FROM HERE TO THERE:

A GUIDE TO CREATING DISRUPTIVE SOFTWARE

FROM HERE TO THERE

A GUIDE TO CREATING DISRUPTIVE SOFTWARE IN ENTERPRISE

As a mature organisation who has had success in a traditional business model over the years, you see a need to evolve with the ever-changing demands of customers through technology. Furthermore, technology is able to unlock hidden value within the business, and present new potential offerings and revenue streams. By using abductive thinking, with a focus on design, carefully understanding customers, and evaluating the strengths of the business, novel solutions surface which often serve as key differentiators for the organisation. People and technology are crucial components to creating disruptive software while iteratively improving and learning. The journey starts now.

WE WILL EMBARK ON A JOURNEY

1. Vision: To create a truly differentiating offering or experience, you need a bold goal.

2. Business: People and the organisation of a business are critical to make things happen.

3. Process: Build disruptive software with cooperative teams. Without structure and ways of work, chaos ensues.

4. Technology: Selecting the right technology and technical approach can impact the quality of a solution, its integration with the business, and time to market.

5. Automation: Do meaningful work fast and reliably. Automate repetitive tasks for robust quality assurance and a reliable digital offering.

6. Measurement: Software is never done, and software must serve the business. Through measuring key indicators, software can be iteratively improved based on use and demand.

CHAPTER 1

CREATING A VISION

It all starts with a vision. Often business-as-usual includes putting out fires, making sure the small things are happening, and meeting short-term targets. For an organisation to truly be disruptive, it needs an overarching vision of where it wants to be – no matter how ambitious it might be. Without an ambitious vision, organisations tend to wither away over time. Let's explore the facets that can guide defining a vision.

01 WE ARE ALL PEOPLE, WE ARE ALL USERS

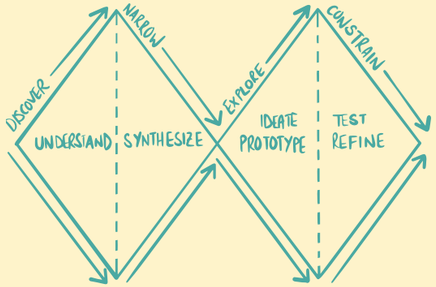
As users, we all know what it feels like to get frustrated with technology, but we also know the immense joy and empowerment it can bring to our lives when a solution is done right. We have apps on our phone that we simply cannot live without, other apps that are deleted within 3 seconds of downloading, and apps that are simply relegated to the app graveyard.

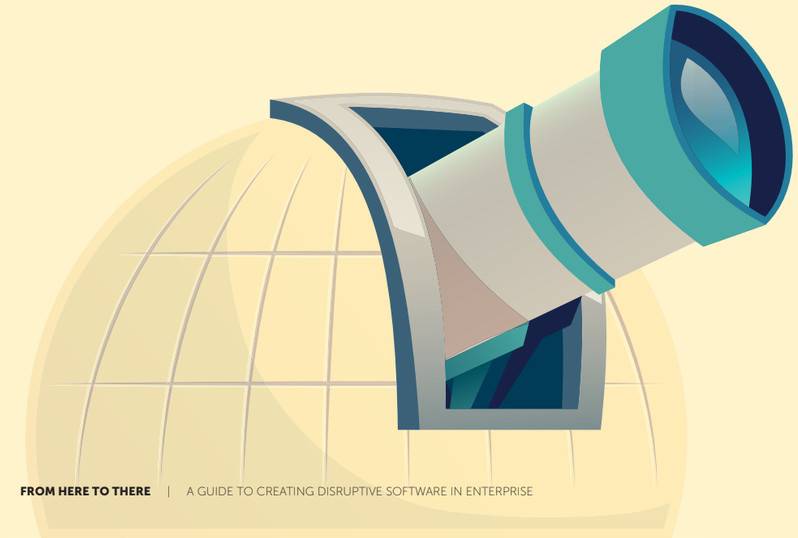
As people, we don't care about features, we care about accomplishing a goal. It might be communicating with loved ones, managing finances, or sharing your memories. The technical bits that make these things happen mean nothing.

The key to the success of a digital solution is adoption by users and that can only be enabled through empathy for all involved, using robust qualitative research (research that's more opinion) and quantitative research (research that's more empirical).

02 HOW IDEAS ARE SHAPED

The life of a solution begins with an idea; the business recognises a challenge and feels it may be solved through technology. Business stakeholders may already have a clear idea of what form the solution should take, what type of tech to use and what the eventual outcome may look like. The important factors often forgotten are seeing the solution as a vehicle for strategic innovation by empathising with the people actually involved in the trenches and being open-minded about who the customers are, what their goals are, and how these ideas can address those issues while providing business value.

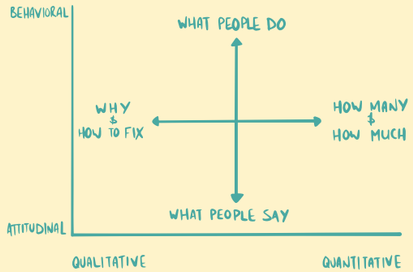




03 THE ROLE OF UX ENGINEERING: UNDERSTANDING

UX engineering means to be involved with business and technical stakeholders from the very beginning to understand the potential constraints and opportunities for any potential solution. This can take many forms, such as stakeholder interviews or focus groups, heuristic evaluation (expert assessments of existing systems) and analysis of existing user data, and business or technical processes. This will allow the construction of a broad framework to direct research and design goals. Without this step, the solution is at risk of being designed solely for the assumed requirements and not the ecosystem in which the solution must live.

While a thorough understanding of the business and technology is vital, it should not affect the necessary objectivity or empathy required when engaging with the users or potential users of the solution. To retain objectivity and guide a user-centred design approach, we must employ rigorous research methods ranging from large quantitative data-gathering to smaller, more targeted qualitative exercises. The type of research methods selected are as important as the research itself and the project should have a strategic UX plan to guide that process. The next diagram shows where different research approaches fit within the UX research landscape.



There are many considerations when choosing which research method to use and the following should be considered at the start of every project:

- What is this research method typically used for?
- What will implementing this method cost?
- How difficult will it be to collect the data?
- How difficult will it be to analyse the data?
- What type of research category does this fall into?
- What is the context of use of the method?

In addition to classic UX research methods, there are many new ways in which UX engineers engage with both stakeholders and users. One that has gained traction over recent years is Design Sprints. Design Sprints were created to condense and target various UX methods such as stakeholder engagement and user testing into a fast and furious 5 days. More on this coming up.

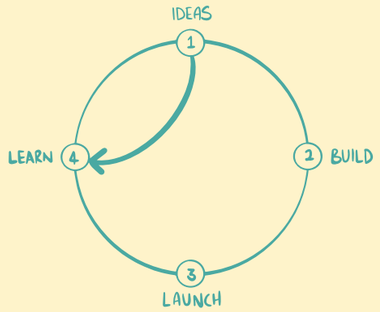
04 THE ROLE OF UX ENGINEERING: INTERPRETING AND EXTERNALISING

The activities and artefacts generated through UX engineering include valuable information from different dimensions of the business and project. These finding are invaluable references for making critical decisions about features and direction.

- Contextual inquiries: Data-gathering technique combing observational research and semi-structured interviews in the user's context of use
- One on one interviews and/or focus group transcripts and analysis
- Thematic analysis: Ways to organise research gathered, such as interviews or observations into useful and consistent themes
- Personas: Consolidating research on users into imagined users to represent a larger set of data
- User journeys: Mapping the entire journey that a user/customer takes during their use of a product or service, which aids in understanding sentiment, needs and goals

05 DESIGN SPRINT: PARTICIPATORY DESIGN

A Design Sprint is an excellent tool in the case where the business needs to be involved in ideation. As mentioned previously, this is a 5-day design process including all those that will be involved in making the project a success. Prior to beginning the sprint, those involved must have consumed the research gathered in the previous phase of the design lifecycle and now have solid grasp of their users.



DAY ONE

We define our goals both for the business and the users. We then define the actors involved in the business problem and then map their journey towards the defined goals. We will also come up with questions and ideas through a process called "How Might We" (HMW). This involves crowd sourcing thoughts in a structured way to provide detailed context about a process. Additionally, it highlights areas of interest through a voting process.

DAY TWO

We'll begin sketching out possible solutions to the problem. There are many methods we use to encourage creativity and allow stakeholders to explore their ability to ideate, free of constraints. We will then display the works, present them and vote on them.

DAY THREE

We'll do the final round of voting and then begin storyboarding the solution using the best parts of each idea or sketch. As you can see, we start with individuals working collectively to form individual ideas, that are then voted on and consolidated into a single vision – this is why the eventual storyboard is so powerful.

