Your customers hate MVPs. Make a SLC instead.

by Jason Cohen on August 22, 2017

"MVP" implies a selfish process, abusing customers so you can "learn." Instead, make the first version Simple, Loveable, and Complete.

"What was initially thought to be a simple process is in fact an incredibly complicated, intricate, and complex system that I've codified and organized into a few easy-to-follow rules that are more difficult to implement than you'd think."

Product teams have been repeating the MVP (Minimum Viable Product) mantra for a decade now, without re-evaluating whether it's the right way to maximize learning while pleasing the customer.

Well, it's not the best system. It's selfish and it hurts customers. We don't build MVPs at WP Engine.

The motivation behind the MVP is still valid:

1. Build something small, because small things are quick and inexpensive to test.
2. Get it into the market quickly, because real learning occurs only when real customers are using a real product.
3. Trash it or hard-pivot if it's a failure, or invest if it's a seedling with potential.

MVPs are great for startups and product teams because they maximize so-called "validated learning" as quickly as possible. And while customer interviews are useful, you learn brand new things when a customer actually uses the product. But MVPs are a selfish act.

The problem is: Customers hate MVPs. Startups are encouraged by the great Reid Hoffman to "launch early enough that you're embarrassed by your v1.0 release." But no customer wants to use an unfinished product that the creators are embarrassed by. Customers want *great* products they can use *now*.

MVPs are too M and rarely V. Customers see that, and hate it. It might be great for the product team, but it's bad for customers. And ultimately, what's bad for customers is bad for the company.

Fortunately, there's a better way to build and validate products. The insight comes by honoring the utility of MVPs (listed above) while giving just as much consideration to the customer's experience.

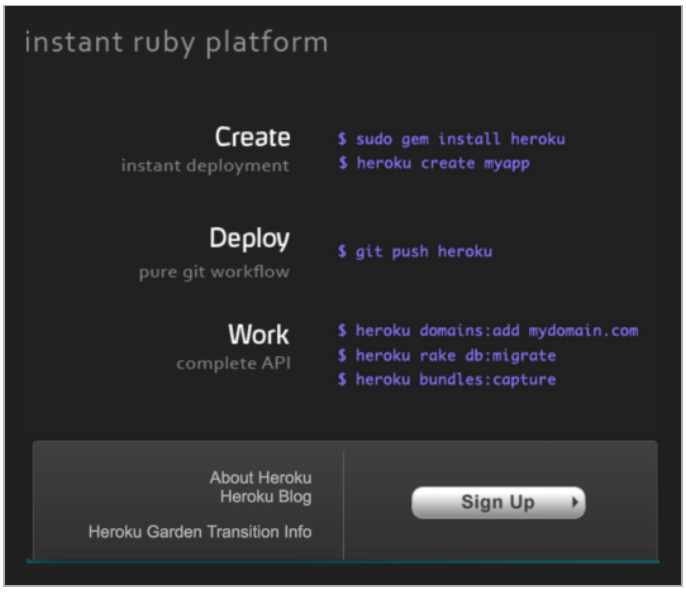
In order for the product to be small and delivered quickly, it has to be **simple**. Customers accept simple products every day. Even if it doesn't do everything needed, as long as the product never *claimed* to do more than it does, customers are forgiving. For example, it was okay that early versions Google Docs had only 3% of the features of Microsoft Word, because Docs did a great job at what it was primarily designed for, which is simplicity and real-time collaboration.

Google Docs was simple, but also **complete**. This is decidedly different from the classic MVP, which by definition isn't complete (in fact, it's "embarrassing"). "Simple" is good, "incomplete" is not. The customer should have a genuine desire to use the product, *as-is*. Not because it's version 0.1 of something complex, but because it's version 1.0 of something simple.

It is not contradictory for products to be simple as well as complete. Examples include the first versions of WhatsApp, Snapchat, Stripe, Twilio, Twitter, and Slack. Some of those later expanded to add complexity (Snapchat, Stripe, Slack), whereas some kept it simple as a permanent value (Twitter, WhatsApp). Virgin Air and Southwest Airlines both started with only a single route —small, but complete.

The final ingredient is that the product has to be **lovable**. People have to *want* to use it. Products that do less but are loved, are more successful than products which have more features, but that people dislike. The original, very-low-feature, very-highly-loved, hyper-successful early versions of all the products listed in the previous paragraph are examples. The Darwinian success loop of a product is a function of love, not of features.

