BUILD SERVICES

Testing and QA

enable®

Introduction

For nearly two decades, companies have been relying on Enable to transform their strategic vision into high quality software. At the heart of this success is a committed approach to testing and quality assurance that ensures exceptional results every time.

This document illustrates the various types of testing and quality assurance activities that ensure Enable delivers the high quality software we are renowned for.

Our goal is for every project to set a new benchmark for quality. We learn from every engagement and feed lessons learnt into the next piece of work in a controlled environment while working to structured quality assured processes. This subtle but constant evolution ensures that we are moving forward while preserving all the elements of our robust methodology that clients choose us for. Our people have room to innovate and experiment within the structure of a highly disciplined approach.

Our software needs to incorporate a number of important attributes as it must conform to the requirements defined in the functional specification and be secure, reliable, efficient and maintainable while providing a high standard of usability for end users. Implementing the strategies described in this document helps to ensure the software we produce incorporates these vital characteristics.

Automated testing

UNIT TESTING

Unit testing is a software development process where programmatic tests are written to confirm the functionality of a small unit of code, which will usually be a component within a much larger system. The goal of unit testing is to take these small units of an application, isolate them from the rest of the codebase and determine whether the unit behaves as expected. In essence, the unit test provides a contract that the unit of code must satisfy in order to pass the test.

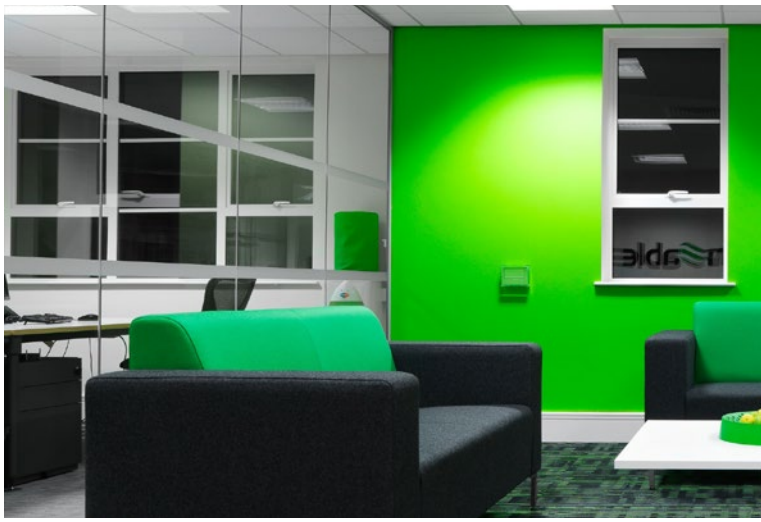
The benefit of unit tests are that they can be run automatically during development work or as part of a continuous integration process. These automated test runs can test in seconds what would normally take many hours with traditional manual testing. Not only are unit tests more efficient, they are also more accurate as they are not prone to human error. In many cases, unit tests can identify bugs before they even have the chance to affect end users. They also help to ensure that changes made to the unit at a later date do not break previously working parts of the unit.

Writing unit tests also greatly improves the quality of the code as a common mistake in programming is to create large units of code which contain multiple pieces of functionality. Software written in this way is generally more difficult to understand, maintain and modify, but unit testing, by definition, naturally encourages smaller units of code with more focussed responsibilities.

Unit tests are beneficial at all stages and not just the build phase of an application. For example, if a bug is discovered during support, specific tests for the bug can be introduced to make sure the same problem isn't reintroduced as the result of future changes.

Unit tests do introduce some overhead in the production of an application, however, during the lifetime of a typical application this initial investment can be repaid many times over by the time savings over manual testing and other benefits.

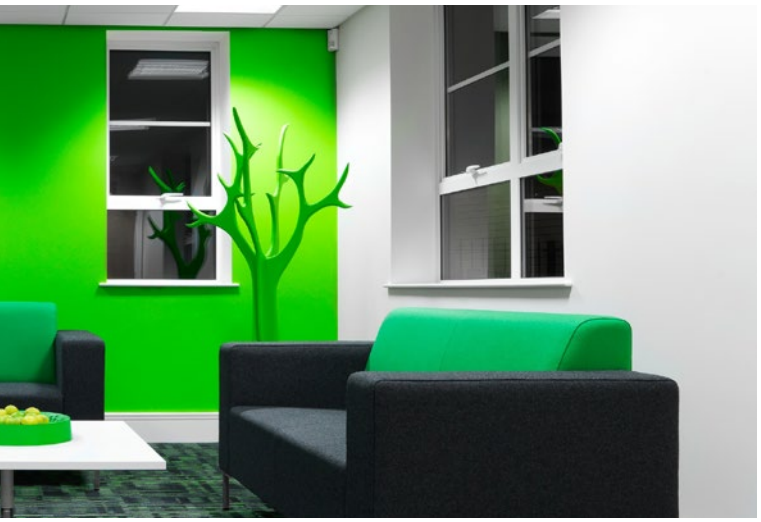
Unit tests also play a key role in test driven development, which will be discussed later in this document.