DAY FOUR

On the fourth day, our design team prototypes the solution. The other stakeholders may be involved if they so desire or if it is necessary to the goal, but it is often just the design team that manages this part of the process. We design a working prototype to test with real users the next day.

DAY FIVE

On the fifth day, we venture out into real-world operations where the solution may be used. User testing is conducted and provides an opportunity to learn and validate solutions. This gives us crucial insights into the viability of a solution or part of a solution which can then be reported back to the stakeholders. In this way, we can know what works and what doesn't with little time wasted if the idea fails and much gained if it is a success.

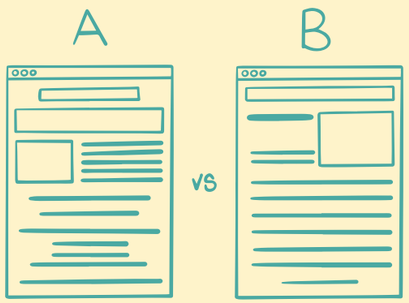
Prototyping and testing are however more involved processes for the solution as a whole. So, while the Design Sprint offers a way to tackle innovation and collaboration on specific areas of a solution, dedicated and extensive prototyping and testing is still required.

06 TEST AND REFINE

The power of the prototype is that one can create a testable design in significantly less time than it takes to develop a solution in code. According to Dr Susan Weinschenk in her paper *Usability: A Business Case*, 50% of development time can be saved by doing UX research upfront. Testing can take many forms, much like the user research mentioned above. Some of the same techniques we saw in the user research phase can and will be repeated here, for example:

- Usability-lab studies: Controlled and observed test of users, using your prototype.
- Eye tracking: Users eye movements are tracked while using a prototype, to understand focus areas and how to optimise the design.
- A/B testing: Testing variations of your prototype or elements of it, to see which designs users favour.

Based on the testing done on the prototypes, the UX engineer can continue to refine the designs until they have been optimised. The reporting done during the prototype testing is again hugely valuable for the understanding of all stakeholders as to why certain design decisions have been taken. In fact, it is ideal if the product owner and team attend some of the user tests, as this engenders empathy and understanding directly.



07 THE RECURSIVE VALUE OF UX

If a UX engineer is included in the process, it will have continuous value to all those involved in the project later down the line. The result should be a set of validated and tested prototypes that consolidate all stakeholders needs into a design vision. It should also enable decision makers, architects, developers and analysts to add value to the solution in new and innovative ways, driven by a robust body of research.

CHAPTER 2

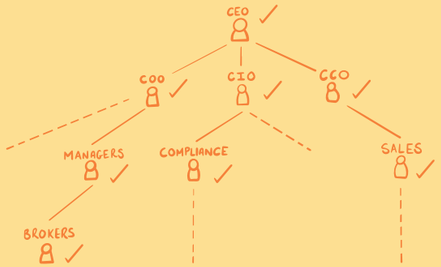
THE **BUSINESS PERSPECTIVE**: AMBITION AND DIFFERENTIATION

As outlined in the previous chapter, the life of a disruptive solution in an established enterprise should always start with a vision. The vision for where an organisation wants to be, what it looks like, and what it offers in future is where innovation happens, as opposed to putting out fires and only addressing the problems that exist now. The business aspect in the lifecycle is critical to the success of the solution. Ultimately, the solution is simply a tool to facilitate the ambitions of the business and possibly differentiate from competitors. This chapter looks at the important facets in a successful solution regarding alignment, evaluating value, mitigating risk, assigning accountability, and understanding the operations of the business.

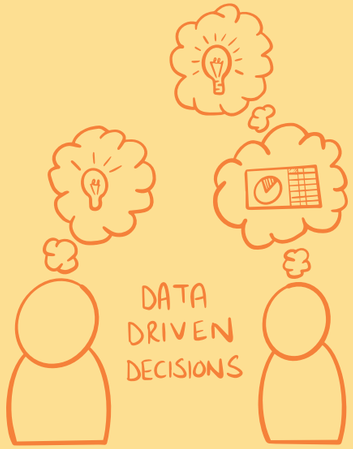
01 ALIGNMENT: STAKEHOLDER BUY-IN AND ALIGNMENT

After the process of exploring and defining a vision - which stakeholders from different areas in the business should be part of - a consensus must be agreed upon by the stakeholders. Key decision-makers, including executives, managers, and operations, must have an understanding of exactly what is trying to be achieved and how it will impact their respective areas. The proposed solution should be well understood and in alignment with existing initiatives and tactics, or at very least, not directly hinder them.

Innovative solutions usually challenge the status quo which causes uneasiness among stakeholders, and challenges the existing tried and trusted ways of doing things. However, if the need for change is embraced, driven top-down and transparent, then the energy and willingness of teams are boosted, which usually reduces friction.



02 VALUE: DATA-DRIVEN DECISIONS



It tends to be easy to make decisions based on gut-feel and push your ideas because, well, they're *your* ideas – you want to add value and be recognised for doing so. But this becomes dangerous when an idea, or at least the core mechanics of the idea, have not been validated.

It can't be disputed that people with years of experience in an industry or business understand underlying trends and invisible consequences of actions, but that intuition is based on the past. Businesses have thrived via traditional models while making small tweaks to react to the market and demands of the consumer. But rapid advances in technology require one to take a step back, and apply knowledge and experience while taking some risk in trying something unheard of.

This risk can be mitigated by making decisions based on data. By utilising data from sales, retention, behaviour and usage, operations, and the respective costs for those areas, decisions can be based on facts rather than a hunch. Analysing competitor performance where they have adopted similar approaches is also useful in validating the viability of a solution. Although these techniques do not eliminate risk completely, they provide guidance and informs decisions going forward.

03 COST VS VALUE

One of the primary considerations for embarking on an innovation project is the cost versus value to execute. What is the return on investment? When will it start making a difference? As with anything, a return on investment, especially a big one, is not guaranteed. However, there are different approaches to consider that can reduce costs, and ease sceptical minds. These approaches include design sprints where ideas are fleshed out, prototyped, tested, and reported on during the week before a large project is undertaken.

Another approach is minimum viable product (MVP) based development. The controversy – MVP is often misunderstood as "what we have plus these things". An MVP should simply contain the "what we have", or the "plus these things", or do something different. Migrating existing features to a new technology is not an "innovation" project; it is good practice if a legacy system is no longer viable. Sure, this might enable future innovation, but we're talking about truly doing something differentiating.

The most successful approach is to embrace change and accept failure. To accomplish this, you commission a team to work on the new project, but priorities are decided on at each sprint, or each month. If they need improvement, great – iterate. If they don't work, throw them away. If the entire idea is doomed, disassemble the team. The cost of running a team on a month-to-month basis while testing ideas and reacting to them is always cheaper than commissioning a team for a long period of time, with a constrained specification and no room to adapt to the ever-changing business and technology landscape. Every cost that yields learning is valuable – the trick is to strike a balance. Cost of learning must be accepted, however, learning and never leveraging it for business value can be detrimental.

04 RISK: CHANGE MANAGEMENT AND SERVICE DESIGN

Apart from the cost of development, the biggest risks in these projects are more related to change management or service design. The individuals in the organisation have learnt the lay of the land, and it is human nature to resist change.

The management of training, education, and roll-out with everyone on the same page is a difficult feat if not enough attention is paid to it. This is true for customers as well. Organisations often change their offerings and receive backlash from customers even though the new solution is objectively better – humans don't like change. Strategic thinking and a pragmatic approach are needed.

05 RISK: CONSISTENCY OVER BIG BANG

Another risk is in setting immovable deadlines for releases of big chunks of the solution. This usually ends in a sub-par experience, poor technical quality, or unmet expectations.

An approach to mitigate this is to release smaller iterations faster, test the behaviour and impact, refine, and repeat for every aspect of the solution.

In implementation, this can make expectations difficult to manage – but if it was easy, everyone would be a world-class, bleeding edge enterprise. More often than not, consistency is more sustainable and successful in the long run over big bang releases.

06 ACCOUNTABILITY: DECISION MAKING

Although there should be buy-in and alignment among a variety of stakeholders, there must be an individual owner and decider. This person should have the strategic insight of the organisation, have autonomy in making critical decisions, and influence change. This individual must lean on others for different insights and direction, but ultimately, projects that rely on many people for approvals typically create blockers, lose momentum, and drain energy due to bureaucracy. Making decisions as a committee is almost always nuanced causing long wait times or poor outcomes due to a single individual not being invested and held accountable.

The decision-maker must be available and decisive. Do you need to allocate office space for this special project? This person should be able to make the call. Need to provision an account for a specific technology or platform? This person should be able to make the call. This individual is ultimately responsible and accountable for the solution and people involved in making it happen. Furthermore, the individuals that comprise the team should have autonomy and use their discretion for micro decisions that need to be made on a daily and hourly basis. By embracing this, measuring solution performance, and being open to iterate and fail, magic happens.

