SalesSeek

Minimum Viable Product

Everything you need to know about designing, building and validating your Minimum Viable Product

TIM HAMPSON &
QAMAR AZIZ

Contents

Tim Hampson
Co-Founder & Product Architect of SalesSeek

Tim has more than 25 years experience in Sales, Marketing, and Product in Europe, US, and Japan.

Tim's start-up experience includes Illustra's first European sales person ($400M acquisition by Informix (IBM)) and Interwoven as VP International Marketing ($1B IPO). Tim has co-founded two start-ups, helped found the European operations of a further two startups, as well as acting as an angel investor and advisor for several others.

Tim on Linkedin

Qamar Aziz
Co-Founder & CEO of SalesSeek

Qamar has more than 30 years experience in Sales and Venture Capital.

At Sybase, Qamar was four time worldwide salesperson of the year. A former venture partner at Accel, Qamar has applied his sales skills to help numerous startups including OnDisplay ($1.4B acquistion by Vignette (Open Text)), Wilytech ($400M acquisition by CA), WhereOnEarth (acquired by Yahoo!). Qamar also sits on the boards of Opsview and ServerDensity.

Qamar on Linkedin

Reading an electronic copy? These links are clickable!

Why a Minimum Viable Product?.....5
MVP within the Overall Startup Process 7
The Pitch Deck.....8
Understanding Your Market 11
The Powerpoint MVP 13
Study the Competition..... 14
Designing the MVP 17
Choosing Features.....18
Community and Marketplace Products.....19
Design Practicalities.....19
Competition not Consensus.....19
Know When to Stop.....21
Customers have no Imagination 22
Product is Perception 22
Copy, Copy, Copy.....23
The Details are not the Details.....24
Building the MVP.....27
Business Requirements of a Technical Architecture.....28
Web API infrastructure.....28
Access Control – Data.....28
Access Control – Functionality.....28
Internationalization.....29
UI Message File.....29
SQL Internal Interface.....29
No Commercial Single Point of Failure.....29
Continuous Operations.....29
Access Control – Clients.....30
Multi-tenancy Flexibility.....30
Security.....30

Managing Development....30
Breaking Down Tasks....30
Minimizing Unknown Unknowns....31
The Time it Takes to Do Something is not the Time It Takes to Do Something....32
Manage Expectations, not Time....33
Ownership....33
Measure to Manage....35
Validating the MVP....37
Will someone steal my idea?....39
What's Next?....41

Why a Minimum Viable Product?

Why a Minimum Viable Product?

When creating a startup, why is it considered so important to focus on the minimum viable product?

It's because of ringtones and widgets.

When I first heard about people trying to sell customised ringtones, I was convinced this was the daftest, most ridiculous idea I had ever heard off. It's now a $B market. I still wince when I hear some silly tune on someone's phone, but the point is I was dead wrong in my market assessment. Whilst this market is today somewhat on the wane, it's still huge and making money for people.

Conversely, I was really excited when I heard about Yahoo's widgets -a way to build browser-less self-contained mini-apps for the desktop using web protocols for communication. I personally invested a ton of development resource for my own company developing widgets. But widgets never took off and finally got end-of-lifed by Yahoo. Once again, I was dead wrong in my market assessment.

Now you might think this just proves how utterly useless I am, but the truth is that no one is any better. This is why venture capital is a portfolio business.

If you could really spot a winner, you would do that and just invest in Facebook circa 2004 and ignore the rest. But you can't, no one can.

For an entrepreneur this means test, test, test, test every assumption. By creating a minimum viable product, you get **the fastest way to test** if there really is a market for your ringtones or widgets. Time is money, so this is the cheapest way to know if you should bet the farm on this.

> *Find your first customer, first. There is nothing more important.*

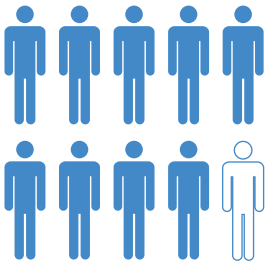
Building an MVP is all about failing fast and making rapid fire corrections.

The plain fact is that **market adoption remains for most startups the #1 uncertainty.** Generally, everything else is "easy" - where "easy" means fixable with blood, sweat, tears, and money. Market adoption, alas, is not fixable by any means, so figure out the answer to the hardest question first, before committing your future. This is why the MVP approach is the most important in building a new product and a new market.

Find your first customer, first. There is nothing more important. Before you build the product, before you quit your job. Mike Schuh of Foundation Capital taught me a great saying, "Until you sell something, you have achieved precisely...nothing". Choosing a name is not progress. Setting up an office is not progress, building a product is not progress. The only thing that counts is getting revenue. This is the logic that leads to the minimum viable product.

MVP within the Overall Startup Process

Before diving into the details of an MVP, it's worth reviewing the overall startup process.



9/10 people say "that's really interesting" when in fact they are thinking "that's the most ridiculous idea I have ever heard".

The high level view is first start with your market. What is your "idea"? It needs to be some combination of your own knowledge, skill, and aptitude coupled with an actual need. The idea does not need to be original, but the need must be real. The first thing to do is to build a short slide deck (because pictures are always better than text) of what you are proposing.

Now test this by talking to as many people in your market as possible. But beware that 9/10 people say "that's really interesting" when in fact they are thinking "that's the most ridiculous idea I have ever heard". Also beware that what people say they like is not what people always part with money for. The output of this process is a **validated value proposition**, and whilst we are at, a sense of the **market opportunity**.

Now you need to think about **team** (ideally in parallel to above).

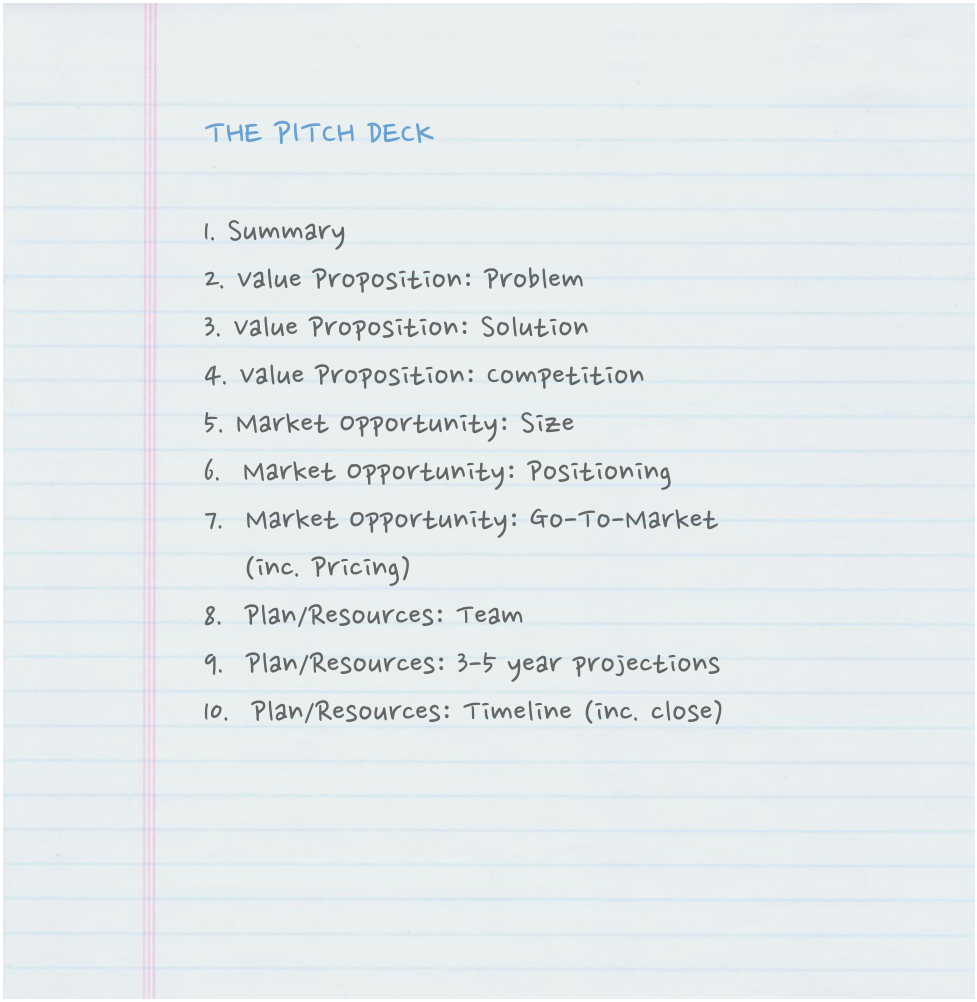
Finally, you need to put together a **business plan**, with some estimates of costs and revenues.

