Things Change (and So Should Processes)

Software changes, as do software-building processes. When old processes outlive their usefulness, let them go..

by Jonathan Kohl | jonathan@kohl.ca

The other day, I had a fascinating conversation with a team that was struggling with its software development process. This isn't uncommon, but it was sad to see that the team members blamed themselves. They had implemented a popular software development process and, after some initial success, found that they had stagnated and felt like they were regressing. They looked sad, defeated, and hopeless. They used terms like "lack of confidence" and "feeling overwhelmed" and powerful words like "fear" and "hurt."

I felt for them. They had worked hard to implement a software process, and it was letting them down, yet they felt that it was their fault. I asked them, "Do you realize that this process was created twenty years ago? How much has the technology that you produce changed in that time? What about in the rest of the world—not to mention market conditions, people's expectations, and other tools we depend on?" They looked shocked, and then we laughed at the absurdity of it. Of course things were vastly different over that time, so why would the software process be any different?

We are undergoing a massive shift in technology right now—mobile devices, more powerful and ubiquitous wireless networks, and massive computing distribution with cloud technology. Big data is changing how we deal with data from a technical perspective, as well as the enormous ramifications for business. Genetic engineering is helping us discover ways to combat injury and disease, while also creating controversy about the food we eat. Nanotechnology may bring about a giant scientific leap by enabling us to do amazing things in fields like engineering, electronics, energy, and health care. 3-D printing enables us to create and share things in a revolutionary way. The recent proliferation of inexpensive sensors means that pervasive computing can enrich the things we interact with daily.

When is the last time you used a Betamax VCR player from the 1980s? Would you prefer it to a high-definition disc or Internet streaming? It shouldn't surprise us that when we upgrade technology, the older, accepted ways of creating technology will also be disrupted and rendered obsolete. Our favorite process may not apply to what we are doing right now, either as a whole or in part.

Our software development processes are not going to last forever. What worked last year may not work this year, and what worked for the last release may not work for this release. There are a couple of reasons for this. First, we get tired of repeating a routine over and over, and we stop engaging after a while. Imagine eating the same thing for lunch every day, week after week. You are still getting nourishment and the health benefits of eating lunch, but humans are complicated and need variety. Ask anyone who exercises regularly about "hitting a plateau" or a weight-loss regime that is no longer effective. Athletes constantly have to adjust workouts because their bodies grow accustomed to old routines and require something new to continue the process of change.

"In regular life, we not only expect a fusion of ideas and a diverse mashup of concepts, we *demand* it because our tastes change."

The same thing happens with processes we follow. What worked well for us in the past may stop working simply because we get bored and we disengage. Our brains aren't stimulated anymore, and we are just going through the motions.

Another reason an older process may not work is that, sometimes, what has worked very well in the past stops working suddenly. On software teams, this often occurs due to environmental (market or economics, customer expectations, etc.) or technology changes. Our generally accepted practices may no longer be compatible with the new technology we need to use.

Here's a simple example I've witnessed on mobile teams. A generally accepted agile development practice is the extensive use of automated unit tests. During the past fifteen years, we have seen an absolute explosion in unit testing tools, frameworks, practices, and bodies of knowledge in this space. It's been exciting to be a part of it. On mobile development frameworks, though, automated unit testing support can be quite weak. That is a cause for concern when we are used to the benefits associated with these tools on web or other older technologies. On mobile devices, *state* is incredibly important—wireless conditions, the movement and optimization sensors within the devices, human interaction, emotions and perceptions, and even things like weather changes and lighting. It turns out that automated unit tests don't really address any of those problems.

And yet, the problems that end-users complain about the most involve mobile apps that don't function for them when they are on the move or when they use the devices in different combinations of the states that I mention above. So, my mobile developer friends tend to rely far less on automated unit testing and instead supplement it heavily with other quality practices.

Some of my agile coach friends have almost lost their minds with concern upon seeing that a mobile development team uses few if any unit tests—only to discover that the team uses different tools that *better suit* the quality criteria their end-users require. The mobile team has most likely started out with a standard development process and toolset, found gaps or a lack of support in certain areas, and adapted the process.

While some people might be uncomfortable with this concept, it is natural, and we have examples all around us of how people create something unique by combining ideas and forming fusions or mashups. Many of my favorite musical artists mix influences like traditional blues styles and heavy metal or Eastern music traditions and pop. In fact, there is so much crossing over in music now that we would be hard pressed to turn on the radio and hear any one pure style. This also extends to the food we eat, the art we enjoy viewing, the clothes we wear, and our relationships with people from other cultures in an increasingly connected, social world. In regular life, we not only expect a fusion of ideas and a diverse mashup of concepts, we *demand* it because our tastes change.

Why, then, do we need to leave our software processes as pristine, unchanging, and implemented exactly the way everyone else seems to be doing it? Shouldn't our innovation also extend to how we create something, as well as what we create?

One of the most painful situations I see repeatedly as a consultant is people who will do whatever they possibly can to bring about some sort of change, regardless of the effectiveness of that change. "If we just got better requirements, or automated tests, or changed to an agile process, or did *X*, then our problems would be solved!" People who are desperate for change fail to realize that the change they so zealously fight for might not be the right solution at this point in time. Or, it might work for a while and then lose its effectiveness. Is it worth it to sacrifice your well-being, your reputation within an organization, and, at worst, your health to get the change through? Be careful about pushing for a change when it just isn't working and people aren't receptive to it. It might be the wrong change.

Another depressing scenario that I see repeated is teams that feel compelled to have process perfection or to follow what is popular at the time, instead of thinking about how their process helps them create technology to create a great customer experience. Rather than innovate in what they deliver, they feel guilty for doing something different. The software field is vast, with very different mixes of technology, people, culture and end-users, so why would you expect a process that seems to work for others to work exactly the same for you? I tell teams that they should feel proud to be unique and that they have a different path to follow. They are the true leaders, not the people who want to follow what is popular. If the process is getting in the way of your ability to deliver great software, the process needs to change.

If you have implemented a process and, after initial success, you've found quality issues, people falling behind, or that you are constantly late and over budget, then it might be tempting to think that you are the problem. Rather than blame the people, look at the process. It may not be appropriate anymore. Is it helping all the people on your team to create value, or is it now hindering them? Also, remember that we need to create value not only for our customers, project stakeholders, and company owners but also for our teammates and ourselves. If any of those areas aren't actively creating value, then there is a problem.

Furthermore, don't expect processes that were created when VCRs were popular and cell phones were the size of shoeboxes to fit the technology you and your team are creating now. In this business, we need to change, or else we'll fall behind. **{end}**