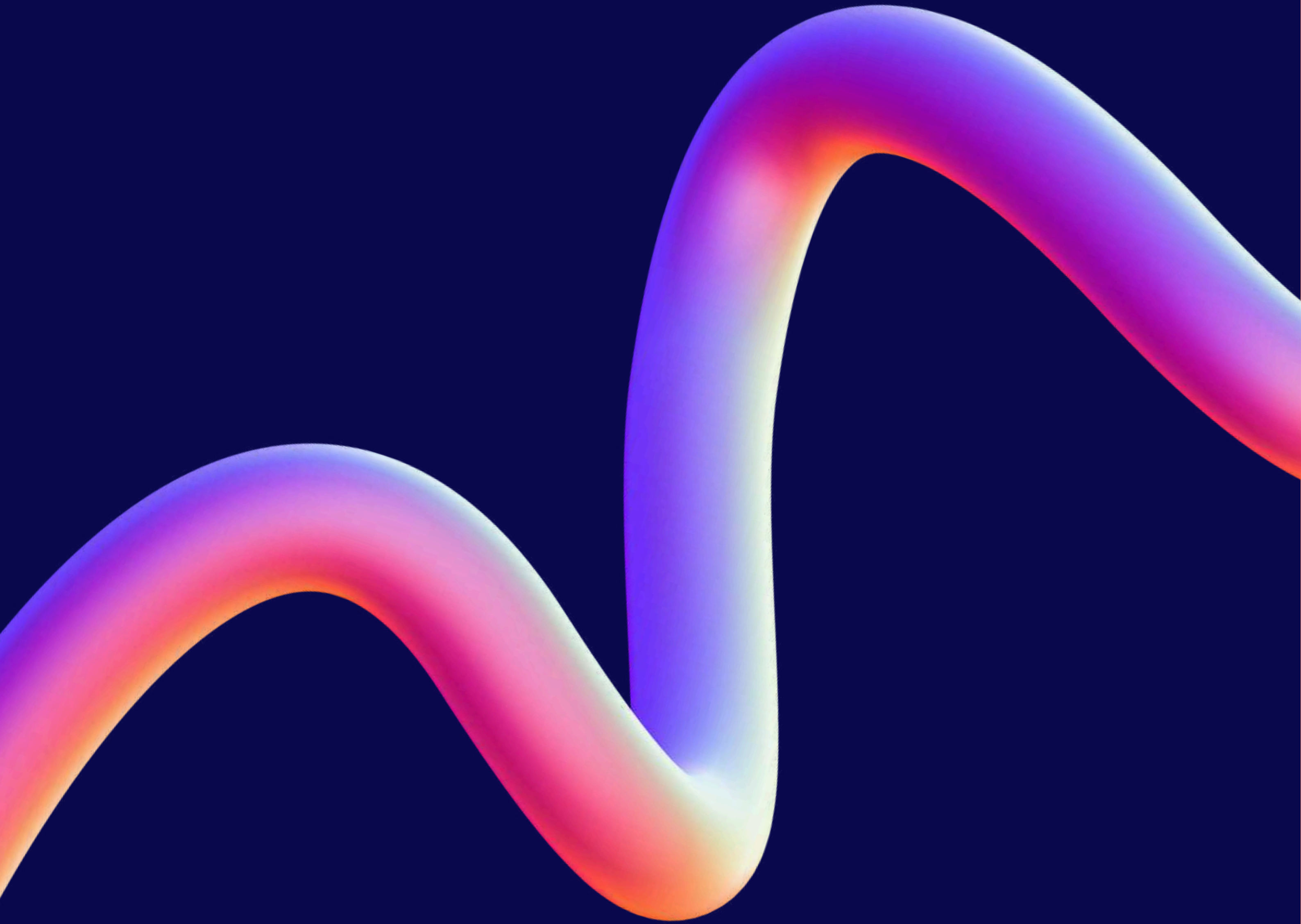
You don’t know what you don’t know.

The Tell-all Guide for Software Product Inventors



putti

Written by John Halvorsen-Jones

Introduction

Software can make you, but software can also break you.

Whilst much comedy has been made of former US Secretary of Defense Donald Rumsfeld's 2002 response to a question on the Iraq war, his words are nonetheless poignant to software development.

"There are known knowns. These are things we know that we know. There are known unknowns. That is to say, there are things that we know we don't know. But there are also unknown unknowns. There are things we don't know we don't know."
- Donald Rumsfeld

Having spent over 16 years running my own software development agency, prior to it being acquired by Putti, I've seen many situations play out from near and afar, and I am more convinced than ever that it's this final category that leads to product failures.

And that bothers me, because there are a lot of highly creative and innovative people coming up with awesome new software and app ideas, but I know from experience that many will end up torpedoed by the things their inventors didn't know that they didn't know.

That is why I have written this eBook; to demystify software product development for non-developers, suggest some key questions for people to ask, and help people understand something of the answers. Then also to offer some insights into the broader software product journey.

Please feel free to contact me with feedback or questions at john@puttiapps.com and to connect with me on LinkedIn: https://www.linkedin.com/in/john-halvorsen-jones

New Product Validation

If we're honest with ourselves, we'll recognise that we have a tendency to fall in love with our own ideas and that, if we're not careful, this can lead us to skipping or minimizing this vital first step.

This step is what's typically referred to as 'market validation', however some products are purely in-house; invented within an organization to improve efficiency, create a unique customer experience, or in some other way deliver a sector leading advantage.

Whether an in-house product, or an external product intended to be sold to others, the first thing we don't usually know is what else exists in the world. We also don't entirely know what people need and what they'll pay for it.

So, here's a quick checklist you can run through before going further:

1. What is the problem my product solves
2. Have I searched globally for something that could do the same job?
3. If there are competitors in any form, have I analyzed their relative strengths, weaknesses and price points?
4. Do I clearly know what unfilled niche my product fills?
5. If in-house, what cost-savings and/or revenue increases could my product deliver to the business over three, five and ten years?
6. If external, are there relevant examples that I can draw on to provide an initial idea of what people would pay for my product and how big the market is for it?
7. If an external product, is there a way I can test market the product to gauge response and run pricing experiments before building it?

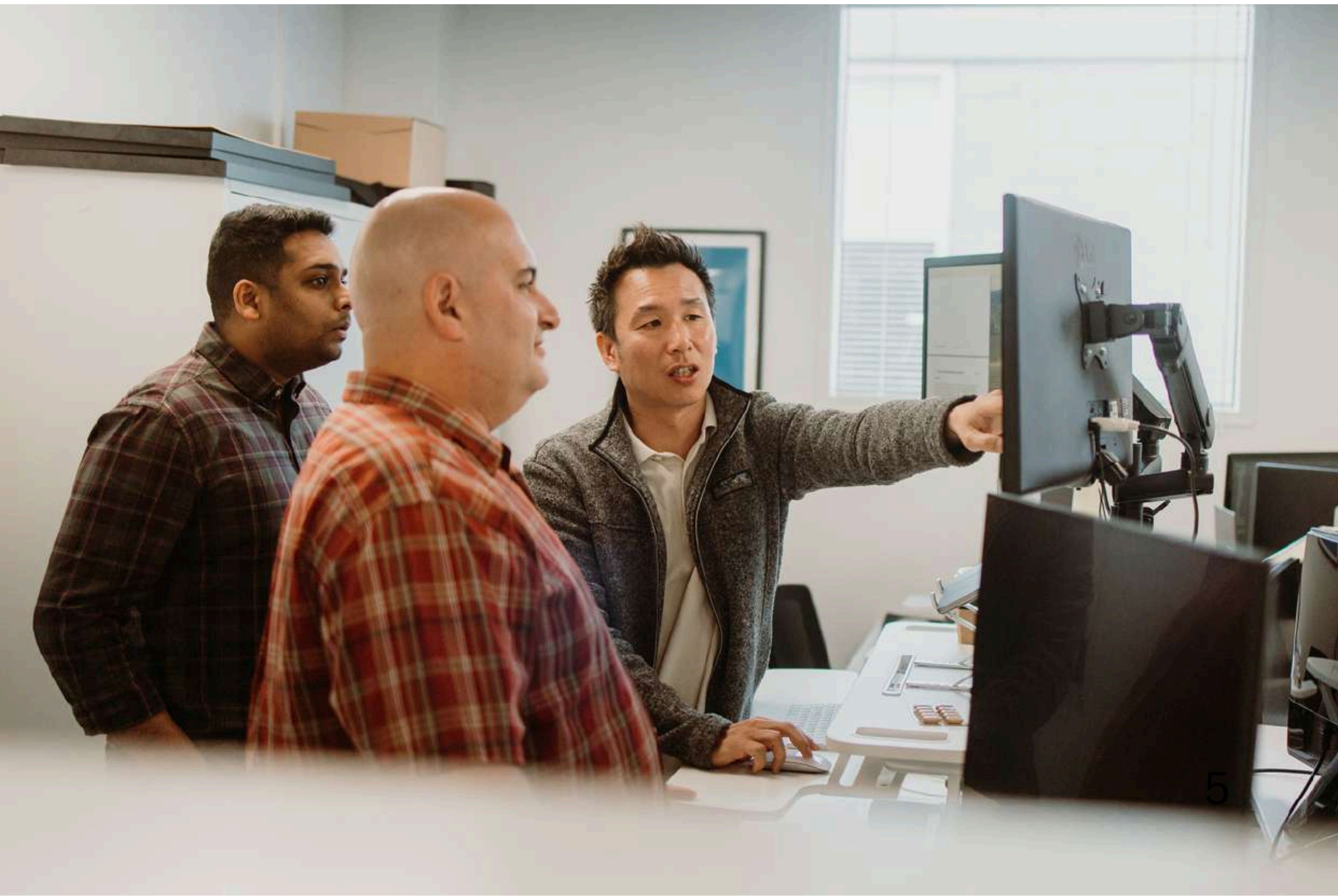
This last point may involve things like setting up marketing funnels and capturing expressions of interest before creating anything, however in many situations you may still need to build an MVP (Minimum Viable Product), in order to effectively test a market. This will be covered further in the 'MVP and roadmap' chapter.

New Product Validation

A great way to focus your thinking in the validation stage is to start by filling out a simple canvas; either a Lean Canvas for a start-up or, for in-house, some form of Product Canvas.

Also consider reading the classic market validation book, “If You Build It, Will They Come?”, by Rob Adams. This will take you through a deep dive into practical strategies and tools you can use to minimize your chance of a new product failing to meet the market.

