## 1.1 What is UX?

**In our quest to learn about UX, an important place to start is to be clear about what we mean by the term “user experience”.**

**We know that companies like Google, Apple and Facebook invest massively in their user experience design teams. But what is this “user experience” that they’re trying to design?**

Well, if you type “user experience” into Google, and you’ll find this definition on Wikipedia. Which kind of makes it the semi-official definition.

It says “user experience includes all the users' emotions, beliefs, preferences, perceptions, physical and psychological responses, behaviours and accomplishments that occur before, during and after use of a product, system or service”.

Now that’s a very technical and very academic definition. And what I think they’re trying to say - and what’s a much more precise definition - is that user experience is what it feels like to use a product system or service.

**So user experience is not just about usability. It’s not just about how fast you can get through a process. Or how few mistakes you make when doing so. And it’s not just about functionality. The ability to do A or B or C. How users *feel* when they’re doing A or B or C is just as important.**

What sort of feelings are we talking about? Often they’re subtle, but extremely powerful. Consider these:

* I feel in control
* I feel confident
* I feel smart
* I trust this product

Simple emotions and very powerful when you consider their opposites. I feel out of control or I have no control. I’m doing something important - like buying a house - and it feels like the system or the process or some anonymous account handler has more control than I do.

I don’t feel confident about this system or product or company. I don’t think it’s going to work well or do what I want it to do. I feel stupid. Or I don’t trust this website or app or the company behind it.

These are not the sort of emotions we want our users and our customers to feel when their using our products.

**So UX isn’t just about solving problems as we discussed in the first video. It’s also about generating positive emotions and we’re going to explain this emotional aspect different types of design that go into making high quality products. The first two - functional design and aesthetic design - are generally understood. But the third - experience design - isn’t as familiar and often gets overlooked.**

Let’s take a look at each one.

Functional design determines what a product is built to do. It defines the engineering that gives a product its capabilities. So with a car, for example, things like horse power, engine size, the type of transmission, how fast it can go from 0 to 60 will determine what sort of car it is - whether its a sports car, a family car or an SUV. Each of these types of car would have a different functional design underpinning it.

The second type of design is aesthetic design. How does the product look? How visually appealing is it? What sort of personality does it have? What do its looks say about the product’s brand? And you can imagine that the sports car, the family car and the SUV would each have a different aesthetic design corresponding to the type of car they are.

The third category of design, the one that’s less familiar to us all, is experience design. What does it feel like to drive this car. What does it feel like to sit in the driver’s seat, hands on the wheel, as you drive this car along the road.

How responsive is the steering wheel? How smooth is the gear stick. How easy is it to adjust the controls on the dash.

What sort of sounds does the car make? What sound does the engine make - for example - when it’s idling in traffic? Or cruising on a motorway? Or overtaking on a country lane?

What’s it like to do mundane things like stick your golf clubs in the boot? Strap the kids into the back seat? Or adjust the position of the driver's seat?

**Consider this important point. All these small details on their own might seem trivial or mundane. But when added together, these details determine what it *feels* *like* to use this car. And companies that are serious about creating great products don’t leave any of these details to chance.**

**Because - here’s another important point - experience design doesn’t happen by accident. It’s a deliberate, intentional focus on the small details that matter to customers.**

Mercedes, for example, doesn’t the workers on the factory floor determine the smoothness of the gear stick. Or the sound of the engine. Or even the feel of the buttons on the dash. Of course not.

And it’s not that their workers aren’t capable. It’s just that Mercedes knows that these details create a massive emotional reaction with customers. And they want that emotional reaction to be a positive one. So these details are too important to be left to chance. And they are defined long before the car hits the factory floor.

**We talked about the three types of design in a sequence: functional design, aesthetic design and experience design. But that’s not how it works in companies like Mercedes. You don’t leave experience design until last.**

All these small little details are defined at the same time as the functional and aesthetic design. The three types of design need to be working together in harmony. And we’ll see an example of that in another video when we look at the design process at BMW.

Finally, the details aren’t there for fun. Each detail has been carefully crafted to solve a genuine problem for users, be it air conditioning, entertainment, navigation or pacifying kids.

Before we move on to the next video it’s worth recapping what we’ve learned about UX. If you’re ever asked in an interview “What is UX”? A smart answer is to say that it’s a problem solving discipline. Identifying user problems and building software to help solve them

On top of that, you can also say that great products solve user problems *and* generating positive emotions while doing so.

**In the next video we’ll look at another example of experience design, and how it shows that functional and aesthetic design, on their own, aren’t enough to create a great product.**

# 

## 1.2 Why experience matters

**Let’s look at another example of the three types of design. This time from the smartphone industry.**

These are some of the many functional design drawings of the technology underpinning the Blackberry 10 smartphone. This was the first Blackberry touch screen smartphone released using Blackberry’s new touch-ready operating system. It came to market in early 2013.

Now, Blackberry had a great track record when it came to technology - the Blackberry messaging service, getting email on your phone, corporate-level security - these were all innovations that Blackberry brought to the handset market. So let’s just assume for argument's sake that they kept up this track record when it came to the functional design of this new phone.

Aesthetically, the phone looks the part. Here are some screenshots of the key screens of the phone, and it looks as good as the other operating systems did in 2013. We’re talking about iOS 6 - the ugly one before the new flat design was released in iOS 7. And long before Google’s Material Design was implemented in Android.

Consider the design of the Blackberry Hub, shown here. And the time this was a new way of presenting a user’s incoming communications: emails, text messages, tweets and so on - all displayed in one stream. You’re probably familiar with the concept, as it has since been adopted by both iOS and Android, but seen first on Blackberries.

So again, it's safe to assume that when it comes to aesthetic design, the Blackberry 10 is doing the business. As far as experience design goes, let’s watch a video of five people using the Blackberry 10 for the very first time.

*[Mashable video here]*

Not surprisingly - looking at that video - the phone was a flop. A few months after the release of the phone, Blackberry had so much unsold stock of the Z10, that it was technically bankrupt and put itself up for sale.

The phone that was supposed to rescue the company had failed badly. And a company that once dominated the handset market, was soon to exit the market altogether.

Why did the phone fail? Why was it so difficult to use? Will explain that in more detail later. The short answer is that the phone did something radically different than existing phones: it had no home button. And the alternative they developed - a series of swipes and gestures - wasn’t sufficiently well designed. Users were confused doing the most fundamental task: navigation…

**But more about that later. The key point here is that functional design and aesthetic design, on their own, aren’t enough. If you don’t get the experience design right, those other two things won’t even matter.**

**Think about the people we saw in that video, who were generally light hearted and laughed their way through the experience. And the reason they were light hearted was because they didn’t pay for those phones. If you had spent several hundred dollars buying a Blackberry 10 - or you were locked into one on a 2 year contract - and you brought the phone home and couldn’t turn it on, I reckon your emotions wouldn’t be light hearted. They would probably range from frustration to anger to regret. And those aren’t the sort of feelings we want people to have when they use our product.**

**Because when people are feeling this way, there isn’t any room in the brains to admire our functional or aesthetic design. Their negative emotions will override everything else. And all the hard work spent on our clever functional design and beautiful aesthetic design will be wasted.**

**Let’s consider the key points from this example:**

* Great products need all three types of design. If your team or organisation is relying on functional and aesthetic design alone, you are being exposed to the same risk of failure as the Blackberry 10. The two types of design on their own are not enough
* Negative emotions have a multiplier effect. In other words one negative experience can outweigh many positive ones. Because people don’t notice good design as much as they get frustrated by bad design. Without a dedicated focus on experience design, the risk of negative emotions outweighing positive ones gets higher

## 

## 1.3 UX is a state of mind

So far we’ve looked at two examples from organizations with large budgets. One who uses its money wisely to make excellent products. And the other who - despite plenty of resources - still managed to get it wrong.

**The final example we’ll use involves an organisation with practically no budget. And I want to use this example to emphasize a key point. UX isn’t about money. Having the right state of mind is much more important.**

This is the exterior of a small guesthouse or B&B on the Dingle Peninsula on the west coast of Ireland. Outside of Dublin, the Dingle Peninsula is probably the most popular tourist destination in the country. So if you run a B&B, it’s a competitive market.

From a functional design perspective, it couldn’t be more functional. It’s just a big house with lots of bedrooms in it for guests.

Aesthetically, it’s definitely got charm. The bedrooms look really nice, and the owners are obviously making an effort. But there’s nothing in this room that you probably couldn’t buy in IKEA. So it’s nice, but it’s not the Ritz.

But when it comes to experience design, the owners are pulling out all the stops. There are gourmet yogurts and Bloody Marys for breakfast. Beautiful flowers in the garden that maximise the sea views. When you order herbal tea, you get real herbs in the pot. And not alone do you get free range eggs for breakfast. You get free range chickens running around the front lawn, which is manicured like centre court in Wimbledon.

