Stanford eCorner

Evangelizing for the Lean Startup (Entire Talk)

Eric Ries, *Author*

September 30, 2009

Video URL: http://ecorner.stanford.edu/videos/2329/Evangelizing-for-the-Lean-Startup-Entire-Talk

Speaker, author, and entrepreneur Eric Ries shares rapid fire wisdom on building nimble, responsive, and efficient online software-based businesses. He also offers his wisdom on streamlining processes and progressing engineering systems, and puts forth front line insight into why some new ideas succeed where others have failed.

Transcript

It is my sincere pleasure today to welcome a really terrific speaker and a really inspirational entrepreneur. Eric Ries was one of the co-founders of IMVU, and after he left that organization, he ended up to go into become a Venture Advisor at Kleiner Perkins, and now he spends his time telling people about all the lessons he learned. And he has a very, very popular blog called "Startup Lessons Learned". So without further ado, I'm going to introduce Eric and he's going to tell us all of his inspirational insights. Wow! This is an incredible turnout so I thank you all very much for being here and especially thank you to Tom and Tina and Steve for inviting me to do this. This is a real treat. So, right at the top, let's cover the ground rules. First of all, I do not want your undivided attention so no need. Everyone here has a cell phone or laptop, forbid just everyone to grab their cell phone, take it out of their pocket, it's going to take a second, right? Everyone knows how to use Twitter, so please twitter amongst yourselves. All I ask is that you use the Lean Startup hash tag.

Everyone understands how to do that? If not, ask a person sitting next to you. Send feedback, comments, anything you find interesting or uninteresting. I love the feedback and again, Lean Startup hash tag on Twitter. This is incredible. I'm really excited to do this. So, how many people in the audience are current entrepreneurs? It's just a quick show of hands. Entrepreneurs? OK. And aspiring entrepreneurs, people hope to be entrepreneurs in the future? OK, that's excellent! The good news, this has very good news for the United States, for Silicon Valley, for our civilization, so I thank you. But let's talk about the bad news, for you, which is that most of you are going to fail. And I'm sorry, it's not personal.

It's not about your level of intelligence or perseverance. That's the fact of entrepreneurship. The majority of companies that are ever started failed and I brought a demonstration. This is Web 2.0. Everyone remembers that trend. Here's our midterm report card for Web 2.0. This is how we're doing so far. These are the companies that are no longer with us. I'm sad to say. And actually even some of the companies that are listed on this chart as successes are just financial successes.

It doesn't actually mean that they succeeded in validating the incredible human potential in energy and talent that we pour into them. And look, this a high-tech entrepreneurship so it's a risky business, and we understand that there's going to be some failure. But I think a lot of the companies that are on this chart and across the whole industry fail for really bad reasons and we can do something about that. I'll get to that in a moment. But fundamentally, the Lean Startup is a vision for how our industry could be different. We don't have to accept the same level of failure that we've gotten used to, if we change the way that we operate and if we are willing to give up on some of our really precious myths of entrepreneurship. So, before we get any further, let's talk about, what is a startup? Because one of the myths we're going to have to let go, is what I call the "Startup Dollhouse Fallacy," that a startup is just a shrunken down, big company. And that it'd be nice if that was true, but it's not. That gives rise to thinking like, for example, big companies have departments. Startups are companies, therefore, startups should have departments, which is wrong.

And we'll talk about why it's wrong in a minute. But, I want to really emphasize this point, that startups are not like other kinds of undertakings that you may have heard about, read about, experienced. And because of that, my belief is that although my background is in consumer Internet high-tech startups, the principles that I want to present to you today have a more broad applicability. That they work no matter what sector of the economy you're in. In as I've been doing the Startup Lessons Learned thing the past year, I have gotten the chance to meet entrepreneurs in non-profits, in enterprise, in government. Two weeks ago, I was presenting these ideas at the Pentagon, to the United States Army, and I wasn't sure whether they would really get that there's an opportunity for them to behave like startups. But they got it right away. Actually I'm sad to say a lot faster than a lot of supposedly savvy tech entrepreneurs. So, the thing about startups is that they are human institutions designed to create something new, under conditions of extreme uncertainty. And it's that uncertainty that really makes it difficult to transplant practices from other contexts into the startup domain.

That's why we usually go wrong with a lot of these myths. And it's a human institution. I include that because we usually think about entrepreneurs as like two guys in a garage, you know, they're eating ramen noodles or something like that. But what makes you an entrepreneur is not the kind of noodles that you eat; it's the kind of business that you're trying to conduct. And so because our goal as entrepreneurs is to create a company, an institution, that will outlive us, fundamentally, entrepreneurship is a management science, which I think is counter intuitive to that kind of heroic mythology we have about entrepreneurs. So, why do startups fail and what make them succeed? If you look at the real stories of entrepreneurs, as I know a number if you will get to do in this class, you'll discover really interesting pattern. It's not that successful startup founders have better ideas than unsuccessful startup founders, actually, and I say this is a startup founder myself. Most founders have horrifically bad ideas at the start. What differentiates the successful startups from the unsuccessful startups is something we call the pivot. The idea is that, as we test those ideas against reality, we discover a surprising truth.

That within every bad idea is a kernel of a good idea waiting to come out. And so if you look at the stories of startups, right, they began as doing as cyber cash for PDAs and then they became the online payment provider for eBay. What? They started up building basic interpreters and then they've had the world's largest operating system monopoly. And I know it's fashionable and there are PR reasons to go back and say they knew all along. That when they made that first basic interpreter, that was just step one, obviously, towards operating system world domination. But that's after the fact rationalization which is unfortunately we're very good at. And so I called it a pivot because when tenacious founders discovered that there's something wrong with their idea, they don't just give up and abandon the whole thing. Instead, they keep one foot firmly rooted in what they learned with the previous idea and move the other foot into the new direction. So it's not a complete change; it's simply changing direction, this kind of zigzag pattern that is the path of all successful startups look like. And the premise of the Lean Startup is actually simple.

It's just, if we can reduce the time it takes us to do those major iterations, those pivots then we can increase the odds of finding product market fit and being successful before we run out of money. Is that makes sense so far? OK. So let me tell you some of my stories because I've had the chance to put some of these ideas into practice and fail quite a few times myself. So let me tell you about one failure which I refer only euphemistically to as startup number one, for reasons you will understand momentarily. This is what startup number one look like. OK. I came to Silicon Valley to join this company and I was really excited. I didn't know what this company did. I just was put through a really challenging technical interview, told to fly to Memo Park, down the street here and show up at this nondescript warehouse and step inside. And here's what I saw inside.