Perhaps the most important aspect of market validation is keeping an open mind, and actually being prepared to give up on your idea. After all, it's better to give up on it before you spend a lot of money building it than it is afterwards. Don’t think of this in terms of being a quitter, but rather as wisely saving your resources to use for a better idea.



DIY Product Design

Many people come to software development agencies with pretty much just an idea, which is totally fine if they're happy to pay for the agency's expertise in helping them to develop it, however they could also choose to save a lot of money by investing in a pencil and paper first.

I mean, if I had a software or app idea, I'd start by sketching out:

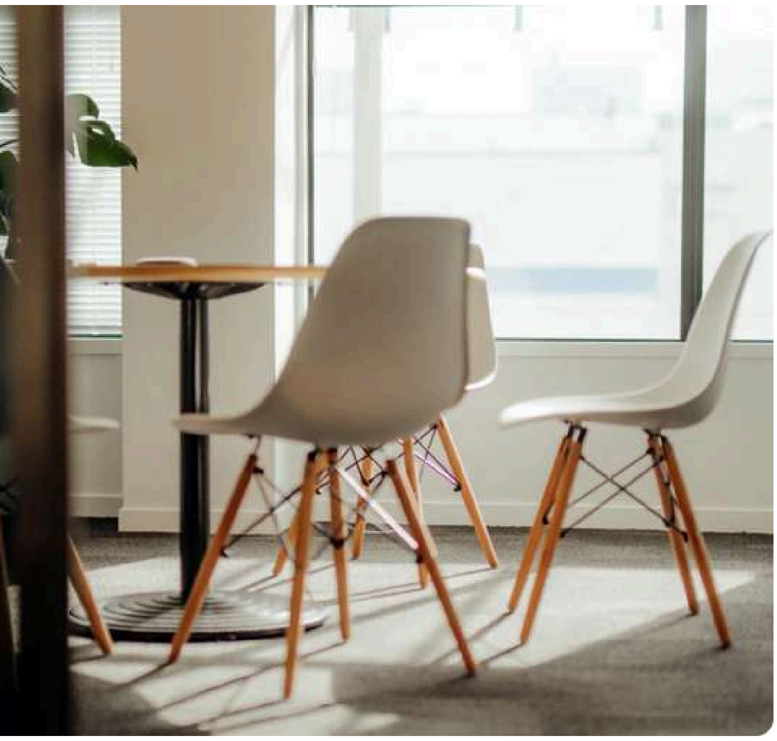
- The screens it will have
- The content of those screens
- The flow between screens
- Notes about things that need to happen that aren't obvious from the screen designs
- Any obvious TXT or email notifications required as part of the process

Doing this would also be likely to uncover any conceptual issues that need to be thought through further.

There are lots of great prototyping tools that could be used to make this look pretty, such as UX Pin (pretty easy) and Figma (a bit more pro), but how far one goes with DIY is a matter of choice; a pencil and paper, or whiteboard with image capture, is sufficient to develop ideas and convey them to someone else.

Whatever process you come up with, the main point here is that there's quite a bit you can do to develop your idea before putting it in front of a software professional, and doing so will enable that person to provide meaningful early feedback, perhaps even some ballpark estimates.

Besides, designing your product is usually the most fun and creative part of the journey, so I don't personally see this as onerous.



This is where things get scary. I mean, there's just so much that can, and often does, go wrong. Even worse, in some cases you may not know it has gone wrong for years.

Perhaps the best way I can illustrate this is to tell a true story. Once upon a time in the land of Tāmaki Makaurau, Aotearoa (AKA Auckland, New Zealand), there was a very forward thinking agency in a specialist area of marketing.

As part of their dedication to providing a world class experience they hired a software developer to help them create a piece of software uniquely fitted to the complex ongoing interaction between them and their clients. They had enough understanding of software to ensure that the technologies their developer chose were good choices at the time. The software got built, and refined, and it worked. Success!!! Well, sort of.

After a couple of years their software developer moved on and this roughly coincided with plans by the business to spin their product off as a SaaS startup. They put a heap of work into setting up website onboarding and so forth, and started to take on a few software subscription customers.

Choosing a Build Partner

Choosing a Build Partner

They also came to me to talk about getting further work done on the product.

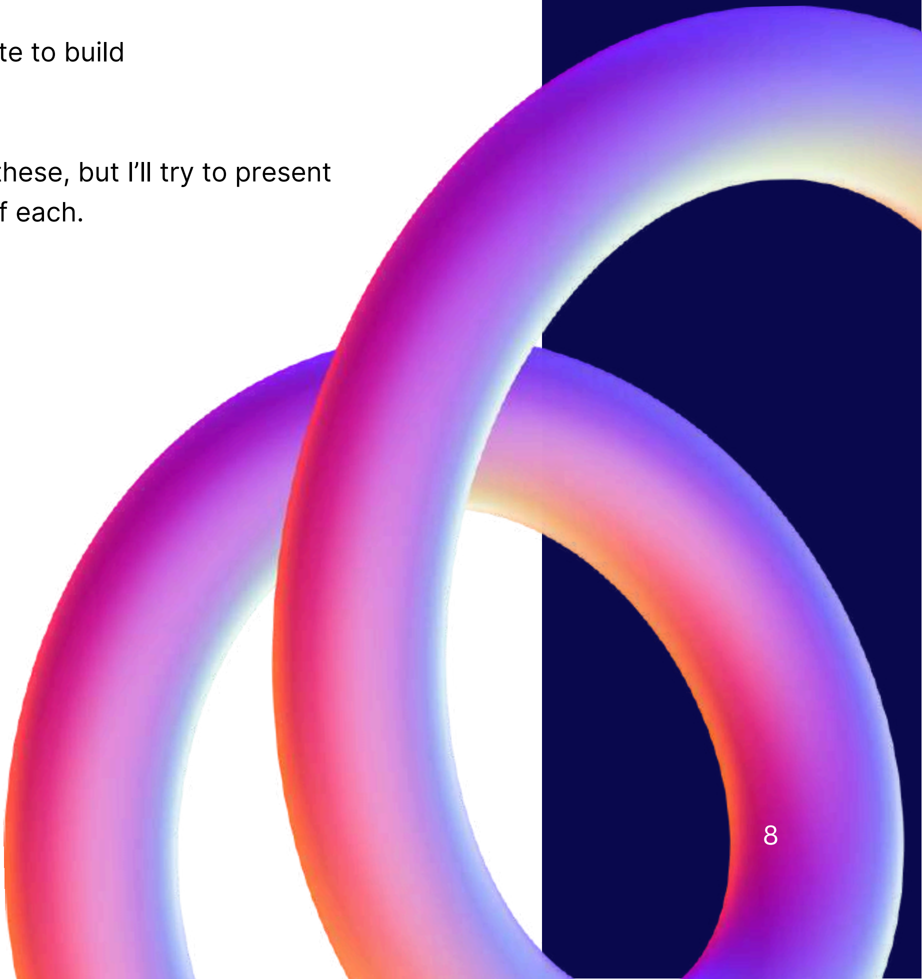
It was this that led us to discover that, under the hood, the software was a bit of a pile of crap; not really suited to taking much further without a complete rebuild. Long story short, there was a lot of repetition where there should have been elegant reuse of functionality, so every time you made certain changes they had to be made in multiple places, without missing any. Well, that was the main issue anyway, but it just wasn't well written for maintainability or scalability in general, two important factors if it was to move from being an in-house product to a commercial SaaS package.

This definitely wasn't the worst software we'd been asked to take over, but it illustrates something many people aren't aware of. Most people think that if software looks good, and works, then it is good. However there are numerous ways in which this assumption can prove to be false.

With that as some context around the importance of getting the right build partner(s), let's broadly consider the options:

- Local software development agency
- Offshore software development agency
- Hiring one or more developers
- Finding a developer friend or graduate to build
- Contracting a one-man band

I'm naturally biased towards the first of these, but I'll try to present a balanced view of the pro's and con's of each.



Local Software Development Agency

In terms of hourly rate, this will be your most expensive option. However, if they have a proven history it will also be the most likely to produce a professional result that will succeed.

Key advantages include the benefit of multiple team members with different specialist skillsets, well developed processes and sometimes significant IP they can bring to bear; where they have contributing IP an agency may not even prove to be the most expensive option.

A well-established agency also provides continuity if staff members change over time, both due to the wider team and because they have experience in the esoteric art of hiring good developers.

Offshore Software Development Agency

By "offshore" people usually mean developing countries with low wage costs.

Theoretically these could provide most of the advantages of a local software development agency, but at a much lower price. In terms of our experience though, this would be a very rare outcome. What we have seen, through people bringing projects to us to review or take over, is a range from practically a scam to just poorly executed work.

With only one exception, we have never been able to take over projects that we've been asked to that originated offshore because the coding quality has always been so low as to be pretty much unmaintainable.

Offshore development can work well, but usually in the context of large projects that have local Product Owners, Architects and CTO's providing very clear technical specifications and a great deal of oversight.

Hiring One or More Developers

Most of us probably wouldn't be confident to hire and manage astronauts, but with developers many people with no experience seem quite willing to just give it a go, even though the hiring expertise required is nearly as specialized.

How can you tell if they're good? What motivates them? How can you ensure they make good decisions? Hiring an in-house team is usually the right thing to do when a new software product becomes a market success, and you're in a position to hire the right people to hire and oversee them, but until then there's a good reason why many businesses who could, don't. When you are ready though, building an in-house team has the potential to increase the output and focus of your development spend.

Thinking ahead from the outset though, when starting with an agency it could be a great idea to ask them what their policy is if, in future, you were ever to request to transition key people working on your project to an in-house team.

Finding a Developer Friend or Graduate to Build

One of our clients was very successful in getting an excellent developer friend to build the first iterations of their software in return for shares in their startup, but they're the only person I've ever known this approach to work for.

I really don't know what the following quote was referring to, but it feels very relevant to what we've seen of this situation.

"Along the way, in their attempt to make their dream come true, many ultimately prefer to stay on the sideline. In the meantime, their packages of good intentions start leaking, or their letters of hope remain silenced by unawareness."
- Erik Pevernagie

Contracting a One-Man Band

In the music world a one-man band could be Ed Sheeran or your local drunk busking for his next whiskey and cola. It's much the same with software developers, with the main difference being that it's harder to tell which is which.

This is also broadly similar to the 'finding a developer friend' scenario, but with the implication that you're paying them, which may help but is no guarantee the outcome will be any different.

OK, I admit, I'm clearly pretty biased towards agencies, but not without long experience to inform that bias.

