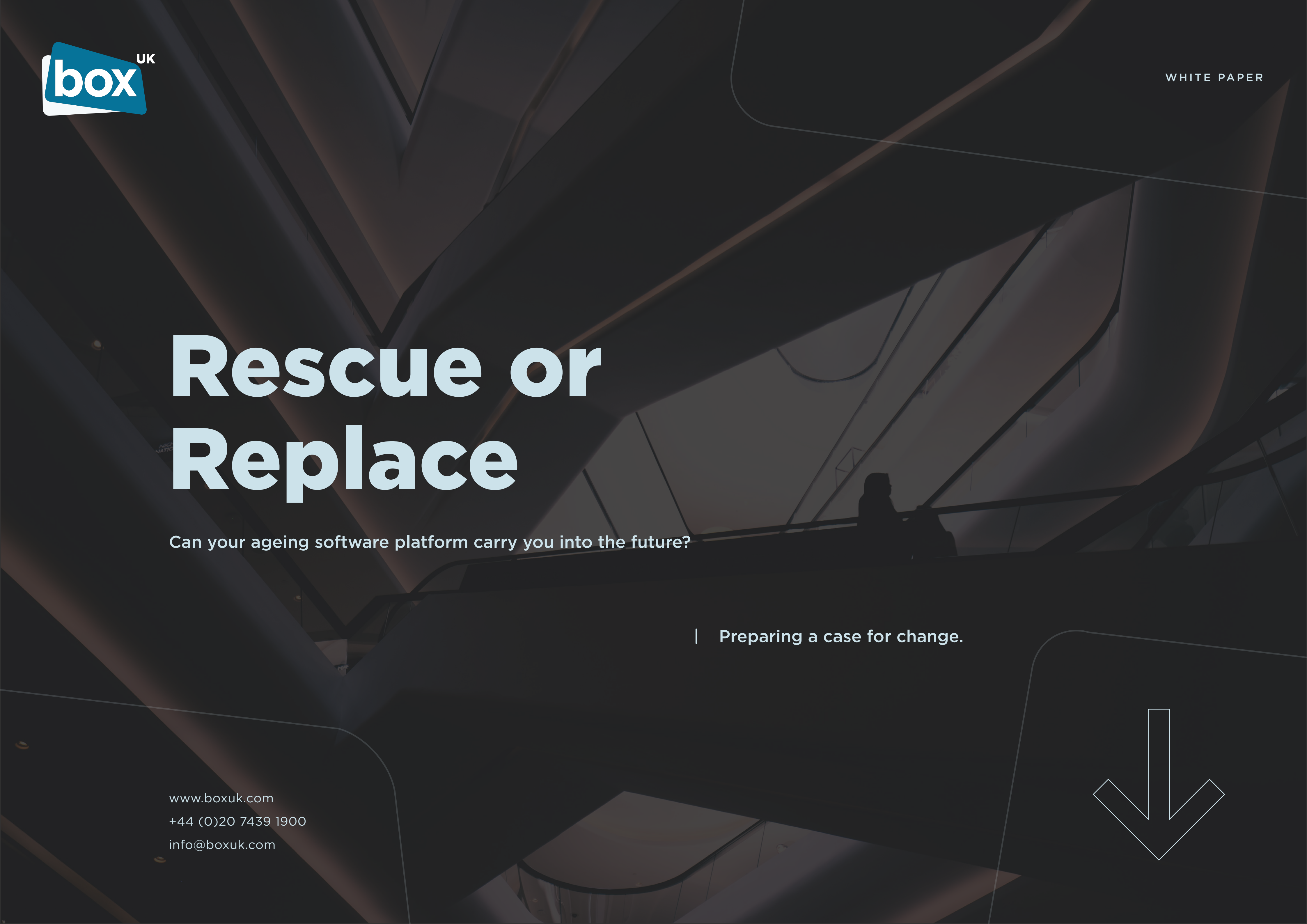


WHITE PAPER

Rescue or Replace

Can your ageing software platform carry you into the future?

| Preparing a case for change.

www.boxuk.com
+44 (0)20 7439 1900
info@boxuk.com

The First Stage of Tech Recovery

It's inevitable. As your codebase ages, problems start to emerge:

- System crashes and downtime, caused by an unstable platform
- Security and data breaches
- Missed targets, from development deadlines to KPIs
- Budgets overrunning, even in non-technical cost centres
- Low agility, making integrations a constant struggle
- Frustrations over usability, sapping productivity and morale
- Over-reliance on the few developers who can fix the code
- Disappointing customer reviews

These and other problems can impact the bottom line, through low productivity, lost opportunity and poor decisions caused by a lack of insight. Ultimately, it all points to one thing: technical debt.