07 PRIORITY AND COMMUNICATION

Friction in priority and communication are common themes in innovation projects. Everyone is excited and looking for how they can leverage the initiative. Having a single solution provide direct value to multiple stakeholders in different departments within the organisation can be problematic. It is obvious that each stakeholder will look out for their own department first and negotiate for their priorities. A strong owner must be able to understand the needs of the different stakeholders, prioritise, and communicate what will happen, when it will happen, and why that decision is preferable.

08 BUSINESS ANALYSIS: UNDERSTAND OPERATIONS AT ALL LEVELS

It is important to understand the operations of the business from all levels. For example, what are the economies within the business, what is the average customer acquisition cost, what is the process followed in branches, what daily planning happens for different teams. A general understanding of these facets reveals hidden mechanisms which mould the solution, helps address problems before they happen, and creates champions for progression in the employee structure if they are involved and heard from the beginning.

After looking at the processes that exist in the business, hidden business rules, exception cases, and ways to handle them emerge. This understanding allows for consolidation of business rules, simplification, and leaner operations. These rules will also be changed and/or developed into the new solution being developed. With more unknowns comes more complexity, cases to cater for, and uncertainty.

Although embracing iterative development, failure, and learning is good, this does not negate the need for a concise and understandable functional specification. This could be a document, but more preferably, user stories that both business and technology teams can understand. The use of collaborative tools to track stories and share details about them is invaluable to productivity. These can manifest in story boards, wikis, and other tools that improve the productivity of everyone – do what works for the people.

09 IT AND TECHNOLOGY

There is a chapter about technology later, but at this point in the journey, it is critical to involve and build rapport with the existing IT teams. Generally, IT divisions have been a function within the company that took care of the infrastructure to enable individuals within the organisation. This included overarching concerns such as provisioning of computers, managing networks, managing software licences, and sometimes developing solutions.

In this day and age, IT is part of almost every business function. IT skills and decision makers are mandated to build and maintain technology. For this to be successful, business and technology must work together and cannot be separated concerns anymore. Involving IT teams in at the beginning creates ownership, buy-in, and energy to work together rather than receiving tasks with little context over a wall. Understanding the existing IT landscape is useful in planning ahead. Existing adopted technology should not stifle the technology decisions for the solution, but planning for how it can be incorporated as seamlessly as possible is a real need which is better embarked on earlier rather than later.

Ultimately, the success of any solution, whether it involves technology or not, is centred around people. We need to facilitate communication and alignment in understanding, mitigate risk, forecast value, and learn from what we do. With these fundamental principles, solving hard problems becomes easier.

CHAPTER 3

QUALITY AND CERTAINTY RELY ON A **PROCESS**

Any engineering discipline requires a reliable process to produce solutions or products that bring value to an organisation. The chosen process should help teams to deliver a quality solution or product within a predictable timeline and at a sustainable pace. In software, this process is called the Software Development Life Cycle (SDLC) which has process elements including requirements gathering, design, implementation, testing, deployment and maintenance. Various nuances of these process elements exist and are applied differently depending on the environment the team find themselves in.

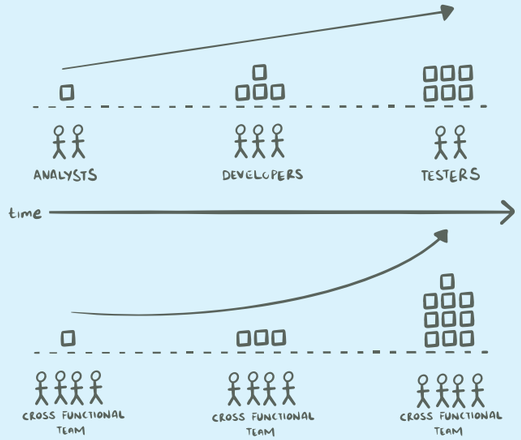
01 AGILE MEANS BEING ADAPTABLE

Software teams struggle when the principles and values of the chosen process are rooted in traditional manufacturing and engineering disciplines. The software engineering profession is fundamentally different in several ways which causes traditional processes to fail.

Traditional planning is centred around predictable tasks or activities with known manufacturing times while software projects usually start with an end goal and the tasks or activities are discovered and figured out along the way.

Progress is not linear to time because in problem solving, people generally start to work faster as soon as the problem is solved. In traditional manufacturing, the throughput rate is usually consistent and linear to time.

Adding more people or resources to a problem does not provide immediate increase in productivity and output. In fact, productivity will decrease while new people get up to speed on the problem space before it increases for the team as whole. In addition, the increased output is not a multiple of the number of people being added to the team as not all people have the same skill level.



In the following sections we will look at some key considerations for implementing a successful software development process. Whether your organisation calls it Agile, Waterfall or any hybrid SDLC in between those two extremes, we advocate to focus on getting four key considerations right for the development process. They are called considerations because this gives organisations the freedom to make it work in their context instead of feeling like they are failing because they do not follow the Agile or Waterfall textbook.

02 SOFTWARE DEVELOPMENT LIFECYCLES

The first consideration is demystifying the noise around the SDLC and whether teams are following an Agile approach. The reality is that Agile is not a process, but rather a set of values and principles that a team lives and applies daily. This gives the team the freedom to follow any process or methodology they like while pursuing continuous improvement in iterative cycles. The Agile manifesto proposes the following values for an adaptive team and development lifecycle:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

If we evaluate the manifesto from a process point of view, it is clear that whatever methodology you choose, make sure you have regular interactions with all stakeholders in the project. The process should make time in everyone's calendars to talk and discuss the project. The goal is to have effective conversations about the project on a regular basis and we have found that co-location and face-to-face discussions is the preferred approach. The best alternative to co-location and face-to-face discussions is to have video conferencing combined with an instant messaging platform geared towards group discussions. When working with a distributed team, the most effective approach is when everyone is remote. A co-located group with a minority of individuals remoting in tends to treat the remote members as second-class citizens.

The process must enable the delivery of working software instead of creating documents which do not solve business problems. Documentation must have an audience and only describe concepts which do not often change, e.g. project vision and purpose, logical system architecture, and business rules and key decisions made during the project that have a significant impact. Specifying features in detail will often not be accurate and maintaining the documentation becomes cumbersome and adds little value.

Economic factors will play a role in customer collaboration, and so contract negotiation cannot be ignored. Projects do not have unlimited budgets and need to be delivered within the economic constraints of the business. The principle here is that if customers are part of the development process, then the solution or project will evolve and change with the team. The scope negotiated at the start of the project should not be enforced to the letter if the final delivered solution still solves the business problem, which is typically not the same as originally envisaged at inception. Being clear on the project vision and purpose at the start of the journey will aid the team to make decisions as the project evolves and test their decisions against this vision and purpose.

Regular planning sessions enable the team to respond to change rather than creating a plan up front and sticking to it no matter what. Different levels of planning from a high-level flight plan, to a sprint plan broken down into tasks all contribute to keeping the team focussed on what must happen next while keeping an eye on the overall project progress. Scope creep or changes in direction need to be embraced but must be done in a responsible way. Changes in planning should be done between iterations to keep the team focussed during the iteration. Having a high-level flight plan coupled with a detailed sprint plan helps the team to determine the impact of scope changes to the overall project scope. The stakeholders must be involved in the planning sessions to make decisions if plans need to be adjusted.

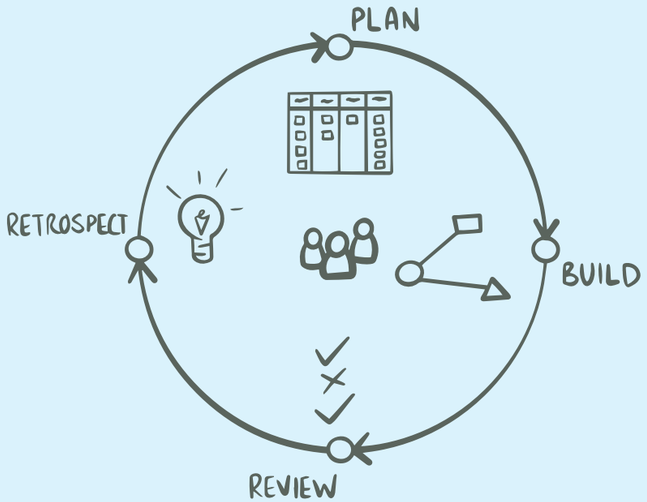
03 CROSS-FUNCTIONAL TEAMS

Other considerations for software development processes include skills and capability, prioritisation, and participation and transparency. Any process is only as good as the people who execute it, and making sure that a team of individuals have the right skill combinations for the problem space, is how we build self-organising teams. This requires people to have analysis, development, testing, data and project management skills within the team. It does not mean that one person is responsible for each of these disciplines but rather the team needs these skills spread across individuals.

