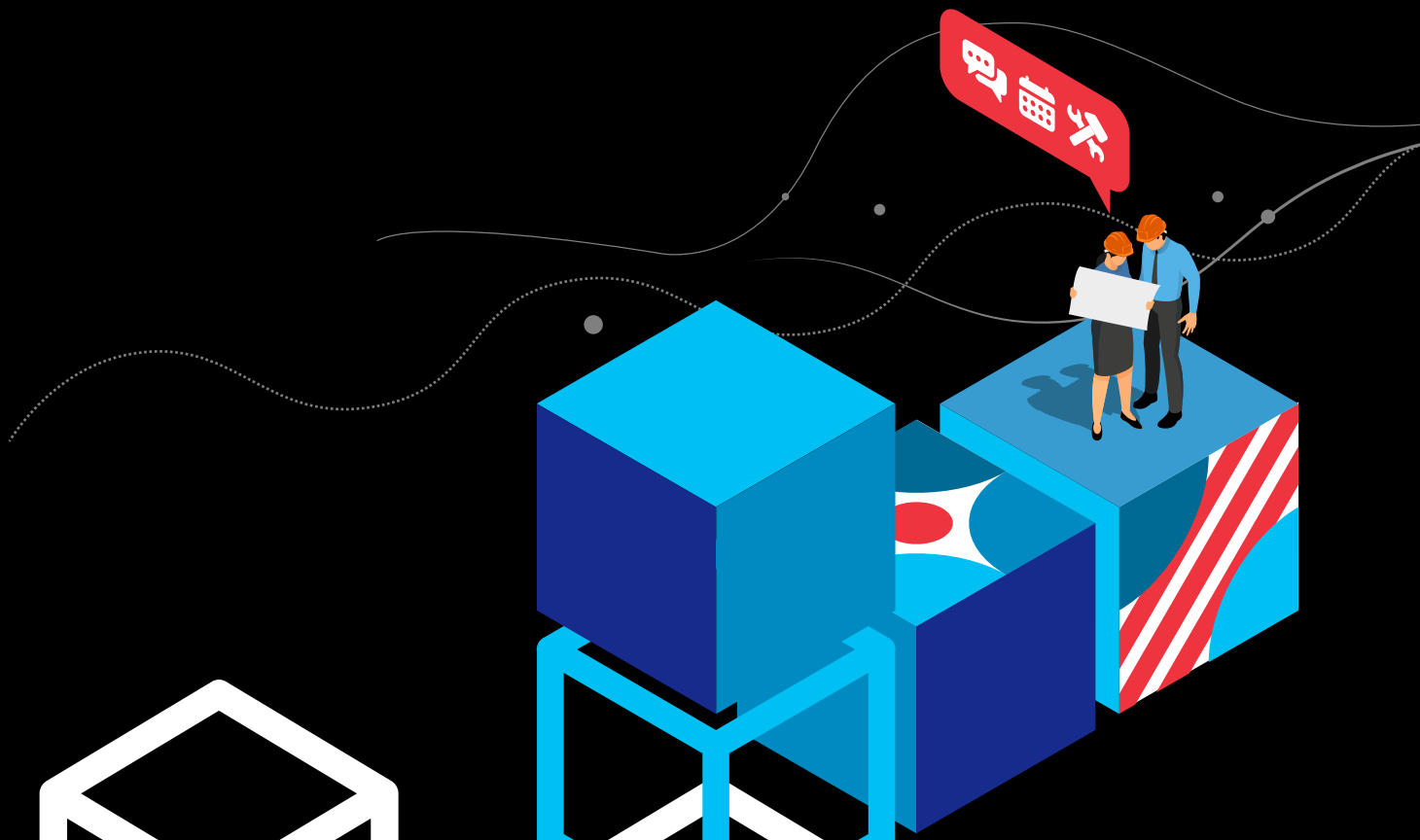
How to scale Laravel projects and applications

10 insights based on 10-plus years working with Laravel.
Foreword by Taylor Otwell



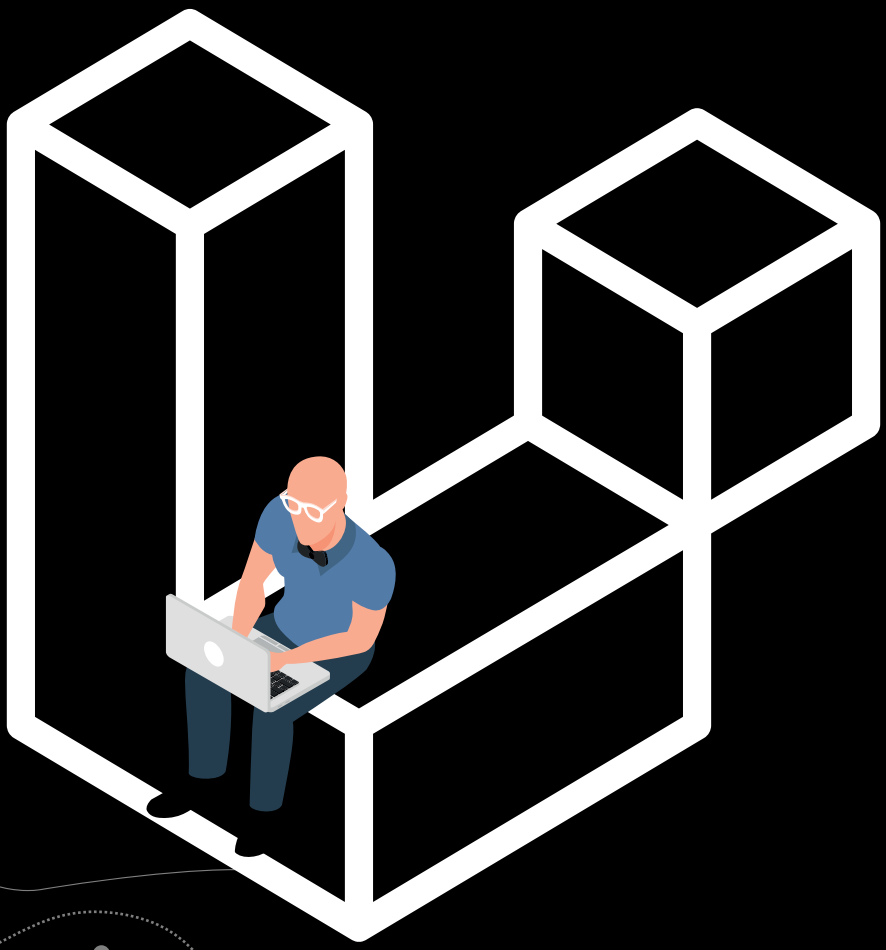
CACI

Foreword

Laravel has evolved into a world-leading web development framework with the depth and breadth to build a myriad of applications and platforms. This white paper covers important topics like planning, architecture, collaboration, design, documentation, and specification. I think there is something in this paper for anyone that works in the Laravel universe.

Taylor Otwell

Founder and CEO of Laravel



Introduction

The reasons why we chose Laravel as a preferred framework for application development over 10 years ago are the same now as then. Built on PHP used by most websites and continually pushed by founder Taylor Otwell and core team to improve its capabilities, Laravel is brilliant for usability, scalability, flexibility, and security to build great digital products with complex business requirements.