This is not a phrase you hear every day, as it's quite an abstract concept. But every business accumulates technical debt to some degree.

It's simply a reality of software development –the outcome of balancing long-term quality with short-term deadlines. The trick, of course, is to strike the right balance, to keep the debt at an acceptable level. But that begs the question:

> **How much "debt" are you dealing with?**

That, perhaps, is the biggest problem of all – because this covert threat is so hard to quantify. Some of your colleagues may be aware of the symptoms, having witnessed the effect on logistics or customer service. But how can they make it a priority, without knowing the actual cost to the business?

Invariably, this lack of clarity puts the problem on hold. There are other priorities in the business, after all. For example:

- A new product launch
- A merger or acquisition
- Opening a new site, or a change of location
- Complying with new laws or regulations
- New dependencies, perhaps with new developers and languages
- An unanticipated surge in demand

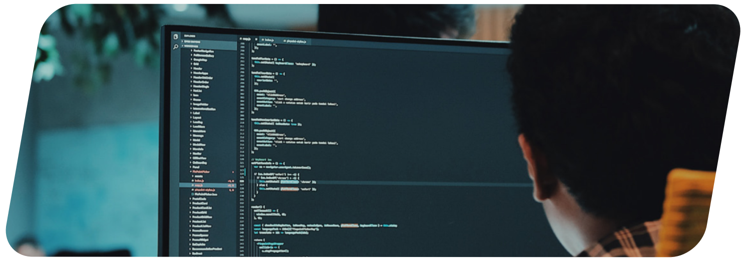
However, any one of these events will compound the problem by subjecting your software platforms to new complexity and stresses.

And even if the business stands still, performance will still dip over time, as systems degrade and new security risks emerge.

So in all likelihood, there will come a time when your platform can no longer cope and becomes a barrier to progress – at which point, it might seem you're left with just two options: refactor existing code or migrate to a new platform.

Refactor existing code

Assuming the code is maintainable, a skilled developer may be able to improve your code quality. But without a clean break, there's no guarantee you'll fix long-standing bugs or address gaps in functionality.



Migrate to a new platform

This sounds like the cleanest solution, but will it address all the issues you've identified? Not necessarily. You could even end up repeating past mistakes, if the root causes remain undiagnosed.

So what's the best way forward?

Well, let's stop making assumptions for a moment.

A warning light on your dashboard doesn't mean you need a new car – it means you need a skilled technician to run some diagnostics.

In the same way, signs of technical debt demand expert evaluation, in the form of an independent code review.

A review will bring the objectivity that's often lacking internally. It will take an impartial view of your codebase, and pinpoint the key areas to address.

In short, it will make technical debt visible – and actionable.

Then, and only then, can you make some educated decisions and prepare a measured proposal...

WHITE PAPER | RESCUE OR REPLACE

Getting The Answers You Need From A Code Review

Getting The Answers You Need From A Code Review

The performance of your business platforms obviously depends on the code quality that underpins them.

Just as good quality code can deliver efficiency and high performance, poor quality code can stall progress, limit functionality and ultimately eat into profits.

There are several reasons why your codebase may be underperforming:

- Business strategy has evolved, but the platform hasn't evolved with it
- Temporary patches have gradually turned into permanent fixtures
- You've inherited a legacy system from a team that has since moved on

Whatever the cause, a code review will provide the means to reduce technical debt and ensure systems are fit to serve the future needs of the business.

It happens in three key stages:

01 Setting The Ground Rules

02 The Code Review

03 Decisions: What Happens Next?

Setting The Ground Rules

Setting The Ground Rules

Before you start the process, the first question is always:

> **Why are we doing this?**

When a code review is commissioned, there's always a context behind it. You need to understand this clearly on day one, as it will shape the scope of the review – for both analyst and client. These questions will establish the context...

What has triggered this?