I don't know if you can read this banner: We are building something that your friends will lust after and beg to beta test, yadda yadda. It's not B2B ecommerce like during the bubble, when B2B ecommerce was all the rage. We can't tell you what it is but we can tell you who's on the team. And then here's the kicker, on startups, it's all about the team. And I had come to Silicon Valley to learn professional entrepreneurship and when I saw this sign, they had me at hello. I was like, Okay, whatever this company does, I have no idea but sign me up because we believe in something I call the "IQ Theory of Entrepreneurship." You just get the smartest, I mean, ungodly smart people in a room together and we'll have them work together on a product. How could they possibly fail? Because they're so much smarter than everybody else in the world and I know we have some pretty smart people in the audience today so maybe this sounds familiar. Here's the IQ Theory of Entrepreneurship put together a plan for world domination in this new industry and this is what it looked like. This is the plan that had me excited about this company but also, this is why this was a hot company in Silicon Valley. I mean, it was really very exciting to be asked to join.

We're going to create a company with a compelling long term vision, so this is not some quick built-to-flip short term thing. We really want to make a lasting contribution to the internet, change the way people interacted online. Of course, we're going to raise plenty of capital from top tier investors. We're going to hire the best and the brightest and our aspiration was to create a product for mainstream customers. So we were going to build it and ship it when it was ready, not a moment before. Because we aspire to be the next AOL, back when I was still cool and data still a little bit. It was considered naive that the next AOL would ship buggy software or would God forbid be consigned to some kind of niche offering. It was only for video game people or something like that, so we did not want that. And how are we going to reach mainstream customers? We had a plan. The plan was we would capture that elusive quality called 'buzz', people know about that? We're going to have the blogosphere go wild in speculation about what we were building because we were going to build our product in stealth.

No one's going to know what we were building. We would just take all these really smart people and disappear them. See it's like magic. And then people start to speculate, what could they be building and then we start to file patents. It gets very exciting. And the idea was, not just that this is a lot of fun believe me, but when we finally did launch we would capture media attention that really resonated with mainstream customers so they would try our product. So, sound like a good plan? I know none of you would do a plan like this. But maybe you have a friend, who's attempted this or may be thinking about attempting this in the future so let this be a lesson. Because we did something I called 'achieving failure'. We burned through about $40 million building this company and after the five years of stealth R&D we felt pretty spectacularly.

But what you got to understand nothing I say the rest of this lecture is going to make any sense at all if you don't grasp that we didn't fail to execute the plan. That was not our issue. We had a flawless execution. So we built a really amazing product, a really compelling technology; we did hire the best and the brightest and we had a great launch. We were in New York Times, Wall Street Journal, CNN, all right, all the right bloggers saying this is the future of the Internet. And we just had one small problem, which is that, although we got all these mainstream customers to try our product, none of them like it. And so we couldn't convert all that energy into a workable business. And so you ask, well how could all those smart people having worked on it for such a long time have come up with such a bad plan? It's because we were crippled by what I call "shadow beliefs". These are beliefs that were universally shared within the company, yet were never ever spoken out loud or ever written down. I'll share three of them with you right now.

But first, that we know what customers want, which you have to believe that if you're going to devote a 120% of your time and energy into building a company. But the thing is, entrepreneurs have this ability we call it the 'reality distortion field'. It's the ability to get people in their vicinity to believe things as if they were true. They're not strictly speaking or literally true. And one of those things is that people right now desperately need this new technology that we're building inside the reality distortion field. And I don't want you to get the idea that there's something wrong with that. All entrepreneurs have it, good entrepreneurs, bad entrepreneurs. The issue is that there's another category of people that can use the reality distortion field and we call them 'crazy people'. All right. So if you want to create a cult, it's extremely useful to have the reality distortion field.

So the crux of the issue is, how do you know if you're in a startup, and for those who were in a startup today, I know you're not in this situation but maybe you have a friend. Think about whether the vision that you're following is the vision of a brilliant startup founder or of a crazy person. We'll get to have to tell the difference in a moment. Number 2, that we can accurately predict the future, which is something that crazy people sometimes do say, also startup founders. You ought to have a business plan if you're a startup and the business plan has to have spread sheet in the back in which you explain to your investors that in year five you'll make a hundred million dollars. Anybody, would like to admit to having that spread sheet in the back of their plan? Now, there's nothing wrong with having a spread sheet in your business plan. That's a good idea. The issue is, what happens when we start to believe those projections are literally true, like we're Nostradamus and we know what's going to happen. In startup number one, we had the spread sheet to say, in year one, we would have million simultaneous users so on the engineering team, we were really excited while that means we need to build a serious, heavyweight architecture to support those million users when they show up. And of course when they didn't show up, not only how we wasted a lot of time on our architecture that no one was going to use, even worse we've lost the agility necessary to change that architecture because it was this big heavyweight monstrosity that was patent pending but ultimately useless.

And number three that advancing the plan is progress. When we're in conditions of extreme uncertainty and we don't know what to do, it's natural to just kind of fall back on what we can see and measure and the easiest thing to see and measure in a startup is milestones. So, in startup number one, we were very rigorous about making sure that we hit deadlines and we did what we said what we're going to do and we made the plan. And there's a reason why we were able to execute that good plan. We held everybody accountable for really doing good work. The only problem is we didn't have a mechanism for asking ourselves, is this plan any good? Is it worthwhile to advance it? And the issue is that, and this really plagued me as a founder too, that founders are exceptionally good at keeping people busy, right? It's very easy to just, if you're a charismatic founder especially, to just sell, sell, sell all day and keep your employees really working hard and your investors believing that everything's going to be fine. You may notice this pattern that even when a startup fails, if you talk to the founder the day before it blows up, everything's fine. It's going great, right? We're going great right into a wall. That's the pattern of startup failure and there's nothing wrong with that. Because every startup founder knows those stories or those companies that were on the brink of failure and they just managed to pull it out of the tailspin.