All frameworks have strengths, but Laravel's use by organisations like BBC and Pfizer speaks to its quality, and it's a joy to use (developers' most starred **PHP framework on GitHub**) for whatever a business needs it to do.

What CEOs, CMOs, COOs and CTO's need digital to do has been transformed in the past decade, with business-critical products, services, and processes now delivered online. But the ability to scale Laravel projects to meet organisations' business ambitions is by no means guaranteed, despite the framework's many strengths.

From our decade-plus of enterprise-level Laravel application experience and many major rescue projects, we've found success rests on the long-term – and continual – alignment of business strategy and technical development. Delivered through brilliant product management, communication, documentation, and maintenance, whether the application is in start-up, rescue, or growth phase. Most of which rings true for all complex digital products, Laravel or not.

This white paper brings together the 10 key success insights from our Laravel experience to help stakeholders, product owners and product leads understand what they should be asking of their development team. And conversely, what they must bring to any Laravel project to ensure it meets ROI and long-term business objectives.

We do hope you enjoy the read. If you'd like to get in touch about your Laravel project, or have any feedback, our emails are below.

Gareth Drew
Chief of Digital Technology
gdrew@caci.co.uk

Sylvain Reiter
Chief Delivery Officer
sreiter@caci.co.uk

Contents

1. Successful Laravel projects align with business vision and strategy

Doing business in the 21st century sees few – if any - businesses unaffected by digital transformation. For organisations with big ambitions, it's critical. Simply to participate in an always-on, global marketplace – let alone succeed in it - requires the right, cost-effective digital tools, processes and platforms. We've found the absolute first step is to align the digital project with the business' vision and strategy before moving into any type of development. This product management approach applies just as much to established companies looking to optimise their operations as it does to start-ups or products looking to scale. While this might seem obvious in theory, in practice, we find it's quite a different matter.

> ***"If you design and build a digital platform solely with a minimal viable product in mind, you've put your business into technical debt before you've even started"***

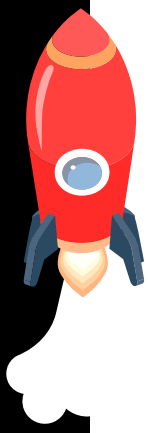
The specific problem we tend to see – and try to avoid – is that of ambition without clear or realistic understanding of how that translates into technical specification, design and ongoing development. Take, for example, a company that has a minimum viable product (MVP) along with the goal of expanding globally within a year. If this company's product team were to design too limited a digital platform, solely with the MVP in mind, they would have put the company into technical debt before they had even started. This technical debt lies in the fact that this basic platform might not be scalable enough to meet the company's goals once it has validated its MVP and begins its growth.

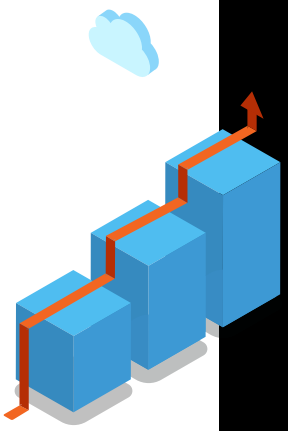
Some of the early indicators that we see of technical debt are attitudes and comments along the lines of "we can just add that later" or "it's just a small addition to the existing feature from the MVP". This attitude can also apply to security and architecture considerations due to understandable desire to get the MVP launched. Unfortunately, we see this happen more with large-scale projects, where it's always easy to drift from clarity due to the scale of the timeline and the project itself. In reality, a successful MVP has the project on too rapid a trajectory to allow costly re-engineering.

And while Laravel is designed, as a framework, to be scalable and flexible enough to conform to a business' needs, it is, once again, only a tool. To ensure that tool works properly and helps the business achieve its goals, all internal teams and the company's digital partners need to understand the business requirements and plan as fully and clearly as possible. This allows them to develop a vision for what the project can or will do for the business and build the software requirements into the development roadmap. Aligning the product with business strategy is not a one-off process, it's ongoing.

Breathing new life into a Laravel ecosystem

Our client, Worcester Bosch, the UK's leading manufacturer of boilers and water heating products has a long-established, enterprise-scale Laravel ecosystem supporting the needs of millions of customers and thousands of service providers. As their extensive digital platform grew iteratively over the years, they required support to ensure that it could continue delivering long-term value for the business and develop at the pace they needed it to. We became the lead agency for its Laravel application ecosystem to help realign their digital roadmap and refresh the application's structure and standards to ensure it could support its ever-evolving requirements.





Supporting a decade-plus of business growth

Our ongoing Laravel project for Cabot Financial has grown from initial launch as a single personal consumer finance management app built on Laravel 3 in 2012, to a multi-organisation and multi-country credit management service digital ecosystem on Laravel 10, including consumer white-labelled websites, native apps, back-office tools and 3rd party API integrations. The success of the product – and relationship – being down to clear continuous communication with key stakeholders as to their ambitions, aligned to a robust schedule of user and technical reviews with the entire team buying into clear project governance. Generating roadmaps that deliver new features, while maintaining the fundamental integrity and performance of the product through regular maintenance and refactoring.

2. A service design and user-centred approach is vital

Once the business vision is clear, the next step is to focus on the user. Many organisations balk at the idea of carrying out additional research, or choose to move through it swiftly, because they're understandably eager to get started on design and development. But again, the real key to successfully scaling a digital project over time is to slow down at the beginning and carry out the proper research upfront. Good User Experience (UX), for example, **can increase a brand's conversions by up to 400%**, underlining why it should be an integral part of the development process.

Online usage has increased drastically over the last few years, and with it market competition and end-users' expectations. There's also the wider context of the 20+% of users with disabilities or literacy challenges. This underscores the importance of designing and developing with a diverse audience in mind. Yes, UX is research-heavy, but this research delivers crucial insight into a business' audience and, in turn, leads to differentiation. Service Design goes another level step deeper than that, which we believe is essential for business-critical applications.

Our human-centred research process not only aims to uncover the product's audiences' needs, but also those of the business itself. We want to know who the customer-side website or app users are, what they're looking for and what these business users aim to achieve with the product. For example, how easily the marketing team can update their CMS to meet their audience's needs, operations teams being able to pull off the right reports to aid decision-making or if the application needs to be integrated into a supplier environment.

> ***"Upfront discovery, followed by continual product management and deep cultural and organisational change, can only lead to better outcomes for all."***

The need to combine UX with business analysis, data architecture, technical discovery and systems mapping in the all-encompassing discipline of Service Design is critical to launch or evolve any product with long-term growth potential. Upfront discovery, followed by continual product management and organisational change, only leads to better outcomes for all.

In many ways, UX research sitting in the broader context of Service Design is not unlike the business alignment step. Its main point of difference, however, is that it considers the business within its wider context – the people, organisations and systems it needs to interact with for success – and identifies the key overlaps between the business and its audience's needs.

Also, as might be obvious by now, the order of these steps is crucial: a user-centred approach will not deliver or scale if it is not built first on a solid understanding of the business' overall vision for success and long-term strategy.