Technology Choices

Technology Choices

In the majority of software builds the technology used is a direct consequence of the build partners selected. Most build partners will have a preferred tech stack, or maybe two to choose from. They will tell you why what they offer is absolutely the best choice for your project, but is it, or is it just what they know?

There are often multiple options that are all fit for purpose, however there can be cases where a tech choice is made that's ultimately ill suited to the product, or just hard to find talent for down the track.

It's important to be sure that all programming languages, frameworks and database technologies your development partner intends to use meet the following criteria:

- Very commonly used across the industry
- Has been around long enough to be well proven
- Has a very high chance of being well supported into the future (which they probably will if the points above are true)
- Is known to scale effectively
- Does not undermine creating a secure application
- Are a good choice for developer productivity
- Are efficient in their use of compute and database resources

Depending on your level of technical awareness, you may be able to assess what's being pitched to you against these criteria, but if not it would be ideal to find an independent software specialist who can provide an unbiased opinion.

Technology Choices

As a quick reference though, here's some very common technologies that fit these criteria. (But feel free to skip ahead if this just looks like monkeys typing)

Web Apps

- Common PHP frameworks, Laravel being the most ubiquitous
- Server-side JavaScript (Node.js, Nest.js)
- Microsoft .Net (though "resource efficient" isn't its greatest strength)
- JavaScript front-end frameworks, the best probably being React.js, Next.js and Vue.js
- Various component libraries based on above

Mobile Apps

- React Native
- Flutter
- Native iOS (Swift) and Android (Java/Kotlin)
- Progressive Web Apps (PWA's)

Database

- MySQL
- Microsoft SQL
- Postgres
- MongoDB
- AWS DynamoDB

This isn’t a definitive list, but it might enable you to tick-off certain technologies commonly put forward.

However of equal or greater importance is HOW they are used; in particular my three S’s of Security, Scalability and Serviceability - serviceability being how easy or otherwise it is to maintain, update and add on to an application.

Ultimately you either need to clearly know why you can trust your vendor's advice, or to get some oversight from an independent expert.

Integrating AI's

There are many spectacular pieces of functionality that once would have been technically infeasible or cost prohibitive to create, that can now be rapidly and integrated into custom software projects for relatively little cost.

Here are some examples:

Integrating AI's

Chatbots that can provide context specific help to users via a chat session, as well as assisting users with relevant content and insights as required. These work much like ChatGPT or Claude, and in fact are often powered by the technologies behind those, just with subject specific training applied to a private model.

Speech recognition capability. This has now reached a fairly conversational level and no longer requires the tedious training of old speech recognition systems.

Smart Search that understands user's queries, including context, and meaningful inferences around intent, in order to provide highly relevant search results from a website, application or other content source.

Image generation tools that assist users with visual content creation within relevant areas of your software.

Data analysis tools that enable users to gain insights from data that resides within the software they're integrated into.

Predictive Analytics that leverage AI algorithms to analyze historical data, identify patterns, and make predictions.

Intelligent Automation where AI enables software to automate repetitive tasks, making them more efficient and error-free.

Image and Video Recognition enabling software to analyze and recognize objects, people, and scenes within images and videos.

Personalization that enables software to deliver personalized experiences by analyzing user behavior, preferences, and historical data.

Sentiment Analysis tools that determine the sentiment behind user interactions.

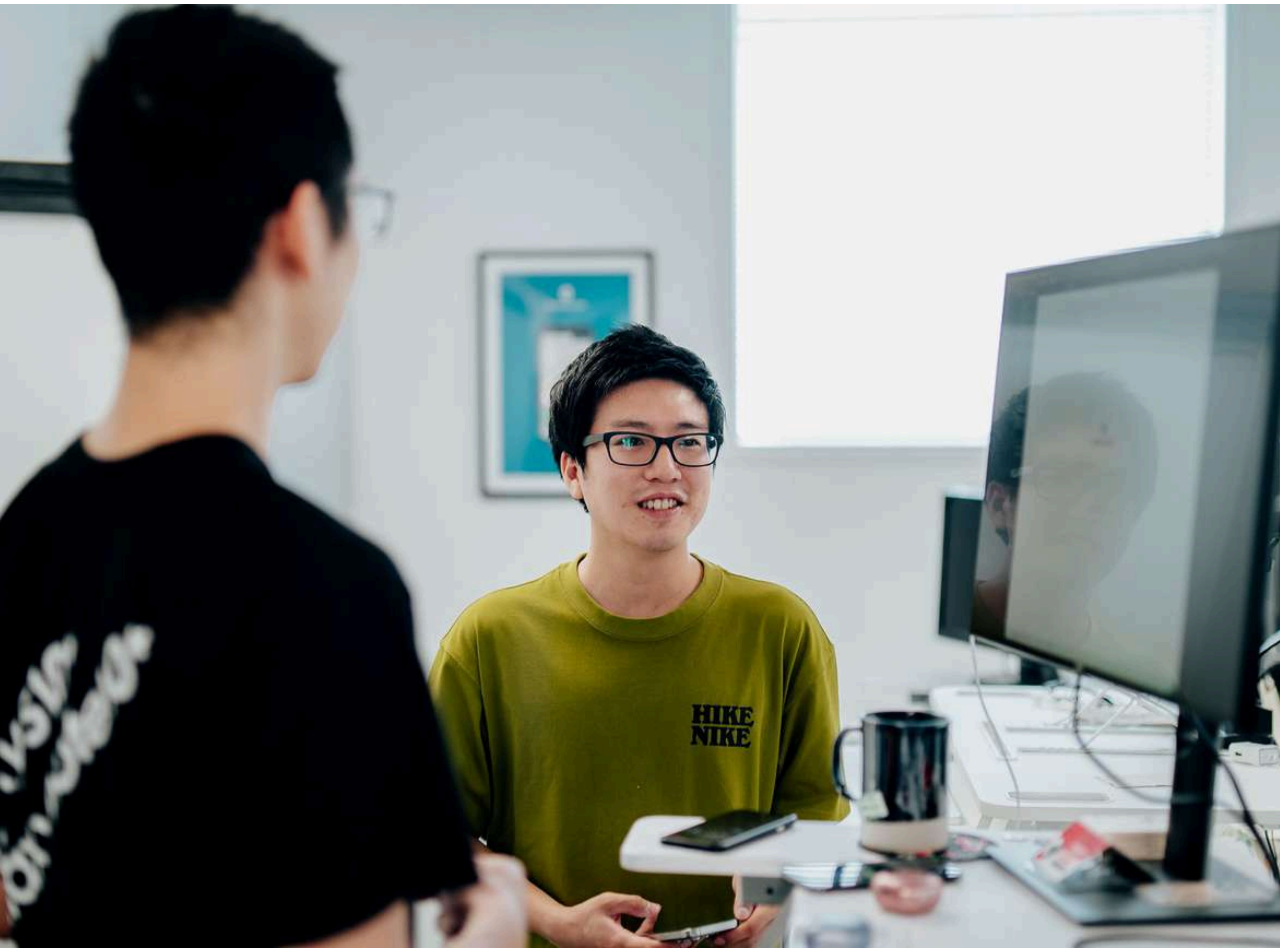
Anomaly Detection that can detect unusual patterns or anomalies within data, allowing software to identify potential fraud, network intrusions, system failures, or abnormal user behavior.

The integration of this kind of functionality is usually handled by passing data from your mobile app or web application to an API that is provided by the AI you are making use of, and then receiving responses back from that.

By way of explanation, an API, or Application Programming Interface, is a standard data gateway that provides structured data services to other applications online. Often these will come with some form of usage cost so, whilst they provide incredible functionality for relatively little development work, you will need to ensure that you understand the cost implications, especially if your application is designed to scale to large usage.

Understanding User Requirements

Understanding User Requirements



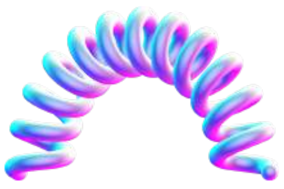
A common mistake in the software world is developers building what the customer asked for. The other common mistake is not building what the customer asked for.

The thing to appreciate is that end users do understand their own needs and problems and, for better or worse, often have ideas about what to build to meet them, but their experience around translating business requirements into a software product is usually limited.

What compounds the issue even more is that, within an organization, people don’t always have the same view as to the details of how things should be done. And it’s common for people to forget to mention the exceptions, or “edge cases”, and these can often prove one of the most challenging aspects to deal with when designing software. A common phrase we hear, that always proves to be untrue, is that “it’s really all very simple”.

So this is a challenging area, however here are some tools and approaches that are helpful.

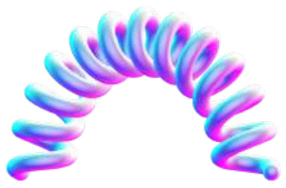
Jobs to Be Done Framework



This is a highly user-centric approach to understanding needs and requirements. It’s worth reading Anthony W Ulwick’s book, “Jobs To Be Done”, however the core technique is easy to explain. Essentially, what it recommends is setting up long-form one-on-one interviews with people who represent different user roles and then really digging into their working lives; their needs, workplace pressures, physical working environment, how things tend to happen in reality, what they find frustrating or encouraging, their view on requirements and the exceptions (edge cases). It is recommended to capture all this on video, both so the interviewer can focus on the interview process and also to enable it to be mined for in-depth notes later.

The main point of Jobs To Be Done is gaining a deep understanding of users' needs, business processes and the environment in which work occurs, in order to discover, clarify and prioritize requirements.

Agile Software Development



At the heart of agile software development is the concept that nobody really knows what the organization needs, not even the people who will be using the software; at least, not until they start playing with something, then they can usually provide some useful feedback.

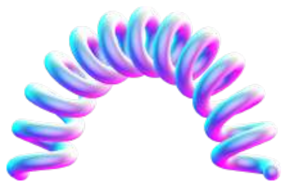
A highly agile approach would be just to get a bare bones idea of what the users needed, an appropriate flow of key events and some examples of the kind of information involved. Then a developer would stitch together an initial idea for an application from which to obtain rounds of feedback. The project would run in iterative loops with lots of feedback from all the relative parties, until the product of everyone's input came into view.

It's a great way of solving a problem that has tended to plague large organisations, which is that of people, including professional Business Analysts, working hard to correctly specify a large project, only for it to be a complete flop when real users try using it.

However, the problem with a fully agile approach is that it can get very expensive. Software development is most cost effective when built in a linear manner against a clear and highly detailed set of requirements. The best way to resolve this dilemma is usually to take a very agile approach to developing prototypes, and gaining feedback on those, until there is clarity on what the product should be, at which point the eventual software can be built out efficiently.