And customers notice these details. This B&B has been ranked number one in its market for several years. Nearly all the reviews refer back to these small details. Two examples here mention:

* The great food and beautiful rooms
* The bed is so comfortable
* The linen smells so sweet
* They turn the beds down at night
* All the touches were provided

The owner of this B&B doesn’t have a big budget or a large team. He’s not part of a chain or a franchise. And although I’m sure he’s never heard the term experience designer, that’s actually what he is.

He’s got empathy for his customers. He observes them closely to understand what really matters to them. And he’s constantly tweaking and improving and experimenting with his product to get the details right.

**And here’s the critical point. His B&B isn’t number one in the market because of its bland functional design. And it’s not number one in the market because of its IKEA-inspired interior design. It’s number one because of all the small details added together that create a knock-out experience that customers love, and vote about, and talk about and spread the word about.**

Now, at this point you might be asking So What? I’ve been watching several videos on this UX course and none of them are about software. Why do I care about phones or cars or B&Bs.

Well, let me explain. Consider these companies. Not all of them are in the software industry per se. Some are in the travel industry or news or retail or automotive. But all of them realise that software is now critical to their success. And they invest massively in UX design.

Why? Because in a competitive/commoditized industry like airline travel- for example - one of the few ways to differentiate yourself from the competition is through great experience design. Ryanair, the European budget airline realised this recently.

**And all our customers are using products designed and developed by companies like Google and Facebook and Apple. And these companies are setting the bar very high when it comes to what people consider to be good software. So if we’re not spending time considering these little details and understanding what details matter to our customers and understanding how we can design them, then our software is going to feel second-rate in comparison.**

# 

## 1.4 Product integrity

**This entire course is structured to explain what an experience designer does. And we’ll be spending a lot of time looking at the tasks and techniques that are the experience designer’s responsibility.**

**But at a very high level, there are two overarching objectives for the UX designer. And these are to do with product integrity and product desirability. Let’s look at product integrity first.**

In any organisation the obstacles to creating a great user experience generally don’t come from the users. The obstacles come from within the organisation that’s trying to create the experience itself.

You might recognise some of these stakeholders and some of these roles. And what typically happens on a project, is that these stakeholders are pulling and tugging at the product, feeding their requirements into the product’s design. YES this feature should be in the product, or NO this feature should not be in the product.

And that’s fine, that’s their job. They’re trying to realise some of their objectives through the release of the product.

**But the problem is none of these stakeholders typically have any product design skills. They’re not thinking about the small little details that go into creating a great experience for the user. Often they're not thinking about the user at all. Instead they’re just thinking of their own narrow piece of the puzzle.**

And what’s missing from this picture are two things. Somebody representing the user. And somebody representing the product.

And the job of the experience designer is not just to represent the user when all these conversations are taking place and these requirements are being gathered… but also to represent the integrity of the product itself.

I’ll explain what I mean by product integrity using this app as an example. If you live in the UK and Ireland you might be familiar with Hailo, a taxi app, very similar to Uber, and certainly a very popular app in Dublin.

Here’s how it works. Once you launch the app, it zooms to your current location, showing you nearby taxis. Press “Pick me up here” and then “Hailo now” and, hey presto, it starts looking for a cab. And in a few seconds I have a driver matched who will pick me up in 12 minutes.

What people love about the app is the immediacy and simplicity. It’s right here, right now, with just two presses a button - no messing around.

But let’s consider all the things this popular app doesn’t do.

* You can’t order a taxi for some other time. You can only order a taxi right now. So you can’t pre-order a taxi for that early morning trip to the airport for example
* You also can’t order a taxi for somewhere else. So you can’t send a taxi to pick your husband or girlfriend or parents up from the station, for example
* And finally, you can only order one single taxi. So you can’t order multiple taxis to take your team out to dinner after work, let’s say

You might think that these are serious weaknesses in a taxi finding app, but it’s precisely because it doesn’t do these things that the app is so great.

You can imagine if the app was made and developed by a larger organisation - let’s say by the organisation that operates taxis in Dublin - that various stakeholders within the organisation would be demanding that these additional features be included. And look what would happen.

* Well users have to be able to tell us what time they want us to pick them up
* They might want more than one taxi so we have to give them this option
* They might not actually be at the address, so they have to be able to tell us where they want to be picked up at
* And now they're ready to place the order…. once they’ve agreed to our terms and conditions of course

So these requirements, while well-meaning, are counter-productive for everybody. They make the app less appealing and more difficult to use. Meaning less usage and less fares for the drivers. A lose-lose situation.

**And that’s what happens in so many organisations, where unstructured requirements gathering takes place without a strong voice representing the product. Yes, some of these features might seem valid, but if they damage the product and the business, are they really worth it?**

**And that’s the role and responsibility of the experience designer; to represent the product and to keep sight of what’s best for the product, the business and the user.**

Before we move on to the next video, let’s clarify what gives a product integrity:

* The product is coherent - it’s clear what it does and what it is about
* The product has a singular purpose - it’s not trying to be all things to all people
* The product is focused on solving user problems - it doesn’t become a vehicle primarily focused on solving stakeholder goals

# 

# 

## 1.5 Product desirability

**The second high-level responsibility of the experience designer, is to focus on desirability. I’ll explain this concept by talking about three key ingredients of any successful product: viability, feasibility, desirability.**

Viability concerns money. Any product has to make money or save money for the company that’s creating it. And the amount of money it makes or saves has to be less than the cost of building it. There might be other reasons for building a product, but often it comes to money.

Feasibility refers to technology. It’s got to be build-able. If not by us, then by somebody we hire to build it. And it’s got to be built at a price that makes it viable, and in a timeframe that’s reasonable.

Desirability means that somebody has to need or want this product. In other words, it has to be solving a problem for somebody - or else it shouldn't exist.

And if it is solving a problem, and people want to use it, then when they do come to use it, it’s got to create a positive feeling that makes them want to use it again.

Many companies focus on the first two variables - feasibility and viability - because they lack the skills and knowledge to look at the third. That's why you see so many products that are viable and feasible, but that nobody needs to use, or wants to use, or enjoys using.

A great example of this is the jukebox 6000 by Archos. This was one of the very first portable MP3 players. Released in December 2000, about 10 months before the first iPod.

From a viability and feasibility perspective, things couldn’t have been better for Archos. They had figured out a way to mass produce this product, at a cost that was lower than the price they sold them for, just at the moment in time when this market, for portable MP3 players, was about to take off.

Absolutely perfect, you would think. Except… when it comes to desirability.

If you remember the first iPod, there were so many things about it that seemed new and different to other consumer electronics products at the time. And these were the things that people loved:

* It was small
* It was minimalist
* People thought it was beautiful
* It was easy to use
* And that ease of use was exemplified by the scroll wheel, that allowed you to scroll through hundreds or thousands of songs in a matter of seconds

And the Jukebox 6000 didn’t stack up on all these factors.

* It wasn’t beautiful or minimalist
* It was relatively big and chunky
* And it was a little heavy

When it came to transferring music from your PC to the device, it was slow. Slower than the iPod.

Also the 6000 wasn’t plug and play, you couldn’t just stick it in to your computer and get started. Instead you had to go to the Archos website, and download some drivers, install them on your PC, and that was never a great experience

And it didn't have a scroll wheel or anything like it. So to select song 385th on your list, you had to tab through your collection 385 times.

**So despite the 10 months head start they had over Apple, despite being feasible and viable, the 6000 wasn’t desirable.**

**And that opened up a gap for a product that solved the same basic problem - portable mp3 playing - but which also delivered a fantastic experience while doing so.**

**And it illustrates the point, that feasibility and viability on their own aren’t enough to ensure success. Your product needs to have desirability as well.**

As an experienced designer it's up to you to ensure that your product is desirable.

Think of it as three concentric circles. The first priority is to identify whether there is an actual problem that your product can help solve - does anybody need it? This is the most fundamental requirement.

If there is a problem, then we have to ask if our product is actually solving the problem. Another basic requirement.

And if it is solving the problem, we can’t stop there. We have to make sure the product isn’t solving it in a mere functional way like the Archos. It has to be solving the problem in a way that creates a great experience while doing so. That’s desirability.

[Colman - possibly introduce the lovable triangle here]

## 1.6 Making a business case

**Throughout your UX career, it’s highly probable that you will spend most of your time working in a business environment. That is, working for an organisation where profit and loss, the bottom line, is the key business driver.**

**And most people working in a business environment don’t have much interest in UX. They don’t care so much about user research, different types of design, personas, prototypes, wireframes and all the rest.**

Instead, they’re much more interested in the OUTCOMES of UX. What can UX do for them? For their team? Their department? The company? The bottom line?

Being a really effective UX designer isn’t just about being good at design. It’s about being good at communicating the benefits of design. The business benefits.

To do that you must distinguish between the inputs of UX; and the outputs or outcomes.