Scaling for success through human-centred service design

Leading B2B and B2C 'door-to-door', seamless delivery solutions company, Argid Technologies, was looking to transform its pioneering brand First Luggage into a bespoke, streamlined digital platform capable of scaling globally into new countries, sectors and markets like relocation services. We started by systematically mapping out Argid's operational and human requirements through user flow audits and service design blueprinting to unravel the business logic, use cases, and the various layers of management and integration necessary. The resulting Laravel application encompasses the back-end Argid Luggage Management System (AMS) and front-end self-service Customer Portals, from which the Argid team now effectively manages over twenty B2B, B2C and white label brands, all of which is aimed at minimising manual input and effort for all users. We continue to support the ongoing maintenance and growth of the application, enabling the rapid development of new features to meet Argid's exciting growth ambitions.

3. Architecture matters – data, design and tool kit

Selecting the right toolkit and architecture for any Laravel project is a critical third step once business and user requirements are aligned. It's important to be clear on the difference between a software's design – or its infrastructure – and its architecture.

Infrastructure refers to the individual components, such as toolkits, API integrations and servers, upon which a system is built and delivers functionality. Architecture refers to how those components are designed and their relationship to one another – its overall blueprint.

Think, how cars are designed to drive on the road and boats are designed to sail on water – and within that whether they are a racing car or people carrier, or rowboat or cruise ship. The architecture, the project's blueprint, shapes what the product fundamentally is and can do, the infrastructure the types of individual components used to build that. Software architecture must also be designed with evolution in mind, as the wider software environment itself continually changes. So while software infrastructure is relatively generic, its architecture will be project specific.

Architecture affects the project's availability for users to access it when they need it, its scalability as the userbase grows or new business processes are added, its overall performance in terms of speed, its security and – critically – its testability and maintainability over time as technology evolves.

This is why designers and solution architects need to ensure the flow of the project is logical from a user and business perspective, with the application's architecture and database following accordingly.

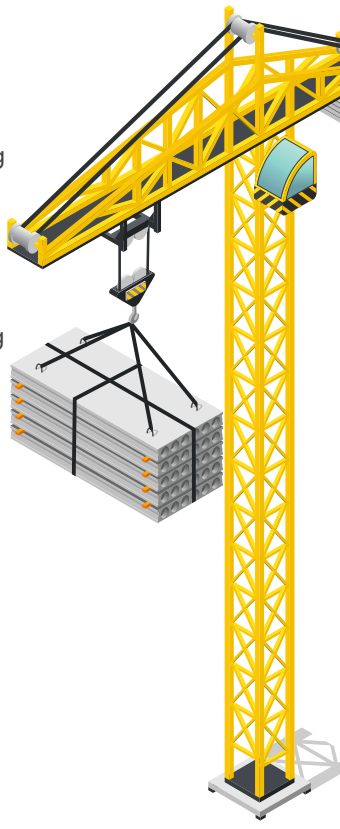
It's particularly critical for product owners to know what data the application needs to collect. Applications only collect what they're instructed to. If your back end is not designed to request a specific metric, for example logging data on which user has done what, when, it simply won't collect it. Getting data collection wrong can lead to intractable issues, for example, an application designed to only capture and communicate tabular data, but later asked to deal with images, voice, and video.

Software architecture follows patterns allowing developers to quickly resolve typical problems they run into when building new applications, letting them deliver a predictable structural output that can be understood by stakeholders and product owners.

Two of the most common are monolithic and microservices. Ideal for smaller applications, a monolithic architecture has an application's database, front end and back end all sharing one server. Microservices, by contrast, have multiple instances to deal with different functions. Think of a movie streaming service that's split into one server for user interactions like login and 'likes', with another server (or microservice) for streaming video to ensure the whole system doesn't slow down if lots of people are streaming at once.

Laravel, by its very nature, is a collection of architectural best practices curated over many years. Its curators selecting only the best features contributed by the community to add maximum value across a myriad of projects, consistently enriching the framework. This makes it extremely powerful when combined with a shared understanding by all stakeholders of what the application needs to do over time.

Yet we are often called upon to rescue Laravel projects that aren't performing or delivering to expectations – sometimes not functioning properly at all. While it's difficult to pinpoint a single reason for the break – other reasons which we'll cover later - we almost always find teams rushed too quickly into building the software. There was a lack of proper, documented understanding of user, business, and data requirements, with a failure to fully scope out a software architecture, translated into a product roadmap with clear timelines.



> ***Software architecture must also be designed with evolution in mind, as the wider software environment itself continually changes.***

In short, a software's architecture is the foundational blueprint for a successful project. Architectural decisions always involve trade-offs, but fundamental mistakes at this stage about the business and its users, will see development teams forced to continually improvise and – even more dangerously – be unable to select the best implementation and toolkits for the job. This will frustrate users, slow down development, and increase costs, sometimes catastrophically in the case of a rebuild, and almost guarantee project failure in the long-term.

A model of understanding

A good option to document and visualise the software architecture is to follow the **C4 Model**. It allows the project team to showcase the whole platform at varying level of complexity. Board Members have an overview at the "Context Level" and Marketers, the key modules at the "Container Level". Product Managers will manage their own modules at the "Components Level" and finally, developers will focus on the code of each component.