So, we want to keep people busy. But when I was a founder, when I would go to sleep at night, I was plagued by this worry. OK, I spent the day of my life and a lot of lives of all my co-founders and employees and investors. We spent money; we expended energy, but did we make any progress? I didn't know how to answer that question and like I said, I know you know you're not afflicted with this condition but maybe you know a friend who has that worry too. So, the shadow beliefs, I mean, it's easy to make fun of people who think they can break the future and crazy people and all that. But I believe, that as an industry, we have this problem that we've talked a good game about how we're going to listen to customers and we're going to innovate and do all amazing stuff. And we don't say these shadows beliefs out loud. But if you look at the plans that we routinely make for companies, I think you'll find them rife with shadow beliefs. And so step one is, of course, to speak them out loud so we can talk about them and then my hope is that, in the future, we can stop believing them. Or at least, call them into question which is something that I got the chance to do in startup number two.

Startup number two, I can actually talk about by name it's called IMVU. It's a 3D avatar, social networking, and instant messaging product. I was actually just doing a version of this presentation in Japan so I have these screenshots. IMVU's now a worldwide sensation and very successful and I'm very proud of it and that's great. But that's not what I want to talk to you about. I want you to go back in time with me to 2004. My co-founders and I had just - we're kind of refugees from startup number one. And we just had this very embarrassing failure way in the public. We'd wasted a lot of money and there are people in this room whose money we set on fire. I'm sorry.

So we thought, all right, we did everything right by Silicon Valley standards and we were praised and we were in this hot startup and everything was supposed to go fine and we got humiliated. So we thought, well, for God's sake let's just try to make new mistakes this time. Because what's the worst that's going to happen, I mean, seriously. Startup number one is about as bad as it gets. And so we built a new plan for IMVU, and I'll tell you a couple of its key components. The first is that we ship this product in six months. So I'm talking about the full, 3D, instant messaging, social networking, virtual currency, user-generated content, blogging, photo galleries, the works - in six months. I can see the smiles if people in the engineering department. I can always tell because I'm doing the math. We call it the 'scope math' in engineering, right? OK, wait a minute.

The budget was fixed, six months, and we didn't have any money. We were brand new startup, and you built all those features so something had to give. And I wanted to be super clear about this--this product sucked. OK, I was the VP of engineering; I was horribly embarrassed to show this to customers. But we were determined that this time we we're going to find out whether anybody wanted to use this product and I admit. We were really a bit nervous. We brought this crappy product under our own name. Any journalist who wanted to could have gone to the website and they could have written an article about idiots from startup number one, you know, foolishly don't know what quality software means. And we were worrying customers would try it and then they would see that it was terrible and they would tell their friends and we ruined our brand. I mean, we have these phobias but we needed have worried because nobody used it because nobody cared.

Those of you who actually shipped the product online know that you don't just put it out there and all of the sudden everybody starts using it. In fact, we did such a bad job in marketing the product that nobody ever discovered how bad it was. Well, not exactly nobody. See the thing is, we charged money for that product, which I know is a bad idea, right? It's a horribly buggy product that would crash your computer and we're charging money for it. But it's not that absolutely nobody would buy that product. Actually we had certain people who were willing to pay us money for product that basically didn't do anything. And you're going to ask, who are those people and what's wrong with them? That's a true early adapter. Those are people who were buying from us, not the product that has existed at that time, but the vision of the product that we thought it would be. And startup founders sometimes always like to hear this but visionary customers are often smarter and more visionary than the founders of the companies that serve them. And so by being in constant dialogue with them, from almost day one of our company's life, we were able to learn a lot of important lessons about what the product needed to become in order to eventually achieve mainstream success.

We also ship the product multiple times a day and this is often considered a particularly bad idea because you think about all the things that could go wrong and I just want to be real clear by what I mean. Engineer would check in codes to the main trunk. There are no branches for those engineers in the room. And 20 minutes later that code is live in production. So I got some thumbs up for part one of that statement but I didn't get any for part two. I got one, all right, excellent. We love it, right on the front row; I like that. And if you think about all the things that could go wrong, right? So some... Thank you, that's excellent. Thank you very much.

Exactly my point, but you know what? Here's a thing. You recover. It's fine. It's not the end of the world; mistakes happen. And I'll talk about at the end of the presentation if they allow us, a practice called continuous deployment, which is the methodology for being able to ship software this frequently without having the kinds of problems you just witnessed. We didn't do any PR. There was no launch. We didn't want to repeat the stealth mistake of startup number one. But we also didn't want to be distracted by serving any constituency other than our customers. And all I can say is that this company has done well.

IMVU's pretty private about its results but suffice to say, here in Palo Alto, it is a going concern and I'm very proud of what we've accomplished. So, I had a chance, as Tina said, to spend some time as an advisor to a number of companies, to working with some venture capital firms and writing these blogs Startup Lessons Learned. And I got to kind of ask myself, how come, when we did everything right, we got a horrible and embarrassing result and when we kind of did everything wrong, we got a better result. Is that's just an idiosyncratic fact of our existence or is there some pattern in that noise? And my study of Startups throughout Silicon Valley and elsewhere tells me that there actually is a pattern here. I encapsulated as something I called the Lean Startup. And so let me just tell you a little bit about what that theory is. There are kind of three basic pillars of a Lean Startup that allow Lean Startups to go faster than their non-Lean competitors. The first is the commodity technology stack. This is the idea that we can introduce leverage into product development and this has been well-worn territory so I won't re-hash it here. Suffice to say that, yes this can make building new products cheaper.

But I think what's really interesting is the amount of time it takes to build really interesting new products is falling fast and even in the last five years, this has gone really exciting. So I was really proud and in view that we don't take six months to put that buggy prototype out. But if you want me to pitch me on that same product today and you're going to wait six months to bring it market, I would laugh at your face. I mean seriously, the ability to build really big interesting new products in really small amounts time is, I think, unparalleled in history. The second is a methodology called customer development, which we'll talk about the ability to discover what customers want before it's too late. And the third is Agile or Lean product development but not the kind of Agile that has been practiced in big companies, kind of tuned for the startup condition. And since we're here, I thought I would just plug on the topic of customer development. This is Stanford zone Professor Steve Blank. This is the definitive book on customer development - The Four Steps to Epiphany, Bit.ly/Four Steps. For those who go on laptops, I just take a moment.