Large organisations tend to employ people to fulfil a specific responsibility in the SDLC, e.g. only do analysis, development, testing or project management. This approach will inevitably lead to silos and bottle necks within the organisation, coupled with a large management overhead.

The size of a team is also a factor which is considered. Having two teams of 5 is usually more effective than having one team of 10. Quality practices like pull requests, code reviews and interpersonal communication are all affected by the size of a team. A truly self-organising team requires a high level of awareness within the team and this becomes challenging in a large team.

Prioritisation and clarity, or a shared understanding of what needs to be built, is the cornerstone of effective software teams. A self-organising team is about trusting the team to figure out how to solve a problem. It is not about letting the team figure out business priority and what needs to be solved. The process for delivering good software ensures that a shared understanding is created between the business stakeholders and development team, which is why a good project vision is key. Techniques like joint application design (JAD) sessions, event storming and user journeys/storyboards are all powerful ways to create a shared understanding.



To effectively prioritise, the team and decision makers need to gain insight into the value vs effort, as well as dependencies to deliver. Clear and concise acceptance criteria for tasks or user stories will enable the team to estimate with a higher level of confidence and point out any dependencies which need to be resolved before development can start. Creating a backlog of user stories or features with clear acceptance criteria and associated effort estimations, enables decision-makers to weigh up different options. This allows them to decide on what the team needs to work on for the next period. This can only happen if the decision-makers are available for the relevant ceremonies conducted in the team for project planning and discussions. Context, continuity and authority to make decisions in the business make for successful projects. When decision making is deferred to a group of people who are not involved in the project on a regular basis, it usually results in rework, missed deadlines and demotivated teams.

04

The final consideration for a good process is participation and transparency from all stakeholders. People enjoy ownership of delivery if they are part of the planning and decision-making process. Making commitments on behalf of the delivery team leaves them feeling like resources and demotivated. A process that enables open feedback with regular demos and a safe environment which values the ideas from all stakeholders lead to a more refined solution. End users will have a better experience using the solution and focus will be given to high value features.

Here are some additional reading links:

- Agile Manifesto
- Agile Principles
- Event Storming

CHAPTER 4

TECHNOLOGY: UNDERSTANDING THE DIGITAL LANDSCAPE

Software plays a role in almost every part of every business. Technical teams are responsible for understanding the digital landscape within the organisation and outside of it. This includes selecting the most sensible tools and frameworks to solve problems, and designing software using best practices to enable maintainable solutions that can evolve going forward.

In order to make system design decisions, some level of understanding is needed of what the envisioned functional and non-functional requirements are. The following questions need to be answered:

- What is the problem we are trying to solve?
- Who/what will use it?
- What does the existing technology landscape look like and what integrations are needed?
- How must it scale and perform?
- Who is going to develop and maintain it?
- What technical skills are available?

Taking the answers to these questions into account, we will be able to make critical decisions like:

- Where the solution will be deployed and hosted?
- Which technologies will be used to build the solution?
- What the plan is for scaling up under load?
- What security measures are needed?
- How does the solution fit into the greater technology ecosystem?

There are several technology and deployment stacks available today, each with their own advantages and disadvantages. There are differences to what each technology is good at and the ease with which they can be used to implement different types of solutions, but often the deciding factor is what skills are available in the team who is to develop and maintain it and where the team is going strategically. As new technologies come out frequently one does want to be a little conservative: one should make sure any proposed technology has a proven track record with regards to scalability and maintainability.

01 ARCHITECTURE

When designing a solution there are many technical architecture considerations known as non-functional requirements that need to be considered. Even if most of these requirements are not implemented at the start, it is essential to keep them in mind from the beginning and have a clear understanding of how provision will be made when the time comes to cater for them. This is needed because these types of requirements typically take much more effort to implement later in the life of the solution if no consideration was given to them early on. Some examples are mentioned on the next page.

SCALABILITY

This is how system resources are added to handle a growing amount of work, e.g. adding memory to the database host or adding another node on a cluster. Some applications (especially legacy applications) can only be scaled vertically. This means more resources like memory or CPU power must be added to a single server in order to get better performance. However this has physical limits, and costs increase faster than the improvement in performance. It is preferable for an application to allow horizontal scaling to allow adding more average powered nodes to a cluster for better performance. This is typically more cost-effective and has many secondary benefits, such as better availability and reliability.

THROUGHPUT

The rate at which the system must be able to handle requests, e.g. the number of transactions per second the system must cater for.

PERFORMANCE

The amount of useful work accomplished by the system with a given amount of resources, e.g. having a low response time or low resource utilisation.

VOLUMES

How much data the system must be able to handle, e.g. storing two million records per year or handle 1MB file imports.

SECURITY

The resilience the system has against potential harm, unwanted changes and data leaks. No system is 100% secure. The key business risks need to be understood for appropriate technical solutions to be implemented. For instance, proper web security and encryption are sufficient for most public facing web applications. However when money is involved, e.g. with a bank website, extra levels of authentication, encryption, monitoring and auditing are required. Threat modelling is a technique for identifying, preventing and mitigating threats in an optimised way by determining where the most effort should be applied as the system and external factors change.

RELIABILITY

The probability that the system behaves as designed. No system is 100% reliable, so it needs to be understood what the risk to the business is if a single transaction fails due to system failure. The envisaged design needs to take this into account and find solutions to have the desired reliability, e.g. the system can automatically retry certain steps that are susceptible to network timeouts.

AVAILABILITY

The proportion of time a system is in a functioning condition. No system is 100% available so the business risks related to any downtime needs to be understood and catered for. If the business loses money for every second the system is down, a high availability like 99.99% up-time may be the service level agreement, thus requiring many levels of redundancy in the system.

MAINTAINABILITY

The ease with which defects can be fixed or new features added to the system. Short-lived research applications might not have the same maintainability requirements as a core business application that must last more than 10 years. This is a balancing act of spending extra effort to make the system more maintainable versus using the time to add new features faster. Developers with different personalities types tend to have wildly different natural biases in this regard, so the vision of the business must be clear so that good decisions can be made by the team.

INTERNATIONALISATION

Should the system be able to be adapted to various languages and regions without engineering changes? If a front-end does not cater for this from the start, it can be a lot of effort to add this later in the lifecycle.

02 FRONT-END

Most modern applications are mainly accessed via a web front-end or a mobile application. Unless there are specific business requirements, it usually makes sense to start off on a responsive web application that works on different devices – the userbase should determine how much effort to put into making it scale well to mobile devices. This is highly dependent on the context of the problem and the users involved.

A public facing application will almost always need to function well on both mobile and desktop environments, whereas a business facing application might only ever be accessed on a specific platform.

WEB

Depending on the business requirements, there are several options for making the system available online. Informative websites typically only contain static web content which is fairly simple to host on premises (low-medium volume) or in the cloud. For high volume static content, a content delivery network (CDN) might be used which spreads the load across the globe.

There are different ways of implementing web applications that support user interaction. Initially web applications were built using frameworks that rendered the final HTML on the server but this does not scale well. Now we mostly host a single static web page containing a JavaScript library (SPA) on the server and let the final HTML rendering happen on the client browser. A Bootstrap page is loaded first and any additional resources are retrieved as needed. Data access is achieved using data exchange paradigms like REST which is more efficient than transferring fully rendered web pages.

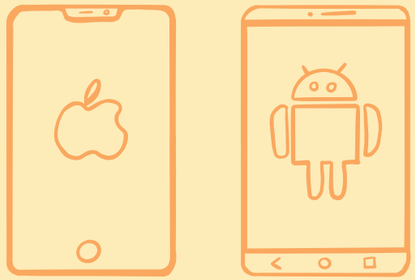
In order to provide a native-feeling application via the web browser one can implement a progressive web application (PWA). They are built using common web technologies including HTML, CSS and JavaScript so they are intended to work on any platform that uses a modern standards-compliant browser. Functionality that a normal web application doesn't typically provide but that native mobile or desktop apps do provide, can be obtained when implementing a PWA, e.g. working offline, push notifications, device hardware access, enabling creating user experiences similar to native applications on desktop and mobile devices.

MOBILE

In order to provide a mobile presence, one can implement multiple native applications using the platform's native tools. This is typically the best user experience for rich applications that need a lot of device interaction and offline capabilities. However, since a completely different application needs to be implemented for each mobile platform (Android, iOS, etc.), it can be costly and finding specialised developers could be a challenge.

There are alternative options like using a cross-platform framework where one application is developed that works on multiple mobile platforms with a lot less effort than supporting multiple native apps. The frameworks are not perfect as there are limitations and potentially a slightly less-than-native feel to the application, but depending on the business requirement this could be an option.