Technical debt can go undiagnosed for years, simply because no-one puts two and two together. Developers are aware of shortcomings in the system, and the C-Suite concerned over falling KPIs – but the correlation can be missed until a tangible problem brings it all into sharp focus.

So, what symptom has brought you to this point? Often, it's one of the following:

- A security audit has raised concerns
- Stakeholders have lost confidence in the maintainability of the codebase
- The CFO wants to capitalise software as an asset, and borrow against its value
- It's part of an annual audit – as required in certain sectors, like healthcare
- Your high-performing development team wants to validate code quality
- Or senior managers are unhappy with platform performance

What are the main concerns about your codebase?

The fitness of your codebase is critical to the performance of your key software systems. It needs to be reliable, maintainable, secure, and adaptable to changing needs.

So at the start of the project, which of these aspects is giving you cause for concern?

Maintainability

In theory, all code is maintainable – but is it viable in its current state, or has it evolved into something you can no longer manage?

Some evolution is inevitable. As a codebase ages, it becomes more difficult and costly to maintain, forcing developers to sacrifice standards for expediency. Then over time, a combination of duplicated code and workarounds will overwhelm the original codebase, forming an impenetrable maze.

When that happens, developers will get understandably nervous at the thought of making changes. Some may even give up trying to understand the code altogether.

> **At this point a serious question mark appears: has your code reached 'end of life'?**

Or can you still see evidence of simple, maintainable code? Such as:

- Short, concise functions
- Clearly defined responsibilities
- Strong naming conventions
- Good commenting
- Lack of duplication

There's no such thing as a perfect codebase, of course. But each time these rules are broken, it becomes less viable.

Security

Obviously, security and data protection are high priorities for your business. The risks have been heightened in recent years by the threat of litigation and ICO fines – which can be anything up to €20 million, or 4% of annual turnover.

The problem is, these risks can be hard to monitor, especially in older, legacy systems. But left unremedied, they soon become a danger to the company, leaving you at the mercy of data thieves and cybercriminals

So the question is:

> **Are there vulnerabilities lurking, unidentified and unresolved, in your codebase?**

It's a serious concern, because the range of security threats is growing daily:

- Session hijacking and other 'man-in-the-middle' attacks
- Phishing and spear-phishing
- Theft of passwords and other personal data
- Cross-site scripting attacks
- Malware, such as spyware, file infectors, worms and ransomware

Any of these could have devastating consequences for your business.

Confidence

Have stakeholders lost trust in your existing platform, or an ongoing development project? Perhaps releases are buggy, or lead times for change unfeasibly long.

Or has your development team lost faith in the technology used? Perhaps estimation has become a challenge, or sprint velocity targets are being missed.

It's a spiral that quickly erodes confidence...

Adaptability

With applications constantly evolving, your systems need to be lean and versatile. They need the flexibility to support not only immediate changes, such as user accounts or CRM integration, but also more innovative developments, such as:

- Virtual and augmented reality
- Voice user interfaces
- Biometrics
- Geo-targeting
- Wearable technology
- Social media command
- 3D printing
- Blockchain

Do your software platforms have the agility to keep pace with all these new developments, and more?

What are you aiming to achieve?

Desired outcomes will vary, depending on the role of the stakeholder. A developer, for instance, will be aiming for bug-free, well-documented software that complies with coding standards – meaning new features can be developed with ease.

Senior management, on the other hand, will be more concerned with the wider business implications: profit, productivity, customer satisfaction and growth. So their desired outcomes might include:

- Maintainability, allowing new developments and enhancements
- Improved code quality, reducing costly rework
- Better planning and more reliable estimates
- Shared knowledge, removing ‘single point of failure’ syndrome
- Defensive programming strategies, safeguarding security and data
- Stable performance, boosting morale and customer satisfaction

Of course, you’ll want to gauge the success of these outcomes. As ever, the best measures will be financial – numbers that can be directly compared with previous periods, such as monthly spend, order levels and customer feedback.

But a code review is not necessarily an end in itself. You might also be considering follow-on projects, such as improving the user experience or implementing new Agile working practices.

Whatever the scope, now is the time to set clear deliverables, so you can measure success and manage expectations throughout.

All of this context is vital in setting the ground rules for the task ahead. It’s where the true value of a code review lies.