Where we want to be with this is the ability to condense all of this into an investor presentation. This is not to say you should look for external investors yet, but rather there is no better exercise to clarify your thoughts and concentrate your mind on one of the biggest decisions you will ever take. There is a good reason why investors ask for this kind of information, as it helps them understand likely success. You are the biggest investor, and you can benefit from the same process.

Ten sections doesn't mean only 10 slides, but you would not want to go above 20 slides.

The Pitch Deck

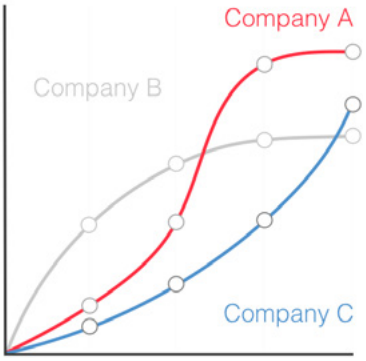
Here is a table of contents for such a presentation. It's split into three main elements - Value Proposition, Market Opportunity, Plan/Resources:



Putting figures on market size and growth rates is difficult. The truth is no one knows how big a market is until you try it. There was no tablet market before iPad. Even in an established category is no guarantee of a market - nor a limit. But you need to put some kind of stake in the ground.

One approach I use is to **try multiple methods of estimating the market size.** One example I did this for was a general business productivity tool. I looked at three different ways to model. One based on some fraction of MS Office revenue, another based on "what would a LinkedIn user likely be happy to pay for this" and finally one based on some fraction of the SaaS market. All three models came in a range $2-5B, so it gave some confidence there might be a reasonable market here. If you can come up with a model, then that normally suggests what data to use (e.g. from the above example "how many business users does LinkedIn have?")

Predicting growth is of course obviously even harder (slide 9.). One way to tackle this is to **present the actual growth rates of similar companies.** Whilst few private companies release revenue numbers, most will release user numbers, and knowing the pricing model you can at least estimate



» *Examine three different methods to estimate the market size.*

the revenue. Even without revenue, and particularly in the case of social networks and community marketplaces, user growth alone can be of interest. The other thing to look at are the S-1s and other reports of "analogous" companies that IPO as they often detail their history. What this does though is to justify assumptions about hockey stick ramps etc. Even if your own plan is quite different, the base assumptions should carry forward.

Both of these points will certainly be a key concern of your investors, so you have to include them. It's largely on these factors that the decision to invest will be made - the other key criteria being what they think of you and the team from an execution standpoint.

All of this should happen before you leave your job or make any other irrevocable commitments.

A key element in any startup, perhaps the key element is resources. In a technical field **you need engineering expertise.**

» *Reid Hoffman, co-founder and executive chairman of LinkedIn. Photo Credit: Wikipedia*

Don't even begin to think about starting a technical company unless you have a CTO on board. *There is no outsourcing option.* Without in-house technical skills you will not be able to either evaluate or manage outsourced development. I've seen this fail more times than I care to remember, and I have never seen it succeed. Software businesses need to rapidly iterate to find product/market fit and that simply doesn't work with a "manage to the functional spec" mindset of outsourcing.

The best quote there is about product development was Reid Hoffman's "If you are not embarrassed by the first version of your product, you've launched too late". The point is, *nothing, absolutely nothing is more important than market validation*, so get something out there now.

Don't think "what is the minimum feature set", instead think "what could we release by end of this quarter?". Setting a deadline forces the unreasonable choices that have to be made. This is the essence of the MVP.

Understanding Your Market

Understanding your Market

Is it better to dive in and learn on the job? Or should you thoroughly plan your startup?

In terms of understanding your market - no, you should do as much research as possible, preferably to the point of getting your first customers/investors before you "start". By "start" I mean give up your day job or otherwise make irrevocable commitments. In terms of understanding startup business - yes, jump in and you will figure it out, as long as you know your customers and your product inside out. Which is why you can't skimp the "understanding your market" piece.

The very first piece of research is seeing what is out there.It's helpful to recast this question as "How do you know if the problems your startup addresses are already solved?" Phrasing it this way makes the answer obvious - you talk to your market.

Depending on the level of granularity, every idea has already been done. Uber is not the first taxi company in the world. Uber is not the first taxi company to "employ" freelance drivers. But Uber is the first to execute on global scale. More importantly, Uber is the first company to try and solve the very real problem that most US taxi trips are unpleasant.

My own bete noire is the electric shaver. I don't like shaving, but I hate not shaving even more as my face gets all prickly and red after a week's growth. Every 5 years or so I buy an electric shaver in the vain hope they might now work. After a few weeks trying, I then go back to a wet shave. So electric shavers aren't exactly a new idea, but as far as this consumer is concerned, my own problem has yet to be solved.

"People don't buy ideas, they buy solutions."

This is part of the reason everyone says "ideas are nothing, execution is everything". You could rephrase the same sentiment as "people don't buy ideas, they buy solutions".

Following on from this, you then should check that your proposed solution does at least in theory solve the problem, and that there is a reasonably sized market of people with the same problem. You don't need to build a product at this stage, just some slides should be able to communicate your idea, sufficiently well to get feedback. This is what I call the Powerpoint MVP.

The Powerpoint MVP

We did this ourselves for SalesSeek. We produced a 10 slide deck that introduced the value proposition with some powerpoint mockups of what the screens would look like. We then sent it to 15 people who would be target buyers (VP Sales/Marketing as this is a CRM/Marketing Automation product), and asked them a set of explicit questions:

1. If you were setting up a start up or small business, would you be interested in this?

2. Would you consider moving over from your existing tool(s) for this?

3. What's the most interesting part of this?

4. Marks out of 10 for the idea (be ruthless)

And here's some of the answers I got back:

RESPONDEE	ROLE	1. GREENFIELD?	2. INCUMBENT?	3. INTERESTING	4. SCORE?
JC	CEO (ex Sales)	Yes	No	Simplicity, Offline	6
RM	BD (ex Sales)	Yes	Yes	Integration	10
MT	VP Sales	Yes	No	Simplicity	6-7
SH	VP Sales	Yes	No	Simplicity, Funnel, Social	7
DS	VC (ex Sales)	Yes	No	Pricing	4
JM	VP Sales	Yes	No	Integration, Forecasting	5
WC	VP Ops	Yes	Yes	Simplicity, Forecasting	5-8
AVERAGE		Yes	No	Simplicity, Integration	6.5

» *Results from the Powerpoint MVP target buyer research.*

Having set questions helps you aggregate responses. Since everyone is time-starved, try to make the questions as simple and short as possible. Also try to compensate for the fact that people who know you seldom say you're an idiot directly. You need to tease that out of them.

