Innovate Within

The underappreciated art of non-disruptive innovation

In the early nineties computer architects believed the x86 instruction set architecture was on its last legs. As a then 20-year old technology, x86 (powering processors by Intel and others) appeared unsuitable to the needs of modern programs: its complex instruction set made it hard to run instructions in parallel and its 32-bit address space limited how much memory could be deployed. Intel had a solution: a from-scratch new line of microprocessors, named Itanium, with a new instruction set architecture focusing on instruction parallelism and extending the address space to 64-bits. Itanium was packed with bold innovative ideas, supported by massive investments, and it failed, while x86 products continued to lead the industry, and are successful to this day.

While this may sound like a story of innovation losing out to a legacy solution, what really happened is much more interesting than that. It is a story of innovation succeeding spectacularly, in large part because of the lack of disruption it caused to a huge and well established ecosystem. However daunting it seemed, teams at Intel and AMD continued to challenge the assumption that x86 was doomed and systematically addressed its technical limitations. They focused their efforts to *innovate within* their existing product line, and because of that, the vast deployed base of x86 software could immediately take advantage of those innovative ideas. Despite the buzz it enjoys in the tech world, disruptive innovation is rarely the best way to turn great ideas into successful products.

We work in a very successful, mature company. Unsurprisingly, many of our systems have an enormous user base, just like x86 had. Despite that, too often we choose to implement new ideas by designing a new thing from scratch, resulting in multiple systems solving similar problems. Besides this being inefficient, it also taxes our developers to navigate the resulting maze of options[1] when trying to build new products, leading to decreased product innovation velocity. On the other hand, implementing a new idea in an existing system will almost always yield greater return on investment – even if the constraints of the existing system might limit the headroom for the benefits of the new idea.

Here's an example: Imagine you have a new idea for a thing that could be twice as good[2] as the existing widely used solution if you implemented it from scratch, but only 20% better if you applied that idea to the existing system[3]. Assume your new thing will require as many people to maintain as the existing system, and that in the first 3 years you will capture at most 50% of the addressable market (optimistic parameters for an established technology area). The overall improvement will be doubling the goodness for only half of the use cases, so the net effect is only 25% better[4], while the total cost of the combined solutions doubles (two teams). Therefore, deploying the new idea from scratch has negative return-on-investment compared to not innovating at all, while deploying your idea in the existing system gives you a 20% positive return... To all your users... In a shorter time frame.

The assumptions in the math above are generous to the from-scratch approach. In practice, our enthusiasm to build our own new thing leads us to overlook how high the bar is for reliability, monitoring, security, privacy, data retention, accessibility, regionalization, user consent, efficiency, etc., in any new production-quality system. A reasonable rule-of-thumb is to only consider implementing a new idea in a from-scratch system if it is so great that well over 90% of existing users would be begging to migrate and take advantage of it right away. A from-scratch solution would also be the right choice for systems that have far outlived their ability to evolve but, as the x86 example indicates, we would be well served to focus our innovation on a new implementation that still supports the legacy interface, however imperfect that legacy interface might be.

Innovating from within, requires enlightened owners of existing systems to be open to trying new ideas, and benefits from a culture that invites non-team members to play in the existing system's code base. Good ideas should come from everywhere and not only from within. With the right engagement model, however, it can be the most effective way to maximize the impact of innovation in mature areas, allowing us to land achievements while skipping the launch altogether.

luiz@barroso.org
2022

[1] If a dev task needs to use 5 tools, each of them with 3 different versions, you have 243 distinct ways to complete that task.
[2] *Good* for the math in this example is a lower number. Think of lower latency.
[3] In many situations there is no penalty for improvements made in the existing system.
[4] Identical formulation as Amdahl's law