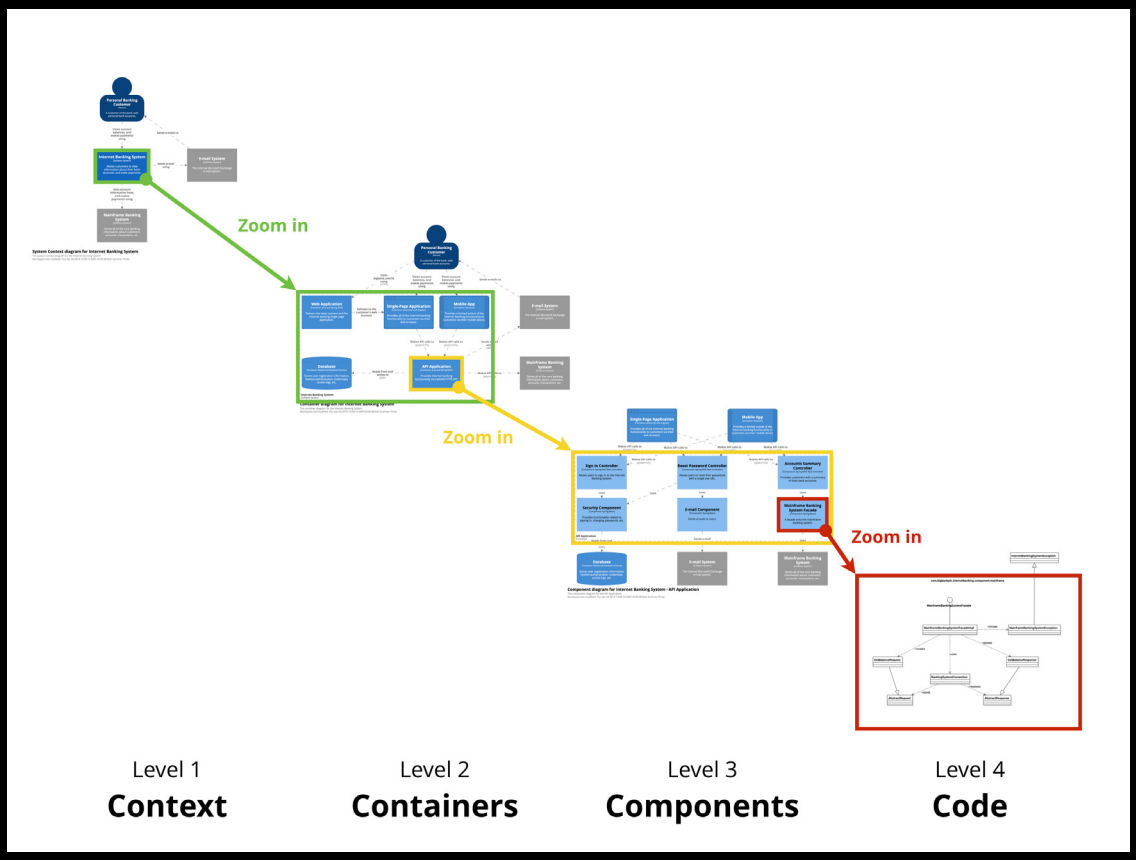
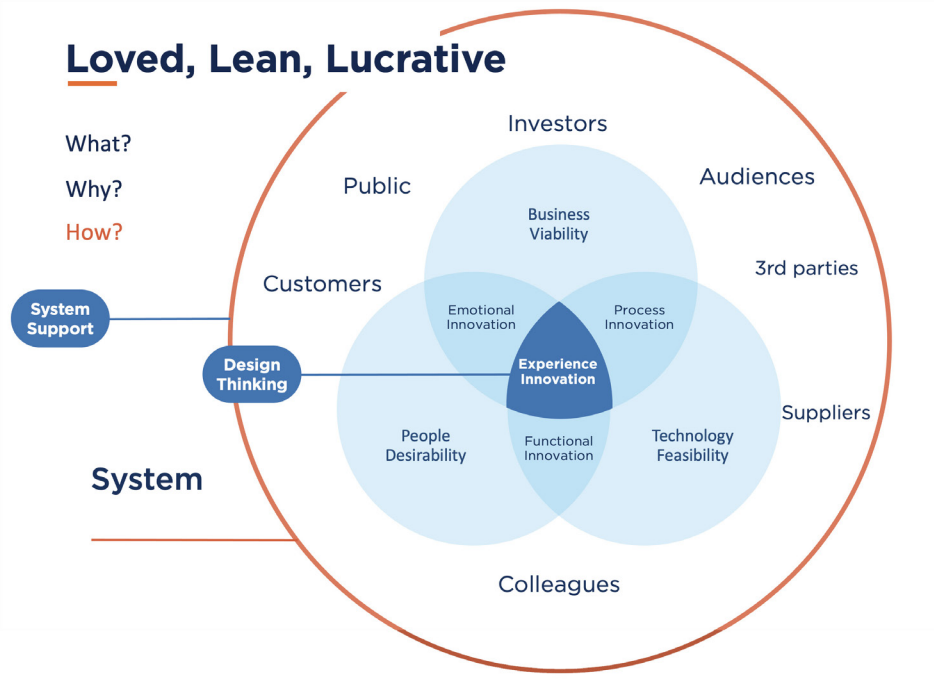


image from c4model.com

4. Less is more – a lean approach delivers sustainability and scalability

We've looked at why it's critical to understand business, user and architectural requirements before building any new project, or application overhaul. But there's a further consideration overlaying this that's also critical to success: sustainability.

While primarily associated with environmental concerns (which we'll touch on later) sustainability in digital products is about delivering innovations which IDEO, the renowned product consultancy, characterise as "desirable from a human point of view, technologically feasible and economically viable". Or, as we like to call it here: 'Loved, Lean and Lucrative'.



Veering too far in the direction of any one of these three criteria, or missing one out entirely, will make long-term success impossible. If a product is amazing to use, offering immersive technologies to deliver the most personalised experience possible, but the licensing and maintenance of that technology involved is prohibitively expensive, the product will fail. Likewise, if an application is efficient to run from an economic and technical perspective, but users find the interface difficult or inaccessible to use, eventually that poor employee and customer experience will see them go elsewhere.

So, viewed from this perspective, over-engineering and adding too much unneeded complexity in terms of features and infrastructure, is just as much of a problem as incurring technical debt in a rush to MVP, for example.

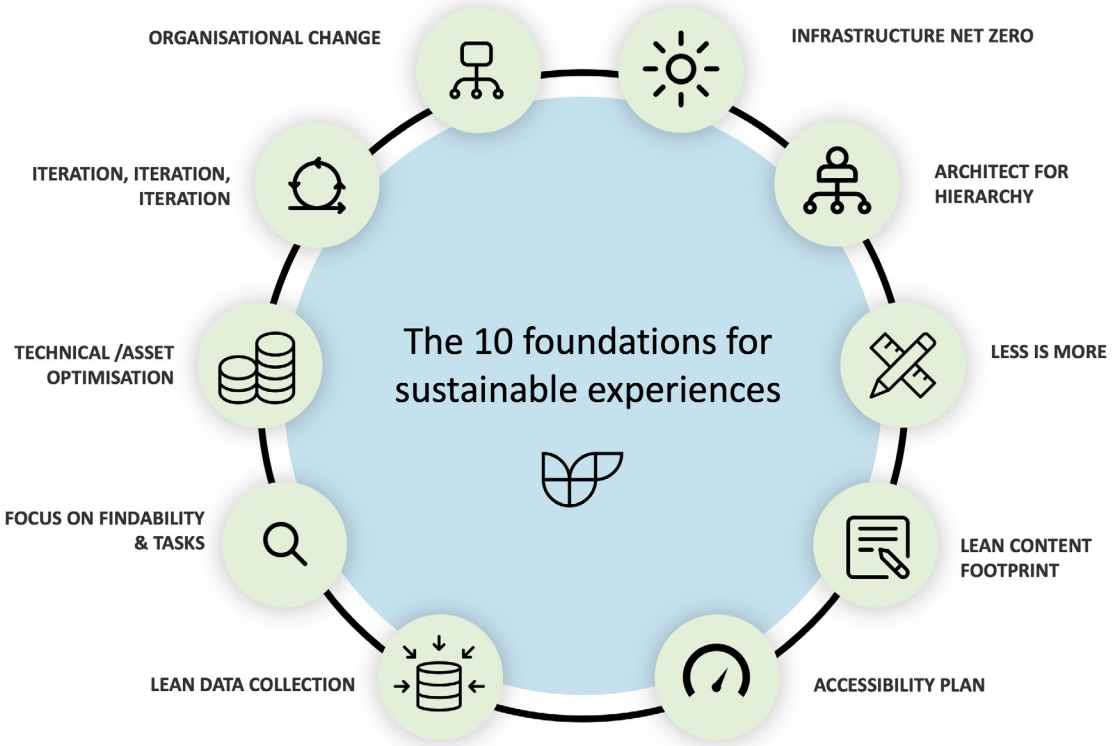
Designing in a sustainable, or intersectional way, says ‘less is more’ as long as the fundamental requirements for each criterion are met. Effectively, saying that we can – and should - solve multiple problems with the same design solutions.