The other half of the equation is estimating market size. Google market surveys etc, especially for customer types etc to get this. For SalesSeek (although CRM is a no-brainer for market size), we got a Gartner figure of $36B for CRM in 2017, and another set of US business figures saying SME spend the same amount as enterprises on CRM. Since SME is our focus, it was an easy calculation $36B x 50% = $18B.

Study the Competition

Many startup founders either completely ignore, or give lip service to **competitive analysis**. It's my number one "request for additional information".

I don't believe in slavishly following competitors. I'd far rather talk to a customer about their problems than research a competitor's solutions, but understanding the competition is key for several reasons:

1. Does it exist?

First, **does it exist?** Competition is usually a function of market size. No competition typically means no market. That's a flag.

2. Market Knowledge

Of course it exists. It may not be direct competition, but your market is doing something right now. When a pitch lacks adequate reference to competition is makes me question their **market knowledge**. That's an even bigger flag. And remember budget competition is still competition.

3. Scope the market

It helps you **scope the market**. How much money is being spent right now in similar solutions?

4. Position Yourself

It helps you **position yourself.** SalesSeek is basically the equivalent of SalesForce (CRM), Google Analytics (web), Hootsuite (social) and Marketo (marketing automation) all in one simplified package for SME. Most people "in the market" understand what that means. What is your niche? What is your competitive moat? How are you differentiated?

5. Go-to-market

It's not just about product, but **go-to-market** too. How do your competitors price? What is their sales model? How do they get leads? Again, there are constraints and opportunities here which guide your own go-to-market strategy.

6. Product ideas

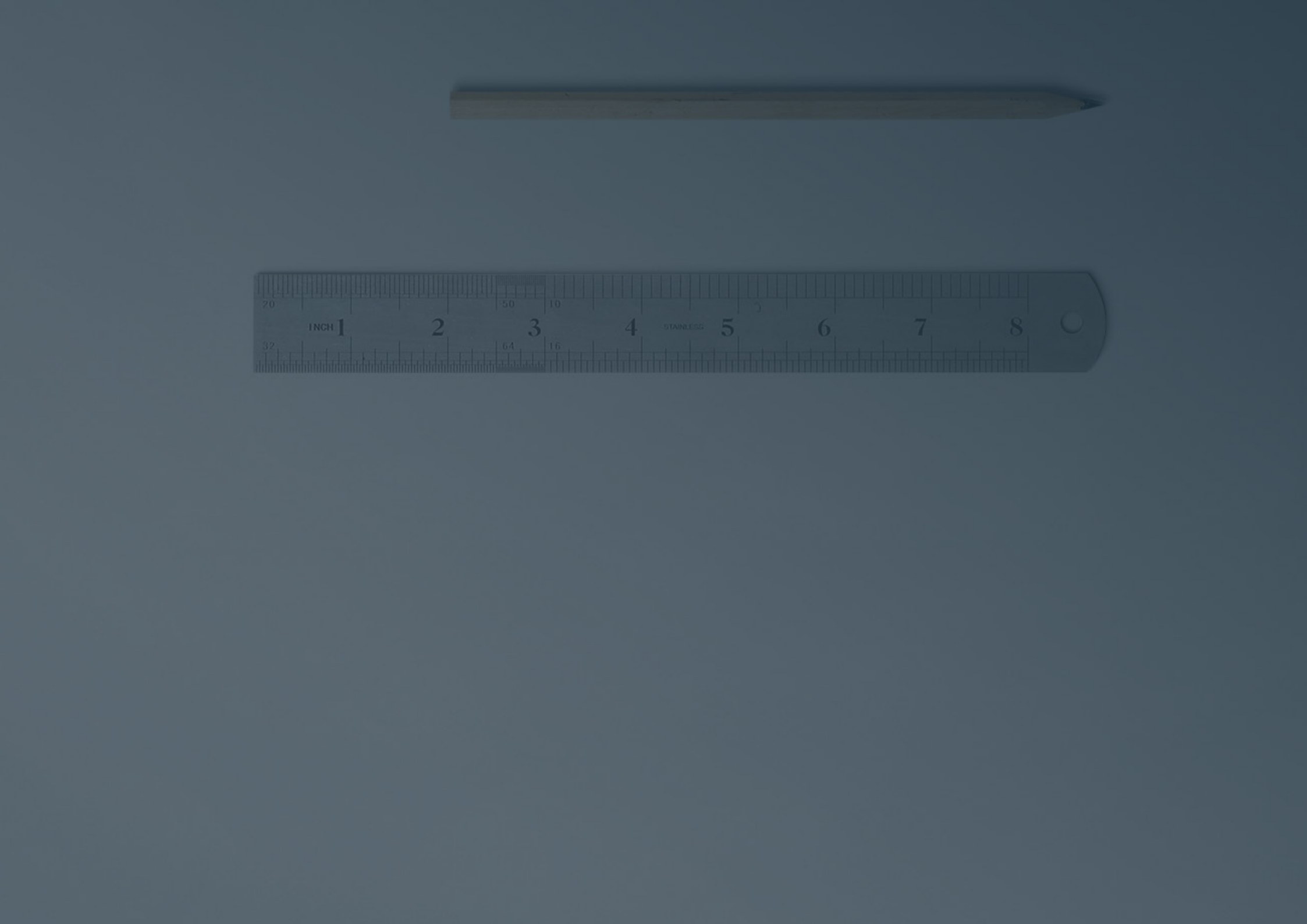
You get **product ideas**. The competition is not awful. The competition probably have some very good ideas. Ditch any propensity to "Not Invented Here" and focus on out-execution, not ego-led "everything they do must be bad".

Of course everyone makes lots of mistakes when setting up a company. Not just first time entrepreneurs either. But different people make different mistakes. The one consistent thing I see across the board though is lack of competitive knowledge, probably stemming from over confidence in your own solution (which isn't bad, but does need to be tempered).

I think the key thing is that the analysis should not just cover the product, but also cover it's sales channels, marketing messages, key customers etc. It's also good to end the report with some actionable sound bites that can be used by sales/marketing to defeat them. A good way to structure such a report is as follows:

REPORT STRUCTURE

1. Summary
2. company/Product History/Profitability
3. Revenues/Market Share/Key customers
4. Market positioning (e.g. are they high-end and us low-end?)
5. Product Functionality - key strengths and weaknesses as compared to us
6. Future product plans (likely to be speculative)
7. Third party analysis/comments/press (e.g. consumer review sites, IT analysts)
8. Marketing (messaging, how much do they spend, what channels)
9. Proposed Product Development Strategy to counter (medium-long term)
10. Proposed Sales&Marketing Strategy to counter (short-medium term)
11. conclusions - maybe an overall positioning matrix of all your competitors, a bit like Gartner's Magic Quadrant.


INCH
1
2
3
4
5
6
7
8
STAINLESS

Designing the MVP

Designing the MVP

Once you understand your market, the next natural step is to design the MVP, this begins with:

Choosing Features

How do you choose the features you require for the MVP? We look at three dimensions:

1. **Value to customer.** This is the obvious one. To choose these it's worth talking to a cross-section of potential clients. This gives you an idea for the spread of functionality (typically everyone wants something different, even if only slightly), and also where the "sweet spot" that maximises your market.