You need to be an expert at the inputs: research, analysis, design and prototyping. But you have to understand that your colleagues mostly only care about the outputs. And those outputs can be summarised in one word: money

**Before we continue, let’s step back and be clear about the objectives for this lesson. We want you to be able to build a business case for your UX projects. We want you to be able to communicate the business benefits in terms your stakeholders or clients will understand. And we want you to be familiar with some of the most common business metrics.**

First let’s talk about business benefits. Ultimately there are only two benefits to a business:

* Increase revenue
* Reduce costs

There might be other ways of talking about them, but really it all boils down to whether the activity the business is undertaking will make money or save money.

When we elaborate a little more, we see terms like these being used:

* Increase revenue
* Reduce cost
* Increase customer acquisition: customers being a source of future revenue
* Increase conversions: getting more people to do what we want them to do - register, buy, contact us, and so on
* Increase customer satisfaction: which will lead to more repeat business and revenue
* INcrease customer retention: keeping customers for longer = more revenue
* Reduce customer churn: churn is another word for losing customers
* Reduce time to market: getting our products into the market sooner so we can sell sooner and - of course - increase revenue

If you work for an organization that sells products or services online, such as an airline, then your projects could potentially help with the following:

* Increasing sales
* Increasing sales / visitor
* Increasing revenue / sales

**Let’s look at a few case studies where UX projects had a positive impact on the bottom line of businesses.**

The first case study we’ll look at is for Supervalu, the most popular supermarket chain in Ireland. Like many retailers it faces fierce competition. In this case the competition comes from other irish brands, as well big UK chains, and the German discounters.

The project at hand was to redesign Supervalu's online shopping, the new frontier in this competitive landscape.

But it wasn’t a redesign for the sake of it. It had to deliver real, tangible business outcomes. And by tangible I mean, outcomes that had an impact on the bottom line, that involve money.

**The bulk of the work was carried out by research agency Bricolage, a little bit of help from ourselves and UX consultancy Each & other. They followed the process, using research to identify problems that could be fixed, designing simple but effective solutions, such as decluttering the interface, and reducing the number of steps at checkout, and then working with the development team to make sure everything was implemented correctly.**

The design outcome was a beautiful new, streamlined shopping experience for users. But the business outcomes are what made it a success:

* 51% increase in online sales: that’s revenue
* 84% increase in registrations: that’s more customers who will bring in more revenue
* 620% in mobile mobile shopping: and this is future revenue. If more shopping is going to be done on mobile in the future, this statistic is ensuring that the shopping platform is ready for that shift

The second case study also involves an irish business, Tesco Mobile, which is a small subsidiary of the giant British supermarket chain.

Tesco Mobile is a small brand in another highly competitive sector, mobile phone networks, dominated by two giants, Vodafone and Three.

The executive at Tesco Mobile wanted to redesign their online shopping platform to make it a more effective sales channel.

Again, by following the process, and particularly doing a lot of user research to identify the problems with the existing platform, the UX agency Fathom was able to deliver some incredible results, including:

* 9.5% increase in customers
* 17.7% increase in revenue
* 170% increase in order value: that is people were spending much more money each time they bought something

These are the types of results that will help a business audience understand the benefits of UX.

Our final case study takes a different viewpoint, focusing on cost reduction. The organization is Mozilla Corporation, a large not-for-profit organisation that makes Firefox web browser amongst other tools.

It’s a big company with half a billion dollars in revenue and over 1000 employees.

Every year millions of users seek support on the company's web forums. And the Mozilla team tried to answer them every day.

Customer support is a cost for every company. It’s easy to measure the cost and it’s easy to identify the top issues that customers have when calling or emailing customer support. It’s always a good place to start when building a business case for UX work.

In this example NNG worked with Mozilla to cut the number of questions being asked on the forums by a massive 70%. From 7000 questions per month to 2000 per month.

The numbers of questions answered within 24 hours increased from 40-60% to 80-90% which means they were delivering a vastly improved service for customers.

For many organisations these results would translate into reduced costs. Less money is being spent on customer support.

**When building a business case what you’re trying to do is to convince your organisation that the money spent on the UX project will ultimately increase revenue or reduce costs by the same amount or higher.**

**So you need to do your homework. Estimate how much will be spent on the project. And estimate how much additional revenue or reduced costs the project will accrue to the business.**

**Be realistic. And make sure you have some evidence for your claims.**

For example, a project that aims to reduce the number of steps on the checkout process. If you can point to statistics that say 20% of customers drop out of the sales funnel during checkout. And that this translates to a loss of 500k. And that your project will hope to reduce the dropout by 20% and increase revenue by approximately 100k. That’s a solid business case assuming the costs of designing and developing the new checkout are less than 100k.

**Your business case doesn’t have to be a large document. A short, well-reasoned case for your work, with clear business benefits is all you need.**

# 

## 1.7 UX is a process

**Process isn’t the most exciting word, but UX is nothing if not a process.**

**And companies that are serious about design, like Apple or Mercedes, are also serious about following the design process.**

And, at a high level, this is it. The classic design process that’s used whether people are designing a car or a phone or a chair or a house.

And it couldn’t be more simple or logical.

Working backwards you can see how that logic unfolds. Before you release a product to your customers you need to test it. Does work, is it safe, will it explode when anybody turns it on.

Before you test it, you have to build it. That’s self-evident.

Before you build it, if you’re smart you will design it. To make sure everybody on the team know what it is we’re building.

And before you build it, you’ll do some research to understand, what is it we want to build, and why.

This is fundamentally it. But in reality it’s slightly more refined. Let’s take a closer look at process. We start with research where we engage with our customers or end users to understand the problems we’re trying to solve for them.

When you conduct research you’re gathering data and you need to spend time analyzing that data to clearly articulate the precise problem or set of problems we’re trying to solve.

Design is where we solve the problem. That is; we design a solution that we think is going to work for our users. And part of the design process is to create a prototype, a mockup of our solution that we can test with users to validate that it does actually solve the problem, before we go to the expensive business of building it.

This is an iterative, circular process that may take a few cycles before we’re happy that our design really work. And only at this point are we ready to build and test and ship the product.

For the moment, let’s go back to the simplified version of the process. And let’s use a real-world example, and see how the design process might have played out during the creation of this building. This is a school in the midlands in Ireland, an award-winning school, that looks nothing like the school that I went to as a kid.

Working backwards you can imagine the design process went something like this:

Before they let the kids into the school, everything was tested rigorously. The fire alarms, sprinklers, Bunsen burners in the science lab. Just to make sure that everything was working properly and safe.

Before they conducting the testing, the school was built. Completely built. There was no scaffolding, or blocks, or workman’s tools lying about when the kids entered the school. Absolutely everything was finished.

And before any construction started, before they started digging the foundations, the school was designed.

And the design would have been incredibly detailed. It’s not enough to say that we want light on the ceiling, you need to specify exactly where they go and what type of light fittings you want; the exact position of the windows and the types of material for the window frames, and so on.

And…. before the school was designed you can imagine that the architects had some fundamental research to do. First, what sort of building do you want us to design? Is it going to be school, or a hospital, or a hotel? And if is a school. What sort of school is it going to be?

**Is it a primary school, or a secondary school, are both? Is it a boys school, or a girls school, or both? Will there be 100 pupils or 1000 pupils? Is it going to be sporty or is it going to be academic? Is it in an urban location, where everybody walks? Or is it in a rural location, where everybody drives? All these questions would need to be answered before the architect puts pen to paper and starts the design.**

And that’s the design process. But it's not all of it.

Because there are even more fundamental questions that need to be answered if you’re going to create a truly great product.

And whoever was responsible for this school — the principal or the board of management — tried very hard to answer these questions.

* What sort of school to we want to create?
* What sort of ethos are we trying to impart form the design of this school?
* How do we want the kids to feel when they come into school every morning.

This is a vision. And it’s the role of the experience designer to help define this vision.

And to create a truly great product you need to spend time up front, before any designing or building takes place, defining what sort of product you want to create. Which is why the research phase - or the discovery phase, or whatever you want to call it, is probably the most important phase of the process.

What’s magical about following this process is not just the output, but several other fringe benefits that make the product development process so much smoother.

First the vision is clear and the product is visualised in high fidelity from the beginning. Everybody knows what they’re building. Everybody knows that the product is going to look like when they’re done.

**And this is particularly important in software, because software is invisible until it’s finished. And it’s too easy for everybody working in a software project to have a different idea in their heads as to what is actually being built.**

The process also has a natural structure around which all the stakeholders and actors in a project can coalesce. This process works for everybody, not just UX designers. In my experience, project managers, developers, testers and and all the other participants always prefer structured projects with clearly defined vision, objectives and designs. It makes everybody’s job easier.

Ideas can be iterated cheaply during the design phase. It’s much cheaper to change your mind when your sketching ideas on paper, than after construction has begun.

And finally, the three success factors: desirability, viability and feasibility, given equal importance upfront. It’s not a case of trying to slap on a bit of desirability at the end.

We have added content here about the business case for UX.

## 1.8 Agile and UX

This totally linear design process has been modified slightly to fit into the not-so-linear agile software development world.