Sense and Sensitivity

One final caveat before you start: a code review can be a sensitive issue. Done without due care, it can harm relationships – so it’s important to tread carefully.

Remember, a code review is not about blame. It’s a unique opportunity to improve code quality, establish best practice and share knowledge.

WHITE PAPER | RESCUE OR REPLACE

The Code Review

The Code Review

Once the ground rules are agreed, the review team will have the context needed to run the code review - for which the key elements are the tech terms of reference, a static code analysis, the human element, and of course the report.

Tech Terms of Reference

The technical analyst's first aim is to understand the scale of this task. So they should start with a fact-find, to identify technologies used, documentation and any third-party dependencies. For example, they might explore:

- Plans, schedules and work agendas
- Database schemas
- Continuous integration policy
- Build environment and deployment pipelines
- Issue tracking systems
- Uptime reports
- Bug reports
- Statistics
- Test coverage

You may need to analyse more than one codebase – for example, a front-end web app and an admin application, with shared dependencies. In which case, you should furnish information on all three.

Even if dependencies don't form part of the brief, the analyst will need as much detail on them as possible, so they can assess the quality of integration.

In short, then, it's not just about the source code. You should aim to provide everything that could help the analyst understand how your system works. The more supporting information you supply, the more comprehensive your report should be.

Static Code Analysis

Static code analysis is an automated process that examines your source code, without actually executing the programs, and reports potential flaws. It's a sophisticated operation, requiring the skills of a highly experienced technical analyst, whose aim will be to assess:

Code metrics:

- Size
- Complexity
- Dependencies
- Inheritance
- Maintainability

Bugs:

- Runtime errors
- Missing commands
- Incorrect syntax
- Logical inconsistencies
- Control flow issues

Vulnerabilities:

- Security violations, like nested passwords
- Weak cryptography algorithms
- Unauthorised access to sensitive data logs
- Shortcuts to high-level database settings
- Application backdoors

Coding standards:

- Do they meet industry standards?
- Do they follow design patterns?
- Are they well-commented, showing intent?
- Are dependencies and frameworks up to date?
- Is test coverage adequate?

To assess these areas, there's a wide range of powerful tools at the analyst's disposal. Some span multiple languages and code quality issues, while others are aimed at specific languages and aspects of code quality – for example:

- PHPMetrics provides code metrics
- Sensiolabs Insight reports bugs
- Nessus identifies vulnerabilities
- PHPCS assesses coding standards

The length of time it takes to run this analysis will vary, depending on the scope, tools deployed, and codebase size. Typically, however, you should expect to see results in 1-2 days.

But this is just the start of the code review process – the gathering of clues for the investigation that follows. The hard work really begins with the interpretation of that data.

The Human Touch

Automated tools have the advantages of speed, depth and accuracy, but they're not foolproof. They can produce false positives and false negatives. In some situations, a tool can report only that there's a 'possible defect'. And no matter how many tools the analyst deploys, there's still no guarantee they'll identify every issue.

In other words, while a static code analysis is a reliable measure of code quality, it can only tell you so much. The bigger issues can't be detected at the click of a button. To understand these properly, you need to go beyond technical diagnostics.

> **You need the human touch.**

This requires the expertise of a senior software consultant, selected specifically for their experience of your particular environment.

While the technical analyst completes their static code analysis, the software consultant should gather vital context to support the evaluation, by interviewing key stakeholders and reviewing documentation.

Discussion topics might include:

- Libraries and frameworks
- Design patterns and style guides
- Test coverage
- Scalability
- Structure

Depending on what you agreed earlier in the Ground Rules, this may extend to business processes beyond the code review itself, such as:

- Architecture review
- Agile working
- User experience
- Usability
- Business case and vision
- Backlog management processes
- Team structure
- Project governance

Transparency is vital for each element, at every stage. From preliminary discussions right through to presentation of the report, you should have clarity and honesty – even when the truth is uncomfortable.

Especially when the truth is uncomfortable.