2. **Superiority to competition.** Some of the functionality is what we call a "blocker" - that means you have to have it, but being any better won't help. A good example of this this might be air conditioning in a car. You have to have it, but almost nobody differentiates on it. Other functionality is a "pusher", which means that more is more. A car example here might be performance or economy. Overall though, the MVP can't be "me too", there has to be some unique value.

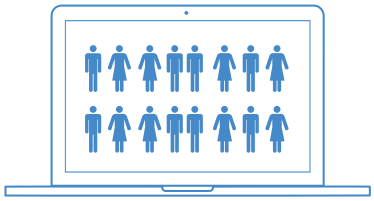
3. **Ease/Speed to build.** A cure for cancer would be great, but may be a bit tricky to develop in the next 4 months. You have to understand the costs and risks.

"If you're not embarrassed by your MVP, then you shipped too late."

The hard part is juggling and evaluating all three, since value and superiority both point to loading more functionality, whilst speed to build forces you to reign in. The best thing to do is to draw up a matrix and aggressively prune the functionality - remember this is just the MVP, so it's not saying never, it's just saying not right now. To do this though you need engineers (who can tell about speed to build), customers (who can tell you about value), and product marketing/management (who can tell you about competition) all together.

If people are pushing you to add more features, just remind them of Reid Hoffman's "If you're not embarrassed by your MVP, then you shipped too late".

Community and Marketplace Products



Your users are the product for your future customers. Put it that way and it's clear you need to collect the users first, and then start selling to your customers.

Imagine that you are building an app to promote events and activities. The users are consumers but the customers are going to be larger corporations that seeking publicity.

The way to look at this though is comparing it to something like Google, or Quora. To be blunt, **your users are the product** for your future customers. Put it that way and it's clear you need to collect the users first, and then start selling to your customers.

The comparison with Google, is that first they had to become a destination search engine site, and then start selling advertising.

This is also a classic example of where external investment makes sense in order to allow that community to build up. Outside of that, the question is how to accelerate this?

Perhaps in this case, you might be able speed things up by automatically scraping/ collecting all available events, putting them on your website to build up an audience and then fairly quickly charging event hosts to "promote" their event - putting on the front page, adding an image etc.

The specifics though depend on your exact value proposition. One thing to look for is adjacent revenue opportunities - in the case of events maybe you could support event hosts by offering some other direct services to them that would be complimentary to your main business model?

Design Practicalities

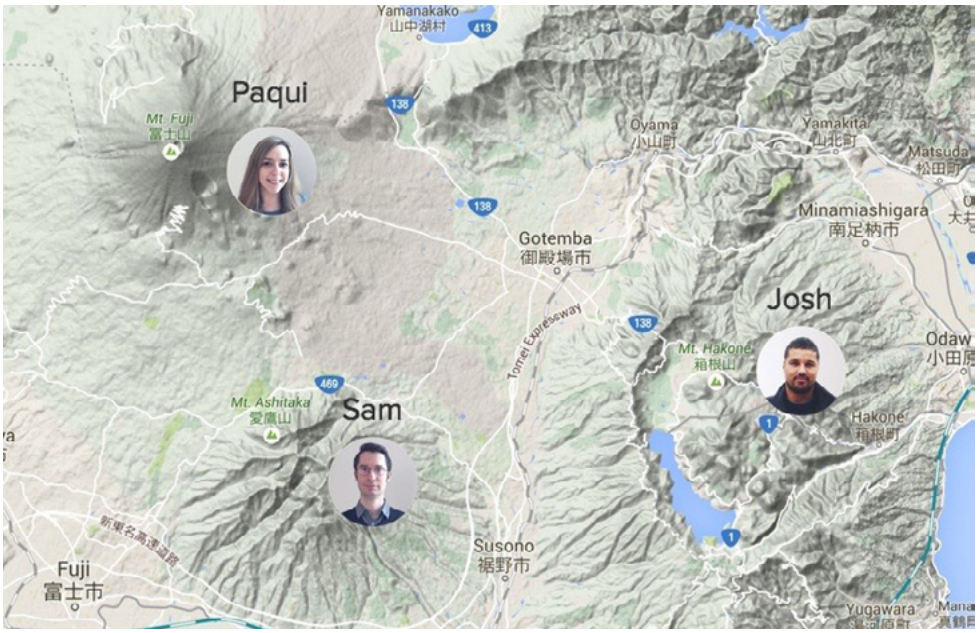
Everyone says design is important, and now that Apple is one of the world's largest companies by value, many are actually believing it too. At SalesSeek, we've tried to apply design principles to our sales and marketing application, but always with our eyes fixed on the goal of a better user experience. But there is more to design than the Helvetica font and a Pantone swatch.

Competition not Consensus

Imagine you were placed in a foggy landscape, and could only see a few feet in front of you. How would you find the highest point, if you could not see it?

One way would be to more or less randomly move about, following the contours of the ground. Sooner or later hopefully, you might detect the ground rising in one direction, then following this for as long as you could until you could no longer go any higher and all directions led to descent. Certainly you would be higher than you started, but could you guarantee you were at the absolute highest point? Perhaps you were "fooled" by a local maximum.

You can think of “design space” as being a little like this mountainous region, where the height represents the degree of optimization, or elegance, or product-market fit (depending on your primary interest). Exploring design space has similar issues. Josh looks happy on Mount Hakone, but unless he’s prepared to descend almost 1500m, he’ll never find the larger mountain to the west.



» *Paqui, Sam and Josh representing the degree of optimization, or elegance, or product-market fit.*

There is a natural tendency to move to the nearest peak - the familiar - by incremental steps and fine tuning. It will certainly produce a better solution, but since design space is effectively unbounded, how can you know that when the fog finally lifts, it won’t reveal a competitor standing atop Mt Fuji?

This is essentially the argument for out of the box thinking, as a way of exploring more of design space, especially those parts not currently so accessible, and also of the brainstorming model of withholding initial judgement. After all, in the picture above, Paqui is only on the lower slopes of Mt. Fuji and lower than Sam, who has reached the peak of Mt. Ashitaka. Sam’s solution is currently better (and indeed, locally optimal), but if Paqui can continue on her path, she will end up with the best solution overall.

Design work needs experimentation, as well as the confidence to throw away everything created up until that point. For both personal and practical reasons, that is a hard thing to do. The worst approach though would be to try and compromise on these different solutions. If Paqui, Sam, and Josh reached a mutual consensus above, it would likely be somewhere south of Gotemba, flat as a pancake, and (no disrespect to their inhabitants), not a solution for anything. Design requires consistency, coherence, and integrity - the very opposite of consensus and compromise, and the origin of the phrase, “a camel is a horse designed by committee”.

A better approach is pursue a number of options, sufficiently far so as to understand their full potential, and then decide between them, more like a competition. Once a solution has been chosen, then a more reliable

process of optimisation should take you to that peak. But trying to get a single designer to pursue several approaches in parallel is inherently difficult - they always have their favourite. Sadly, the dream option of having multiple designers advance comps is often too expensive for most companies. But for consumers the good news is that the market itself provides this function.

Porsche's iconic 911 sports car is a case of how continuous development and refinement can make the best of a sub-optimal solution - in this case hanging the engine out at the back, where it acts like a dumbbell swinging out the back end when you least want that. But (almost) all other sports and racing cars have their engines inside the wheelbase to minimise that swinging effect when cornering. Even Porsche now.

Know When to Stop

Exploring design space takes time, and is non-linear in terms of results. You can hit upon success immediately, or not at all. Vincent Van Gogh painted Sunflowers seven times over several years and was still not happy with it.