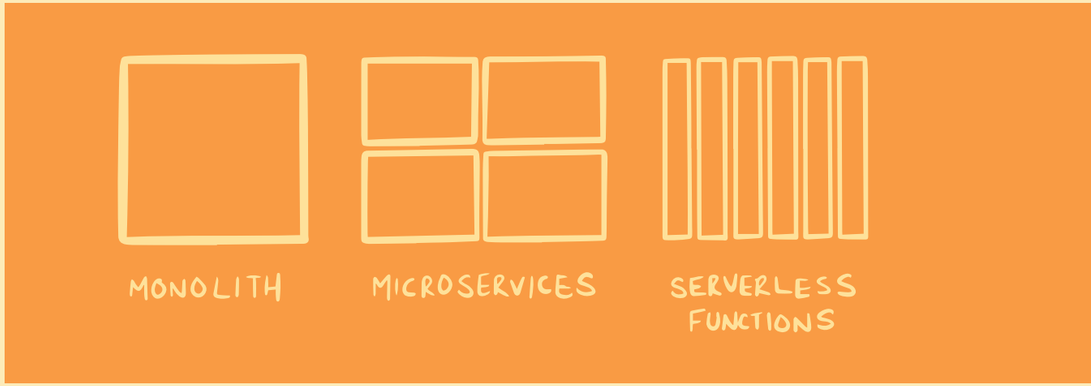
As mentioned in the web section, PWAs are also an option with even less specialist skill requirements and effort, but with more limitations.



03 BACK-END

As applications have grown larger and more complex over time, with higher demands on scalability and availability, application architectures have adapted accordingly to support them. Instead of building a monolithic application that is hard to scale out and maintain, applications are typically broken up into several smaller applications or micro services – each with its own focus, maintenance lifecycle and non-functional requirements (scalability, availability, etc.) .

Splitting up applications into small pieces helps teams to understand what each service does without keeping the complexity of the entire application in mind. This requires deployments to be automated, services to be automatically discoverable and connections routed accordingly. Asynchronous communications help the system to cope better with spikes and overall stability.



As back-end systems are broken up into smaller pieces that can be maintained by separate teams, it can make sense to also split up the front-ends. This ensures that each team is only responsible for maintaining their little piece and allowing different maintenance and framework upgrade cycles – which can be difficult in a big front-end application. There is typically an overarching application that stitches together the individual pieces.

Historically all applications were hosted in private data centres that had to be maintained by teams of specialised system administrators. Cloud computing seeks to commoditise computing infrastructure by implementing large scale data centres that can be optimised and provide highly reliable and scalable infrastructure with less effort to get set up and maintain. Most hosting is quickly moving to the cloud, as it is more cost effective and reliable in most cases. However, many businesses still use physical and virtual servers. Private cloud hosting can be an option, but one would still need a team that can support it.

To be cost effective in the cloud requires that the application be designed to work efficiently in the cloud. The traditional mindset of having a virtual server that you maintain manually in the cloud will be more costly. With the 'serverless' model, the application pieces are hosted on shared infrastructure that is highly tuned for each type of workload. That way the business just pays for each usage of the system instead of renting fixed hardware that sits idle most of the time.

04 PERSISTENCE

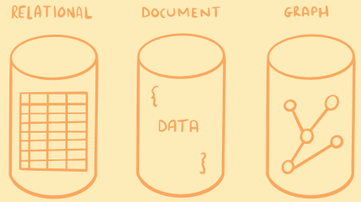
This is an area where incorrect decisions can have a long-term impact, beyond the life of the solution itself. Data stores tend to outlive the application itself and have value to the business outside of just what the application it was built for does.

There are some important questions to ask when choosing a persistence layer:

- What is being stored, and for what purpose?
- Is the data heavily numeric with a requirement to perform complicated calculations on that data?
- Is it high-volume event data from IoT devices where the individual events are less important than trends and anomalies?
- Is it unstructured, where there's no defined or expected schema, or where the schema is expected to change frequently?

TYPES OF PERSISTENCE

There are several types of databases available today. The relational database - historically the most popular database type - is joined by document databases, graph databases, key-value stores and more. The choice of database type should be guided by the type of data being stored, not by what is popular in the media. A poor decision here can have major repercussions throughout the application's development and usage.



PICKING A SOLUTION

The type of data being stored is not the only consideration. Consistency and data security are two other important aspects that must be taken into account before deciding on a database platform.

Consistency is about the behaviour of the system under concurrent reads and writes. If three processes write to the database and then a fourth one reads, is it important that the fourth process sees all three of the writes? Is it going to break anything in the application if it only sees the first two? Or if it only sees the third write? Typically, relational database engines only support strong consistency, meaning that in the example the fourth process would always see all three writes, while non-relational databases support eventual consistency, meaning the fourth process could see any combination of the writes, including none of them. There are exceptions however, with some modern database systems offering selectable consistency models.

Data security has always been important, but the increasing frequency of data breaches and the existence of laws such as GDPR have brought it into focus. This isn't just a matter of whether the database engine has any known security vulnerabilities. It's also about auditing, encryption, granularity of permissions and a lot more.

Is there sensitive data in the database that needs to be stored encrypted? If so, and the database chosen does not support such encryption, then the application development gains extra complexity as the encryption would have to be implemented within the application. Depending on the requirements, it may also be worth investigating how and where keys can be stored, what kind of work would be required to rotate keys, and whether the encryption protects the data against the administrators of the servers or not.

Does access to certain types of data need to be tracked and audited? Do changes need to be audited? Do schema changes, in the case of relational databases, need to be audited? These are often not obvious requirements when the initial application requirements are detailed, but they are critically important nonetheless.

Once all the technical requirements are out of the way, there's also the question of existing skills and investments. This isn't just a matter of what the development team is familiar with, but also what your operations teams are familiar with. Training is costly and time-consuming, so introducing a completely new database engine may not be desirable. This goes for infrastructure and cloud investments as well. What you have in terms of infrastructure and where they're investing in a cloud footprint (or if you're staying completely on-prem) can also affect the choice of database engine used for the application.

05 BUSINESS INTELLIGENCE

Today businesses collect large amounts of data and information that can be used to derive real-world value. This is what business intelligence (BI) excels at, as it comprises of the strategies and technologies used by enterprises for data analysis of business information. This information is often used to provide historical, current and predictive views of business operations and facilitates efficient and effective decision-making, and enables automation of current business processes. Time to value, e.g. BI projects have demonstrated that even in organisations that are improving their BI maturity, the time to value is very low and insights can be obtained in a few weeks. However, to ensure your initiatives are sustainable, the organisation will eventually have to mature and get the necessary infrastructure in place.

Most of the market-leading BI tools these days are so similarly matched that the biggest contributing factors in deciding which tool to use comes down to how much data transformation needs to be handled by the tool itself and the total cost of ownership. When a business decides to embark on the BI journey, it's important to take an approach that will future-proof the solution. This means spending most of the time curating your data effectively. Start by establishing a data warehouse solution whereby business principles and logic are built into a foundation layer that is centrally located and controlled. What this will enable is the ability to implement any BI tool on top of this established data warehouse, and provide the versatility and flexibility to easily substitute the BI tool being used, should there be the need to do so. This ensures the sustainability of an entire BI solution.

Once a healthy foundation is set, the BI tools themselves need to be compared to determine which tool would be the best business fit. Some of the aspects that need to be considered are:

DEVELOPMENT TIME

This involves the amount of time required to get the BI project up and running. The development time directly impacts the overall development cost of the project.

USER EXPERIENCE

Choosing a BI tool that is easy to use, simple to understand and aesthetically pleasing to look at, will aid in the tools adoption rate and overall success of the solution.

SECURITY

The capability of restricting data to only be viewed by certain individuals or ensuring that all the data in a BI model is safe has become a standard in all BI reporting tools. There are however various methods that can be used to facilitate this function.

SELF-SERVICE

Generally, in any business there should be a few "power users". The ease with which these users create a particular view, dashboard or develop their own model to save on development costs is referred to as the self-service aspect of a tool.

LICENCE COSTS

The costs associated with implementing and maintaining the tool generally involves a licensing cost. This could either be a single enterprise wide licensing cost and/or licensing cost per user. This is one of the most important aspects to choosing the correct BI tool for an organisation.

FUNCTIONALITY

Although a lot of the market-leading BI tools share similar features and have more or less the same capabilities, careful consideration should be given to the overall goal of the organisation and the capabilities that will be required for the particular task at hand.

CUSTOMISABILITY

BI tools offer a wide range of customisability in terms of the look and feel. This usually involves adding corporate identity to any model that will be developed.

EMBEDDING

Embedding of models have become a very popular addition in many BI tools. This allows models to be incorporated into a web interface allowing a seamless transition from, for example an intranet site to the BI tool. This greatly improves the adoption rate within an organisation as there is no need to alternate between two interfaces to access the tools.