You can go and buy it now. Go ahead. Everyone has a copy. I trust by now. OK. Good. So let's talk about Agile product development and what I want to do is try to convince you that the experience I had in Startup 1 and Startup 2 was not just my idiosyncratic experience but, in fact, represents a kind of a larger trend and so I'll do that by showing what I went through schematically. And a point of disclaimer is that they are not everybody is interested in software startups per se. Let me just say that that is my experience but I think these principles have broader applicabilities, I said at the beginning. But I also want to say and I don't mean to be melodramatic but I do believe that the future survival of our civilization depends on our ability to reliably shift new kinds of software.

Why? Because every new product in the industrialized world is either contains software in it now or is built with software assist. So if we accept the same kinds of failure rates that we've gone used to in a software industry, globally, then I think we're in a real trouble. I don't think we have to wait for robots to take over our civilization. We're doomed anyway. That's a digression. It's traditional in our talk about development methodologies to beat up on the waterfall methodology. This is the kind of a traditional way people are thought to build products. This is how I was trained as an engineer so I will. Waterfall is this idea that you take a product idea. You turn it into some kind of requirements' document than you collect specifications.

You build the design for; you implement it. You handed it off to some kind of QA function and it enters into maintenance mood. So a fundamentally linear batching queue way of building products and although that's fun to beat up on because, of course, this is what makes achieving failure possible because you're getting this positive feedback about how you're advancing the plan, even when you're advancing the plan off a cliff. It's actually important to understand that waterfall is appropriate methodology to use in a certain context. The context when both the problem and the solution you are trying to solve are relatively well understood. When you can model what's going to happen in the future, you want to do this. The issue is that in almost all high-tech product development, this is not true. So if you look at the academic research on, for example, on an IT projects that are built using waterfall, something like 6 out of 10 of them failed outright. Think about that. The lucky 4 out of 10 are the ones that come in way late and way over budget.

Six out of ten never finished. They just entered the batch size death spiral and you never hear from them again, so luckily as an industry, we've been working on doing something better. It's called Agile product development. And the inside of Agile is that we can eliminate kinds of waste from our development process in waterfall. For example, when you build the specification document that goes stale or you have a meeting where you accomplished nothing, or you build extra APIs that you might need in the future but then actually while not needing them. All that's a waste and so what we want to do is build the product itself iteratively so that we change our unit of progress from just advancing to the next stage to creating a line of working code like the canonical Extreme Programming, example Extreme Programming is an Agile methodology as diagrammed here. It's something like a big company needs new payroll system and so when you're building payroll, you don't really have to ask what problem are we trying to solve. OK. Every company makes payroll or it's going out of business pretty soon and so what you would do in I know traditional waterfall, even those projects fail. But under Agile we would actually collapse the feedback cycle top so we would take an in-house customer, a product owner, who knows a lot about payroll.

And we'd sit them to the engineers who are building the products so that if they have a question, like how does the deferred amortization work? Or how should the screen work? Or what is payroll seriously? How does it work? They have somebody they can turn to and get authoritative answers, "Excuse, in-house customer, can you explain this to me?" And then they have a dialogue on the spot, at the moment that the question arises. And so under Agile, we can do big IT project, substantially better because we're living in a world where the problem is known. It's the solution that's unknown. And if only startups live in this world, we'd be fine. But, of course, this is what it looks like in startup lab, where both the problem and the solution are unknown. So let's say you want to do Extreme Programming, who are you going to sit next to the engineers, if you don't even know who the customer is. Startup faces problem that there is no authoritative answer yet to the questions they want to ask. And so what they need to do is combine an iterative process of customer development within iterative process of agile development tied together in a company-wide feedback loop. And what's interesting about this is, it changes the unit of progress in an interesting way, and I had to learn this the hard way. And where I talked about how an agile progress is a line of working code, so when you're doing agile, fundamentally, if you wrote working software today and you go home at night, you can feel good that you made progress.

At EnView, we had the strategy. It was brilliant. Here, let me tell you. We we going to deliver this instant messaging product and the people here are familiar with the theory of "network effects"? Right, that the value of the communication network increases with the square of the number of participants called the Metcalfe's law. And the theory would say that you can't create a new instant messaging network because every one is already on some other IM network and in order for them to use yours, they would have to bring all their friends with them, and that's really cumbersome so that creates a barrier to entry and, therefore, you just can't create a new network, not an org. So we had this brilliant strategic idea. We would create an instant messaging add-on that would interoperate with all the existing networks then we would solve this problem by not requiring people to bring their friends and learn a new buddy list and all that stuff. And at the whiteboard, that seems like a very clever strategy. The only problem is that, everything I've just said about the strategy of network effects et cetera is wrong as are all incorrect statements. They just sound good and look good at the whiteboard.

Leaving aside the question of why it's such a bad idea, I want you to empathize with me personally, for a second, if you would. It's all about me. I'm speaking, all right, so... I was the one who wrote the software to do IM interoperability across all those networks and I had to be drawn kind of kicking and screaming to believe that actually all that code, even though it's well factored and I had the unit test and it was well documented. It was actually all waste. So if I spent the last six months of my life just doing nothing, building no software at all, I would have contributed just as much value to our corporate bottom line. So if I killed myself writing the software, I mean, I couldn't believe it. Here, I've done everything right. Again, I'd used the extreme programming, Agile, everything, and yet, I created this totally wasted software. Why? Because the biggest source of waste in startup is building something that nobody wants.

And that's not a technical problem which is kind of frustrating. Right? My theory was I could delegate the business issues to somebody else. I'd just focus on building good technology and I would be successful. And unfortunately, startups don't fail because the technology doesn't work. They fail because nobody wants what they're trying to build. And that's what I talked about at the very beginning that a lot of startups are failing for a bad reason. It's not considered taking a lot of risk. It's because they're building something that, basically, they could have found out ahead of time nobody wanted. And that's why we have to break down some of these myths. If you want to operate this way, you got to get rid of the traditional departmental silos that we build companies with.

So my suggestion is, instead of we have business and marketing sales, forget that. We have one cross-functional problem team trying to answer the question: Who's the customer? What problem are we trying to solve while we concurrently have what we used to call engineering ops and QA, a solution team that's trying to answer the question: Is the current hypothesis for our product any good? And these two teams operate independently but in close synchronization. That's what allows us to discover that the current pivot we're on is no good and pick a new one. So to simplify a little bit, here's the fundamental feedback loop that power startups. This is based on the work of fighter pilot named John Boyd, if anyone's familiar. I highly recommend you look up his works. And they're called it the OODA Loop. The idea here is that every software company, every startup really is a catalyst that turns ideas into product, and then we measure that product against reality, discover if it's any good. We collect data about that, qualitative and quantitative and then, hopefully, we learn for the next iteration. OK, that's not controversial.