Take GDPR and similar data protection laws. Merely storing data – including in cloud and hosting storage- counts as ‘processing’ that data. Which means if your application is collecting unnecessary data, it also exposes you to unnecessary risk. And on top of that, its storage has both economic and environmental costs.

It’s now estimated that digital’s carbon footprint is actually now greater than the aviation industry, and at minimum **equivalent to being the 5th largest country carbon emitter** responsible for 2-6% of global GHG emissions, depending on various academic estimates.

In addition, a typical data centre uses about 3-5 million gallons of water per day - the same amount of water as a city of 30,000 to 50,000 people. Meaning that how businesses design and operate their digital ecosystems, particularly enterprise-scale ones, draws increasing scrutiny on how it might impact their ESG performance.

Our approach towards Laravel projects takes this intersectional, lean approach. Delivering maximum user experience, findability and accessibility for business efficiencies, with the minimum necessary content, data collection and infrastructure. Going for 'less is more' allows us to extend and maintain applications easily over time, as we only build what is truly needed, reducing waste, expenditure – and environmental impact.



5. Collaboration, communication and documentation build trust and success

A good part of our Laravel work involves rescuing a dysfunctional project or system, and one of the main reasons we hear when we ask product owners why they came to us for support is they've lost trust either in their internal team or current digital agency.

The best countermeasure we've found to this issue is transparency through clear communication, collaboration, backed up by robust documentation practices.

Everything we've mentioned until this point requires excellent communication, but when the team moves into development and things are dynamic, it's mission critical. More (and better) communication gives everyone, regardless of their technical ability, awareness, and clarity on what's being done and – critically – why. Ruling out any unexpected, or indeed nasty, surprises. It's why we actively build transparency into our product management and delivery approach.

Broadly speaking, it means bringing multifunctional teams together in the right meetings, in the right way, and on the right cadence. As a minimum, this means having the Technical Leads in the same meetings as the Product Owners, and the Delivery Manager in regular meetings with the business-side stakeholders. It means a focus on good, plain language that everyone can understand and not using jargon, whether business or technical. It means not working in silos, assuming a few lines of description on a ticket is enough.

New discoveries and problems often occur as digital projects unfold. Good communication keeps key stakeholders abreast of such findings. Examples include content writers noticing inconsistencies in the messaging scenarios, or designers and engineers needing more details early in the project. A late-stage problem could be the business considering a slight change in the requirements themselves.

Having the right frequency of meetings and communication ensures everyone is updated – in almost real time – regarding changes in strategy, alignment, and technical challenges. This allows problems to be communicated quickly, the development team to mitigate risks and manage expectations, and the whole project team to work together to resolve them. This increases the speed and quality of execution, saves time and money, strengthens relationships, and builds trust.

Robust documentation – the systematic process of recording product requirements, timelines, audit discoveries and development activities - cements the above. The fact that nearly all Laravel rescue projects we've taken on have limited, and sometimes almost entirely absent, documentation, perhaps speaks for itself...

Laravel users cite the excellent quality, accessibility and in-depth examples of the framework's own documentation as accelerating development and better code quality through architectural best practices.

We utilise this Laravel documentation within our own robust documentation framework - prioritising team transparency, client transparency, workflow management and audit checklists - to get, and remain, organised. It's challenging, given the time taken to complete and manage the documentation, documentation workflow and version control, etc, but this organisational habit is vital for success.

Considering documentation from both sides, it helps developers understand the business-side pain points and to report clearly to the product owner on progress. The technical audits we conduct at the start of every project also helps to document and present this complete picture to the key stakeholders. This allows for more realistic expectations, a clearer problem breakdown into smaller tasks, and to establish the right governance for the project tasks and timeline.

> ***"Documentation reflects and evidences good project governance and a collaborative, honest approach"***

Conversely, documentation helps product owners and stakeholders with transparency and traceability over the project, for example having the correct information on tickets for example, including acceptance criteria. Any review of a well-written ticket will explain the what, why, how and when of what development tasks have been undertaken.

This also lets us work at a consistent fast pace over the long-term. If we're fixing a bug or adding a new feature, the code will link back to relevant tickets and documentation providing the full context of what was done and its purpose. Allowing new changes to be made more easily and lowering the risk of errors.

Documentation, in effect, should reflect the project status quo at any given moment in time. So keeping it up to date and accurate is essential. This is especially important in the case of any audits or disputes, which protects all parties. But most importantly documentation reflects and evidences good project governance and a collaborative, honest approach. All of which builds teamwork and trust.

6. Investing in an upgrade and version migration strategy is critical

If we had to choose a single reason for including a software upgrade or migration strategy to these insights, it would be that we frequently discover in the rescue projects we take on that the baseline frameworks are several years out of **End of Life** (EOL) – in some cases by up to 6 *years*.

Using EOL software opens a product, and therefore the organisation, to numerous operational and security risks. Vulnerability to viruses and other breaches because the software vendor no longer provides patches, bug fixes, and/or security upgrades. Another example is the added time and cost of engineering workarounds for new features.

Maintaining your software is a lot like maintaining your car: without an annual inspection and necessary care, they develop little problems which, as they accumulate, usually become one big, expensive problem.

With Laravel, it's entirely avoidable. Laravel constantly delivers precise and clean upgrade documentation, as well as robust set of upgrade tools, ensuring the burden of maintaining parity with the latest version is minimised. So, the failure to do this in practice is really one of prioritisation and understanding it really is that business-critical. Yet what we discover in most of our audits is companies simply don't notice the problem until the entire system breaks down.

> ***"The huge costs and delays major upgrades can cause ironically are the exact issues companies try to avoid by not upgrading on a regular basis"***

It's almost impossible to add new features to a product and scale it without upgrading the underlying framework. Upgrading or migrating everything past EOL is far more drastic. In one case for a large client, it took four months, with all other development at a total standstill, to upgrade a framework 18 months past EOL. For smaller businesses, this would be significantly less, though the timeframe remains in "weeks" as opposed to "hours".

What's most important to remember is the huge delays these major upgrades can cause, as well as their cost. Ironically, these are the exact issues companies are trying to avoid by not upgrading on a more regular basis – i.e. they don't want to take the system offline long enough to upgrade, or they want to focus their development team on creation, rather than maintenance. But these organisations aren't considering the longer-term impact this cost-cutting has on their business.

Software development is about delivering on both short- and long-term fronts. When quality and velocity start to drop because of underlying software issues, the trust factor begins to come into play. This applies to the business as much as it does to the development team. Customers will simply vote with their wallets if a site or app's quality drops, meaning businesses owe it to themselves to follow a good maintenance schedule to remain competitive.

We talk more about the importance of ongoing maintenance in section 8, but as a rule of thumb, we like to dedicate 20% of our development time to maintenance. This makes it more manageable and actually clears the remaining 80% of our time to more predictable and higher-quality delivery.