But here we're talking about design not art, and time is important. All designers, regardless of expertise, will produce a better design if given longer. The problem is that for commercial enterprises, time is money. Designers generally are never happy. They will continue to both experiment and tune forever. **That's actually good, but at the same time, there has to be a point at which we say the design is "good enough".**

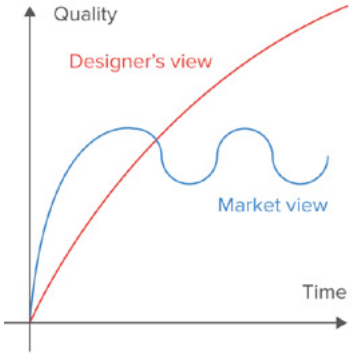
» *Sunflowers by Vincent Van Gogh. Photo Credit: Fair Use.*

The challenge for both designers and the rest of the business is to understand how this compromise between time and quality is best met, when the ultimate arbiters are the overall market - Mr & Ms "Average". They may or may not share the finer points of optimization, and if you do narrow your appeal, you need to be certain you can increase your value to that smaller market enough to compensate.

The issue is at the high end, much design can be highly subjective, and that does not always map to commercial success. Van Gogh's sequence of Sunflowers represents part of his own artistic journey, but I have no clue how to put them in chronological sequence. They all look equally awful to me. You may think me a philistine, but I am part of your market nevertheless.

There is, of course, a floor of minimum quality any design solution must make in order to sell anything, and just as there is a concept of "minimum viable product" for functionality, so there is for overall design quality. And just as there is a market for both feature-rich/ expensive as well as feature-lite/cheap products, there is space for both Apple and Dell in the design stakes.

Customers have no Imagination



» *The Designer vs Market perception.*

One common mistake is to rely too much on market research, or, indeed any form of customer feedback. The problem is that customers have no imagination and can only at best think of tiny incremental improvements over what they have currently. It's why design is hard. If all you had to do was listen to the what people wanted, then you probably wouldn't even need designers. Logo briefs would look like "give me a rounded off square in mustard yellow, with a big S in sans-serif font". Clients/customers/markets need to see something before they know they like it.

I once worked on a software application to integrate in with an existing product and demonstrated it to the client. The screen had a neutral grey background. Initially the feedback was somewhat negative - "it doesn't really look much like our app". I changed the neutral grey to their own steel blue grey and showed them again - "Wow, that looks really good, you've come a long way". It was only 30 seconds work to change one entry in a CSS file, but the fact remains most people can't even mentally swop out neutral grey for blue grey.

This means it pays to be cautious when showing clients work in progress. Generally you shouldn't. All they will see are the "mistakes" and will not be able to extract any of the winning features. Expect feedback along the lines of "could you make it, err, maybe, better?". Any attempt to press them further will merely pull out curve balls and bum steers. But build it, and they will come. Maybe.

Product is Perception

» *Oversimplification is not always the simple solution.*

If a tree falls in a forest with no one there to hear, does it make a sound? From a design perspective, a product only exists in the mind of the user. This means that a product is not an external objective thing, but rather an internalized subjective experience. Specifically, it's not enough to look at the concrete properties of a product, but rather how it is modelled inside the user's head.

A classic example of this is the shower faucet (tap). Upscale hotel bathrooms often select elegant shower controls that may be as minimal as a single lever, often bereft of any unsightly markings. As to whether you push in or rotate for volume or temperature, or whether it's clockwise or anti-clockwise for hot and cold, that is left for the jet-lagged, half-awake traveller to determine. Since there is typically some lag between the control and the actual water temperature, it's common to go through several scolding/freezing cycles before actually finding something bearable.

The counter-point to Antoine de Saint-Exupe's "Perfection is achieved not when there is nothing more to add, but when there is nothing left to take away", is Albert Einstein's **"Make everything as simple as possible, but not simpler."** In the case of this shower faucet, it has been over-simplified.

The simplest approach is to have one rotary control for temperature, clearly labelled blue to red, maybe even marked with a 38 degree point that most people consider acceptable for a shower, and another with volume, again graphically marked from small to large with symbols, not text, for those that speak a different language.

Why this wins is that it's understandable. The user has a clear mental model of the operation, and can safely operate and predict its behaviour. It is the ultimate in conceit to think collapsing everything into a single control is worth compromising the user's ability to operate it. Anytime/everytime you say "but we can document this in the manual", you have failed.

My own personal bugbear is the prevalence of vanity panels. You see these commonly on video players, and also in software in the shape of extra button clicks. Here is an example from Google. Why the extra click? It's not like they lack space.

» *The Google menu oversimplified.*

> *"Hiding controls for the sake of beauty is not simplicity, it's lying, by creating a false promise and adding to the user's cognitive burden in achieving their goal."*

I know Google make their clean search page integral to their brand, but hiding stuff is not a great user experience - especially the additional "More", which seems particularly gratuitous. There is a reason aircraft cockpits look "busy" - all those instruments are there for a purpose -it makes it simpler to operate (i.e. fewer crashes).

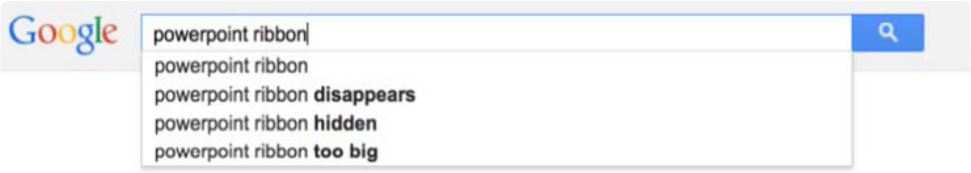
It's not what the product looks like in the showroom, or on the screen. It's what it acts like in users' heads. Hiding controls for the sake of beauty is not simplicity, it's lying, by creating a false promise and adding to the user's cognitive burden in achieving their goal.

Copy, Copy, Copy

If you were starting from scratch, you would not design a steering wheel to direct a car. Nor probably would you devise a time system of 7 days, 24 hours, and 60 minutes. But of course, you are not starting from scratch. The familiar is always easy to use, and more productive than a new unlearned approach, even if it is "worse" in some senses where that familiarity is not taken into account. The problem is that the universe where you don't need to take into account past experience does not exist. One of the best examples of this in recent(ish) times is the ribbon interface introduced with MS Powerpoint 2007. Powerpoint has been around for a long time, so the majority of its users are pre-existing, which means they have not just to learn an interface, but crucially they need to unlearn

the old one. I personally hated this, and it was one reason I kept on the previous version when using Windows and a significant reason for my moving to Keynote when I moved to Apple (if I'm gonna have to learn again, I may as well...).

You know you're off to a bad start when Google helpfully autocompletes for you...



And Microsoft need to write a support article on how to find things with it. It talks about the interface being "simple and discoverable", but the very existence of this help screen mocks that.

» *Austin Allegro with the squared off steering wheel.*

Everyone copies. Not in the sense a back alley warehouse knocks out Louis Vuitton fakes, but in the way artists have copied (and crucially, extended) each other for centuries. If something isn't broke, don't fix it.

In the dying throes of the British car industry in the 70's, Austin brought out their new Allegro with a squared-off steering wheel. Creative, but it didn't take off and was quickly replaced.

Don't reinvent the wheel. Really.

The Details are not the Details...

...The details are the product. This quote is from Charles Eames, who along with his wife, Ray, formed one of the most successful design partnerships of the 20th century. You'll see their chairs in board rooms across the world.