Agile has also spawned a whole set of methodologies around controlling time, cost, feature prioritization and quality. Almost all projects now incorporate some or all of these methodologies. What does need to be assessed on a case-by-case basis though is the balance between up-front specifying and iterative changes based on reviews along the way.

Prototyping



Prototyping is a critical step in the software development process that involves creating preliminary versions of a product to test and refine ideas before full development. Various approaches to prototyping help product owners, designers and developers explore different aspects of a product's functionality, usability, and design. Here are some common approaches:

Paper Prototypes: This involves sketching interface elements on paper or a whiteboard. This is very useful for early-stage brainstorming as it enables ideas to flow without the need for technical development.

Wireframes: Wireframes are usually neater and more detailed than paper prototypes and provide a skeletal framework of the software's interface. They focus on layout, content placement, and navigation without detailed visual design.

Interactive Prototypes: These prototypes simulate user interactions and workflows using digital tools. They range from low-fidelity (clickable wireframes) to high-fidelity (fully interactive models with realistic design elements). Tools such as Figma, InVision, and UX Pin allow designers to create interactive prototypes that provide a more accurate sense of how the final product will function.

High-Fidelity Prototypes: High-fidelity prototypes closely resemble the final product in terms of design and functionality. They include detailed visual elements, realistic content, and complex interactions. These prototypes are often used for usability testing to assess user experience with a more polished representation of the product.

Wizard-of-Oz Prototypes: This approach involves simulating functionality manually, often using a combination of human operators and rudimentary software interfaces. For instance, users may interact with a seemingly automated system that is actually controlled by a person behind the scenes. This can be a great approach for initial market validation or user testing.

Each of these approaches has a different time-benefit trade off, and the suitability of each can also come down to the size, complexity and stage of the project.

User Experience

User Experience

Since the earliest computers the shift that's been occurring in human-computer interaction is that of progressively moving from humans having to adapt their interaction to the workings of computers, to computers being adapted to the needs of humans.

Arguably the greatest leap-forward in this progression was the advent of the Graphical User Interface (GUI), first commercialized by Apple and later by Microsoft as Windows. Other significant steps forward have included touch-screen devices and, from a very different angle, AI's becoming mainstream.

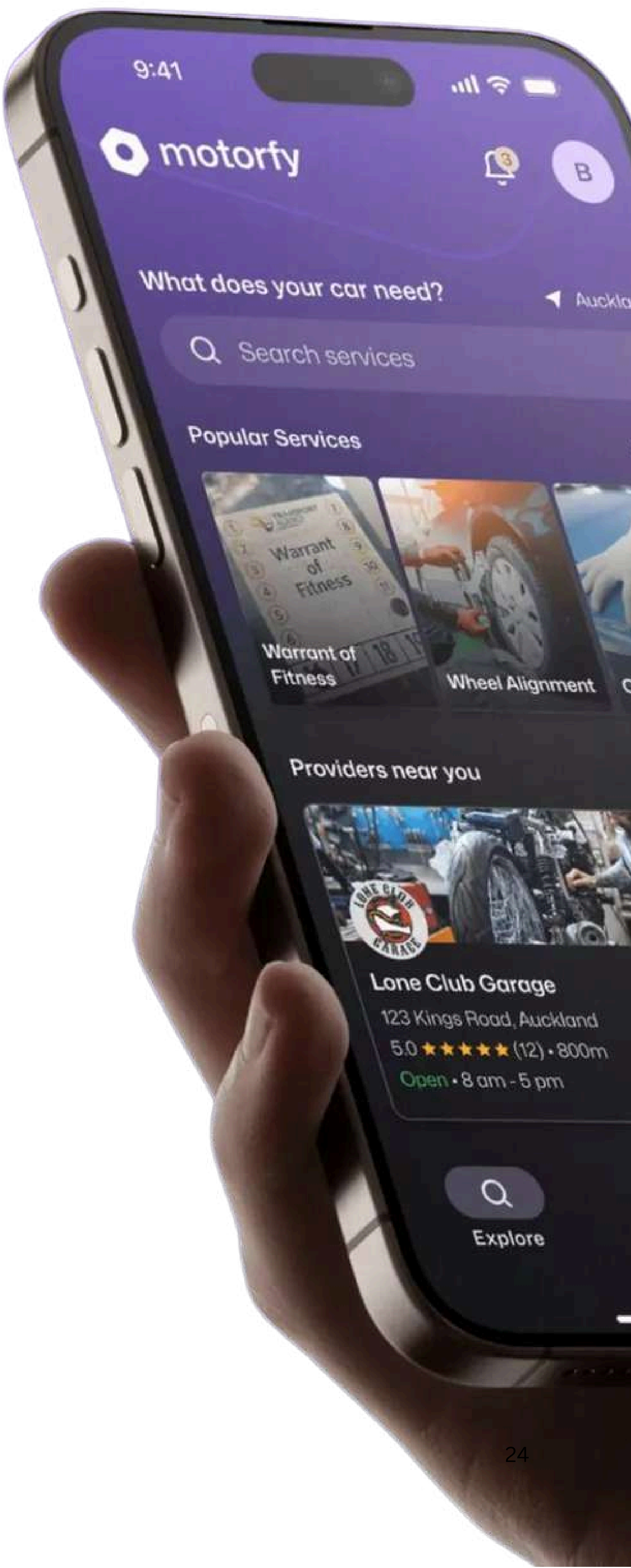
The science of UX encompasses all aspects of end-users' interaction with a product and its goal is to make that interaction as intuitive, seamless and enjoyable as possible.

A seminal concept in the evolution of UX thinking was, "Don't make me think", an idea introduced by Steve Krug's book of the same name. As the awareness of UX has continued to develop, the following principles have become commonly recognised:

User-Centered Design: Focus on understanding and addressing the needs, preferences, and behaviors of the target users throughout the design process. This includes using terminology familiar to the target audience, rather than jargon or acronyms that may not be.

Usability: Ensure the software is easy to use, with intuitive navigation and clear, understandable functionality.

Consistency: Maintain uniformity in design elements, interactions, and terminology to help users build familiarity and reduce learning curves.



User Experience

Accessibility: Design to accommodate a wide range of users, including those with disabilities as much as possible. Examples can include screen reader friendly pages, ability for the user to enlarge font size, text-to-speech functions and so forth.

Feedback: Provide timely and clear feedback for user actions to help users understand the results of their interactions and guide them through tasks.

Efficiency: Optimize workflows and minimize the number of steps required to complete tasks, making interactions as efficient as possible.

Memory: Avoid ever requiring the user to remember something; for instance, if you show a phone number, including an easy control activate phoning it is better than the user having to try and remember or copy the number into the phone app. Likewise, people should be able to access information known within an application when required to enter it somewhere else.

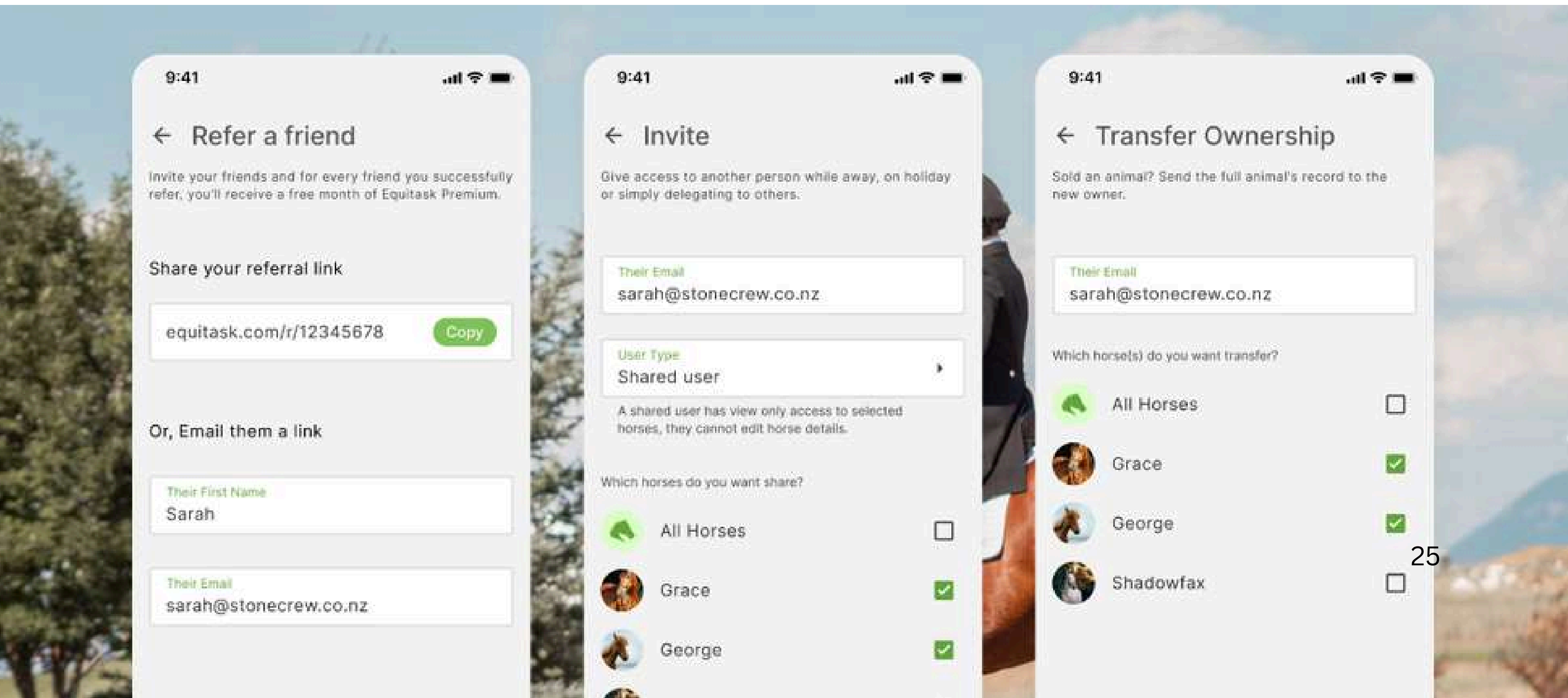
Error Prevention and Recovery: Design to minimize the likelihood of errors and provide helpful error messages and recovery options when mistakes occur.

Visual Hierarchy: Use layout, color, typography, and spacing effectively to prioritize and organize content, guiding users' attention to important elements.