A good upgrade or migration strategy therefore delivers greater security, increased compatibility, and efficiency, among other benefits – like developers motivated and happy to work on cutting-edge code – all of which lead to reduced costs, scalability, and an optimised customer experience.

> ***"The solution is ongoing, small-scale audits and upgrade schedules built directly into the product roadmap."***

7. Iterative development makes better products, but needs a roadmap

To ensure the software delivers business value from the very start, we strongly recommend following an iterative (versus incremental) development approach. A further, immediate benefit is that this delivers working software right from the start in the case of new projects.

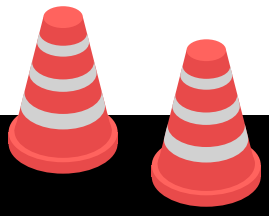
As with many of the steps we have mentioned so far, there are, of course, similarities between the two approaches. However, the critical differences, in our opinion, are that the incremental approach doesn't allow much flexibility as the business or product scales, along with its requirements.

This is not to say that the iterative approach should not come with a roadmap. In fact, no matter what approach a team takes, we continue to strongly advise using one. The roadmap brings together the principles of business vision and strategy, documentation, and communication, without which development projects can easily go off-track.

Instead, we recommend the iterative approach because of its focus on what needs to be built next and its more flexible, “experimental” nature, meaning that it allows for more ongoing discussion and adaptability to changing requirements.

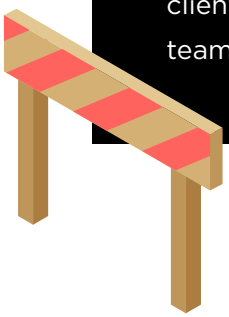
The main prerequisite for iterative development is a clearly aligned and well-managed system architecture. Following the steps we’ve already outlined should already go a long way towards meeting those conditions, but it’s worth mentioning that the Laravel framework, with its built-in flexibility, aligns itself well to the architecture adjusting course along the way.

In this way, Laravel and the iterative approach to development work well together for their ease at mitigating and managing change throughout the process.



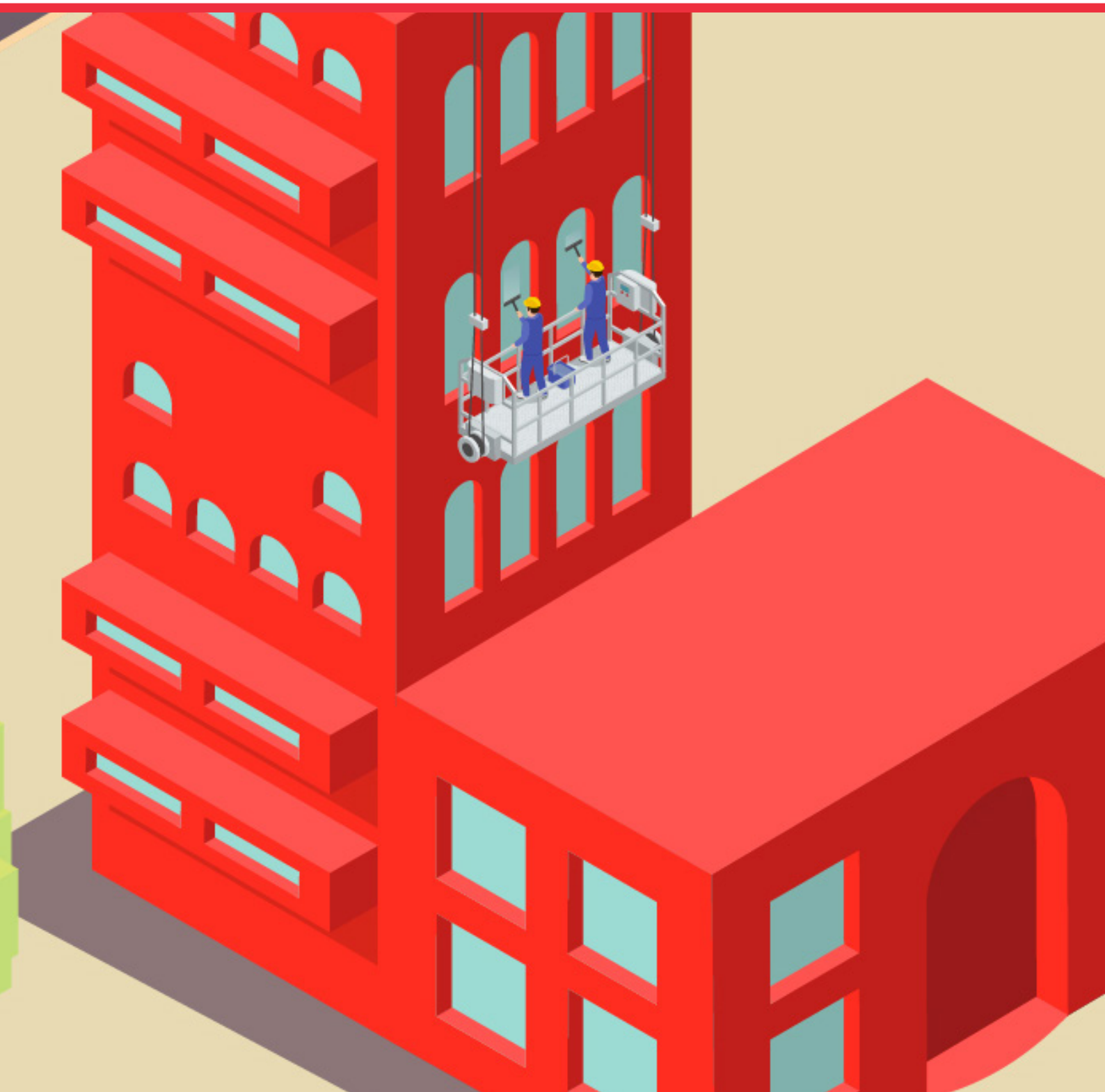
Identifying roadblocks ahead

A good project example highlighting the success of this approach was for a large manufacturing product owner who came to us with a well-defined 12-month roadmap, enabling us to identify dependencies between modules in the project. And were so able to explain to stakeholders early on about the need to build a new Single Sign-On feature before they could consider integrating the new CRM they were prioritising. It was the most efficient technological solution, which enabled the client to better prioritise its business roadmap, giving them enough notice for all teams to be aligned to make the new features successful.