Details telegraph care. Luxury is not the presence of anything, so much as the absence of irritation. It's the door lights in a Mercedes. It's the way

“Luxury is not the presence of anything, so much as the absence of irritation.”

a Laguiole corkscrew perfectly distributes pressure across the palm, and the way an Apple power cable clicks magnetically into place.

This applies to every part of the product experience, including its purchase, maintenance, use, and eventual disposal. Just following that one phrase “avoid irritation” and you will be ahead of 99% of your competition.

Form follows function has been the overriding principle of much of modern design, but it still seems to observed as much in the breach as in its observance. Design is about usability, not appearance. Good design applies to all products and even processes and services. Taking note of these tips here should help you achieve the best possible design for your users.

Building the MVP

Building the MVP

Business Requirements of a Technical Architecture

Software architectures are evolving all the time. The best ones though, last the longest. The biggest point, and the one that has paid off for us at SalesSeek many times over is having all access via the Web API. What follows is not so much a technical architecture as the business requirements of a technical architecture. Clearly it is written from the perspective of a software product, but the principle is applicable to any complex product where one might expect the need to rapidly make changes to the product to support partners, evolving market requirements or new technologies.

HERE ARE SOME EXAMPLE TESTS

Web API infrastructure

TEST

Enable "add contact" functionality programmatically

System should be architected to insulate client calls and responses, effectively turning the application into an API, with the web interface being just one such use of the API. This is a moderate overhead, but will pay dividends in supporting multiple clients as well as enabling a publicly accessible API layer at the appropriate time. Again, the advantage in starting out this way is avoiding expense later, as well as accelerating development on multiple platforms such as emerging smartphones and tablets.

Access Control – Data

TEST

Give Managers read only access to their sales people's forecasts

Whether implemented or not, every logical data item must in principle be able to be controlled by role based security with groups. This is needed to support internal privacy controls dependent upon market requirements, and so likely to require frequent change and evolution, hence the need for architectural flexibility.

Access Control – Functionality

TEST

Package a pure contact manager low-end offering

All logically grouped functionality to be access controlled. This will enable us on a per client basis to configure a functional set (e.g. create/read/update/delete contact is a single set). Part of this is to support tiered functionality at multiple price points. Within this, the client administrator should also be able to further restrict availability of functionality, if we choose to surface this ability.

Internationalization

TEST

Support “Die Großväter” as content

All parts of the application that touch data should fully support Internationalisation. This is a requirement prior to any international expansion, since even in the UK, both contact details and currency symbols may extend beyond the standard character set. If architected in from the start this should be minimal overhead (many components such as Java support this natively).

UI Message File

TEST

Change menu item “add contact” to “new contact” in production in less than 30 seconds (assume automated deployment)

In the short term, this enables rapid changes of textual content on screen as the application is developed. Longer term (2-3 year horizon) this will also ease the translation of the UI to support localisation, and allow such work to be performed independently by a partner. This is a very low cost overhead to start with, but can be extremely expensive to reverse engineer in at a later date if missing.

SQL Internal Interface

TEST

Enable commonly available query/ visualization tools to access data.

For ease of expansion and using third party reporting tools, the system should provide a SQL interface to data (including available database drivers). This need not be the sole, or even primary mode of application data access, but should be an available option. This will open up integration options with third parties.

No Commercial Single Point of Failure

TEST

The company credit card used to pay for Google services expired on Saturday.

Business Continuity Planning needs to come out of a cost/benefit analysis, but should not be restricted to purely technical failures. Commercial issues and disputes do occur, and these can lead to a suspension of critical service from a supplier, such as a hosting vendor. This should be anticipated and the backup plan (e.g. moving backup site to separate organization) should explicitly cover instances of commercial failure.

Continuous Operations

TEST

No need to ever advise clients of system downtime.

The system implementation should allow for continuous operations, that is, all software and hardware components (not just the application) should be hot-swoppable without suspension of service. Degraded performance and requiring the user to log in again is acceptable during such periods. By removing the need for maintenance windows, this provides much greater deployment flexibility.

TEST

Moving clients A-K to system 1 and L-Z to system 2, without disruption

Access Control – Clients

All client data should be logically grouped on a per client basis to allow backup/ encryption/ load balancing etc operations to apply to the client(s) as appropriate.

TEST

Migrate everyone to the new version except two clients

Multi-tenancy Flexibility

It's highly desirable to always run a single image for all clients. However, there will always be situations (e.g. demos, testing) where multiple images are needed, as well as potentially the flexibility to support different clients with different images.

Security

TEST

No opportunity for anyone to criticize security policy

Security is a cross-cutting concern, so it's important that no architectural choices limit the ability to implement best-practice security. This primarily affects Web API and Access Control areas, but does potentially cover everything. Further, security should be embedded in the architectural level where possible. Whilst no system can guarantee 100% security, equally there is no reason to make unforced errors such as LinkedIn made recently (not adequately protecting passwords) which damage market trust.

Managing Development

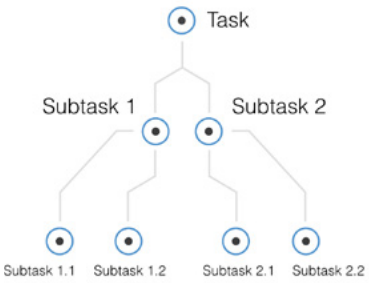
Accurate scheduling is the bane of all product development. "It will be in production end of April". Everyone rolls their eyes. No one believes it. Not even the person saying it. It doesn't have to be this way. Here are six key elements to put your software development estimates back on track:

Breaking Down Tasks

This shouldn't need to be stated, but given the mushrooming of IT jargon around Agile, sprints etc, it seems what ought to be simple common sense apparently isn't. There is an active competition between software developers and management consultants for jargonizing statements of the bleedin' obvious.

Breaking down larger tasks into smaller ones has been part of our communal knowledge for all of recorded history. You can build pyramids with it. Since even apes and dogs know this (Department of Developmental and Comparative Psychology), it's not clear to me why we need an Agile manifesto (Principles behind the Agile Manifesto) to remind us that continual delivery is a good thing.

Despite the obviousness of all of this, there are some common issues, which while everyone will agree with in the abstract, somehow don't seem to get implemented as rigorously as they might.



» Break down main tasks into simple and manageable subtasks to ensure deliverability.

Granularity. The tasks need to be broken down into sufficiently small pieces that they are fully understood. Practically speaking, this might equate to a < 10 line function in Python or Clojure or a < 200 line class in Java, described as a single sentence. You should be able to accurately determine how long it will take you to write and get working such a piece. Of course this means doing most of the design and all of the architecture up-front. This is time consuming, but the only alternative is guessing, and that hasn't worked out well for the industry. It's also likely that at this level of granularity, some (lots?) will be black holes, but then the only solution is to do a quick proof of concept to validate the approach in this area. Again, the criticism is that this takes time, but again, the guesstimate option ultimately leads to more pain. If you take the approach above, a mid-size project of something like 50K lines of Java, would require 250 lines of pseudo code to estimate, or about a 10-20 page document. It's not an unreasonable overhead. The level of granularity is determined by the level you can accurately estimate.

Decouple. Reductionism is not just about breaking things into smaller pieces, it's crucial that those elements are independent. A common criticism of the approach here is that whilst yes, you can describe a system with a pseudocode, just like functional specifications, there are likely to be significant holes in the logic that are only apparent when you try and write the code. It is an issue, but the solution is to focus on decoupling the functionality as far as is humanly possible. There should be a religious obsession with micro-API's at every point. Coupling not only introduces additional bugs, but it also slows development as John waits upon Jane to complete something, rather than both working to a fixed interface. If any of these dependencies are external, then of course that makes the problem even worse.