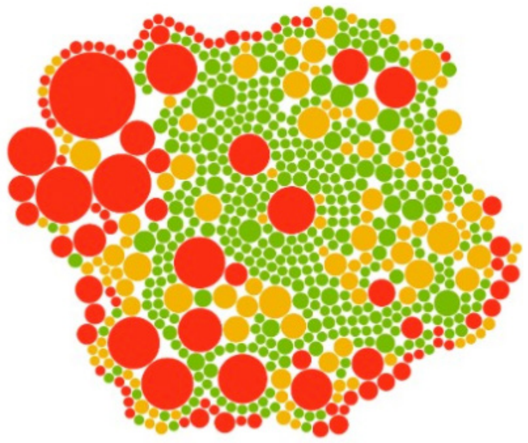
The Report

The final stage of the code review is for the technical analyst to collate the various findings and present them in a report – in a way that's both honest and accessible. Covering everything from code metrics to platform strategies, this is an intricate task.

The report must cater for two audiences – the development team and the C-suite – so the analyst should include an executive summary as well as technical results. The final picture should be an unbiased, forthright appraisal of the issues affecting your business platforms, including:

Visual representation of code quality

The report should include high-impact images: charts and graphs can provide visual illustrations of your codebase, showing findings in a logical, readable format. For example, bubble charts can be used to highlight code hotspots. Small, green circles represent good code quality, while larger, red circles denote areas for improvement.



Based on metrics drawn from 31,000 lines of code, the above example shows reasonable code quality with some areas for improvement, alongside the following observations:

- 6 months of technical debt
- Potential security risks
- Industry best practices not followed
- Third party frameworks not leveraged
- Lack of test coverage
- Lack of documentation

Visual elements are helpful to any audience, but they should also serve a practical purpose, allowing you to drill down to pinpoint problematic files – or further, even to the problematic lines within a file.

Code performance

The static code analysis will quickly reveal specific issues that give cause for concern, such as:

- Code repetition
- Bugs
- Coding issues, such as buffer overflow, memory leaks, and null pointers
- Usage of obsolete libraries or frameworks
- Unnecessarily complex code to perform basic routines
- Unmanaged dependencies
- High coupling and low cohesion
- Poor version control
- Naming conventions not followed
- Lack of commenting

Not only will these factors impair code performance, they may also have a damaging effect on team morale. Which in turn, affects coding standards.

> **In other words, a vicious circle.**

Could poor code performance be an indication of more fundamental problems? Consider:

Best Practice

Failure to follow industry best practice is invariably the precursor to poor coding standards. (There may be sound reasons for applying different standards or design patterns, so the analyst should be wary of any cognitive bias.)

Training

Lack of training and inexperience lead to a spiral of uncertainty and delay. The more developers deliberate over a task, the less they focus on its purpose – meaning efficiency and even functionality may be lost.

Deadlines

Unrealistic deadlines create unreasonable pressure. This incentivises developers to cut corners, ultimately eroding confidence in the codebase.

Documentation

Poor documentation leads to poor quality. If style guides are incomplete or out of date, developers will start to do their own thing – undermining standards and creating the perfect environment for a 'blame game'.

Knowledge Sharing

When knowledge isn't shared, best practice becomes the preserve of a few, and standards begin to erode.

Test Coverage

Without adequate testing, issues can rise to unmanageable levels. Then technical debt could become technical bankruptcy.

Issues and risks

The code review report should provide a clear, prioritised summary of the issues and risks identified by the static code analysis, including:

Libraries & Frameworks
Patches, versions, long term support.

Design Patterns
Structure, classes, objects, behaviour.

Code metrics
Size, complexity and depth of inheritance.

Coding standards
Best practice, custom conventions and violations.

Testing
Strategy, speed, coverage and evidence.

Dependencies
Versions, robustness, security practices, libraries, languages, custom code and testing.

Structure
Maintainability, speed of development, onboarding of new team members and documentation.

Security
Authentication, authorisation, session management, data validation, error handling, logging and encryption.

Consider these findings in the context of your original codebase concerns: maintainability, security, confidence and adaptability.

Is your codebase maintainable?
The question has huge implications: if the answer is yes, you'll have options to develop it further; if not, you'll have tough choices to make.

Are there security vulnerabilities in the codebase?
According to Herjavec Group's 2019 report, cybercrime is expected to cost the world around £5 trillion in 2021 – double the figure for 2015. Attackers are armed not only with new technology and faster computing speeds, but also with an ever-growing encyclopaedia of system vulnerabilities.

Can you rebuild trust in the codebase?
A tech strategy with consistent quality standards and realistic deadlines will go some way to restoring confidence.