INTEGRATION TESTING

Integration testing verifies that different parts of an application work correctly together and, unlike unit testing, it involves several units of code operating together along with application infrastructure concerns, such as a database, file system, network resources, or web requests and responses. While unit tests use fake or 'mocked' objects in place of these concerns to test the unit in isolation, integration tests confirm that individual units work together with other units and infrastructure to fulfil functional requirements. Although integration tests offer many of the same benefits as unit tests, they usually take longer than unit tests to write, and also are computationally expensive and so take a lot longer to run than unit tests. For these reasons, we tend to only introduce a few carefully-selected integration tests for aspects of the system that really benefit from them.

Integration tests allow us to be confident that changes have not introduced bugs and that the system is working in accordance with the functional specification. The key difference is that integration tests can catch the problems that unit tests can't, such as issues with the integration of different units of code or integration with infrastructure such as a database. We use the top-down (outside in) approach for our integration testing which allows high-level logic and data flow to be tested early on in the process. This approach also has the advantage of consistency as the integration testing is performed in an environment very similar to the real application.



TEST-DRIVEN DEVELOPMENT

Test-driven development (TDD) is a software development process which builds upon the ideas of automated testing. This process provides measurable quality and confidence in the software along with a number of unit tests which verify the code that was written.

In TDD, a developer follows a cycle in order to create each unit of code. During this cycle, the implementation of the unit is only advanced in response to the creation of unit tests and, as the name suggests, unit tests are used to drive development. TDD follows a process known as "Red/Green/Refactor" which consists of these three stages:

— Red: a unit test is created which describes a small part of the unit's desired functionality, and tests are run to verify the test is failing. This validates that the test is working as expected (as the unit's functionality has not yet been added, meaning the test is expected to fail).

— Green: just enough of the unit is implemented to make the test pass.

— Refactor: once the test passes, the current implementation of the unit is assessed for any potential improvements. These might involve removing duplication or improving the readability of the code. Unit tests are also run again to ensure that all the tests still pass after refactoring.

This cycle is repeated several times until all the requirements of the unit have been met and all the unit tests pass. This approach has many benefits including:

— Faster feedback: our developers get immediate feedback on their implementation. There is no need to run the entire application to verify one particular feature works as expected.

— Better unit tests: by writing tests first and only writing code to pass these tests, we can ensure that all requirements are covered by unit tests.

— Better design: TDD leads developers to think about code in small units. Code written via TDD is often more extensible and easier to maintain.

Specification testing & validation

THE ROLE PLAYED BY SPECIFICATIONS

At Enable, we consider it essential that every delivered project fits the client's needs exactly. All our time and resources are devoted to fulfilling the user's requirements and there are a number of approaches to validating new functionality against the project specification to ensure validity is an integral part of the project process from start to finish and not limited to the testing phase alone.

CONCRETE OBJECTIVES

A deep understanding of the specification is vital from the outset, before any development work starts, and clarifications are requested from the client if any requirements are not clear. Ensuring the objectives are concrete before any technical changes are made to the system reduces the amount of development time spent during the user acceptance testing (UAT) phase and ensures that any adjustments more closely align with the client's needs.

For projects with multiple phases, each specification is always considered in the context of previous specifications. Requirements which are not altered in a further specification are brought forward and the system must be valid against these. Previous specifications are referenced if there is any uncertainty about how new implemented functionality should cooperate with existing functionality, and further clarifications are requested if there is any doubt.

The development process emphasises to developers the need to adhere to the specification from the start, as it is the context in which all planning and implementation is performed.

DIVISION OF RESPONSIBILITY

Each developer is assigned tasks from a separate section of the specification and planning is done to ensure that they can be worked on independently to minimise potential bottlenecks. These tasks are allocated based on the developer's strengths and areas of expertise while ownership over an area allows for in-depth knowledge into it and a robustness as the related points are carefully worked on in tandem.