But here's the key idea: We need to measure every proposed process, change, employee, product, everything. It needs to be measured against this simple question: Does it minimize our total time through this feedback loop? And that's the problem. With most of the practices that we transplant into startups from big companies and elsewhere, they generally are designed to optimize only one state of the feedback loop. For example, when I was head of the engineering department at IMVU, I had a "no metrics" rule. I was like, "Look, customers don't care if you have good metrics so building them is a kind of waste." And I thought that was really clever because I was able to eliminate the kind of waste I've seen in startup number one, which is generating all these reports that nobody reads. The problem is that when we shipped that product I mentioned that basically nobody used, we couldn't figure out why. We just assumed that we liked the product so other people would like it. And so we would start to bring in customers. You know, I was also our first VP of marketing. I did our first one -- no, $5 a day ad words campaign -- to bring 100 customers a day to our site.

And I assumed that we would get half of them to buy it, which any of you who have done direct marketing would be laughing because we'd be very lucky if we got one in 100. Of course, as you know, we got basically zero. And day in, day out, we would make the product better. And zero out of 100 people would buy it. And we just assumed that that was a payments issue, like we needed to improve our marketing at the point when people were trying to make the sale. That didn't do anything. So then we started to do, "Well, all right, we'll do a little bit of metrics." Like, "How many people even downloaded the product?" Zero. Huh. "Well, how many people tried to download the product and failed?" Zero. You start to work your way back up the funnel to really like, "Well, how many people got from page one where they landed on to page two?" Zero.

Okay. So the problem that we had was not that we had this horribly buggy product that crashed. It was that we couldn't even get people from page one to page two. And so we had to start to add metrics to measure the process of users coming through our product to understand what was happening. But our goal wasn't to create as many metrics as possible. It was to create as few metrics as necessary to get through this feedback loop. All right? So everything I've said so far is completely theoretical and I wanted to just try to blitz through a few specifics before we have time to take some questions. What I thought was: Let's talk about three specific practices. And what these practices all have in common is my belief that each of them, although they operate at one stage of the feedback loop, actually optimizes the total time to the feedback loop. That's the test you got to use whenever you're evaluating a startup process.

So I'm going to talk about continuous deployment -- something they call minimum viable product, and then five why's. Here's continuous deployment. That's what I talked about before that 20 minutes between checking in code and having it live in production. So here's the magic that allows you to do that without, you know, taking the site down. In order to do continuous deployment, you have to be able to like actually physically deploy the software quickly, like I said -- about 20 minutes. Most of that 20 minutes is actually not the mechanical process of doing the deployment. It's trying to certify that this change is non-harmful to the business. So just to give you an example, let's say, like, it's April Fool's Day and you were back at my old office at IMVU. So, hey, this would be hilarious. Let's check in, I don't know, an infinite loop to the front page of IMVU.com so it doesn't work anymore.

And let's deploy that because that would be hysterical. Right? OK. You know we go back to my desk; we'd make the change; we try that on my local sand box; it would blow up; we check it in. And some time in the next 20 minutes, I would get an email from the cluster effectively: "Dear Eric, thank you so much for attending to check in that change. It's a terrible idea, and it has automatically been reverted. And not only that, we've notified everybody in the team that this happened and that something has gone wrong and nobody else can make a change until someone gets to the bottom of what has happened." We, of course, had to be able to do that revert quickly; and that's got to be true, not just at the level of like unit test and automated test because the example I just gave is kind of easy. OK, you know, if I make the whole site not work -- like, we can have tests that catch that. Let me give you a better example. This is actually a real example. Instead of doing that, I'm going to change the check-out button in the e-commerce flow for IMVU.

I'm not going to review it because the test would catch that. I'm just going to make it white on white so it's invincible to humans. OK? So now we're going from having a business to having a hobby -- not a good change. And again, the same thing would happen. In 20 minutes, that thing would be automatically reverted because the system has the intelligence necessary to know that we've made the change that is going to impact our business metrics in a catastrophic way. This idea of shutting down the line when it happens is really important. This is kind of by metaphor to the Toyota production system. Everyone has heard that story of the andon cord, the cord that any worker can pull to indicate that there's a quality problem that stops the whole production line. Even though it's very expensive to have the whole production line stopped, it's even more expensive to let a quality problem kind of fester and proceed down the line. So we use the same idea to lean startup.

So anytime we have a test failing or a regression that's hit production, we want to stop everything and create the space for human beings to get to the root cause of what happened. In order for this all to work, we have to work in very small batches. So just for the engineers in the room, we have one engineer working on a project, say, for three days without checking in and deploying. We would consider that to be an extremely large batch. It will kind of be like: What's going wrong here? We certainly have a whole team of five working on something for two weeks all by themselves generating integration risk -- no way. And that means we have to break large projects down into small batches, which is actually a lot easier than it sounds because 80% of the work of any new feature is actually all the, like deep interchanges we have to make to reconcile that feature with all the other features in the product -- make sure we didn't break anything. So there's just hundreds of little changes that are all supposed to be without side effects, right, and our theory is: If you make a change, and it supposedly has no side effects, let's deploy that change right now and be sure. Here's what it actually looks like from a technical point of view, for those who are technical in the room, you know what I'm talking about. For everybody else, these are some key words you can take back to your startup someday and say, hey, are we doing any of these things? We run all our tests locally in a sandbox. Every one in the company has the ability to run a simulated version of the product on their own desktop.

We do continuous integration using a service called Buildbot that means we have unit tests, functional tests, acceptance test -- every kind of automated test you can imagine. We do that process of incremental deploy that's monitoring the health of our product as we're doing the deployment to see if there's a problem so we can do -- revert it fast. And, of course, we do real time -- what we call predictive monitoring which allows us to get a human being up in the middle of the night if we have a catastrophic problem. But here is the most important things we do: Whenever somebody sees a failure, we want to get to the root cause of why. So we're going to ask: How come our defense mechanisms, the cluster immune system didn't catch this? And every time we fail to catch it, we're going to make our defenses a little bit stronger. That's the essence of continuous deployment. Let me say a word about minimum viable product. I know people will have heard of this phrase a little bit. The idea here is we want to kind of -- most startups are torn between these two different approaches to building product. One, which I call maximizing chance of success, says "Look, we only got one chance at this so let's get it right." Right? That's what I talked about in startup number one.