**Some agile practitioners would claim that there is no place, and there is no time, for design in agile. In fact, you don’t need design if you’re truly agile. All you have to do is get your working code in front of users and modify it based on the feedback you receive.**

**While this sounds attractive – the fact that we can save time by not doing design – in reality it’s a false economy. The only thing you learn from putting badly designed software in front of users, is that people don’t like badly designed software.**

But it is a challenge to fit what is often a three or four month design life-cycle into a two or three week sprint. So let’s take a look at how it’s done.

First of all in an ideal world the linear design process wouldn’t actually be linear. In an ideal world we’d be in a continuous improvement cycle.

Once we’ve designed, built and tested our product, and released it to customers, we’re in a position to carry out research on the live product, and start designing improvements and repeating the process over again.

This is the ideal world of continuous deployment. And if you have a software as a service product, or a popular app in the app store, you probably live in this world.

In the classic waterfall project management process, those steps can be played out over a 12 month period, depending on the scale of the product that was being built.

But if you think about it, this is quite a risky approach. Anything could happen in those 12 months. You could run out of money during the design phase. And all the work completed up to that point would be thrown away.

You get a new boss while your working on your prototype. And he or she decides to change course, and cancels the project. All your work thrown away.

Competitors could release a new product shortly before you go live, changing your strategy. And rendering all your work up to that point, null and void.

Anything could happen over 12 months, and waiting that long to release any code into the wild could potentially see a lot of time and effort and money wasted.

One of the tenets of agile is: wouldn’t it be better if we released software more frequently and reduced this risk. Rather than waiting 12 months for one release how about we have a release every three months. And if you are being very rapid and are working on two-week sprint cycles how about four releases in two months.

And this is where people began to wonder, how or where does UX fit in to these very tight short development cycles.

And the answer came from these guys, IDEO, a product design company based in San Francisco, and Google Ventures, Google’s venture capital arm. These firms use a rapid fire methodology that allows them to work with their clients to identify problems, solve them, and build product prototypes, all within a short period of time, such as a week.

And their approach was adapted by Jeff Gothelf and Josh Seiden, in their book Lean UX. To summarise very briefly, what they are saying is that you still follow the design process but you just squeeze all the steps into a much shorter, more compact timeline.

And instead of trying to solve the big problem upfront, take the agile approach and start solving elements of the big problem, smaller problems essentially, piece by piece.

**~~Later in this course will take a more detailed look at how you can compress user experience design into shorter time periods. That module will be called running a design sprint. But for now it’s enough to know that the process is the process and it doesn’t change - despite what some people might say - even when it's being compressed to work in an agile development environment.~~**

# 

## 1.9 Humans are not machines

**There are things that are inherent about technology that make it complicated. And then there are things that we humans do that make it even more so. And we are going to look at both of these factors.**

**And there are good reasons for us to discuss this. If technology wasn’t complicated, there would have been no need for the user experience industry to appear and help simplify it. UX exists to solve a problem and it’s important and useful to us to understand what that problem is.**

A lot of material in this module is taken from two highly influential books in the UX space. The first is “The Design of Everyday Things” by Don Norman, first published in 1988.

The second is “The Inmates Are Running the Asylum” by Alan Cooper from 1998.

These books really set the template for what has become the UX profession. And we will touch on some of what they cover in this course, but we’ll only be scratching the surface. So I’d strongly recommend you have both these books in your UX library.

Back to the topic of technology being complicated, the first factor is that humans and computers are completely different beasts, as summarised in this table from Alan Coopers book. My laptop, for example, can remember 1 million phone numbers without making any mistakes.

But it doesn’t recognise my face. And I’m the only person it knows and I use it everyday.

Me, on the other hand, I can remember about three telephone numbers, on a good day. But I could recognise somebody on the street that I haven’t seen in 20 years.

Computers are incredibly fast, they’re error free – it’s the software we put into them that typically makes the errors – they are totally logical, and totally predictable.

Humans, in comparison, are incredibly slow, we make mistakes all the time, we’re not logical at all – we’re irrational, emotional, and unpredictable.

**And a lot of friction you see when you see people frustrated with computers, is this clash of the human and the machine. The human meeting the machine, and the machine is programmed to expect the human to behave like another machine - to be totally logical, not make any mistakes and to be really fast which, of course, we’re not.**

**And it's our job, the experienced designer’s job, to make the machine behave better. To be polite, to be nicer, to expect us to make mistakes, to expect us to be slow, and not to punish us for these human qualities. To be more considerate and understanding instead.**

## 1.10 The danger of features

**Another reason technology is complicated is because of features. While this may seem counterintuitive – surely features are a good thing – it’s actually one of the most important concepts for a UX designer to grasp. Features are not always a good thing.**

**Features and technology go hand-in-hand. What technology actually allows us to do, is to add more features to our products.**

We’ll examine this by looking at telephones. Some of you might be old enough to recognise this type of phone – known as a rotary phone in the United States.

This phone has only two features. You can make a call by lifting the handle and dialling the numbers or you can receive a call by lifting the handle and starting a conversation.

Because of the limited number of features, there are only two controls on the phone: the handset and the rotary dialler.

Because there are only two controls, the phone has a really simple uncluttered interface. Simple interfaces are more intuitive, and much easier to understand.

So simple in fact, that my three-year-old daughter knows how this phone works, even though she’d never seen her parents use a rotary phone before she came across this one at an exhibition.

Fast forward to the 1980s. We have new technology – integrated circuits, also known as microchips. These have allowed us to add new features to the phone. And these features are activated by these buttons here. So we have more features requiring additional controls, and now the interface is more crowded. And because of this, it has become less intuitive.

But hold on a second I might hear you say. Is it really less intuitive? It looks simple enough to me. Well this is an experiment that we conduct in our public classes. Hands up everybody who knows what the MR button does? People have a variety of answers.

“it dials the last number”. Are you sure? No.

“it’s memory recall”. What does that mean? I’m not sure.

Of over 800 people who came to our courses last year, nobody was able to confidently say what the MR button does.

**So straight away we have moved from a product that even a three-year-old can use, to a product where people are uncertain about what exactly it can do.**

What the MR button actually lets you do is save numbers to the phone. You can save up to 10 telephone numbers to the phone to save you the trouble of tapping the buttons everytime you want to call your mother for example. It was an early version of speed dial. So, if I want to save my wife's telephone number to the phone, what do I do? Again, this is one of the experiments we conduct in our classes and not one of our students has ever been able to get it right. Because of the convoluted process involved:

To save a number to the telephone, you need to do this:

* first you need to lift the handset
* then press the MR button
* then press the number of where you want to store it in the sequence of 1 to 10
* So if I wanted to save her as number 2 on the sequence, I would press button 2
* then you enter the telephone number
* then you replace the handset

Then to call my wife, what they need to do is lift the handset, press the MR button, and then press the number two.

It’s so complex people ended up writing down the names of the numbers they had saved and in what sequence. This defeats the whole purpose of the feature.

And there are two really important points here.

**First, features involve a trade-off. Each new feature crowd out the existing features. Each additional feature adds complexity, makes the interface more cluttered, and places an additional mental burden on your users, however slight.**

The second point is that just because you can add a feature, it doesn't mean that you should. Did this feature really add value to users of the telephone? Did anybody ever use it? Or did it just add clutter to the interface, and sow little seeds of doubt in people’s minds every time they use the phone.

If we fast forward to the present day, we see phones like this. A Nokia feature phone. The amount of technology and features in this phone is on a completely different scale compared to the previous phone. Let’s just list out some of the things this phone can do:

* it can make phone calls
* it does voicemail
* it does SMS, email, and you can get on the Internet
* it’s got a camera, calculator, radio, MP3 player, and Calendar
* it’s got a clock, alarm clock, and a stopwatch
* it’s got apps?? like YouTube and Facebook
* and you can configure it and customise it to your own preferences, such as adding wallpaper and setting favourite contacts.

On one hand, you have a remarkably complex device from a technology perspective. Yet, it’s not complex from a users perspective. This is actually the phone my parents use, because they think it’s actually easier to use than a smartphone.

**So how come the phone with only a small number of features was so complicated? And this phone with a huge number of features is relatively easy to use? Well, that difference is called “design”.**

The Nokia is really well designed. And even though Nokia missed the smartphone bus, they have a great track record at designing easy-to-use feature phones.

And this brings us to an additional point about features. They must be designed. You can’t just throw in a feature to your product and hope for the best. That’s what they did with the MR button and it didn’t work out so well.

**Let’s look at the consequences of ignoring all three of these points, using remote controls as an example.**

This is the very first wireless TV remote control designed by a company called Zenith in the 1950s. The remote had a limited number of features, such as channel up channel down, volume up volume down – and only had four controls – four buttons – to activate these features.

Fast forward to the present day, and this is a remote control for a DVD player. It has 43 buttons which are used to activate and control the many features of the DVD.

The cost of including these manifold features is that the really important features get crowded out. Like the play, rewind and forward buttons.