Does your codebase have the scalability to adapt to a dynamic market?
Compliance with industry best practice could be key, if your codebase is to have the scalability to take advantage of future software innovations.

Recommendations and roadmap for future development

Investment in technology is a decision fraught with risk – risk often obscured by the distractions of doubt and assumption.

The conclusions of the code review should provide sharp relief, with a clear set of recommendations.

For example:

- Strategies for moving away from legacy systems
- An actionable risk matrix of security issues
- A plan for ongoing maintenance of the development pipeline
- Proposals for stabilising and future-proofing infrastructure
- Guidance around the necessity of any disaster recovery review
- Advice on establishing best practice development standards
- A plan for implementation of test strategies – from unit tests to full integration tests
- Looking beyond the scope of the code review, to options that might enhance existing performance, such as an information architecture review or content audit

> **Building on these, a roadmap will provide further clarity, giving definite directions for the way forward.**

A good report should cover all these bases, so you can take a critical view of your technical debt. Then build a long-term tech strategy that will:

- Improve maintainability of your codebase
- Reduce risks and vulnerabilities
- Boost developers' morale
- Restore the trust of managers and users
- Increase agility of your platforms

In short, create a software platform that's perfectly tuned for the growth of your business.

What Happens Next?

What Hapens Next?

Now you have your code review report, you have three key questions to consider:

1. Did the review meet its objectives?
2. What's the way forward?
3. How do you get other stakeholders onside?

Did the review meet its objectives?

With the report in your hands, you now need to make sure the objectives of the code review have been met. A checklist might include:

Codebases
Does the report analyse all codebases identified in the tech terms of reference? Does it cover all relevant dependencies?

Issues
Does the report highlight all risks and issues identified by the code analysis? Has it excluded all bias? Are all findings from consultant research documented?

Recommendations
Does the report address all risks and issues? Does it respond to initial technical and executive concerns? Does it relate to desired outcomes? Is there a roadmap for the way forward?

For the code review to meet its stated aims, you should be able to tick all these boxes.

Not your experience?

This has been an 'ideal world' picture of the code review process. Sadly, though, it's not everyone's experience. Perhaps you've been disappointed by a poor code review in the past – where an analyst promised the world and under-delivered?

If so, you're certainly not alone. It seems the most common grievances are:

Stating the obvious
Telling you what you already know, or making simplistic recommendations

Academic exercise
Tcking technical boxes, but with no regard for the C-suite

Lip service
Agreeing to address your concerns, then simply going through the motions

Artistic licence
Finding non-existent faults in a thinly disguised sales exercise

Poor diplomacy
Blunting the morale of your development team

These things can happen, in an industry that's often careless or blind to best practice. So your choice of independent reviewer is critical.

What's the way forward?

Reduced line counts, up-to-date dependencies and other technical objectives are all perfectly valid – but they might be of little interest to your senior colleagues. They'll be more concerned with the wider business implications: profit, productivity, customer satisfaction and growth.

So, let's consider the tangible improvements you might be looking for as a business:

- Larger customer base
- Higher average spend
- Increased order levels
- Improved click-through rates
- Reduced maintenance costs
- Higher uptime levels
- New revenue streams
- Better customer feedback
- Boosted share price

Obviously, these are common goals for any business. But now, armed with the code review, you have a clear advantage over the competition: a chance to create an action plan that will make those goals more achievable.

> **So what kind of work will feature in that plan?**

- Wrapping a new third-party library around existing code to improve functionality?
- Consolidation of existing platforms?
- New coding standards, style guides and version control?
- Implementation of Agile working practices?
- User Experience review, to increase end-user engagement and loyalty?
- Migration to a cloud-based platform, and development of new apps?

- Or perhaps a combination of several tasks, as part of an overall strategy that paves the way for full migration.

Whatever your decision, ultimately it should be based on the best value outcome for the business. So keep an open mind and be ready to challenge preconceived ideas.

How do you get other stakeholders onside?

For the C-suite, technical debt has probably flown under the radar – until now. But the code review will change that, by giving the debt a context and a tangible business impact.

> **Be aware, though – some of the findings may prove controversial.**

The report may pose uncomfortable questions about your business functions – like working practices, development standards, or even organisational structure. A delicate touch may also be needed if a key stakeholder sponsored or project-managed the original development.