Flexibility and Customization: Allow users to adjust settings and preferences to fit their individual needs and work styles, though there can be a balance between this and overcomplicating a product.

Performance: Ensure the software is responsive and performs well under various conditions, providing a smooth and reliable user experience.

Emotional Design: Consider the emotional impact of the visual design, creating a positive and engaging experience that resonates with users on an emotional level.
User Testing: Continuously test with real users to gather feedback and validate design decisions, making iterative improvements based on this input.


9:41
Refer a friend
Invite your friends and for every friend you successfully refer, you'll receive a free month of Equitask Premium.
Share your referral link
equitask.com/r/12345678 Copy
Or, Email them a link
Their First Name: Sarah
Their Email: sarah@stonecrew.co.nz

9:41
Invite
Give access to another person while away, on holiday or simply delegating to others.
Their Email: sarah@stonecrew.co.nz
User Type: Shared user
A shared user has view only access to selected horses, they cannot edit horse details.
Which horses do you want share?
All Horses ☐
Grace ✓
George ✓

9:41
Transfer Ownership
Sold an animal? Send the full animal's record to the new owner.
Their Email: sarah@stonecrew.co.nz
Which horse(s) do you want transfer?
All Horses ☐
Grace ✓
George ✓
Shadowfax ☐

The Role of The Product Owner

The Role of the Product Owner

The Product Owner role is typically very key to a successful software project. In an agency situation this role may end up split between people within the client's organisation and agency staff, and the role names may therefore be different, however it's important that all areas of responsibility are covered.

Whether collectively, or encapsulated in the Product Owner role, the main responsibilities in view here are to:

- Own the vision developed with stakeholders
- Assist with product strategy
- Act as a liaison between the business, the developers & other vendors
- Deeply understand the needs of users
- Translate business processes into technical requirements
- Proactively manage the project and report on progress
- Identify and manage risks
- Control project budgets
- Manage project documentation
- Own UAT (User Acceptance Testing)
- Define roll-out and support requirements
- Continuously research, gain feedback on and prioritize the roadmap
- Monitor and report on ongoing costs

Where there is no formal Product Owner in place it is important to at least map out who will be across each of these functions, and ensure there is still a central person in the project who will be proactively holding all the pieces together; a project manager or similar. Also bear in mind that some agencies offer a Virtual Product Owner service, so it may at times be possible to get the complete role without having to hire a permanent full-time person.

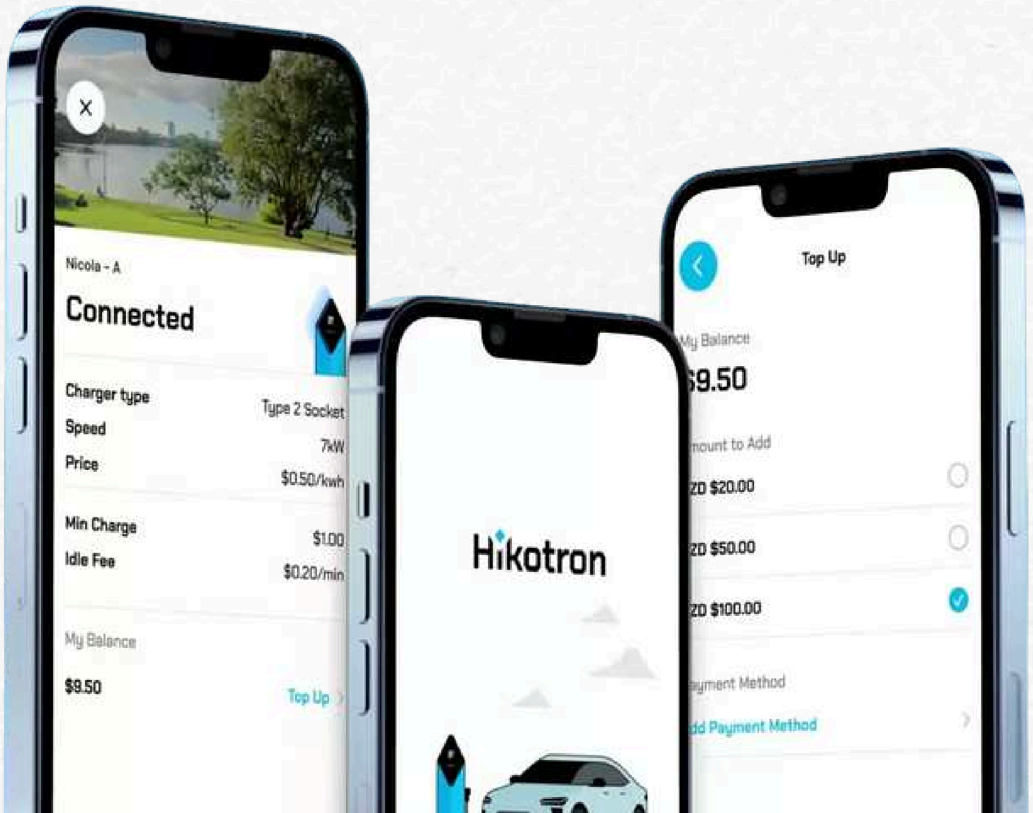
Let's consider some of the common pitfalls of software projects and how the Product Owner fulfilling their role well can mitigate them.

Common Pitfalls of Software Projects

End Product is not Fit for Purpose

The responsibility for ensuring end products are fit for purpose sits squarely with the Product Owner as they are the bridge between the intended users and the developers.

They must deeply understand the users needs and the business processes that need to be fulfilled. They must also be able to translate these into clear feature specifications for the development team and provide meaningful feedback as features take shape. If the Product Owner has any lack of clarity around either the requirements side or what's actually being built, this introduces significant risk of the eventual product not being fit for purpose.



Time and Budget Overrun

Although time & budget overrun may well originate in the development team, the Product Owner plays a critical role in monitoring budget, reporting back to stakeholders and working through options where the budget is going off-track. Options may include simplifying the project, accepting that areas of it require more time than expected or solving problems, including any team problems if that is having an impact.

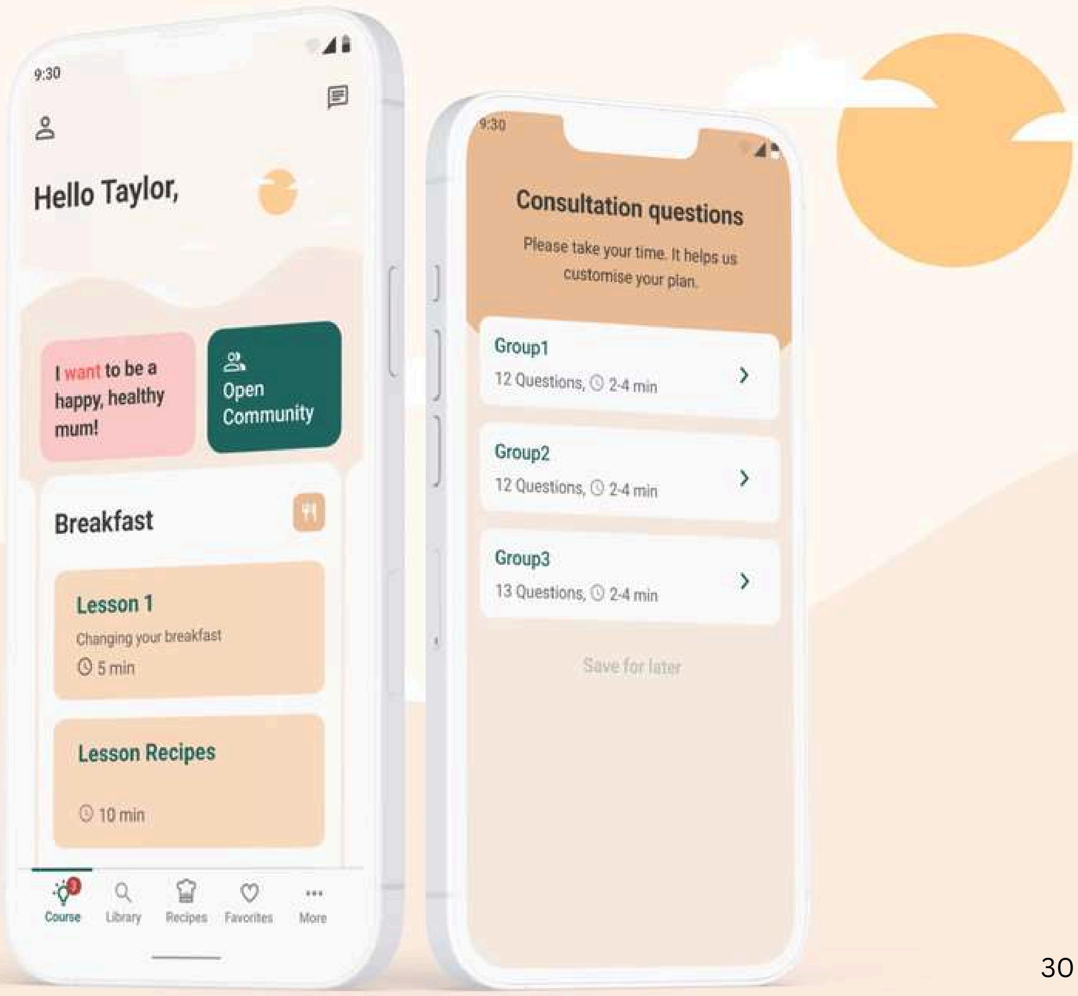
It is the product owner that needs to set up appropriate project management processes that provide visibility on project stages, features, feature completion and overall estimates of completeness for both the team and stakeholders. Again, the Product Owner's depth of understanding of both the requirements and implementation will greatly affect their ability to provide quality oversight.

Poor technical decisions lead to a software cul-de-sac

A Product Owner is not usually a Software Architect, however they need to be experienced enough in the software development industry to have meaningful conversations with software agencies, architects and the like. It's important that they grasp the implications of different tech stack and architectural decisions, at least sufficiently to liaise with stakeholders and ensure that stakeholders need will be well matched to technical decisions being made. Otherwise, a project may run for years, only to end up in a cul-de-sac of scalability, security, maintainability or extensibility problems.