There are many ways to generate love. "Minimum" and "viable" are definitely *not* among those ways. The current-in-vogue way is through design: Elegant UX combined with delightful UI. But there are other ways. The attitude and culture of the company itself can generate love, such as Buffer's blog with its surprising transparency or MeetEdgar's blog genuinely helping entrepreneurs or HubSpot's blog which early on was at least as instrumental to their customers' success as the actual product. Another way is through a deep connection to the psyche and work-style of customers, like Heroku who broke with marketing tradition by filling the homepage with command-line examples instead of benefit-statements, thereby connecting instantly with their geeky target customer:



See this article for many more examples of how to generate love.

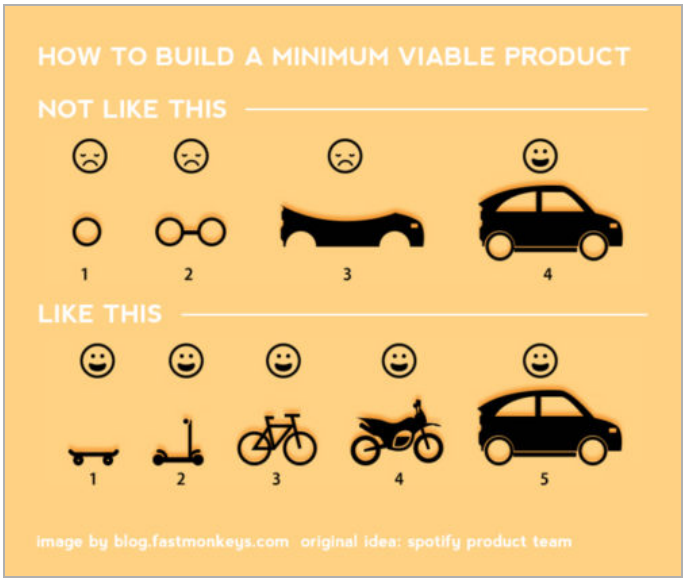
From this reasoning, years ago I named what I believe is the correct alternative to the MVP: Simple, Lovable and Complete (**SLC**). We pronounce it "Slick." As in: "What's the 'Slick' version of your idea?"

Another benefit of SLC becomes apparent when you consider the next version of the product.

A SLC product does not require ongoing development in order to add value. It's possible that v1 should evolve for years into a v4, but you also have the option of *not* investing further in the product, yet it still adds value. An MVP that never gets additional investment is just a bad product. A SLC that never gets additional investment is a good, if modest product.

Although not called SLC, there's a popular meme in product circles that neatly encapsulates the idea of SLC in a diagram: The Modes of Transportation example[1] from the Spotify product team:

[1] Critics correctly point out that Tesla did not follow this system—they didn't make a skateboard and then a bike and finally a car, they just made a car. This is a valid critique, when you know that the only product goal is "car." Indeed, there isn't a SLC version of a car—it's a leap of faith. Still, human society *did* evolve somewhat like this, with basic wheeled objects, later a bike, and still later a car. Also this model is especially useful when your startup set out to make a simple product, not even knowing that a car could exist. Software companies often do evolve this way; the main text gives a real-world example.



A skateboard is a SLC product. It's faster than walking, it's simple, many people love it, and it's a complete product that doesn't need additions to be fun or practical. At the same time, you can evolve the skateboard by adding a stem and handlebars, to create a scooter—only slightly less simple, and definitely loveable and complete. Next, you could grow the wheels, add a seat and some gears, and you have a bike. Again, less simple but now you have a product with massive benefits of speed, distance, and energy-efficiency.

Zooming into one of our examples above, Snapchat took an SLC progression similar to the transportation metaphor. The first iteration of the product was a screen where tapping anywhere took a picture that you could then send to someone else, at which time it disappeared. No video, no filters, no social networking, no commenting and no storage—simple, yet Lovable and Complete, as evidenced by its massive adoption. The insight of "no storage" was critical, but many people have theorized that the simplicity of the interface was also critical. The very fact that it was simple, while not sacrificing loveability or completeness, caused its success.

Later they added lots of stuff—video, filters, timelines, even video cameras inside sunglasses. It's OK for products to become complex. Starting out SLC does not preclude becoming complex later.

With SLC, the outcomes are better and your options for next steps are better. If it fails, that's OK; it's a failed experiment. Both SLCs and MVPs will sometimes produce that result because the whole point is to experiment. But if a SLC succeeds, you've already delivered real value to customers and you have multiple futures available to you, none of which are urgent. You could build a v2, and because you're already generating value, you have more time to decide what that should look like. You could even query existing customers to determine exactly what v2 should entail, instead of a set of alpha-testers who just want to know "when are you going to fix this broken thing?"

Or, you can decide not to work on it. Not every product has to become complex. Not every product needs new major versions every two quarters. Some things can just remain simple, lovable, and complete.

Ask your customers. They'll agree.