> **So once again, sensitivity is crucial.**

As we've already said, the reviewer should tread carefully and avoid blaming individuals. And now, as you escalate their findings, you'll need to follow the same rules – stressing that defects are always structural, never personal.

That said, diplomacy should never override honesty. If the report makes for tough reading, then so be it – remember, both parties agreed at the outset that transparency should be an integral part of the process. Senior managers want the unvarnished truth, not soothing platitudes.

For the report to be of true value, it's vital to stick to these principles. Only then will it have the authority you need to make a business case for change.

Then, at last, there could be light at the end of the tunnel.

Next Steps...

Your codebase may have delivered robust, efficient performance for years but...

✗ Has it become a barrier to progress?

✗ Are you stuck with legacy hardware, understood only by a select few?

✗ Are integrations now a perennial struggle?

✗ And let’s not even get into the security risks...

This is the harsh reality of technical debt. No business is immune. But you can take steps now to turn the tide.

Your first priority is to make your technical debt visible, and actionable. To do that, you’re going to need an in-depth, independent code review. And for that, you’ll need an expert partner you can trust. A partner who values quality and transparency, with an outstanding track record.

Why Box UK?

Specialists in enterprise software development, Box UK's track record of success spans hundreds of high-priority and mission-critical software projects, delivered over more than twenty years.

Our developers have performed code reviews for clients including RS Components, Sodexo and Jaguar Land Rover, and we can do the same for you too. Plus, whether your code review signposts a legacy system rescue, new web platform or upskilling in development practices, with the diverse skills of our team we're always ready to respond.

So, what might a Box UK Code Review look like? To learn more, simply request our Product Sheet.

www.boxuk.com
+44 (0)20 7439 1900
info@boxuk.com

What can you expect when you work with Box UK?

Expertise
We recruit from purposely diverse backgrounds to offer an unrivalled depth and breadth of experience that sets us apart from the competition, delivering digital solutions that push the boundaries of creativity and innovation.

Partnership
At Box UK we work with you to understand the evolving needs of your business and agree shared goals – building robust client relationships, many of which span 10 years or more.

Agile working
With a process that's geared towards maximising productivity and eliminating waste, projects are delivered faster, without any compromise on quality.

Quality
Quality sits at the centre of everything we do, as our ISO 9001:2015, ISO 27001:2013 and ISEB accreditations demonstrate.

High coding standards
Based on leading development standards, best practices, peer review, automated testing and deployments, the exceptionally high quality of our code has been recognised by numerous independent reviews.

Some of our work

When our code review identified performance and security issues in Sodexo's Employee Benefits codebase, we stabilised the existing platform, then migrated over 300 legacy sites to a new high-performance solution.

The new platform supports thousands of clients, and hundreds of thousands of end-users – maintaining an impressive 99.9% uptime since go-live.

"Box UK is a fantastic partner for our business... providing proactive analysis and recommendations to the challenges we face."

James Allan, Head of Product Management, Sodexo

When RS Components set out to improve DesignSpark, their online community platform, our code review revealed high levels of technical debt, built up over many years.

Our developers transformed the platform – generating a huge increase in engagement, and more than doubling DesignSpark's membership to over 750,000 engineers worldwide.

DesignSpark also contributes directly to revenue targets now, with third-party advertising generating additional revenue.

"The Solutions Team were totally blown away with what we have achieved so far – it more than meets our expectations and takes us to a new level."

Dave Cole, Web Platform Manager, RS Components

Keen to know why Sodexo and RS Components rate us so highly? Request our Code Review Product Sheet, and we can start that conversation...

Box UK specialises in the design, development and delivery of large-scale digital platforms that are relied upon by millions worldwide, with a track record of success spanning more than twenty years.

Our multidisciplinary team act as partners to global clients including Jaguar Land Rover, Sodexo, the British Medical Journal and RS Components, delivering impressive programmes of work that elevate and accelerate their digital growth.

Across hundreds of high-profile projects, we've gained a reputation for quality that's recognised at the highest levels – with a host of awards from leading industry bodies, including the Webbys, Communicators and UK Digital Experience Awards.

www.boxuk.com
+44 (0)20 7439 1900
info@boxuk.com