Feedback. Software is not written until the end user says it is. Again nothing new in this. We called it Rapid Application Development in the 90's, Agile today, and I'm sure we'll invent a new term for it in the next decade. There is a separate problem in that functional specifications are seldom worth the paper they are not written down on which makes matters worse, but regardless it is essential to continuously check back with the end-user. This does not just help the obvious point about keeping things on track, but by continuously saying "this will be ready in 3 days" and delivering in 3 days, your reputation is built up more rapidly than the more infrequent "this will be ready in 3 months". Also, whilst most people will forgive you being a day over on a task, they won't be quite so charitable about being a month over - even if proportionally it's the same. Breaking down something into smaller parts also means breaking down the delivery.

Minimizing Unknown Unknowns

"Every estimate should start with the phrase "In my experience...", because if it doesn't, then you have no basis for that estimation."

Every estimate should start with the phrase *"In my experience..."*, because if it doesn't, then you have no basis for that estimation. This highlights absurd situations of estimating novel work, which by definition, cannot be done. You can't say "In my experience, it will take me XX months to write an application in a language or framework I've never used before".

You would expect a builder to be able to estimate pretty accurately how long it will take to build your new conservatory, but that is because the builder has built ten just like it previously, using the same techniques and materials, and knows all the common pitfalls. But ask for a conservatory

in the shape of a geodesic dome made of 3D printed parts, and you can't expect any meaningful estimate of how long it will take.

The moral of the story is that if hitting a deadline is important to you, then don't introduce any new techniques or tools. If you need both, then you might consider doing both - during both the second world war and cold war, the military would frequently commission two or more systems with ostensibly the same purpose - one would push the envelope of what was possible, and the other would be a boring incremental development, but with very low risk (V bomber). Today, with defence spending slashed, the military just focus on the glamour projects, but given the novelty of what they are attempting, every project ends (or gets cancelled) with ridiculous cost and schedule overruns because fundamentally these are largely *research* projects, not *engineering* projects as advertised.

If the project introduces some novelty, then do a proof-of-concept before estimation. Make clear the separation between research and engineering. Make sure management understand that you haven't done this before, which means not only is there no possibility of creating a schedule, but that it may not even work anyway...

The Time it Takes to Do Something... is not the Time It Takes to Do Something

> *"Generally I would say that a day's worth of work for me would take me a week in elapsed time to deliver."*

It's important to be realistic about not just how long something will take in actual effort, but also how that translates to elapsed time. Nobody can exclusively focus on a single task to the exclusion of other work and personal commitments. Generally I would say that a day's worth of work for me would take me a week in elapsed time to deliver. Your own mileage may vary, but I'm pretty sure you won't be able to clear your personal and work schedule sufficiently to deliver a day's worth of work within one day unless you are marooned on a desert island, or in prison.

As to the point made previously, any coupling between project tasks will automatically introduce delays as people have to wait on completion. It's for this reason that the famous project management rule of thumb that to halve your schedule, you would need 4 times or more the number of people because of the communication/synchronization overhead. It's even worse as there is an absolute floor, beyond which the schedule cannot go below, and indeed will merely increase with additional resource. See any government IT project for examples.

Management don't really care about how long something takes you (actual time), all they care about is when they will get the result (elapsed time). Make sure the estimate takes this into account. Don't say (or even think) *"if we made this my top priority, I could finish it by the weekend"*. Do say *"given everything else, it won't be ready until the end of the month"*.

Management will attempt to make this your top priority, but you need to push back, because they say that about every project. Always build in your own personal contingency -the kids will still need to be picked up from school, regardless of the company deadline.

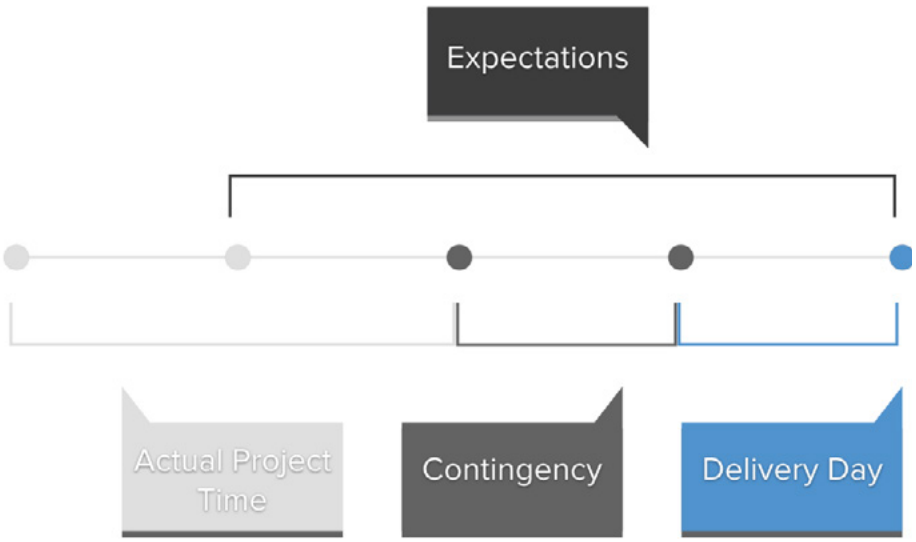
“There is no such thing as time management.”

Manage Expectations, not Time

There is no such thing as time management. The only way you can manage time is to travel close to the speed of light and you don’t have the budget for that. There is, however, such a thing as expectation management.

It’s better to deliver 3 projects on time, than deliver 2 projects early and one late. The reason is that your work does not exist in a vacuum, but is a part of the greater whole. Being early does not necessarily help, but being late most definitely hinders. The key is to be consistent and reliable.

The way to ensure this is to operate a “bank”. If you finish a task early, avoid the temptation to mouth off how quick you are. Tuck it away in your top drawer. Then, next week, when you hit some roadblock in the rest of the project, you can maintain the illusion of progress, by magically pulling this previously completed task out of the hat.



» *Operate like a bank - store away your completed tasks for when you hit a road block. Always leave contingency.*

Just like dogs and infants, management need consistency and a regular routine if you want them to behave properly.

Ownership

Personal credibility and accountability are vital. I hate it when a developer tells me something is late because “it took longer than expected”. Paraphrasing is not answering, and it sure as hell is not delivering. Drilling down (a painful process for both of us) typically highlights some or all of the issues above “couldn’t get Framework X to work with Y, waiting on Susan, had to take some personal days, didn’t realise we needed to implement Z”. What gets me is that so many people think these excuses are simply acts of God beyond their control, or, more to the point, beyond reasonable anticipation.

What does this mean in practice for developers?