We're going to ship it when it's right and that actually is perfectly rational. If you only have one shot, you want to take the best shot you can and build the most perfect product you can. The issue is, of course, you know, you can spend, I don't know, say five years of stealth R&D building a product you think customers want and then discover it to your chagrin that they don't. So the other possible extreme approach is to say, "Well, let's just do 'release early, release often'." People have heard that phrase. And this is -look, we'll just throw whatever crap we have out there and then we'll hear what customers say and we'll do whatever they say. But the issue there is if you show a product to three customers, you get 30 opinions, and now what do you do? So minimum viable product is kind of a synthesis of those two possible extremes. We want to figure out what's the minimum set of features necessary to engage with those early evangelists to start the learning feedback loop going and sometimes, I get to be asked the question: "Well, how do you know if you've found the minimum, minimum viable product?" And from a theoretical point of view, this is quite challenging. You could make a really interesting argument that any given feature is absolutely 100% necessary to learn. But the good news is, there's no reason to deal with this theoretical issue because if you're like me and like every entrepreneur I know, what you think the minimum viable product is, is way too big, probably two orders of magnitude too many features. I'm not exaggerating.

So the easy formula for finding out what the minimum viable product is, is take what you think it is right now and cut it in half and do that two more times and ship it back. And I know, I know, customers are going to absolutely hate that thing. It's only one-eighth as big as you thought it should be, right? And that's fine. If you ship that and customers say, "You moron, how could you have shipped without having features X, Y and Z, the things that were all going to be on your road map anyway." You can say, "Good idea, good point," and then go build features X, Y and Z. But you may be surprised, well, of course, not you, but maybe you know a friend who would be surprised to ship a product as I did and nobody cares. They don't say, "You idiot, it should have features X, Y and Z." The worst fate of any shipping of any product is that nobody cares. You don't get any feedback at all. Right? That's what most features or most products do. They're just dead weight. So what we want to do is try to eliminate those and ship without them.

Of course, that's because visionary customers can fill in the gaps. Right? Early adopters can be very forgiving of missing features. They see the vision and you can be in dialogue with them going through that learning feedback loop. Here are the kinds of the reasons why people don't do the minimum viable product; I'll just try to address them really quickly: first, the fear of the false negatives. So I ship my minimum viable product. If it had just that one extra feature, customers would have loved it; but because it didn't, of course, they hated it. So, duh, why would I bother shipping something I know customers will hate? And there's nothing wrong with that reasoning. All I want you to do is ship anyway so maybe customers will love it even though it doesn't have the feature X, and then you can kind of go on and be very successful. And if you're wrong and I really do need feature X, then you can just build feature X. If you want to do minimum viable product, you have to be prepared to iterate.

And so you have to have the courage to say, "Yeah, we'll ship something, get negative feedback and respond." The real startups, real products always have these key features that are always one milestone away. All right, there's like the A features that we're going to build next, you know, right now; and then the B features that are going to happen in the milestone after. And if you've ever done this in real life, you'll discover there are these features that are perpetually in category B. They never manage to get into the current period. They're essential features that just -- but customers actually want this thing over here. And so you need to just accept that. That's fine. It's OK to have the fantasy that, one day, you'll build this amazing other product. Meanwhile, you're being really successful with the product you have today. You need to allow that to happen, give yourself the freedom for that to happen.

The visionary complex that customers don't know what they want so why the hell am I asking them -- and that's true. Customers do not know what they want. But when you show them a prototype and get their feedback, you can learn interesting things about what they do want. So they may say I hate this when, in fact, they would have liked it if it had a few more features. The idea of this is not to advocate your judgment and responsibility for figuring out what customers ultimately will want. You're acting on their behalf, not at their request. And then, of course, my favorite, which is that you're too busy to learn so sometimes, I get the question: "How can you afford to spend all this time collecting all these data?" And it's like, "How can you afford not to?" I don't really understand that one. And the last, I'll just go through briefly, is something called five why's. The idea here is that whenever something goes wrong, we want to understand what the root cause is. And this is a little bit frustrating especially for the more engineering types because it means that behind every supposedly technical problem, there's actually a human problem that caused it.

And if you don't find out what those human problems are, you'll never really going to learn and make progress. So I'll just give you one quick example: Some code gets checked in and it takes the site down. And you're like, huh, well, why did that happen? Some engineers use this really obscure API that if it's not used properly, it takes the site down. It's like that's interesting. Why did they do that? Although, there are new engineer and they were never properly trained in the use of this API. Well, that's odd. How come they were never properly trained? Oh, it's because their manager doesn't believe in training. What? We thought we had a technical problem with our server going down and we actually have a managerial problem that relates to training. My suggestion is that you do that analysis whenever something goes wrong, and then you make what we call a proportional investment at each of the five layers of the analysis. So you don't say, "Oh, we found a problem; therefore, we're taking the next six months off to do prevention." But you also don't do nothing.

You try to find at least one improvement you could make at each of those five stages. So we'll bring the servers back up, we'll fix that API, we'll go train that engineer. Of course, we'll have a conversation with that manager. But the manager might say, "Oh, sure, I'm happy to do a new training program for new engineers. You know, that will take about eight weeks. So, you know, if you don't want me to do anything else for the next eight weeks, I'll work on training," which is basically a fancy way of saying, "Screw you, we're not doing it." All right. That's manager speak for not going to do it. And the idea of five why's is this, instead of having that negotiation that's all or nothing, you say, "That's fine." I want you to do the first hour of your eight-week training program. It's like, "What? I can't do anything in an hour." You say, "That's not true. You could set up a training wiki and just create a page that says training program goes here.

And that's it. That's all we're going to do this time. But, see, the next time we have a training problem and we do five why's, we're going to notice this problem again and we're going to do another hour's work on it. And that person will say we'll all create the training wiki. It's like, sorry, dude, that's already done. Now, you need to do hour two. And if this problem keeps happening, we will naturally invest time wherever the problems really are in the kind of technical human combination system that is our product development team. That's five why's. There's a lot more that I wish we could talk about. Of course, we have very limited time today.