PUSH REPORTS

Many years ago, it was common to pull your reports from the BI tools, however, a shift has come whereby push reporting capabilities have been incorporated. This allows users to be sent reports in various formats via email. Removing the need to go to the tool itself and pull data.

INTEGRATIONS

BI tools allow integration between other components such as Python, R and other statistical tools. Additionally, the capabilities of connecting to various data sources forms part of the integration capabilities and is an important consideration as not all of the tools are able to connect to an equal amount of data sources and some tools require additional expenditure to facilitate connecting to particular source systems.

06 TAKE AWAYS

A team with experience using a variety of technologies can give expert advice from lessons learned, what the new trends are and common pitfalls. Make sure the vision for the solution is clear and the functional and non-functional requirements are understood in advance to avoid future re-work, but don't try to do everything at once: figure out what the MVP is so that a working application can be delivered quickly. Then feedback and changing business requirements can be incorporated to meet the business demands as needed.

FURTHER READING

Non-functional requirements:

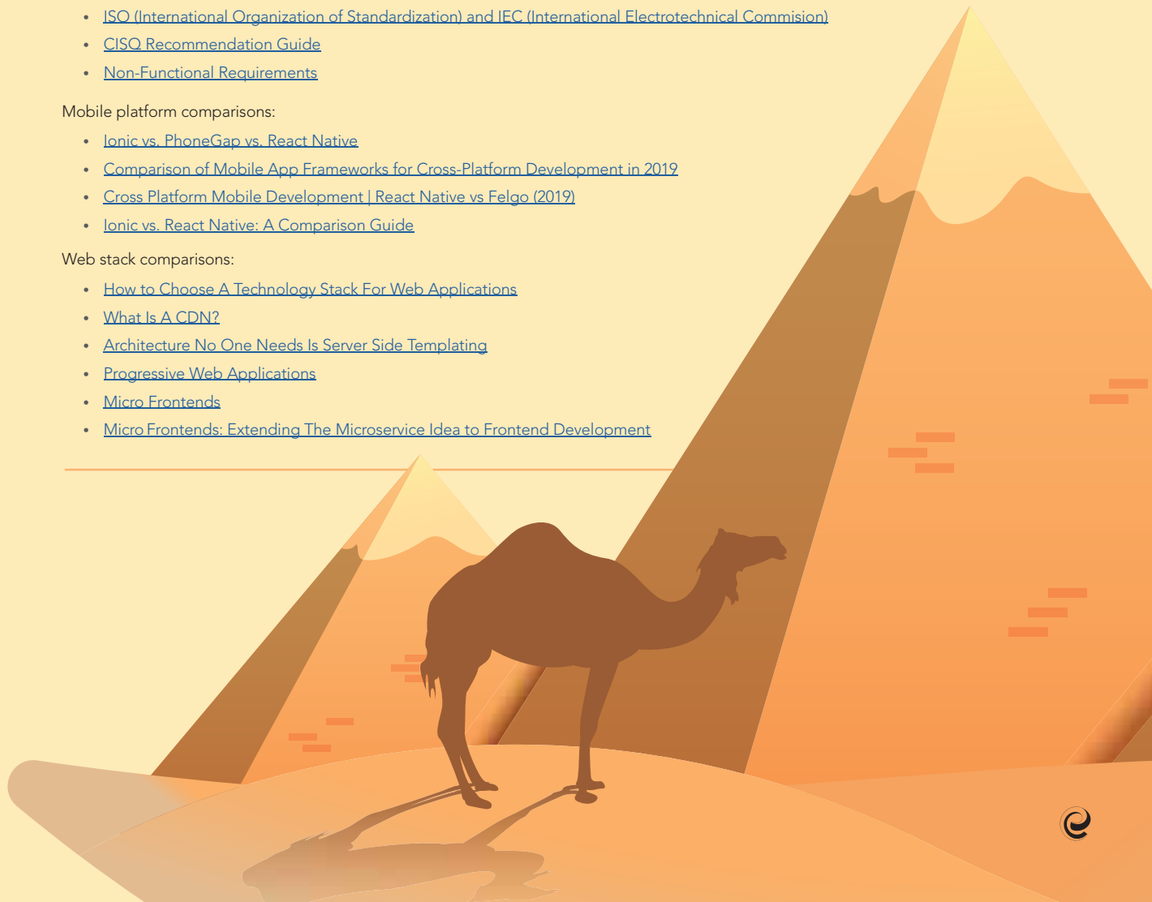
- ISO (International Organization of Standardization) and IEC (International Electrotechnical Commision)
- CISQ Recommendation Guide
- Non-Functional Requirements

Mobile platform comparisons:

- Ionic vs. PhoneGap vs. React Native
- Comparison of Mobile App Frameworks for Cross-Platform Development in 2019
- Cross Platform Mobile Development | React Native vs Felgo (2019)
- Ionic vs. React Native: A Comparison Guide

Web stack comparisons:

- How to Choose A Technology Stack For Web Applications
- What Is A CDN?
- Architecture No One Needs Is Server Side Templating
- Progressive Web Applications
- Micro Frontends
- Micro Frontends: Extending The Microservice Idea to Frontend Development



CHAPTER 5

AUTOMATION LETS YOU MOVE FAST AND LEARN FAST

Developers are hired to solve business problems, but in the world of software development there often end up being many menial, repetitive tasks. There are multiple issues with developers performing these tasks manually:

- They are not providing direct value to the business when distracted with these tasks
- Humans are prone to error when doing the same thing over and over (as we become familiar, we become over-confident and make silly mistakes)
- The tasks are not challenging to the mind of a developer which can cause boredom and a lack of job satisfaction

Fortunately, many of these repetitive tasks are nothing more than a set of well-defined steps that need to be executed in a specific sequence, often depending on the output of the previous step. These are perfect candidates for automation.

The following war cry sums it up well: **"If it's important and repetitive, then automate it!".**

01 FOUNDATIONS OF SOFTWARE DEVELOPMENT AUTOMATION

An extremely important practice is required to enable automation. This practice is source control, or sometimes called version control.

Source control allows a team to track changes made to the source code as features are developed. Developers can see a comprehensive history of changes made as well as jump around that history to revert to a specific point in time.

These practices also allow for the "branching" of code which enables developers to make sweeping changes without affecting the work of others. There are various strategies around branching, each with their own pros and cons, but the two most popular are:

1. Trunk-based or master-based development: All work is continuously merged into the main branch, even if the feature is not yet complete.
2. Feature branches: Work is done on a separate branch and only merged into the main branch once complete.

Besides the primary benefits of a source control system, maintaining the source of an application in a centralised, version-controlled repository is the key to unlocking many of the automations mentioned below.

02 TYPES OF AUTOMATION

There are many kinds of tasks that are performed across all software projects, there are also unique tasks per specific project. Most of these tasks can be automated. Let's have a look at some generally accepted best practices for automation in the life of building a solution.

CONTINUOUS INTEGRATION

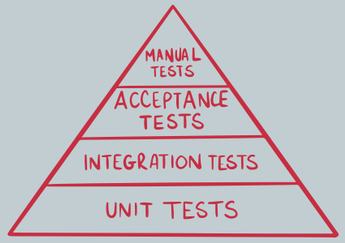
Eventually, most software projects grow large, and the development teams expand to cope with the demand for new features and maintenance. This results in many parallel branches of the code existing at the same time and in various states.

These branches will have to be merged together (and back into the main branch) at some point. There is a high risk of merging errors, as well as logic bugs when this code comes together, especially if the branches have lived alone for any extended period. The solution is to merge all branches regularly and to make sure everything is always working all the time. This process is known as "continuous integration," and is defined by Martin Fowler as "a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including automated tests) to detect integration errors as quickly as possible."

Another advantage of this automated integration is a consistent build. We have a well-defined automated process that produces binaries to be deployed onto some environment. This removes the risk of a developer having some specific setup that no-one else can replicate and resulting in only that developer being able to build for a release. We also remove the commonly used "but it works on my machine" excuse. If it doesn't build on the build server, it doesn't go to production.

AUTOMATED TESTING

An extremely important part of continuous integration is the execution of all automated tests. These tests should be written by the developers as a safety net that gives them confidence to make changes without accidentally breaking something else in the code base.



In the diagram, the bottom 3 levels can be automated with varying degrees of difficulty.

Unit tests are the most straight-forward to automate as they should be small self-contained tests that only confirm the functionality of a single piece of code under very specific conditions. These tests should have no outside dependencies (or all dependencies are mocked out) so that they can be executed over and over with no side-effects.