It’s a classic example of a product succumbing to the “feature arms race”. Let’s try and beat the competition by adding more and more features, regardless of whether anybody needs or

uses them.

Some of the features on this device include depth enhancer, multi-remaster. And advanced disc review. And if you’ve never heard of them, you’re in good company. Neither have we and neither have the hundreds of people attending our training courses each year.

**This arms race breeds a form of insanity. Adding features that nobody needs, nobody uses or even understands.**

Finally, the interface is not designed. At least not intentionally. The buttons are just thrown together. With no prioritisation, organisation or obvious logic.

**It’s like getting all the ingredients for a cake and pouring them out on your kitchen table, and claiming “this is a cake”. Of course, it’s not a cake, it’s just a mess. And so is the interface for this remote control.**

There’s a lot to learn from this example. So let’s quickly recap the key points:

First, features add complexity. This is a stone cold fact. The more features you add, the more controls you need to activate them. More controls make interfaces more complex, this is unavoidable. You need to decide whether your new feature is worth the complexity it is bringin to your product.

Second, features must be designed, which adds time and expense to your product development efforts. You also need to decide whether your feature is worth the time and money designing, building testing, supporting and maintaining it.

Third, just because you can add a feature, doesn’t mean that you should. Features must serve a purpose. They must solve a genuine problem for end users. Otherwise it is a lose-lose-lose situation: more complexity, more cost, no added-value.

Finally, here’s a handy checklist of questions to ask before deciding to add a feature to a product.

First of all, does anybody actually need it? And by anybody I mean an actual user, not a stakeholder in the business. What is the problem its solving? And Who is it solving the problem for? Can you prove that people need it? it shouldn’t just be random guesswork.

Second, what is the trade-off? What additional complexity are we adding to the product? What impact will it have on the core features and user experience?

Finally, what is the cost - in time and resources - of adding this feature? And is it worth it?

Answer these questions before adding features, and you’ll be making better more judicuous decisions.

# 

## 1.11 Engineering culture

So what is it that makes otherwise logical organisations add so many features to a product that nobody needs to use, wants to use, or in many cases never actually uses.

**What’s the logic? Well there isn’t necessarily any logic to it, but it does come down to culture. Specifically, engineering culture.**

The most famous consumer electronics and computer companies were all founded by engineers. Many consumer electronics companies were founded in the interwar years, and computer firms in the 60s and 70s.

Back then the wonder of these machines must have they worked beautifully but that they worked at all. People didn’t care how cumbersome the washing machine was to use, because it saved them so much time and backache compared to washing clothes by hand.

And people didn’t care about the leader transistor radios were, the fact that the radio didn’t take up half their living room more than compensated for any usability issues.

But as technology advanced, products were able to do more and more and this “it doesn’t matter how it works” philosophy began to cause problems.

Devices such as video recorders, became bywords for poor usability. They’d become so complex, people just didn’t know how to use them. There were so many features included that the really important features such as recording TV programmes were swamped with too many controls and unwieldy workflows.

This was largely driven by the engineer’s mindset, the more features you can pack into a product, the better that product is. Our CD player has 35 features, and the competition only has 20 features. Therefore, we’re killing them.

There is also a perverse logic from the sales and marketing perspective. This is a full featured product costing this much, a 20 feature product only costs this much and here’s a bargain basement 10 feature product costing only this much money.

And this set the tone, and established the culture whereby usability was never serously considered during product design. It was all about features and packing in as many as possible, regardless of how difficult this meant the product was to use, regardless of whether anybody actually used them.

**There are some depressing statistics about features, one of the most well known is that 80% of your users will only use 20% of your features. Another is that up to 9% of consumer electronics features are never used by anybody. Which makes you wonder about all the time, energy, and expense put into designing, building, testing, releasing and selling them.**

When the first personal computers came on the market in the 70s, a lot of them had similar issues. You needed to be an expert to use them. And around that time we saw the emergence of a new type of product - software that can run on the personal computers. And the companies that made software were also founded by engineers.

And the same habits emerged. Success was not defined by the quality of the experience, but the number of features included in the product.

**So this “feature first, usability last” approach is deeply ingrained the cultures of both the software and hardware industries.**

Steve jobs was not the first user interface designer, but he was definitely the person that brought design awareness into the mainstream.

After his second coming at Apple, the success of his products beginning with the iMac and the iPod showed everybody that design could be a competitive advantage. And this was the catalyst for the huge surge in demand for design skills all across the technology sector.

But most of us don’t work for an organisation with a Steve Jobs in charge. Many of us are working in culture’s where the engineering mindset dominates.

Before we move on to the next lesson, let’s revisit the remote control example and shine some light on the decision making process an engineering culture fosters. The more you understand about the decision-making, you are in a better position to prevent them happening again.

Why does the remote control have so many arcane features that nobody understands or uses?

First, because the team were designing the product for themselves not the wider audience. The made the common mistake of assuming their target audience was just like they were. We all understand what depth enhancing means and think it would be a great feature. Therefore everybody else in the world does too.. But the team are audio/video experts and don’t represent the vast majority of DVD users. Don’t fall into this trap. See the lesson called “I am not the target audience” later in the course to understand more about this concept.

Second, the didn’t do any customer research. Here are some drawings that our classroom students do when we ask them to design a remote control for a DVD. From thousands of drawings we always see the same patterns. A huge reduction on features. No multi remaster or depth enhancing. If the team in Panasonic did a similar exercise, they would see the opportunity for a more streamlined, more cost effective, more useful product.

Third, they are technology led, not customer led. The engineers discovered a way to add a new technology into the product. And before considering whether the technology was solving a problem or not, it was added. Smart companies start with customers not technology. They know that people don’t buy technology. People buy solutions to problems. Learn what the problems are first, and then find the technology to solve them.

# 

## 1.12 Features v goals

This engineering culture has allowed many bad habits to creep into the standard software development process. In the next four videos we’ll take a look at some of these habits.

The first returns to the topic of features, and how typically on software projects we tend to focus on developing features rather than helping users achieve their goals.

This is a brilliant quote because it sums up perfectly how people approach your product. They’re not using your product just for the sake of using it, they’re using your product to help accomplish a goal or a task.

I would go further and say they are not buying the hole in the wall, what they are really buying is a new picture hanging on the wall. Or a new mirror hanging in the bathroom.

And your product is just a mere tool to help them do that. And this is something that’s sometimes hard for us to admit.

When you need to wash your clothes, you don’t think, oh great I’ll go downstairs to my lovely Zanussi 8kg ecoSafe Washing Machine and try out some of the features. Of course not. You probably don’t even know the brand. You just think, I need to wash my clothes. And the machine is just a tool to help you do that.

And when you’re preparing dinner, you don’t think: “Great, I’ll get out my John Lewis Classic Cooks Knife, and see what it can do”. No, you just think I need to chop the onions - and the knife is just a mere tool to help you do it.

And it’s the same with software. When people are using Microsoft Word, they’re not thinking about kerning or macros or autosummarize or all the other features packed into the interface. What they are really thinking about is writing the report, completing the memo, sending that quote to the client.

**Microsoft, along with many other software companies, has lost sight of this, and has packed Word with so many features that it sometimes becomes difficult to accomplish the more basic everyday task.**

Google is eating Microsoft’s lunch in the word processing and spreadsheet market. No doubt using lots and lots of data, it has figured out 20% of features that 80% of people use. and it has created a word processing tool that is streamlined and elegant and just focused on the most common, most critical tasks.

Just to emphasise how misleading feature focus can be, let’s conduct a little thought experiment from Alan Cooper’s book. If you have a pen and paper to hand listen to me call out the features of this product and write down the name of the product as soon as you think you know it.

* This product has four wheels with rubber tyres.
* It has an internal combustion engine
* A suspension system
* A transmission system connecting the engine to the drive wheels
* The transmission and the engine are mounted on a metal chassis
* It has a brake mechanism
* Steering wheel
* And t has a seat in line with the steering wheel

I’m sure you’ve made a pretty good guess by now as to what this product is. But lets look at some of the goals of the product.

* It’s comfortable to sit on for long periods
* And it cuts grass quickly, easily and accurately

In our classes most people assume its a car or a van or a tractor. The point here is that by focusing on features everybody could have a different idea in their head as to what it is we’re building. It’s only when we start talking about user goals that it becomes crystal clear that we’re building a lawnmower.

**Like I mentioned before, this clarity is even more important when it comes to building software, because software is invisible until it’s finished. Leaving even more room for different interpretations.**

Another way of looking at this is to consider the most common reasons why startups fail. Among the many different reasons, the most common is that there was no market need for the product being developed.

In other words, these startups had created products that didn't help people achieve any particular goal. They hadn’t identified any problem they product was going to solve.