A lot of these techniques are discussed in the blog, Startup Lessons Learned. I also am available to do speaking et cetera, et cetera. You can learn about that online if you are interested in learning more. Before we close though, if you're willing to indulge me, just -- I'm going to ask everybody, just do this exercise. Just trust me. It will only take a moment. Can you all close your eyes? I'm always interested to see what ratio of acceptance I get to this idea. OK. Thank you very much for doing that. My belief is fundamentally you are ready to do this right now.

No matter what company you work for or what your job is or what's happening, you can do this today. So I ask you visualize something you could do that you thought of in the last hour. It could be something I suggested. It could be something I should have said but it didn't. It could be the opposite of what I said because you know better, whatever; something that you could do in one day to impact whatever company or organization you work for now. And just visualize what that thing is. And that's it. So thank you all very much. I really appreciate the opportunity to be here. This is my contact information.

Please get in touch. Thank you all very much. So, Eric, thank you very much. Thank you. I'm Steve Blank, and our class, The Spirit of Entrepreneurship, is I think at least a quarter of this audience. Can you raise your hands, class? Good. Wave. Anyway, you did the wave. So our class actually meets before and after all the ETL lectures, and today's extra bonus for our class is, as we end today, Eric will walk over with us to our classroom and that is the exclusive for being in MS and E 278 as we get Eric. But in the meantime, Eric, we have a couple of questions called from the massive amounts of questions I've gotten.

And I want to ask you the first one which is: Why do you think the Lean startup concepts are only now being adopted? Only in the last two or year or two do I start hearing about them. What has changed? Is it the venture communities' mass adoption? Is it change in technology? Does this only work for Web companies? What is it and what is it we should understand that's happened? Yeah, there's like a real interesting confluence of events that has made this interesting now. I mean, Steve and others have been talking about some of these ideas for quite a long time, but there are a couple of trends that are really interesting. One is just the technology trends. There's a fundamental democratization of entrepreneurship as a result of new technology so talking about the commodity technology stack. What that means is that, there are incredible new products that you can create without having to ask anybody for permission, without having to get a proprietary license agreement or having some kind of government approval. Basically, if you can imagine it, you can build it. It's pretty much the de facto way that we operate in a number of industries so that means there's just a lot more startups available. That's one trend. The other is that as high-tech industries get more mature, we're starting to move away from technology-risk businesses to market adoption-risk businesses.

So we used to say, you know, I want to build a new semiconductor or like someone is like, hey, I'm going to invent teleportation. And the issue isn't like, if you invent teleportation, will anybody want it? No, we understand that there will be a market for teleportation. The issue is, what reason do we have to believe that you have -- you can build that, the difference in technology and market risks. And then the third thing, I think, is economic downturn. I think that when money was plentiful and there's a lot of structural reason, there was just an incredible amount of money being invested by private equity and pension funds, intervention capital funds, we then passing it on to entrepreneurs. So there was really an opportunity to create the most successful kind of business there is, which is, a Ponzi scheme. Right? You just collect as much venture money as you can and pay it to yourself by trying to convince people that you're making progress. And, you know, there was a time when in fact it was -- if what you wanted to do was get rich, that was a pretty decent way to do it. But I think now we're seeing that it has kind of dried up and so people are trying to remember that entrepreneurship is not actually a get-rich-quick scheme. In fact, entrepreneurship is not a very good way to make money.

If you want to become rich, there are a lot more rational and easy ways to do it in a career that has a rational career path where effort will advance you to the next level and you eventually become wealthy. So, as entrepreneurs, if what we want to do is build companies of lasting value and the venture money is starting to become a little bit more rational, venture investors are trying to get a little more savvy. There's this opportunity to talk about, well, what is the best way to build a company to maximize its chances of success. I think that's what's happening. And I'll take the last question. We'll do 278, and then we'll open it up to the audience. When you were at IMVU, you and your partner made a series of critical decisions about the company which eventually led you to lessons learned and the lean startup. But what were two or three of those critical decisions as an entrepreneur, as the head of engineering and head of marketing, that were obvious and non-obvious at the time as an entrepreneur? Yeah. What's interesting is that, you know, I talked about how it's easy after the fact of rationalization. I said the key decision was having me in the company and not have all the good ideas and therefore, of course, that's what causes us to be successful.

But if we're being honest, you got to really look at, not what are the smart decisions that we made but what were the previous decisions we made that caused us to be in a situation where we would make good decisions on an ongoing basis. That's why I talked about entrepreneurship as a management science. It's crafting a new institution such that everybody in it will make good decisions on average. So, like, for example, when we ship that early product, we had a revenue plan that we tried to make. I think the first month's revenue target was $300, and then it was like $350 the next month. I mean, we used to have investors that would say, "Are there zeroes missing in this graph? All right, is this in thousands?" And we're like, no, it's in ones. And look, it's actually a lot harder to make $300 of crappy product than maybe it sounds. And so what happened is after a few months, we couldn't make the revenue plan anymore. I mean we sold to all our friends and family and everyone we knew got them all to buy -- not as easy as it sounds. And then we started to do that $5 a day ad words and bring people in, and they wouldn't buy.

And we started to get ourselves in a situation where we had these even very small targets that we couldn't hit. And it was that total failure to do what seemed like a very easy thing that caused us to be interested in why is it that customers aren't buying. So then, we made a lot of good decisions like all right, well, let's do some metrics and measure. Let's bring some customers into our office. We did about three usability tests a week talking to customers and saying, "Hey, is this the right thing?" And then I'll talk the flip side of it which is that, because we kept failing to make the revenue plan and we were still working really hard, eventually, it became clear that the reason we weren't succeeding is not that we weren't working hard enough. I mean, you can kind of believe that for a certain amount of time; but, eventually, you start to say, you know what, maybe, just maybe the reason we can't drive these numbers up is because we have the wrong product so we need to pivot. And that's kind of the interesting pattern that you see with pivots. Even though it's the problem team whose job it is to be trying to figure out hey, are we solving the right problem, it's generally the solution team, the product development team that discover it's time to pivot. Because they're just banging their head against the wall, right? The theory says if we make this product just a little bit better, we'll do a little bit better with customers and yet zero, zero, zero. Eventually, the solution team will say, "You know what, it's not that we're not working hard enough.