1 Minimise moving parts. One theme We've been trying to bring out here is that software development is not uniform. There is software engineering - basically doing something you have done before and software research - which is doing something you haven't done before. It's important to be really clear on the distinction, to understand and communicate which goes into which bucket. Whilst it may be boring and won't do much to further your CV buzzword count, your management will want you to minimise software research. But earning a reputation for accurate estimation will earn you far more respect than tinkering with 63 javascript frameworks (A JS framework on every table)

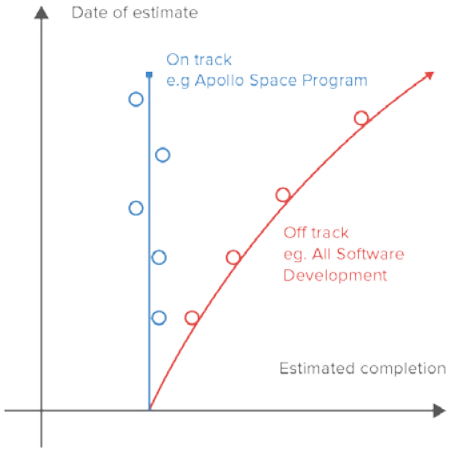
2 Take the smooth with the rough. Yes, you read that right. No one can work 60.hour weeks continuously. But you can work one 60-hour week if it's topped and tailed with two 30-hour weeks. As an experiment to demonstrate this, you might like to try running a marathon, but sprinting from the start line as fast as you can. After 300m, you will be ahead of most of the field, but after 1km you'll be collapsed by the side with cramp. Pace yourself. There is no more need to be guilty about finishing early than there is to brag about working late. Let the boss think you just work 40.hour weeks, week in, week out, because you are on top of things (which of course, you actually are). Look at how sales people work - they don't do guilt. I've never seen anyone work as hard as sales people, nor goof off as much. It's not a contradiction.

3 Just say no. It's OK to tell your boss to 'eff off if you are presented with an unreasonable schedule (which by definition is any schedule you didn't personally create). It may seem better to be diplomatic, but not at the expense of clarity. Being held accountable is not all bad. It means you get considerable freedom - essentially commensurate with your track record - to run things as you want. But just as doctors and lawyers will refuse to treat/represent clients in ways they disagree with, software professionals need to be prepared to say no to unreasonable scheduling. Yes, it might end up with you being fired, but then either they don't deserve you, or the've seen you deliver sufficient times to know that you know what you are talking about, and take your advice reluctantly as they would from any other professional. Just never say "it took longer than expected". Never say "I didn't agree to that schedule anyway", because unless you threw the chair at your manager at the time, you did agree by continuing to work on it. But once you've committed to a schedule that you created, then even if it takes an 80 hour week, just deliver. Own it.

Measure to Manage

Whilst I am intrinsically more of a "if you need to measure an improvement, then no improvement has been made" sort of person, software project estimation is one place where the adage "if you're not measuring, you're not managing" holds sway.

Well-known software commentator, Joel Spolsky has an excellent article on this where he introduces a complete system - evidence based scheduling - Evidence Based Scheduling. In fact for most organizations, merely recording how the estimate changes over time would be a massive improvement on the current vacuum. It's worth noting in passing that sales people and their sales forecasts have the precisely same problem, although sales people are as a group are usually better at expectation management than their developer cousins.



» *The BS Curve*

It's colloquially known in project management circles as the 'BS curve' - a chart of projected completion date against date of estimate. A well-managed project tracks on the line, the out-of-control project starts to curve off. Monitoring this for different projects and developers is a useful communication and analysis tool, in forcing people to confront past mistakes and modify future behaviour.

There are few areas in business where the gap between what is needed and the current state of the art is as large as in software development estimation. But it need not be that way. JFK said he would land a man on the moon before the decade was out, and he did. Estimating how long it takes to implement an accounting package ought to be possible.

"Sometimes the key to moving faster as a team is to go slower as individuals."

It does take focus, and a lot of seemingly hard work, but it can be done. Perhaps the most important non-trivial takeaway from all of this is that the detailed design should be completed before any estimation is attempted, and a ban on all research. Don't just wave a finger in the air and then dive into code. Writing pseudocode and design documents has become unfashionable in today's hacker culture, but the moment you have other people needing to synchronize their own work with development milestones, it becomes essential.

Sometimes the key to moving faster as a team is to go slower as individuals.

Validating the MVP

Validating the MVP

Simply having an MVP is not enough. It must be validated with actual users.

Here are some practical techniques to motivate feedback.

1 **Bounty system.** Well established for soliciting security bugs, this could easily be extended. The reward does not need to be in cash - it can be as much as bragging rights, or some free use of the product.

2 **Intercom in-app messaging.** I first came across this when I was evaluating Quip and found it very useful as a new user. We've implemented this ourselves at SalesSeek and it's had a massive impact on user engagement. Being able to chat there and then, in the context of the app, reduces friction. Intercom has a great mobile app, so your staff don't have to be in the office to reply - at SalesSeek we respond 24x7, even if half the time we're in the bar...

3 **Act on their feedback.** There is nothing more satisfying than recommending some new enhancement and seeing it implemented. Conversely, there is nothing more demoralizing than sending in feedback to a black hole and hearing nothing. If you look at some companies forum support sites you see a lot of bad examples - I know one SaaS company where users have been asking for custom fields for years, with no recognition of that from the vendor. It's a massive turn-off to anyone thinking of submitting a new idea. Instead make a point of praising all new ideas and clearly saying a) we will do this in X time frame or b) we won't do this for the following reason.

To bootstrap this, write a blog or otherwise highlight elements of the product that were the result of user feedback.

Will someone steal my idea?

If the idea is good, it will be copied. Some of these may be better. But it's important to keep this in perspective. **Large companies are the last people to worry about. See the article below for more info**:

Startups: Why Google/ Facebook/ Apple/ SalesForce (probably) won't steal your lunch money

Facebook is interesting because it came into a market already dominated by MySpace, yet still managed to "come from behind" by fine-tuning the detail. MySpace's boss puts it down to Facebook using real identities rather than MySpace's pseudonyms. This is an interesting thesis, because right now a lot of people are equating Twitter's fall from grace with the explosion of vitriol on the site - a consequence of anonymity.

Another common view was that MySpace, in allowing unlimited customization produced a visually chaotic network, was uglier than the more constrained Facebook pages. Others simply blame acquisition by "suits" at News Corp for cultural mis-match and mis.management.

Whichever reason you subscribe to, the point is the same -**the devil is in the detail.** This is a level of execution that is seldom copied as it would be too obvious.

A bigger worry are other startups - some of whom may already be out there but you don't yet know about. It's worth **understanding completely your competition** to figure out what your "using real rather than anonymous names" edge will be.


ONE WAY
ONE WAY

What’s Next?

What's Next?

So now you have your MVP, does this mean you are ready to approach investors?

That depends. A community based product needs a community before you get the V in MVP.

If you had, say, something like a graphics design tool, you could build the tool, show it to a few graphics designers and (with their feedback "this is awesome, I'd pay $$$$ for this") then hit the investors.

But if you take a tool like Facebook - there isn't a whole lot there as a pure application. It's a website where you add stuff like text and pictures, and other people see it and can comment on it. It's assumed anyone could code that - because anyone could.

> *"Minimum Viable Product is not just about version 1.0. It's an approach that should carry forth into every product development."*

What's difficult is getting a community, which presumably you are seeking to monetize in some way? If so, then practically speaking, **the community is your product**. Clearly the website is part of it, but that is 1% of the overall MVP here. It's tempting to think of your community solely as your users, but in reality they are both your users and your product.

Whilst no one expects you to hit 1M users on day one, they will need to see clear evidence of momentum. Without be Angels who buy into your vision, but the issue with taking investment at this stage is that the risk is so high, your valuation would be minimal. This caps the amount of money you could raise as well as reducing your upside.

Minimum Viable Product is not just about version 1.0. It's an approach that should carry forth into every product development. It's the first step of the Agile methodology, variously described as ship early, ship often. Apple's iPad pretty much invented the tablet market (Yes, there were tablets before, but they had no customers). At the time of it's launch many commentators were mystified that such basic functionality as printing was missing. But it was the ruthless removal of functionality that enabled Apple to ship qucikly and with the simplest possible experience. Of course now, the iPad supports printing, but this is because now is the right time. This is the MVP in action.


SalesSeek
www.salesseek.net