**So while these startup products were no doubt beautifully designed, and had lots of elegant features, they had missed the key point. Products are not about features. Products are about helping people achieve something: either solving a problem or accomplishing a goal - which are just two ways of saying the same thing.**

In contrast, some of the most well-known and successful start-ups focus from the very beginning at solving genuine customer problems. Not just customer problems, but business problems too. Because remember the product has got to work from both a business and a customer perspective.

Back in 1999 the global recording industry had revenue of $27 billion. In 2008 nine years later, that revenue was down to 14m. Almost half, in less than 10 years.

The business problems here are quite obvious. The decline was largely due to music piracy. And because of the prevalence of piracy, music was no longer seen as a legitimate product - it became something free in the minds of customers, something they didn’t need to pay for. All of which contributed to the steady decline in revenue.

**To try and solve this business problem, the four major record companies EMI, Universal, Sony, Warner, along with an organisation called Merlin - a group that represents independent labels gathered together in Sweden in 2008.**

**And they chose Sweden because it was the home of Pirate Bay, a notorious website where people could, amongst other things, download music illegally.**

What this consortium did was to give over their entire combined catalogue of music to a Swedish start-up called Spotify. Essentially they were conducting a grand experiment, to see if there was any possibility of generating revenue from streaming music.

But they weren’t just solving a business problem. There was also a genuine user problem that needed to be solved as well. This was the anarchy of managing music in the digital era.

If you wanted to rip your CD collection to your PC, it could take you several months, if not years, depending on the size of your collection. Then depending on what format you used when ripping, the music might not play in every computer in your household. If it was ripped on a Mac, it might not play on a PC for example.

And if you purchased music using iTunes the issue was sharing that music, even to your own devices, and even with members of your own family. You only have 5 registered devices, and sometimes it’s hard to keep track of them all.

And with the advent of smartphones getting your music from your PC to your smartphone became another elaborate task.

**So, at the dawn of the digital music era people were spending more time managing their music than actually listening to it. Your music library had become a source of pain.**

Faced with these two clear problems the team behind Spotify formed a hypothesis. They said that we can generate meaningful revenue from streaming music. If we make it incredibly easy to access all the music in the world. On all devices. All the time. Because it would be a better experience than iTunes or piracy.

That was the critical point. To really solve the problem it had to be easier than the two existing solutions - iTunes or piracy. And the experiment, so far, has proven to be successful.

In 2018 Spotify have 157 million users, 71 million of them paid subscribers. The annual revenue is close to $3 billion. Still a long way to go, but definitely a strong proof of concept - streaming music can be a source of revenue for the music business.

Another example of a problem-focused product is Dropbox. At its inception, Dropbox faced a number of business problems. They were one player in a crowded market of other startups trying to solve the same file sharing and file storage problem.

The problem itself was complicated, and was a tough technological nut to crack. Even worse, all the solutions that were on the market were unreliable, giving the whole space a bad reputation, and making it difficult to raise funds.

From the user’s perspective they were looking for an easy way to share large files with friends and colleagues and clients. And a reliable way to backup their computers, without using cumbersome external hard drives, which themselves could be damaged or lost.

Again Dropbox formed a hypothesis based around solving the business and end-user problems:

* We can build a successful business
* If we make it incredibly easy to share, store and sync files. With 100% reliability.
* Because the market is huge, and every other service sucks.

And this laser-like focus on problem solving, not features, has given Dropbox great success. It has over 300 million users, about $300 million in annual revenue, and a market valuation of several billion dollars.

Both Spotify and Dropbox used their clarity on problem-solving as a tool for innovating and disrupting their markets.

In contrast, let’s look at a product that wasn’t designed with clear user goals in mind.

The final example here is Google Plus, conceived to give Google more relevance in the social media space. And also to compete head-on with Facebook, considered to be an arch rival.

But from the user’s perspective what were the problems being solved? Unfortunately, there was none. Facebook was working fine and nobody really needed or wanted another one.

**And despite its beautiful design, and incredible user interface features, Google Plus is a ghost town. It wasn’t and isn’t solving a problem for users. And no amount of gorgeous design and fantastic features can make up for that fact.**

## 1.13 Taking shortcuts

Another reason technology is more complicated than it should be, is that often in software projects the first two phases of the design process are skipped. And if we look at the more detailed view of the process, this means that the research, analysis, design, prototyping and validation are all avoided.

**Anybody who’s been there knows the pain involved. And in my personal experience this is the number one reason why software projects fail. Either fail completely - as in don’t get finished - or fail in one of the other factors - quality, speed or cost.**

And this is such a common practice - starting a software project in the build phase - that many of the early UX agencies started growing their business by being asked to rescue projects that had gone astray because of this failure to follow the process.

But don’t just take my word for it. Consider this book, The Definitive Guide to Project Management. It’s published under the Financial Times imprint, by Prentice Hall, a very reputable educational publisher owned by Pearson, which itself is the largest educational company and the largest publisher in the world. So you would imagine that these guys must know what they’re talking about.

Yet, on page 157 of this definitive guide they make this statement: “on software projects for example, the design phase and build phases are synonymous”

Now, if you were to substitute the word “construction” for software you see how ridiculous that statement would sound. “On construction projects for example the design phase and the build phase are synonymous”

**Imagine building the Golden Gate Bridge or the Sears Tower and designing it at the same time. There is no way the writers would make such a statement about construction projects. Yet such is the prevailing culture in software, it seems perfectly acceptable to say it about digital projects.**

But just because the Financial Times is saying it doesn’t make it any less wrong. If you do a search for “software fail” on Google you find a small cottage industry devoted to cataloguing high-profile, hugely expensive software projects that were either never completed, or if they were, produced disastrous results.

A great example of this is P-PARS, a hugely controversial software failure that occurred in Ireland during the last decade.

P-PARS stands for personnel, payroll and related systems and the background to this project is the National Health Service in Ireland.

Up to the late 1990s Ireland had around nine different regional health services, such as the Southern health board, Eastern health board, the Western health board, the South Western health board, the South Eastern health board and so on.

In the late 90s it was decided to merge these into one national health service, to make the system more efficient. As part of this, it was necessary to merge all the different administrative systems that each regional health board had.

Things like personal systems, payroll systems, accounting systems, employee review systems and so on.

And that’s what the P-PARS project was about. The unglamorous job of blending together these disparate systems, to make efficiency gains, to cut costs, and reduce unnecessary duplication of effort.

In 1997 when the project started, the estimated budget was 9 million Euro. A lot of money. But by 2008 as the project was a lingering its death throes, the actual budget was €180 million. 20 times higher than originally costed.

But despite spending 20 times more money, feedback from users wasn’t great. This is some feedback from the Chief Executive of St James Hospital, one of the biggest hospitals in Dublin, which was trialling the software.

The CEO said that the software had so many flaws that it threatened the hospital’s basic functioning. And that should not be extended any further within St James’s Hospital or anywhere else in the health service. So not alone was it not solving the original problem, it was creating so many more problems that it was threatening the viability of the hospital.

As word of mouth goes, this feedback couldn’t get much worse.

When news broke of the huge overrun in costs, it caused a political scandal. It was around the time that the global financial crisis erupted and Ireland was being bailed out by the IMF. It became a symbol of political waste and incompetence.

So a report was commissioned by the controller and auditor general to find out what went wrong. And this is the executive summary of that report.

Overall the report said that there were seven high-level reasons for failure, three of them highlighted here were to do with research and design or rather the lack of either of them.

The first highlighted line refers to a failure to develop a clear vision of what strategic Human Resource management actually meant. Unlike the team behind the school we looked at earlier, this project didn’t have a vision, there was no clear definition of what it was supposed to achieve.

The next point raised the fact that substantial variations in pay and conditions, organisation structures, cultures and processes existed between the various regional health boards. This shouldn’t have been a surprise, because this was the whole point of the project - merging all these systems into one.

However the full extent of these variations was not known before commencement of the project. Essentially they hadn’t done even the most basic research to understand “what are all these variations, what is the nature of the problem we’re trying to solve” before they started coding. And by “commencement” they mean actual software coding. These guys went straight into full build mode.

Finally the report states that there was an inability to definitively freeze the business blueprint or business requirements. So the design, such as was, was never finalised, never nailed down. It was a moving target, and anybody who’s worked on a project with a moving target rarely has good things to say about it.

Think about the expense and frustration of building that school if the design was changed regularly after the builders started work. It will give you some idea of the pain and madness involved.

**So while none of us might ever work on projects with such a high profile, and such a high cost overrun, and with such disastrous consequences, no matter what size project we work on, if we try to cut corners and try to short-circuit the first two phases of the design process, we’ll end up in a mini version of P-PARS project. Spending more money, taking more time, and producing low quality results.**

I think there are two reasons that these shortcuts are taken so frequently.

One - people skip research and design because they don’t understand them, and they rush straight into build because it gives them a sense of progress, they're getting their hands dirty, and something is being built. But it's a false economy, a delusion.

Another reason is just the terminology itself. People balk at the term research, because it conjures up images of long-term studies, huge costs, and reports that run to hundreds and thousands of pages. But as we’ll see, that’s not what research is necessarily about.