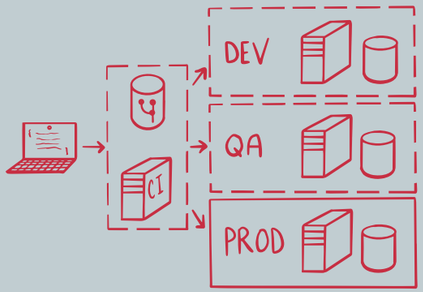
Integration tests are used to ensure that blocks of functionality can talk to each other successfully. This may mean accessing a database or file system, or calling a service over the network. These types of tests may also check the flow of data through the entire system. Automated integration tests are harder to set up, as they often require a valid database to access (with valid test data pre-populated) or real services to call.

Acceptance tests exist to make sure that the end-to-end solution being built meets the requirements specified by the business. In systems with a user interface, these often take the form of scripted steps through a specific flow, recording and verifying along the way that the system is behaving as expected. For non-UI systems, these tests may include loading production-like data into the system and checking the output. Acceptance tests require a deployed instance of the application to be executed on.

Whatever the type of test, running them manually would put a huge strain on the team and severely affect delivery. Executing the tests on every code check-in (or once a day for heavier integration and acceptance tests) provides confidence in the changes being made with very little effort from the team required (beyond writing the tests when functionality is built).

AUTOMATED DEPLOYMENT / CONTINUOUS DELIVERY

Deploying an application into an environment can take many forms. In most cases there is a file copy involved, and often there are multiple steps involved before and after the file copy. These steps could theoretically be documented as a recipe for someone to follow when deploying, but there is still room for error when a human gets involved with simple, repetitive tasks.



An application could also be deployed to various testing environments on its way through the production environment. To get feedback quickly, it's a good idea to deploy the latest changes to a test environment as they are made. Doing all these deployments manually would be a huge time sink for the team.

At a minimum, deployments should be executed by a script. In an ideal world, that script should be controlled by a system that can put permissions and auditing around when a script is executed, who executed it and what the results of that execution were.

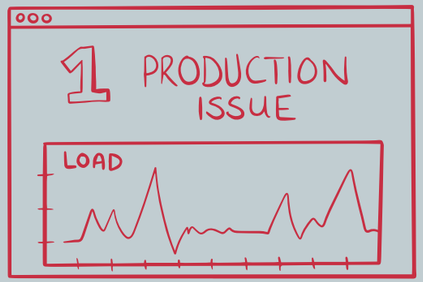
PROVISIONING AND SCALING

In this era of cloud-based computing, it is also possible to automate the provisioning of resources for the consumption of our application. We can spin up an entirely new environment just to run a set of tests and then tear it down again when done.

We may have a system that provides functionality for adding new customers each with their own dedicated resources. These resources can be automatically provisioned on demand.

MONITORING

Once an application is released into the wild, it is extremely important to ensure that it continues running well and that any errors or unexpected behaviour is picked up immediately. Systems should be put in place that alert the team to anomalies, instead of requiring someone to watch dashboards all day or sift through thousands of system generated error reports. Artificial intelligence can also play an important role in identifying unusual trends.



03 AUTOMATION TOOLING

As with many tools available in the software development world, there is a wide variety of tools and systems that will help a team accomplish its automation goals.

SCRIPTING TOOLS

Scripts allow the chaining together of commands into a single parameterised command. Initially, the team would execute the scripts themselves until they feel confident they are performing as intended.

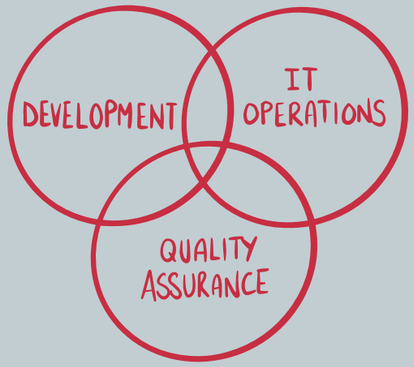
DEVOPS TOOLS

The term DevOps is used to describe a set of practices that combine software development and IT operations teams in order to shorten the cycle of delivering working software and realising business value. Automation falls into these practices and there are many software solutions available in this space.

A team should invest in these systems so that they are able to automate and control the execution of the scripts (or extend them with further capabilities) and provide additional functionality such as permissions, logging, remote execution, scheduling, etc.

There are many players in this field and the choice of system would depend on many factors:

- What technologies are used to build the application?
- Are there operating system constraints on the build tools?
- Does the organisation have preferred software vendors?
- Is it suitable to execute builds and tests in the cloud, or must it be on-prem?



CHAPTER 6

MEASUREMENT QUANTIFIES YOUR SUCCESS

Once an idea has made its journey into an implemented solution, it is necessary to make sure the original need has been met. Just because something is in production, it does not necessarily mean it is doing what it should be, and often it's difficult to tell straight away.

There are various techniques that can be used to measure the success of an implementation, both from a business point of view as well as technical.

01 MEASURING TECHNICAL SUCCESS

There are many things that could potentially fail in a live system. Many of these risks would have been considered when making technology stack choices and while implementing the solution. But there is still a very important need to keep a close eye on what is happening in real time and to be prepared to act as soon as something fails.

GATHERING METRICS

A software system has the potential to generate large amounts of data related to the way it is functioning. This data can take various forms:

- Text log files
- Database log entries
- Hardware performance metrics

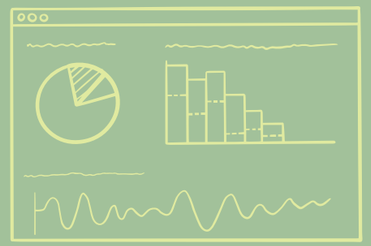
Gathering all this data presents some unique challenges, substantial amounts of storage are required and there is a risk of overloading support staff to the degree that important information is missed or ignored.

MAKING SENSE OF METRICS

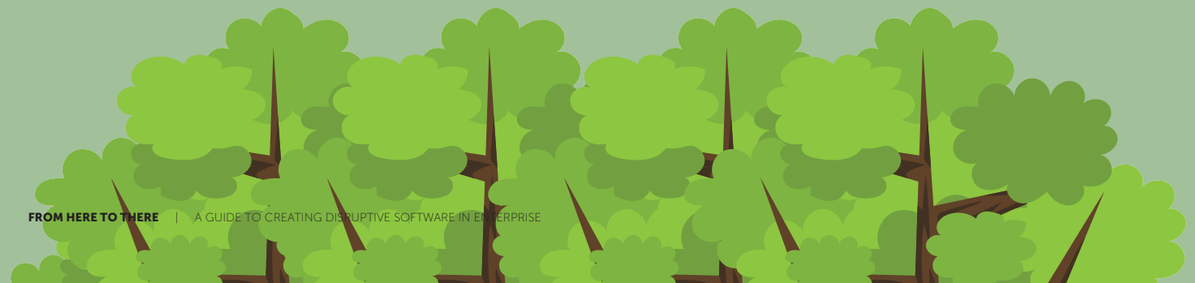
Simply gathering data is not useful to the business or technical team unless they are able to glean insights from it. The data needs to be processed and analysed (in an automated tool) so that trends and anomalies are easily identified. Artificial intelligence and statistical analysis have the potential to find things in the data that most humans would miss.

RADIATING INFORMATION

Once trends and worrying issues are identified, the relevant stakeholders need to be made aware of them. Simply emailing reports is likely to get them filtered away and ignored. The insights need to be on display all the time and radiating throughout the workspace of the people responsible for the health of the solution.



Permanent dashboards displayed on TVs are a great way to radiate information. They do, however, need to be well thought out to be the most effective. Make sure that anomalies are instantly visible, show trends over time so that unusual changes stick out. There is also room for showing successful metrics to promote confidence in the health of the system.

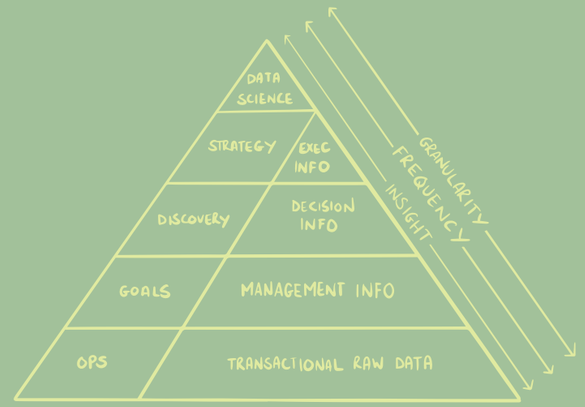


02 MEASURING BUSINESS SUCCESS

As part of the implementation process, a business intelligence (BI) solution would have been chosen and put into place. Now is the time to leverage that effort to measure the business success of the implemented idea. The chosen BI stack will allow stakeholders to slice and dice the data in order to extract key metrics and indicators.