User Experience is Poor

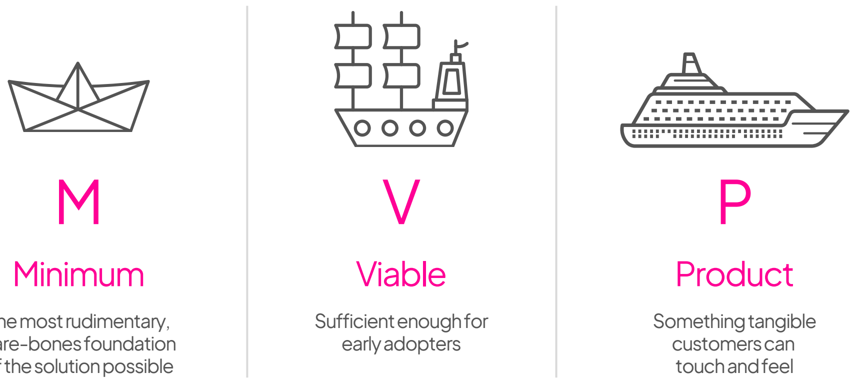
Whilst again a Product Owner is not expected to be a UX specialist, they really do need a good working understanding of the field to contribute to UX decisions as they arise. As with software Architects, the Product Manager needs to be an effective bridge; able to select and converse meaningfully with specialists, and to translate between specialists and stakeholders.



MVP and Roadmap

MVP and Roadmap

Diagram source: geeksforgeeks.org


M
Minimum
The most rudimentary, bare-bones foundation of the solution possible
V
Viable
Sufficient enough for early adopters
P
Product
Something tangible customers can touch and feel

The term MVP, or Minimum Viable Product, refers to an initial version of a new product that includes only the essential features necessary to meet the needs of early adopters and validates the product's core value proposition. The primary goal of an MVP is to test the business hypotheses with real users, at minimal cost.

The other significant benefit of starting with an MVP is that any further development is then informed by user feedback on the live product. No matter what feedback mechanisms you employ prior to MVP, there's nothing like feedback from real users carrying out their real work.

A caveat around MVP, put forward by Eric Ries, author of The Lean Startup, is that your MVP must still also be valuable to users; he later coined the term MVVP (Minimum Viable Valuable Product) to reinforce this point.

In trying to stick to an MVP, one of the common challenges product owners face is catching everyone's ideas and enthusiasm for new features, without bloating the initial build. A good way to handle this is to run an ideas board that's separate to the MVP feature set, and then to evaluate those ideas to see which ones merit being brought over into the product roadmap.

The product roadmap is the collection of new features in the pipeline at any point of development following MVP. Typically roadmap's are prioritized and re-prioritised around business value; a quasi-scientific assessment of how many people or operations a feature touches, and the significance of the value delivered to each. Roadmaps also usually contain an outline of the key functionality of each feature and some degree of estimate around the resources required to deliver it. Sometimes very high or low estimates will trump business value in determining a feature's order in the roadmap.

From a psychological point of view, often just being able to say that something is "on the roadmap" encourages users to to feel that their needs have been heard and that they will, at some stage, receive what they've been asking for, even if you know that stage is going to be just after hell freezes over.

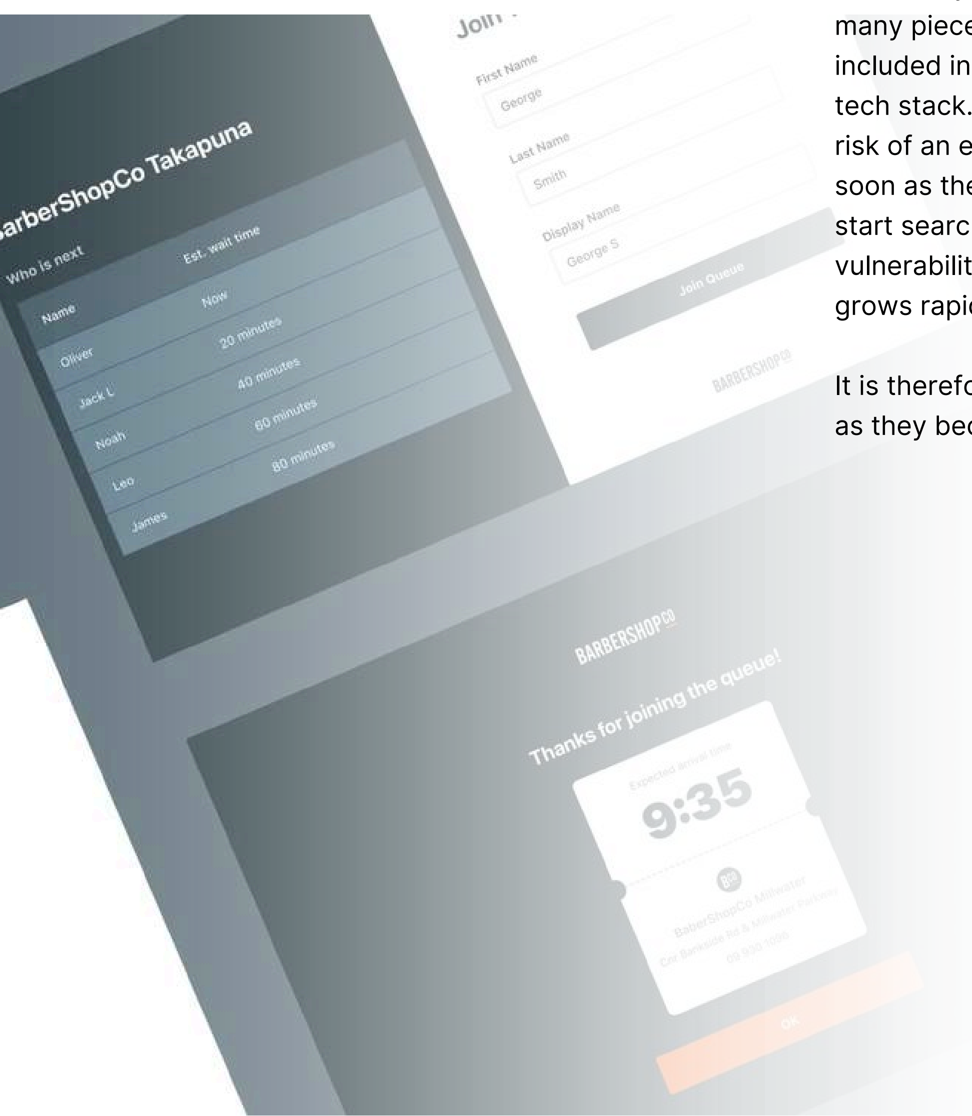
Releases and What Lies Beyond

Releases and What Lies Beyond

From ages 16 to 19 I trained almost every day to reach my goal of being a black belt in Karate and, when I finally got there, I'll never forget what was written on my black belt certificate; "The Beginning".

When the whole product development journey culminates in releasing your new product, you may breathe a sign of relief and feel like it's all over and that you can put your feet up. In reality, especially for successful products, you've only reached the beginning.

Not only will releasing your software to the real world potentially expose various unforeseen issues, but it will also invite feedback beyond the scope of any pre-release user testing. Listening to feedback, and the needs of people using your software will be, from that point onwards, one of the most important drivers of your product direction.



Who is next

Name	Est. wait time
Oliver	Now
Jack L	20 minutes
Noah	40 minutes
Leo	60 minutes
James	80 minutes

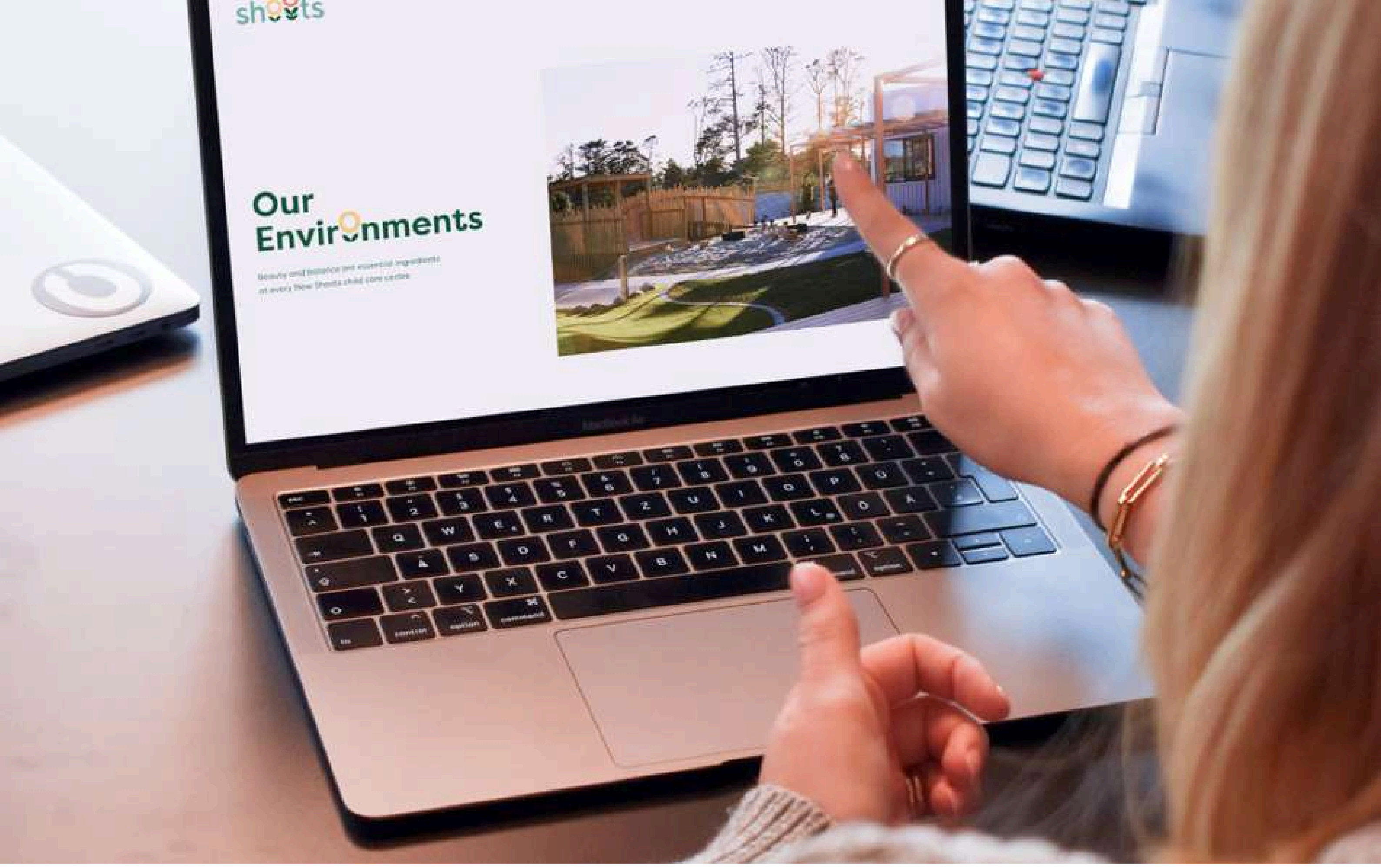
Another thing that's important to consider when planning your post release journey is the need for ongoing maintenance and code upgrades. Although sometimes applications just work day in, day out, with no real maintenance, there are multiple risks that come with a lack of ongoing maintenance. These include:

Performance degradation. Databases in particular need ongoing optimisation to stay performant, otherwise they are likely to become slower and slower as the number of records builds up.