Also the term “design” sounds shallow and insubstantial to some ears. It conjures up images of fashion and they conflate design with styling, which is something else entirely. And they feel they don’t need it. Or it can be added later.

But really what these terms mean is that during research we’re defining the problem. The software is being built to solve a problem and we’re just taking time to understand exactly what that problem is.

And design is just a word for solving the problem or defining the solution. Which is a logical thing to do before you build the solution.

So whenever I’m approaching clients and discussing what we will do during the project, I try to avoid terms like research and design. And you should too. Istead, I would strongly recommend using the following vocabulary:

First, tell your stakeholders that you think it would be prudent to spend time to understand the nature of the problem we are solving before we build the solution. It will reduce the risk that we end up building the wrong thing. And that’s expensive.

Second, tell them you think it would make even more sense to spend some time *designing* the solution before building it. Tell them it’s like designing a house before building it. It also reduces a lot of risk.

This language sounds more logical and less expensive to uninitiated ears. And it also has huge doses of pragmatism and common sense built into it, which makes it more difficult to argue against.

# 

## 1.14 Low fidelity design

**Many organisations do actually include a design phase in their software projects. But while they may think they are doing design, and they might be paying lip service to design, they are not actually designing.**

The fidelity of your designs is critical. And by fidelity we mean how close does the design replicate the end-state of the product. The higher the fidelity, the closer the designs resembles the finished product.

To explain this is first take a look at how they do design in the auto industry. Take a look at the next video in this module and then we’ll resume our discussion in the video following that.

*[BMW clay modelling video]*

## 

## 1.15 Low fidelity design (continued)

That was BMW, but it’s not just BMW that creates clay models for each car. All car manufacturers such as Mercedes, Volkswagen, Toyota, and Ford will spend a couple of months, and up to a quarter of $1 million, creating clay model of the car before production begins. Only after a clay model is built will projects get the green light or not.

**The reason they do this, is because automakers know that cars our highly interactive, highly visual products. And they want to make sure about the quality of the aesthetics and quality of the interactions, before they commit to mass producing them.**

So car manufacturers want to see, touch, and interact with the product before they decide to proceed with the expensive business of making thousands of cars. What is it like to sit in the car, play with the dashboard; How does the car interact with the wind, the sun, the rain and so on. That is high-fidelity design - the design looks, feels, and interacts like the finished product.

Essentially, they are making a prototype - an essential component of the design process. Remember this diagram? The car manufacturers are here. They’ve designed a solution and are now prototyping it and validating it before they go the the very expensive and irreversible business of mass producing the car

And it’s not just car manufacturers that go through this process. In the construction industry buildings are also highly visual and highly interactive.

It’s why architects produce high fidelity blueprints and 3D CAD drawings. And cardboard models. These are prototypes that help the team and the customer understand what the building will look like, and how people will flow through it, before they commit to building it.

Clothes are also highly visual and interactive, which is why the fashion industry creates high fidelity designs and prototypes before committing to mass production.

Even in cookery we see what the meal is going to look like before we commit to cooking it.

But in software, which also produces highly visual, highly interactive products if we do design, oftentimes it looks like this. Spreadsheets or Word documents that list line after line of requirements or specifications.

**There’s nothing wrong with these documents. In fact they are very important. But this is not design. It’s just not possible to express aesthetics, or communicate complex flows and interactions on a spreadsheet.**

As Frank Zappa once said “talking about music is like dancing about architecture”.

And trying to communicate visual design and interaction design through spreadsheets is something similar. A project team could read these documents and each person could have a wildly different vision in their head of what it is they’re building.

What’s a prototype look like? Here are some examples of our team testing prototypes of an iPhone app with users of a property portal a few years ago. We wanted to test a new mapping feature before we built it and rolled it out to the app store. It took several iterations of the prototype before we were happy it worked. A lot of effort, but it avoided the pain of shipping a feature that didn’t work to 500,000 active monthly users .

**Prototypes are an essential part of the design process. Prototypes are just high fidelity designs. Later in this course we’ll do a deep dive into prototyping and show you that unlike the auto business, development software proptypes can be fast and cost effective.**

**For now, we’ll recap why they are so important:**

First, they reduce ambiguity and make it clear to everybody - this is what we’re building, and this is what it will look like and behaves like when we’re done.

Second, prototyping gives you the time and space to validate before you build. Don’t think about it as extra time or cost. Think about it as reducing the time and cost of fixing something that doesn’t work

Third it reduces risk. The risk of spending extra time and money. The risk of shipping low quality solutions. The risk of damaging your brand and reducing customer trust and loyalty.

## 1.16 Failure to prioritise

**The fourth reason that technology is often more complicated than it should be, has to do with prioritisation. Or, more specifically a failure to prioritise.**

**To explain this we first need to discuss the concept of use cases.**

A use case is a way of describing what we want our software to be able to do. Or a scenario we want our software to facilitate. Or a task we want our users to be able to complete.

And let’s imagine that each of the lines in this diagram represent a use case for an email application we were designing .

* One use case could be that we want users to be able to open the email application and and create and send a new email message
* Another is that we want users to be able to open their email, look at their unread messages, and send a response to one of those messages
* And a third would be the ability for users to look for an old email by running a search

In fact there are endless different ways somebody could use email software. But trying to design software for all the different use cases can be challenging.

**But here’s a really important point. Not all use cases are created equal. Some use cases happen far more frequently than others. Think of the 80-20 rule.**

So 80% of the usage of your email software might come from only a small number of use cases. Such as the three I just mentioned.

And some use cases might rarely if ever happen. These are known as edge cases. So for example, an edge case might be somebody who wants their personal email forwarded to their work email, but only at certain times of the day and only if the email title contains a certain word or words.

It might happen, but it’s rare. It’s not the common use case.

**And the trick for software designers is to prioritise designing for the most common use cases. And to de-prioritise the edge cases. Because if you try to design an interface to accommodate all possible scenarios, you risk tying yourself up in knots, and your software, and your users along with it.**

Think of Gmail. It prioritises the three most common use cases I just mentioned; create a new email, read new messages and search old messages. These three features are front and center on the interface and easy to access. And if you want to forward certain emails and filter while doing so, well… you have to dig around and work a little harder. And that’s fine because that means the minority have to work a little harder while the vast majority have a nice clean interface.

Here’s a great example of how not to do it from the Irish Rail website, a website I use frequently when travelling from Dublin to other cities in Ireland. This screenshot was taken during the purchase process after I had selected where I was travelling to and from.

As you can see here I’m travelling from Heuston station in Dublin to Cork.

And this section here is asking me about the preferred delivery method for my ticket. In fact they don't deliver tickets any more, and what they are really asking me here is what station would I like to collect my tickets from.

The obvious answer here is Dublin, I mean where else would I want to collect the ticket? Or to put it another way what is the one station in Ireland that I am definitely going to be in at the start of this journey? Of course it is Dublin Heuston.

But Irish Rail, in their wisdom, have given me the option to choose any station in Ireland to collect my ticket. Now there are 132 stations in Ireland and Dublin Heuston is about 40th on the list. So every time I book the train with Irish Rail I have to tap 40 times through this list to choose my departing station.

Why did they do this? I’m guessing that during the development process the team came up with this scenario: what about people who are passing another station on their way to the departure station, and it’s more convenient for them to pick up their ticket there, instead of queueing at the departure station for the ticket when they might be under a time pressure.

Now this is a possible scenario – a possible use case – but a very unlikely one. A classic “edge case”. And anybody can think of edge cases if they have enough imagination to do so. And far too often in software development, edge cases like this are given just as much priority as more common use cases, resulting in efficient interfaces like the one on Irish Rail.

**And this is important. Because what we want people to do is to flow through our software as smoothly as possible. And If we design our software to accommodate all sorts of unlikely edge cases, we’ll interrupt that flow just like we saw in the example there. And if we include too many of these interruptions, we’ll end up creating seriously annoying software.**

Let’s look at another example, from Windows 7. This is the screen that allows you to search for files on your PC. And some of the different options are as follows:

* You can search the entire computer
* You can search the current folder,
* Or all subfolders,
* You can search by date modified,
* The kind of document,
* And the size of the document.
* You can look back at your recent searches,
* You can look at advanced search options – (because this isn’t the advanced page these are just the regular options)
* You can save your search,
* And then there appears to be, what looks like to me, two different input fields for entering you search term.

Again based on anecdotal research conducted during our public courses, people never search for a file on their computer just by date modified. Not that they sorted search results by date modified, but that they actually searched by date modified. Nobody in our class has ever raised their hand for that.

A tiny number of students said they searched by size of the document, but nobody has ever said they have saved a search on their PC.

Which begs the question why is Microsoft cluttering the search interface with all these features that nobody or only a tiny minority of people, will ever use?

Apple, on the other hand, treats search like this. One input field only. They have designed their search feature based on what they think is the most common scenario, that people will search for a file using the name of the file. This is what they think most people will do. And this is how the feature works.