> ***We recommend the iterative approach because its more flexible, “experimental” nature, allows greater adaptability to changing requirements***

8. Maintenance pays dividends



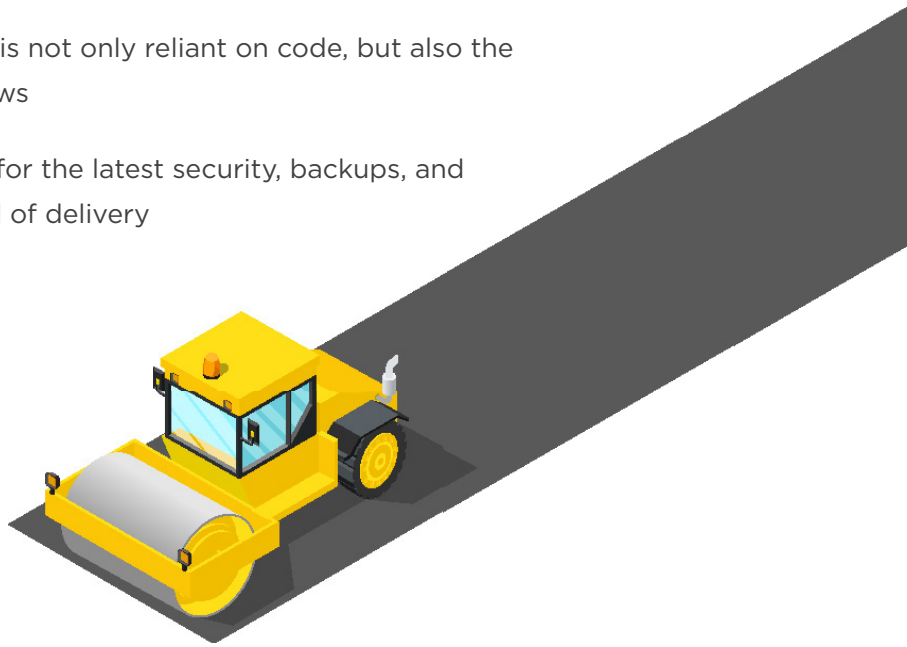
We make a distinction between a project's software upgrades (addressed in insight 6) and its wider ongoing maintenance and environment. Generally speaking, this means the project's security, hosting, performance, and other fundamentals, to which we add a monitoring category to further optimise efficiency. All of which we tackle through our DevOps approach, underpinned by a commitment and adherence to the Open Web Application Security Project (OWASP) secure design principles.

Because software decays with time, maintenance can be summarised as carrying out a regular code audit. Software decay is also known as code rot, or software erosion, and happens because the code simply becomes less responsive to the changes in its environment. This is not to suggest that the decay is physical but refers more to its age and age-related quality.

Of course, the more code, the more maintenance required, and the longer this code is left unchecked, the worse it gets and the more time-consuming it becomes to improve its quality.

The maintenance areas we focus on are:

- Application architecture, which involves checking and improving the system architecture in line with the business and product strategy
- Code readability to unlock further improvements and speed for the development team
- Test automation, to create and use the most cost-effective testing for the previous two points, and to better support manual testing
- Product documentation, which is not only reliant on code, but also the high-level strategy and workflows
- Recurring framework upgrade, for the latest security, backups, and features, as well as faster speed of delivery



As mentioned in the section on upgrades, consistency is what makes maintenance work. We see too many companies defining scalability as constantly adding new features without paying attention to the maintenance. Effectively companies are building on top of a steadily decaying foundation until the foundation itself fails and the software grinds to a literal halt. In one case, we saw a client shut a product down for nearly 5 months while the infrastructure was upgraded.

In this case, these companies end up boxing themselves into a situation where they spend 80% of their time repairing their systems and only 20% in delivery. Our recommendation of regular maintenance as part of the development process results in a ongoing balance of 80% development and delivery with only 20% on maintenance. So what might seem like cost-savings not undertaking continual and preventative maintenance, always incurs a greater cost in the long-run. The flipside of this when there is a robust roadmap that maintenance can align with. If new features are planned, maintenance and improvement of existing functionality can be carried out in advance, making a far easier implementation or integration when delivering them.

To be more time-efficient with maintenance, we include as a standard – and recommend including – monitoring tools. Specifically, these monitoring tools visualised into dashboards allow us to easily keep an eye on the application's performance and any errors that might affect its performance, which allow us to be more pre-emptive and target our maintenance priorities.

Maintenance is a time-consuming but necessary task, not unlike doing the laundry or maintaining a car. It's always made easier by performing it regularly, which keeps its demands smaller and more manageable. More than anything else, however, it keeps the product healthy and scalable and responsive to changing business requirements.

> ***"What might seem like cost-savings by not undertaking continual and preventative maintenance, always incurs a greater cost in the long-run."***

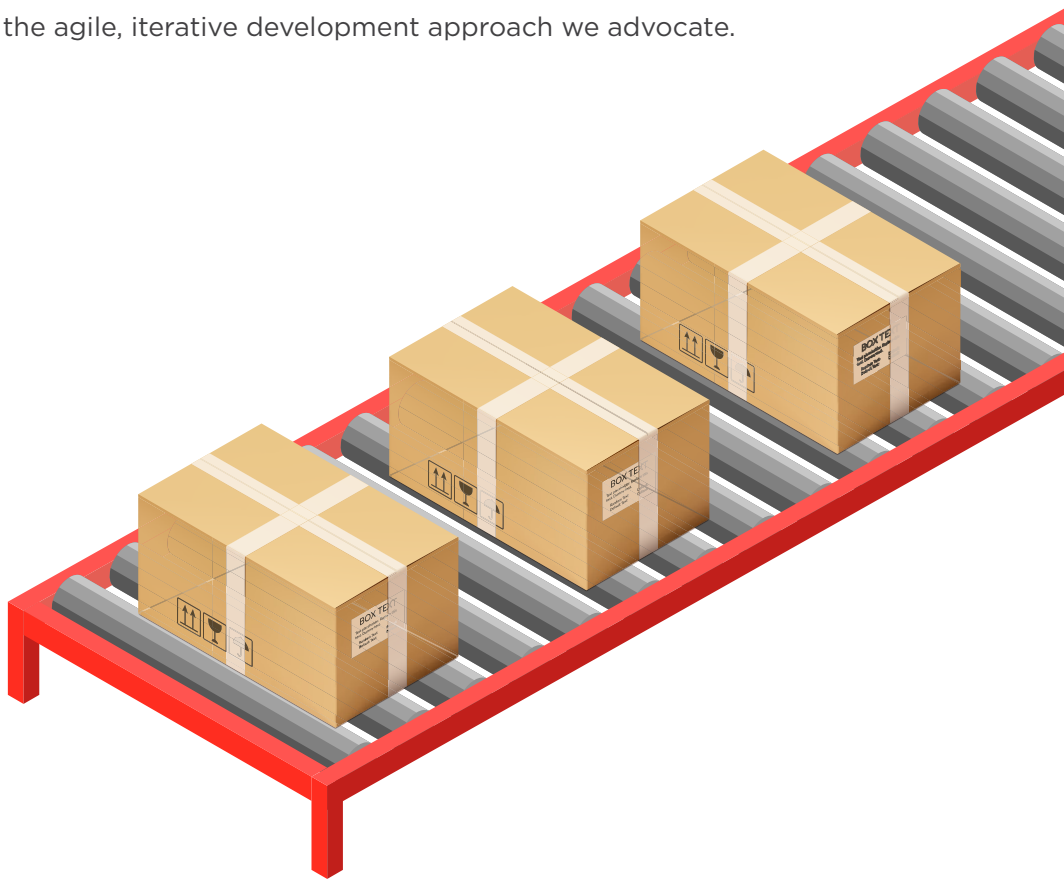
9. Automation adds value – to everything!