THE DATA PYRAMID

The data pyramid shows how reporting, roles and use cases for the data change in a mature data environment. The insights gained, frequency and granularity of the underlying data, and audience viewing the reports change, depending on the layer of the pyramid.



At the bottom layer, reporting is primarily based on transactional or “near real time” information. This serves the purpose of showcasing the as-is processes within the business and serves to improve visibility of business processes.

On the upper layers, reporting becomes more analytical and strategic in nature, providing a more aggregated view of information which could help better infer trends, which assists management in doing comparative analysis and scenario planning, among others. This helps identify where a business needs to be and what adjustments need to be made in order to better reach business goals. This is achieved by enabling an accurate view of key indicators tracked against organisational targets.

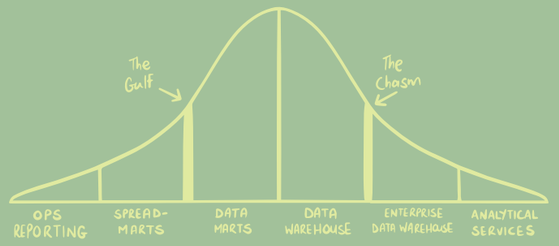
The capstone of the data pyramid is predictive and prescriptive analytics that come into play through the use of data science. Predictive analytics indicates what is likely to happen in future (i.e. what will happen), whereas prescriptive analytics refers to recommended actions and strategies (i.e. what to do about it).

The BI maturity model and the data pyramid work hand-in-hand. Having a mature data environment naturally enables more analytical data use cases through the insights journey below.

THE BI MATURITY MODEL

Each organisation or division is on a journey to an optimised data warehouse (DWH). Even a mature and optimised DWH will still change as new analytical requirements are discovered and as new trends emerge in the field.

The reporting and analytical requirements of an entity change significantly as they move through the steps in the BI maturity model, as explained to the right. Please note that an entity could be in more than one level across different facets of their business.



STAGE 1

This stage represents the pre-data warehouse environment where an organisation relies entirely on operational reports for information. In general, these reports are static and inflexible and show a limited range of data for a limited set of processes providing little benefit to the organisation. During this stage it is mostly spreadsheets on local machines that are used to support the reporting requirement of the organisation.

STAGE 2

The entity's first attempt at a BI solution, where an initiative is started by a department without fully understanding all the fundamental requirements of a successful BI solution. The first BI tools are introduced - primarily for ad-hoc queries, reporting and basic OLAP - where licences are given to only a handful of power users and analysts. Benefits are starting to be derived, but only by a handful of people within the organisation.

STAGE 3

At this point, a business unit recognises the value of consolidating these silos of information into a single data warehouse to save money and gain greater consistency in the information it uses to understand and analyse the business.

STAGE 4

An area is entered where BI/DWH delivers a strategic, enterprise resource that enables organisations to achieve their key objectives. This is accomplished through a unified data warehousing architecture defined by a common set of semantics and rules for terms and metrics shared across the business.

STAGE 5

This phase completes the cycle by converting core BI/DWH capabilities into services, both technical and commercial, and redistributing development back out to the business units via centres of excellence.

STAGE 6

With a well-established curated DWH solution in place, true value can be derived from BI by using predictive and prescriptive analytics that come into play by incorporating data science. The focus shifts from looking at the past, to trying to look into the future by learning from mistakes made or improving on past successes.

IT ALL COMES DOWN TO

HAVING THE RIGHT KNOW-HOW AND THE **RIGHT PEOPLE**

To build something truly differentiating and disruptive, it all comes down to the right know-how, and the right people – navigators, implementors, and decision-makers. A vision of where you want to be is an important start. Having a bold and ambitious idea of what your organisations offerings could be breaks out of the business-as-usual mindset and creates inspiration and urgency to become more. The vision must be driven from top down, and involve people from all levels in the organisation. People from different departments, operation levels, and responsibilities that may be impacted or have an impact. Alignment and buy-in from everyone spurs new ideas, collaboration, and mitigates risk.

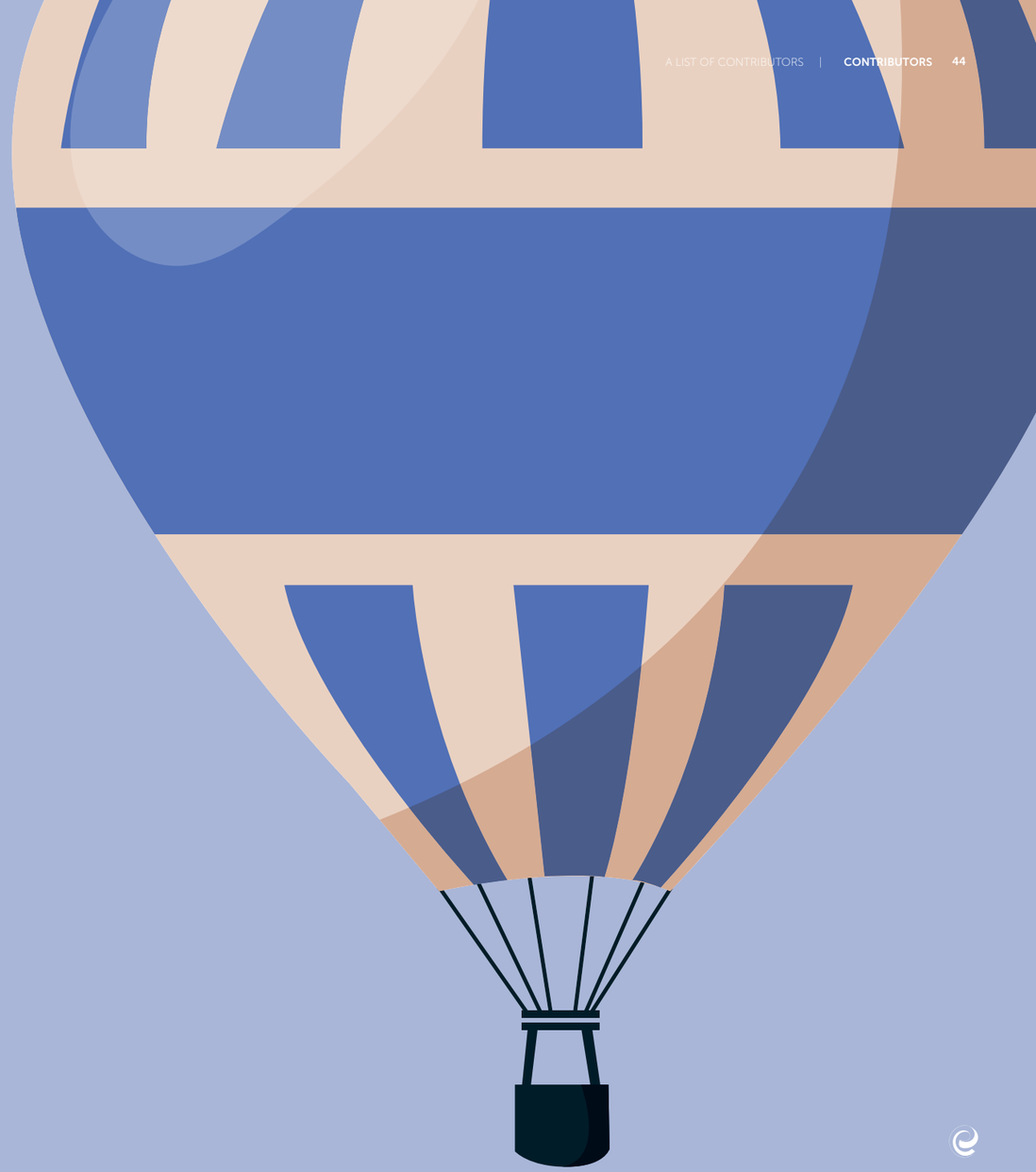
A structured yet adaptive process for intake of ideas, mapping them to the vision, execution, and measurement must be defined; more importantly, the right team of cross-skilled, knowledgeable, and ambitious individuals trumps all. Technology choices and approaches are virtually infinite. Choosing the right technologies in a pragmatic way to solve problems efficiently is critical to any innovation project, while automating menial or risky tasks such as deployments, infrastructure changes, and patches mitigate human error, and allow for rapid feedback cycles between the users and business. Finally, everything would have been for nothing if the success of the solution cannot be measured, and learnings cannot be incorporated into the solution easily and incrementally.

By embracing these different facets of building a solution, with the right team, **everything is possible**.

WHO MADE THIS

A LIST OF CONTRIBUTORS

Chris Nezar
Duane McKibbin
Gail Shaw
Greg Schroder
Jacques Du Preez
Jeffrey Russell
Jolene van Heerden
Justin Weldon
Marius Kruger
Matthew Butler
Rian Lauwrens
Rishal Hurbans
Ruan Greeff
Safiyya Laher
Sean Buch




entelect.co.za