It's that we are fundamentally going on the wrong direction." And that can create the context for the company to have the brilliant product insights that then we all take credit for later and say, well, I knew all along. But that was, to me, the key feature. Great. So let's take questions from the audience now. And let's see. Let's start in the back. And, Eric, if you could repeat the question when it gets asked. Sure. Ralph? So you mentioned the feedback loop, that's really great. But the hardest part there is like the measurement part of it.

And you mentioned you need to get users and that means the right -- sort of right kind of users. So the problem is with this day and age where there are like thousands of startups especially in the Web where they're trying to compete for their users. So how do you get those users and how do solve that key distribution to the problem that that makes the cycle go around? Oh, so the question is: Feedback loop is great; but the hardest part is measuring what customers want. How do you even get those early customers to begin with? I love questions that start with X is great which means, actually, X is not great. So, well, thank you for being -- for asking a challenging question which I hope other people will. I don't think this is a distribution problem. So in a big company, when we're talking about getting users, we're basically talking about distributing a product with many people as possible. That's a marketing or sales function. Here, we're talking about getting users for a very different purpose. We don't want to get as many users as possible.

And guess what? We don't have to worry about getting the right kind of users -- not a problem. The only customers who will talk to you at all if you're in a startup are early adopters by definition. So you don't need to worry about like, if someone is willing to talk to you, don't be like, "Well, they're not the right users." The fact that they're willing to waste their time talking to a startup means they're probably an early adopter of something. So the specific tactics of how you get those users really depends on specific industries. I'm a big believer in search engine marketing that has really democratized access to new customers. You know, Google Adwords is a great product for doing that. I've also had heard stories of people having really good luck with very cheap campaigns on Stumbleupon which allows you to buy clicks very expensively. There are some people who are using Facebook ads to be able to target specific kinds of users. And one thing that I have heard a really interesting story about is, you have a product that's designed to compete with some other product, and you create a Facebook ad that just says that other product sucks. That's the name of the ad.

Who clicks on that product? The people who are frustrated with the status quo and then that gives you an opportunity to talk to those people so that's the kind of technique you need to use to get to those early customers. Yes? I think we got a question over here. You said that at some point in developing IMVU, you're like really wondering are we doing the right thing, are we going in the right direction; then you said, we realized we were working on this new thing so we had to make a pivot. What did you do? What did you change? Oh, yeah. Well, the question is at a certain point, I said we noticed that we were kind of barking at the wrong tree and it was time to pivot; what did we change, how did we do the pivot? The earliest pivot we did was from that instant messaging add-on I described that interoperated with other clients to a stand-alone instant messaging network. That was really counterintuitive to us but we discovered by doing a lot of in-person customer interviews and usability tests. And we were really just -- we thought we were going to be building a product that was in a social gaming space so, therefore, the target demographic would be kind of stay-at-home moms; and we were really surprised to discover that it was all teenagers who were using it. And we kept being like, these teenagers are clogging up our system; we got to get them out of the way so we can talk to our target customer. So we tried to lock the person out of the system. We couldn't do it.

We couldn't get them away. They were all teenagers. And so we started to do these interviews with teenagers where we would have them come in, create an avatar and use the product. And we had this really interesting phenomenon. They'd say, creating an avatar, that's cool. I would say, "OK, great. It's time for you to invite one your friends on your instant messaging network." And they kind of look at us and say, "Yeah, no, thanks." We say, "Why not?" "Like, yeah, I don't know if this thing is cool yet, and I don't think I want to invite of my friends that something that might turn out to be lame." See, we didn't understand social capital is to, you know, your average teenage high school student. So we'd be like "No, no, try it, try it. You know, it will be so cool." We tried everything we could go to convince them that it would work. And they said, "No, pretty much a deal breaker.

And they kept saying, "Well, I want to try it by myself first, and then, once I realize that it's cool, then I'll invite a friend." And we said, "Oh, we're from the video game business. We know what that means. That means single player mode." So we built a version of the product with single player mode -- this is what I'm talking about, the solution team trying to come up with solutions in response to customer feedback, kind of still in line with the overall like plan. And we would have teenagers, they come in and they say, "Oh, the avatar is cool." Then we say, "OK. They try it by themselves, and they could kind of use their avatar and dress it up and that kind of thing. Then we said, "OK, time to invite one of your friends, and they would say, "No, thanks." We'll say, "Why not?" "This thing isn't cool." "But we told you it wasn't going to be cool. Right? Like, it's a social product. You have the network effects and the switching costs." And they're looking like, "What, old man, what are you talking about?" I'm like, no, it does not compute. And so we're like, "Well, but you don't want your own separate buddy lists in instant messaging network." And they're like, "Why not?" "Well, because of the switching costs and this and that. You know, you don't want that." Well, I already run eight instant messaging clients so I don't mind another one.

We're like -- there are eight instant messaging clients? And so we were so far, we made these assumptions about what customers would and wouldn't do that were just, I mean, in retrospect, ridiculous. And so when we finally were willing to accept that and once we got over my anger in having built all these code I had to throw out, we then made it into a stand-alone product and a stand-alone network and it started to take off really from that day. OK. Let's take one more question. Up here? A question about building the teams and choosing partners -- what's the right team -- right synergy? How do you pick your partners and what's the right synergy of the team? Boy, I don't know that there's a general purpose answer to that question. I mean, the successful startups that I have seen run the gamut from people who have been friends forever and really have a tight bond to people who basically hate their -- hate each other's guts but they want to use this product to resolve the dispute, you know, longstanding dispute. So, really, it's about, you know, it's really a chemistry question. Like what makes any team great? You know, I would refer to something like the wisdom of crowds for like the basic rules of how do you create a successful group environment, which is you have a mechanism for having diverse opinions be aggregated together. So that's -- you know, you want the right level of tension and compatibility where people don't all have -- don't drink the exact same Kool-aid but they do have a shared passion for the ultimate vision that the product's going to go. And then what I would say is, great entrepreneurs don't have better ideas, they have better process.

That's really what fundamentally what I'm trying to evangelize. And so I believe that teams that come together from the very beginning with the idea of having enough humility and openness to say, we know we're going to iterate, we know we're going to pivot, I think are more likely to be successful. That's my belief. OK. Eric, so thank you very much. Oh, thank you very much.