You enter in your search term, you see the results, you have an option to view the results in the Finder, which is equivalent to Windows Explorer. When the Finder window opens, at this point, you can then modify the search results and sort them alphabetically by size of document or kind of document or date modified.

But Apple only gives you these features when it believes you need them, which is after you've made your initial search. It’s a technique known as “progressive disclosure”. Just give people the information and the features they need when they need them. Don’t overload people by giving them everything upfront.

And of course if we look at Hailo, the taxi app, that we discussed in module 1, we’ll see that it was designed for one single use case. I want a single taxi, right here, right now.

If they had designed it to accommodate other use cases, such as users who want to specify the time, or order more than one taxi and so on, it would have had a much different - more clunky - interface. But because the Hailo team were good at prioritising, they created a knock-out app, that doesn’t work for all people all the time, but works for many people a lot of the time.

There is a writer called Scott Berkun who writes about software and design amongst many other things. He wrote a great blog post in which he defined his rules for prioritising when designing software.

**His key point was that if you’re not prioritising, you’re not designing. Design involves making these hard decisions all the time.**

And he says as a rule of thumb this is how you should prioritise features:

* The things that most people do, most often – these are your top priorities
* The things that some people do, somewhat often – these are secondary, still on the interface perhaps but not as prominent
* And the things that few people do, infrequently – these are your edge cases, you can still accommodate them, but people have to work a little bit harder to find them

We’ll show a more sophisticated prioritisation tool in our analysis module, but this is a great place to start. Don’t clutter your software with a profusion of little used features designed for a minority of users.

## 1.17 Finding a design target

**Research sounds intimidating. It conjures up images of specialist technicians who have PHDs, wear white lab coats and carry clipboards. But really, it’s just meeting and talking to people.**

**So, if you were building a new house and hired an architect - it’s likely that before sketching some designs, he or she would come to meet you to learn about your likes and dislikes, and understand what sort of house you’re looking to build. That’s research.**

**And in this lesson we’ll look at what it is we want to learn about people when we conduct research. There’s a lot of information we could find out, but only a small amount of it is actually useful when it comes to designing products.**

Research allows us to avoid the common pitfalls we saw in the previous lessons. The more we learn about our users, the more we understand what they are trying to do, the better placed we are to make decisions about what we should or shouldn’t include in our software.

What we are trying to avoid is building products that try to be all things to all people, software designed for all possible scenarios. What we really want to do is to create products that only include the features that will help our users achieve their goals, and doesn't include the features that won't help them achieve their goals.

But a lot of people push back on this and say “Colman, that’s easy for you to say but I work for a bank, for example, and our potential audience is everybody in the country over the age of 16. Our online banking software has to be available and useful to the entire population. So, in spite of what you say, I need to design software that has to be all things to all people.”

**That’s a very difficult thing to do and it’s almost impossible to do well.**

Because the reality is, even if you are designing banking software, and your potential audience *is* the entire population of Ireland - that audience will only be using your software with specific goals in mind, in a specific context, and using a specific set of behaviours.

And these three things - goals behaviours and context - become, what we call, our design target. And here it’s critical to understand the difference between a target audience and a design target.

The target audience could be large - 4.5m inhabitants of Ireland. The design target is more specific: their goals behaviours and context when they conduct banking transactions online.

These are the three most important things we need to learn about our customers. And we’ll discuss these important concepts in the next video.

## 1.18 Goals, behaviours and context

**When researching our users, there are many things we could learn about them. Their age, their gender, their income, their political persuasion, where they live, how many kids they have and so on. Most of this information, however, is not useful when it comes to designing interactive software products.**

**Instead, the three things *we have* to learn about our users are their goals, their behaviours, and their context.**

And we’re going to explain these somewhat fuzzy concepts by conducting a thought experiment with these two chairs.

The chair on the left is called an Aeron chair, designed and manufactured by a company called Herman Miller in California. These were first manufactured in 1992, they retail for about $1000, and since 1992 Herman Miller has sold over 7 million of these chairs. If you do the calculations you realise that this is a very successful chair.

The chair on the right is called a La-Z-Boy. Designed and manufactured by the La-Z-Boy Corporation since 1927. Some of these chairs can retail for over $1000, and they sell so many of these chairs that every year the *annual revenue* of the La-Z-Boy Corporation is in the billions. Another very successful chair.

So what we have here are two chairs, both of them top of the market in their respective categories of the chair market.

Now let’s look at the goals, behaviour and context of the users of each chair.

First the Aeron chair. And let’s look at the context first, and by context I mean the social and physical environment where its used. Now it is obvious by looking at this chair that it’s an office chair and the context is the office professional environment or work environment.

During our classes when we talk about goals, a lot of people say the goal of the person using this chair is to be alert, or ergonomic, or stop getting back pain.

**But really when you come into the work in the morning, you’re not thinking I want to be ergonomic today, or I really want to avoid getting back pain today. Instead what you’re really thinking is I really need to send out that email, or I need to get that report finished.**

So your primary goal when sitting in this chair is really to do your work. And your secondary goals are to be alert, to be comfortable, to be safe and injury free. And what this chair does it helps you achieve your primary goal by making sure your secondary goals are taken care of – you do your work better if you’re comfortable, alert, injury free and so on.

Behaviours are the things that people will actually be doing when they're sitting in this chair. So that the sitting at their desk, typing on their PC or laptop, reading books or papers, swiveling around to talk to somebody behind them, or moving the chair around the office to and from a breakout area. Just some of the things, some of the behaviours taking place while people are sitting in the chair.

Before talking about the Lazy boy, it’s worth looking at the history of the Aeron chair Before the design the chair they conducted ethnographic research with their target audience - office workers in big tech firms in Silicon Valley, the likes of Dell, HP, Microsoft and so on.

By ethnographic research, I mean they went into offices and watched people working at their desks - observed them in their natural habitat, so to speak.

And they observed many of the behaviours we just mentioned. Swivelling, taking to the break out area, typing and so on. But most office chairs at the time, looked like this - standard chair with four legs. You weren’t able to swivel and would have to carry your chair - not wheel it - to somebody’s desk or a breakout.

They also observed that people had different styles of sitting. Some people sat ramrod straight. So slouched forward or backwards, some people liked to sit with their keyboard in their lap.

And unless you had very correct posture - sat bolt straight - you would have a difficult time being ergonomic in one of these old chairs.

They combined all these learnings into their chair design. This is why it has wheels and a swivel mechanism. And why it has levers that allow you to adjust the resting position forward,backwards, and so on. The chair is flexible enough to work for everybody.

And now most office chair is most offices most everywhere in the world, more or less look like this and have these features. Herman Miller set the standard for office chairs everywhere because they did their research before they designed the chair.

Looking at the La-Z-Boy, we see that the goals, behaviours and context are completely different. The context is home or in a hotel lounge definitely not a work or a professional environment.

The goals of the person sitting in the chair could be manyfold: to watch TV, read a book, pet the dog, with the ultimate goal being relaxation.

And the behaviours could be watching the TV, petting the dog, falling asleep, drinking a glass of wine, curling up or kicking back.

**Now the thought experiment is this: let’s assume I arrived at your office and your boss gathered around all your colleagues, introduced me and said this is Colman and he’s going to show us a new chair that’s going to revolutionise how we do our work in this office.**

And… I wheeled out this chair.

I’m pretty sure most people would start laughing, they’d think it’s ridiculous. It wouldn’t be impossible for us to get work done sitting in a chair like this. But the question I have for you is – this is a really well designed chair, top of the range, earning billions in annual revenue for the La-Z-Boy corporation. If you define success as revenue generated, this is possibly the best designed chair in the history of chairs.

What is it specifically about this chair that wouldn’t make it work in your office? Because it’s hard to find fault with a product like this from a design and revenue perspective

**And the answer is that even though it’s possibly the best design chair ever, it’s not optimised for office users**

Despite being the best design chair in the world, it’s not optimised for the office context.

Despite being the best design chair in the world, it’s not optimised to help people achieve my goals when I’m working at my desk. In fact, it would make it more difficult for me to achieve them.

And even though it’s a really well designed chair, it’s not optimised for the common behaviours that I conduct when I’m at my desk. It would be very difficult if not impossible to conduct these behaviours in the La-Z-Boy.

**What’s the lesson? If you’re designing interactive products, you must have a clear understanding of your users goals behaviours and context. Without this knowledge, you’re just guessing and you run the risk of creating a product that is as wildly unfit for purpose as the LAZYBOY chair is in an office environment.**

**And if you already have a successful product, and want to make it better and more competitive, to get those difficult marginal gains of 5-10%, you have to know the goals behaviours and context. Because guesswork is not efficient.**

**And the more specific you get about these three things, the narrower your focus, the better your product is going to be. And products that are really good and doing a small number of things, tend to be more popular than products that are mediocre and doing multiple things. . It’s a concept called the paradox of specificity, and we’re going to discuss it in the next video.**

# 