The most commonly advertised feature in software these days is automation. The listed benefits are streamlined processes, time savings, and reduced cost and human error — automation in product development is no different.

From a technical point of view, we talk a lot about testing automation. Between functional and non-functional tests, there are at least a dozen types of tests that development teams need to carry out, like interface testing, security testing, stress testing, and regression testing, to name but a few.

The opposite to automated testing is manual testing, where a developer, instead of a tool, tests the software step by step, without the use of test scripts. While both approaches have their strengths and weaknesses, manual testing is notoriously slow and tedious. For one client, for example, we had to carry out almost 1,100 tests, where each test held multiple assertions. For another, every one hour of developer work required four hours of testing, so it becomes obvious why automating that element becomes important.

There will always be a degree of manual testing, especially where the scenario being tested is more complex. It's also worth remembering that automated testing requires coding and test maintenance. That being said, the greatest advantage of automated testing is that it allows for more testing in less time, thereby increasing productivity, maintaining a sustainable delivery schedule, and better supporting the agile, iterative development approach we advocate.



A FinTech client requested a regression on its Laravel project, which took over two weeks with multiple people to test the entire application. The automated test suite runs in less than a minute and can test everything. Any bugs found in manual testing are then added to the automation testing suite to continually lower risk of bugs in production. Saving the business time and money that can be used to develop new features.

Automation, however, won't work properly if all the groundwork we've discussed isn't properly in place. Even down to the product roadmap and planning the testing ahead of time, the automation groundwork means having a clear understanding of the target user and the business vision, which provide a defined purpose around which the developers can write the testing code and schedule. It also means clear documentation and communication for transparency and problem resolution.

> ***"Automated testing allows for more testing in less time, increasing productivity and maintaining a sustainable delivery schedule."***

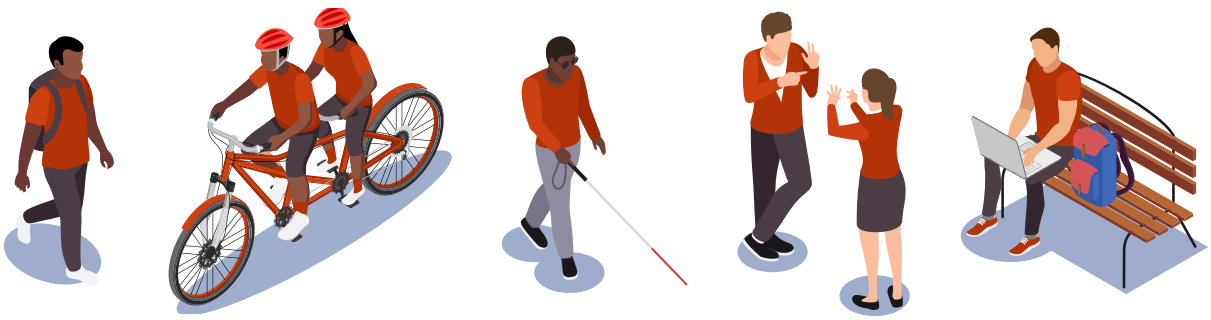
10. Being part of the Laravel community is a must

The benefits of support, learning, and sharing are perhaps obvious to anyone within a professional context in general. But in open-source development, community is absolutely vital.

Software and their development frameworks are only ever as good as the communities that support them. Specifically, developer communities give access to tools, tutorials, coding snippets, and even code reviews and bug discovery. These, in addition to the discussions and shared expertise, can actively improve in-house workflows and help solve problems faster, the result being reduced support costs and faster development.

As a high-level Technical Lead, but even as a non-technical person like a CMO or marketing team member, joining a developer community like on **GitHub**, **Stack Overflow** and **Reddit** can help understand the language, mindset, and thought processes, thereby further facilitating communication within the larger team. Our Laravel specialists immerse themselves in the various Laravel communities. **LaraChat**, for example, is a Slack community of close to 35,000 members allowing Laravel developers to discuss and help build a better PHP framework. **Laracasts** is a subscription service focusing primarily on providing Laravel development courses. There's also **Laravel.io** and **Laravel News**.

One of the reasons Laravel has such a large ecosystem is that it is one of the most used PHP frameworks in circulation. It has a substantial library of applications and packages that include authentication, server management, subscription billing, browser testing and automation, and more. The long-term success of Laravel projects rests on any developer or agency being immersed in such developments. Many of the team are Laravel Certified and we've used many packages from the open-source community.



We've also contributed new features or fixes to many of these over the years, as well as producing our own open-source packages and tools, including:

We've also contributed new features or fixes to many of these over the years, as well as producing our own open-source packages and tools, including:

- Docker image suitable for developing Laravel
- Mail testing
- Interacting with spreadsheets
- Address lookup and sanitation

For any Laravel project owner wanting to grow their Laravel project over time, knowing their development team is an integral part of its community is vital. The size and activity of the active Laravel community is not only a strong indication of the framework's quality, but also enables those who continue to participate in it to develop better, both now and in the future.

> ***"For any Laravel project owner wanting to grow their Laravel project over time, knowing their development team is an integral part of its community is vital."***

About Us

CACI is committed to designing, delivering, and operating innovative, high-quality, complex digital solutions through strong, collaborative partnerships that transcend geographical boundaries. This is firmly underlined by CACI Ltd's acquisition of Cyber-Duck, the original and premier Laravel partner in Europe.

Its entire team moved across to join the CACI Digital Solutions business unit in November 2023, bringing along with them a decade-plus expertise gained from using Laravel since version 3.1 to deliver over 35 clients' projects, including award-winning work for Worcester Bosch, Cadbury, Cancer Research Technology, Cabot Financial and Eurofighter Typhoon. Maintaining and growing long-term Laravel projects and client relationships through robust project delivery and open communication

This now 200+ strong – and growing - team of strategists, service designers, UX & UI creatives, software engineers, cloud infrastructure engineers and QA testers is focused on empowering people-centred digital transformation and experiences through the approach set out in this white paper. With additional scale and cost-effective delivery capability provided through CACI India's seamless integrated offshore software development offering.

The team regularly contributes to the Laravel community, producing open-source packages and tools that help with mail testing, interacting with Excel spread sheets, address lookup and sanitation, and **Docker image for a php-fpm container** crafted to run Laravel based applications, giving them improved capabilities in high traffic situations.

Laravel Partner

Acknowledgments

The following contributors from CACI were involved in the research and preparation of this white paper.

Laravel specialists

Gareth Drew, Chief of Digital Technology
Sylvain Reiter, Chief Delivery Officer
David Murray, Technical Strategist

And the wider team compos

Danny Bluestone, Chief Digital Officer
Joanne Bell, Brand and Content Strategist
Josh Smitherman, Visual Designer

and everyone at CACI who supports the team delivering Laravel projects.

CACI

www.caci.co.uk

Partner Since 2012

www.laravel.com

Revision control

This document was first published in February 2022.

This version (v.2.0) was released in March 2024 and contains the following updates:

- Updated copy following the acquisition of Cyber-Duck by CACI Ltd.
- A new insight 4. focusing on lean and sustainable development
- Additional examples of project work undertaken by CACI's Laravel specialists