Server outages. These are more likely to occur if usage of resources is not monitored since the server may become overwhelmed in one or more areas as usage of an application grows.

Security vulnerabilities becoming exposed. There may be unknown vulnerabilities in the many pieces of third-party technology included in any software and its underlying tech stack. While they remain unknown the risk of an exploit is extremely low, however as soon as they become known bots will typically start searching for applications with these vulnerabilities, so the risk of an exploit grows rapidly.

It is therefore important to patch vulnerabilities as they become known.



Errors and failures due to code rot. Changes in the environment (phone operating system, browser, etc), or underlying tech stack, can cause your application to start having problems, even though it hasn't changed ... actually, because it hasn't changed in sync with other developments in technology. Without proactive updating of the frameworks and other technology in your application this one will get you sooner or later, it's only a matter of time.

Some of the tasks required to avoid these pitfalls are ongoing; daily, weekly or monthly as appropriate, whilst others are large and irregular, such as upgrading the frameworks in use. It is ideal to have a maintenance budget that not only addresses the regular needs, but is also building up a savings pool to address the larger needs as they arise.

Also bear in mind that, whilst infrastructure, maintenance and other running costs may start off being quite low, as the usage of your mobile app or web application grows, you'll need to allow for the growth of all of these underlying costs if you want to maintain a stable and performant product. The good news is that, even so, software businesses tend to be far more scalable than bricks and mortar businesses because scaling doesn't usually require the commensurate scaling of things like office space and people. You may need more of both of these, but in a software company these costs usually take up a much lower portion of your revenue growth.

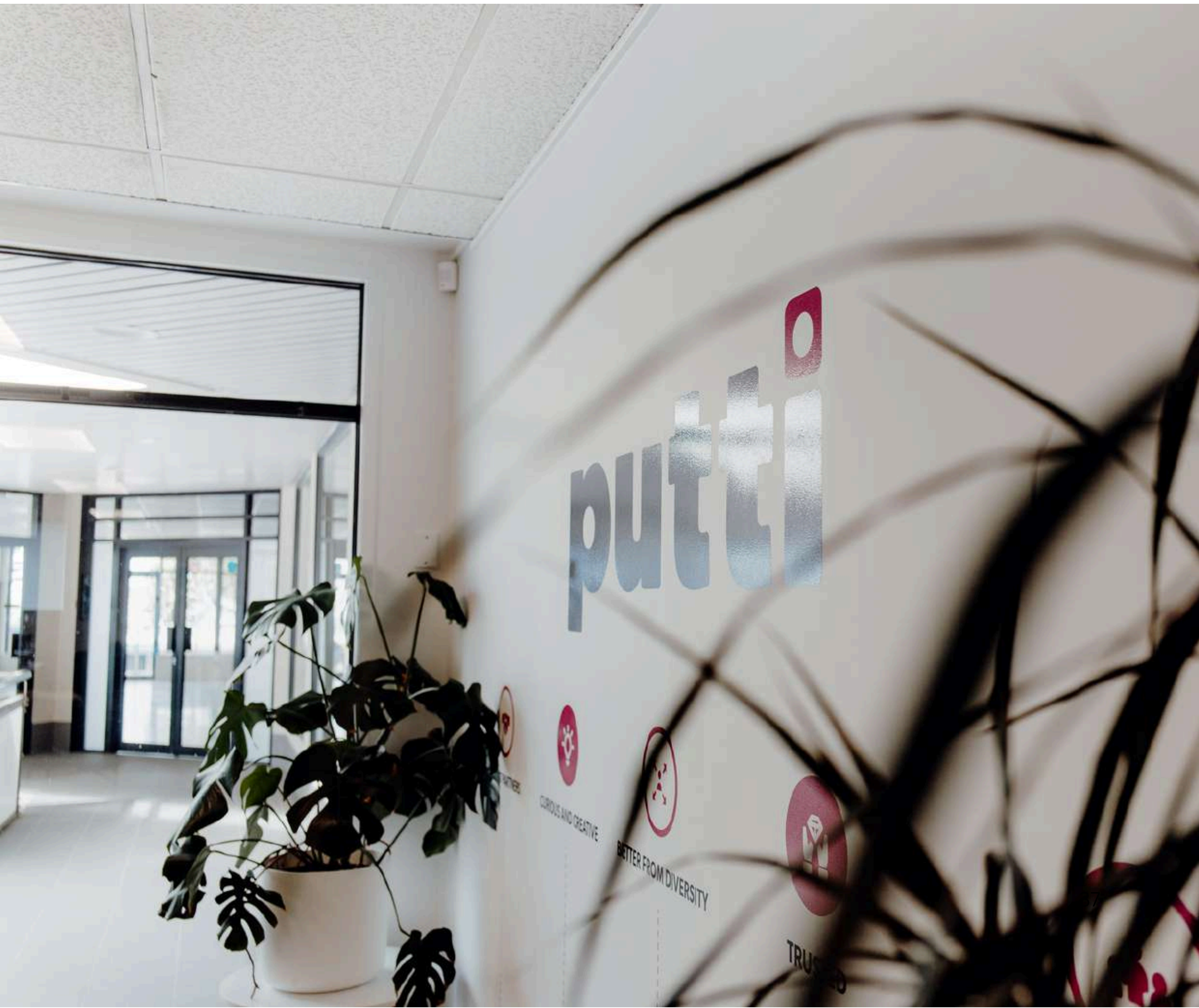
Startup Funding

Startup Funding

In many startup founders' minds, outside investment has the same allure as winning the lottery. Funding is hugely important to the success of most software startup’s, however if you plan to raise it from professional investors, angels, VC’s (Venture Capitalists) and the like, then it is important to have a clear understanding of:

What is required to get funding
The usual stages of capital raising
The potential pitfalls to watch out for

**“When you take that money your life is over”
– Paul Sinclair, OFNZ**



Getting Funding

Validation, validation, validation. Your product idea may sound very exciting but, when it comes down to it, can you validate everything you're saying?

- Product-market fit
- Competence and experience of your team (which can be augmented by a good dev partner)
- Size of market
- Competitor analysis
- Pricing
- Cost of customer acquisition
- Go to market cost
- Margin
- Likely development costs, including ongoing maintenance
- Other costs, including specialist team members you'll need to achieve your goals
- A realistic exit plan for investors

The hardest of these to validate tend to be pricing and CAC (Customer Acquisition Cost); so much so that often it's easier to get to a point of having some revenue so that you can run pricing experiments and test marketing in order to determine them in the real, rather than theoretical, world.

Whilst some Angels and VC's will consider investing in pre-revenue startup's, the degree of validation they're looking for tends to be nearly as much work as actually doing it for real (if you've already built a working product).

Stages of Capital Raising

Pre-Seed Funding

The pre-seed stage is often referred to as the "idea stage" at which founders are primarily working on conceptualizing their product and determining the viability of their business idea.

Typical funding sources during this stage include:

- Self-funding / bootstrapping
- Bank loans
- Friends and family
- Angel investors (thought these usually come later)

Usually the amounts involved are between $50,000 to $500,000 and primarily go towards building an MVP (Minimum Viable Product, market research, and legal fees related to company and shareholder setup.

Seed Funding

The seed stage is where the product moves beyond the initial prototype or MVP, and the startup starts gaining traction. Seed funding is typically used for product development, market research, hiring early team members, and building a customer base.

Typical funding sources during this stage include:

(cont'd from Seed Funding stage)

- Angel Investors
- Seed Venture Capital Firms (that specialise in early stage investment)
- Crowdfunding

Usually the amounts involved are between $500,000 to $2 million. This funding allows startups to refine their product, expand the team, develop marketing strategies, and validate the business model.

Series A

The Series A stage is one of the most critical milestones for any software startup. At this point, the startup should have a clear product-market fit, with customers actively using the product and a validated business model.

Typical funding sources during this stage include:

- Venture Capital Firms
- Corporate Investors (usually where your product is relevant to their industry)

Usually the amounts involved are between $2 million to $15 million. Series A investments typically focus on helping startups scale their operations, refine their products, and expand to larger markets.

Series B

Series B funding is for startups that have successfully proven their product-market fit and are now looking to accelerate their growth and expand their business significantly. Typical funding sources during this stage include:

- Venture Capital Firms
- Private Equity

Usually the amounts involved are between $10 million to $50 million. The money raised during Series B is typically used for product refinement, sales and marketing efforts, hiring additional staff, and expanding into new markets.

Series C

Series C funding is for companies that are already leaders in their niche market and are looking to expand globally, acquire other companies, or develop new product lines. The goal of Series C is often to prepare for an exit, such as an IPO (Initial Public Offering) or acquisition.

Typical funding sources during this stage include:

- Venture Capital Firms (top tier)
- Private Equity and Hedge Funds

Usually the amounts involved are between $30 million to $100 million. The funding is generally used to fuel international expansion, major marketing campaigns, or large acquisitions that allow the startup to capture more market share.

Series D and beyond

While Series C typically marks the final round of funding before an IPO or acquisition, some startups may enter Series D, Series E, or even later stages of funding. These rounds are usually for startups that are looking to fuel specific growth initiatives or take advantage of unforeseen opportunities.

Amounts and funding sources can vary a lot, though are often similar to Series C.

The use of money raised in later stages can vary greatly, but is often used for acquisitions, scaling operations, or preparing for a public offering.

