediately (this week's velocity)
- rate of developing features within this quarter (this quarter's velocity)
- rate of developing features next year (next year's velocity)

1. Adapted from a list Viktor Grgic assembled. ↩

2. Yes, I have seen this as an *implicit* goal that was surfaced during training. ↩