Pitfalls to watch out for

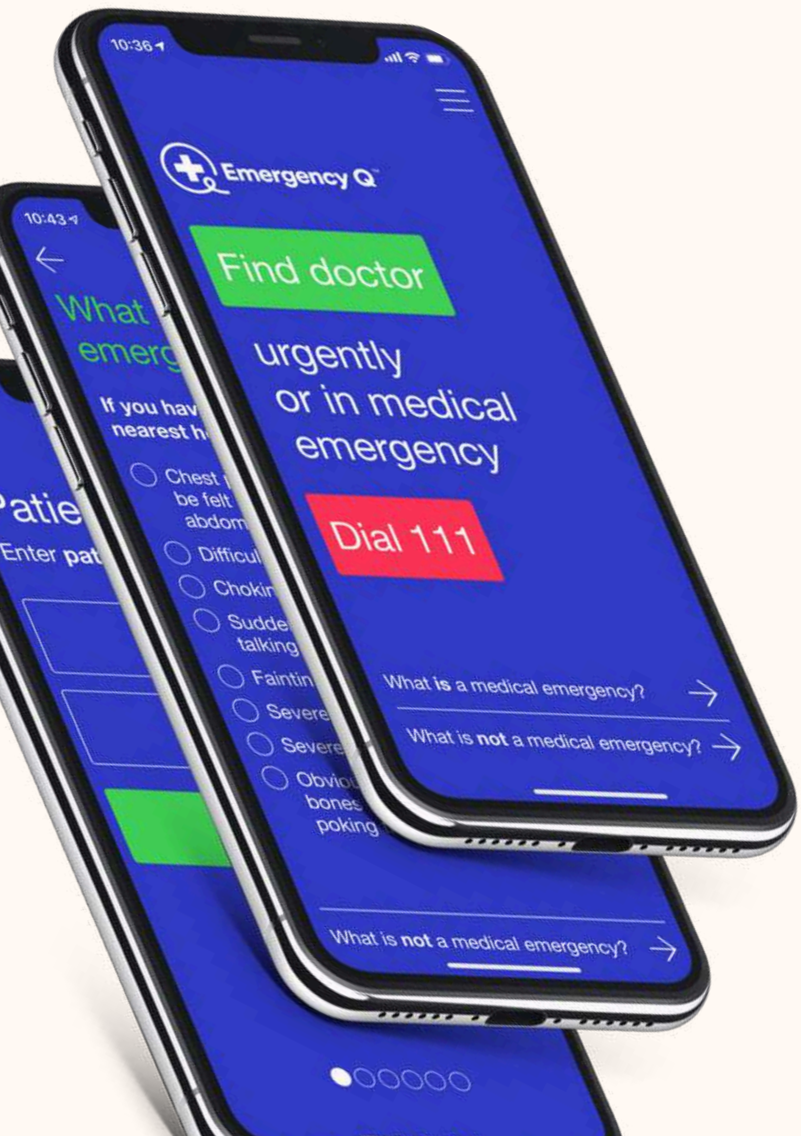
It's well worth attending events that attract past startup founders and hearing some of the war stories; every way that people were ripped off is somewhat unique, however here are the most common threads to stories I've heard.

Co-founder / investor problems.

These are probably the most common; often people's ideas on where things should be going diverge along the way and human elements of greed and desire for control can also come into play. A great example of this is the divergency between the original co-founders of Facebook, Eduardo Saverin and Mark Zuckerberg, as portrayed in the 2010 film, The Social Network. There is no easy answer to this; in general terms though it's worth considering early partner and shareholder compatibility with the same degree of seriousness as contemplating marriage. Ensuring that you're dealing with people who are open, reasonable and reciprocal is key to being able to resolve the differences that will arise.

Unscrupulous corporate sharks.

Often it's very alluring when a large, powerful corporation takes a keen interest in your product and tells you all the things they can do to make it a massive success. These situations can indeed be great opportunities, but don't be naive, no matter what bond you think you have with the people you are dealing with, the heart of the beast is that of a psychopath. Strong words? The point is that ultimately large corporations will make the decisions they deem to be in their own best interest and neither empathy, loyalty or even supposedly binding contracts will get in the way of that. Also, the ultimate decision makers may end up being people you have never even met. Often your IP, ideas, or unique knowledge turns out to be more valuable to a large corporation than whatever deal you officially entered into.

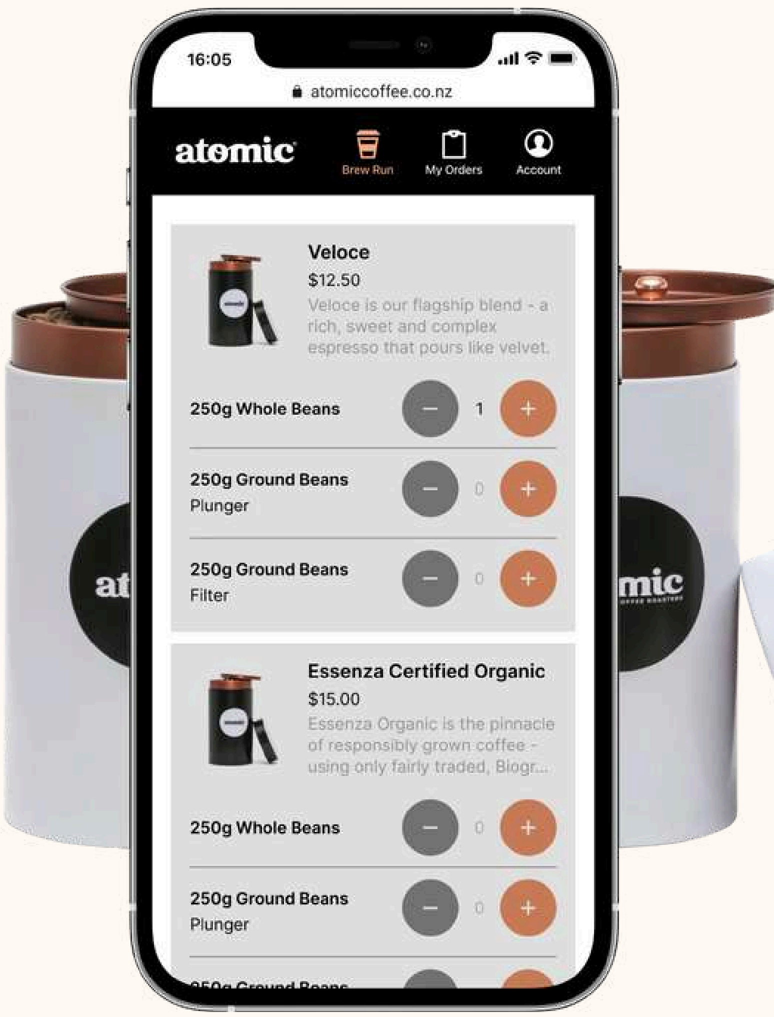

Emergency Q
Find doctor
urgently or in medical emergency
Dial 111
What is a medical emergency?
What is not a medical emergency?

Overbearing VC's

This comes back to the opening quote of this section, "when you take the money, your life is over". The very nature of the VC business model is that the successful investments need to produce returns that more than cover the unsuccessful investments; it's a high risk, high return, environment. This means that, to minimise chance of failure and maximise return, VC's will usually expect a lot from founders. Also, due in particular to bad experiences with some founders, VC's will usually want a high degree of oversight and accountability to be in place. The best thing you can do is to consciously choose whether or not you want to go down this path and, if you do, embrace all aspects of it as part and parcel of your choice. However, it may also be worthwhile to ask around the startup's community to get some insights into the nature of the particular VC's available to approach; just like any company the culture of VC's will vary greatly.

Loss of control.

As soon as you start taking other people's money, some loss of control is inevitable, and that can be OK if it's well managed. However, the problem comes when you end up diluting your shareholding to the point where you really don't have any control; at that point you are really just a CEO with a share package, and if you don't please your masters you can be fired like any other CEO. For some opportunities that consume capital for a long time this may be semi-inevitable; if it is, just ensure that your competence rises with the company such that investors want to keep you and want to listen to what you have to say.



Veloce
$12.50
Veloce is our flagship blend - a rich, sweet and complex espresso that pours like velvet.

250g Whole Beans 1
250g Ground Beans Plunger 0
250g Ground Beans Filter 0

Essenza Certified Organic
$15.00
Essenza Organic is the pinnacle of responsibly grown coffee - using only fairly traded, Biogr...

250g Whole Beans 0
250g Ground Beans Plunger 0

Government Grants and Co-Funding

These obviously vary from country to country, however, in the context of New Zealand, the main sources relevant to software businesses are:

- R&D tax credit scheme
- Callaghan grants and co-funding
- NZTE grants and other services

R&D tax credits are especially worth going for as you pretty much have a right to them as long as you can make the case that you're solving previously unanswered problems. They can also be applied retrospectively if you have carefully tracked costs and portions of staff time used. To prepare a good application usually requires the help of a consultant though, so this only tends to be worthwhile for six-figure projects upwards.

Criteria for other grants and co-funding will obviously change over time so it's best that you obtain fresh information, however in general they usually seem to require a reasonably mature operation to obtain. It's worth noting too that some of the Angel investor networks have relationships with Callaghan to obtain co-funding that adds to the amounts they're investing.

Also, when it comes to the various government and partner organizations that provide funding, it's good to think beyond just money itself; some offer other great services such as introduction to investors and potential overseas customers, helpful advice and relevant courses for product startups. My experience with NZTE in particular is that it felt a bit like a free incubator, however to my knowledge they do only deal with companies that already have seven-figure revenue.

Where your business meets eligibility criteria these government sources are great in that they don't dilute your shareholding; in most cases the way to look at it though is not so much either-or situation, but rather as a companion to other investment pathways.

Closing Thoughts

This eBook is only a light overview of the areas it covers, however I hope it has raised some things worthy of further consideration.

For anyone fairly new to software development, I also hope it has enabled you to take stock of the road ahead; it's not as easy as media stories of overnight success, or some fly-by-night agencies, often portray.

On the other hand, what I hope I haven't obscured is that software development is fun! Yes, if you're working with the right people it really is. There's nothing else I know that enables you to just dream and create like software and app development.

And when it does result in a new valuable product, software is also the most scalable business known to man. In the context of New Zealand, the media tends to focus on a few of the biggest players, but there are a surprising number of successful SaaS companies that fly below the radar. One reason they get very little media attention is that many of them have created a 'best in the world' product within a narrow niche, but then scaled globally within that niche to the extent that they're not actually very focused on the NZ market.

If you do choose to be a software product founder, I believe you're taking a road that will thrill, challenge and stretch you. I'd encourage being a student of the process and appreciating the growth it brings. And I'd also encourage you to surround yourself with capable people; the number one reflection from my journey so far is that successful technology delivery is all about great people. Sorry AI's, you're just not there yet; maybe one day, but that day is not today.

As mentioned in the introduction, please do feel free to reach out to me: john@puttiapps.com or via https://www.linkedin.com/in/john-halvorsen-jones/