Before work begins, the team lead will go through the specification and discuss with each developer the technical approach. In addition, team members are allocated time to familiarise themselves with the specification and are expected to maintain an understanding of parts of the system that they do not implement themselves. This promotes an understanding of the assigned task within the greater context of the whole application as developers are mindful of how their changes will impact the system as a whole.

AUTOMATED TESTS

Each solution includes a suite of automated tests that validate the behaviour of various components of the system while every project factors in time to allow developers to implement further tests for any functionality they add or modify.

The tests are automatically run each time a change is submitted to ensure that developers are alerted if an alteration breaks existing functionality. Although modifications to the system are carefully made so as not to affect other areas, it is inevitable that some bugs will result and the tests help safeguard functionality not just technically but conceptually against earlier implementations.

APPLIED EXPERTISE

While strict adherence to the specification is important, the development team endeavours to interpret any ambiguous areas in the specification to maximise applicability to the user's needs. The primary example of this is through the attention taken to user interface design as the optimal user experience cannot always be conferred through words. In this instance, the Enable team will use its expertise to plan and deliver functionality that meets the user's business requirements.

MANUAL TESTING

A substantial amount of testing time is designated for each project with the development team working in a concerted manner to analyse all new implemented parts of a system to ensure technical soundness and conceptual validity. Time is allocated for every area of the specification to be tested by a developer separate from the one who implemented the area, with any bugs logged in detail and passed back to the original developer for resolution.

Only when the developer who raised the bug has verified that it has been fixed is the bug closed while each bug references the area of the specification it is related to and lists any other specification points that are affected. Each alteration to the code is mapped to the task that detailed the issue — every change must be justifiable and is efficiently tracked.

Where possible, developers are asked to test within the same testing environment; an internally hosted version of the site that mirrors the system that is delivered to the clients for the UAT process. Where this is not possible, each developer must periodically update their local copy of the system to reflect any recent changes which have been made.

The testing environment is populated with test data in order to reflect the nature and volumes of the data that are likely to be encountered by the client in everyday use — as outlined in the specification. This is performed using a propriety software package that is capable of producing realistic data for any number of required database records. This is especially important for performance analysis conducted to ensure that the system is able to process data effectively and deliver the specified user experience.

Each developer will test not only that the technical implementation is sound, but also that the system conceptually follows what is specified. The developer will scrutinise all parts of their assigned area, covering as many user stories as possible as well as testing unusual data. The aim is to ensure that the system under test fully meets the specification and is as as robust as possible.

USER ACCEPTANCE TESTING

Although the aim is to reduce the amount of work that needs to be addressed in it, the user acceptance testing (UAT) process is a period that allows for bugs to be addressed within an environment reflecting everyday use by the client. Within this period, clients have the opportunity to request changes to any part of the system which does not match what was specified. This also allows for further refinement of the delivered project until the client is completely satisfied that it conforms to the specification.

Security testing

OWASP TOP 10

The Open Web Application Security Project (OWASP) is a not-for-profit charitable organisation focused on improving the standards of web application security. The project is an online source of expert information on the best security practices for web developers via freely-available articles, methodologies, documentation, tools and technologies.

The OWASP Top 10 is a regularly reviewed list of the current most critical aspects of web application security as decided by industry experts. The aim of this list is to bring these security risks to the attention of companies and developers so that they can be addressed by the industry. For each of the top 10 the list details:

— How the risk might be exploited by an attacker;
— Advice on how an attack can be prevented;
— A categorisation of risk in terms of its ease of exploit and the potential impact of a successful attack.

We regularly review all our applications against the OWASP Top 10 and ensure that all our developers are fully aware of the common known threats. When new entries occur we hold a companywide training session to make sure that everyone understands the risks involved and the actions that are required to avoid them.

PENETRATION TESTING

Web applications have become common targets for attackers as they can leverage relatively simple vulnerabilities to gain access to confidential information or even gain full control of the targeted environment. While traditional firewalls and other network security controls are an important layer of any Information Security Program, they can't defend or alert against many of the attack vectors specific to web applications. It is therefore critical for Enable to ensure that its web applications are not susceptible to common types of attack. By performing penetration testing before the site has gone into an externally accessible environment we can ensure the following common forms of attack are mitigated against:

— Cross-Site Request Forgery (CSRF);
— Cross-Site Scripting (XSS);
— Path traversal;
— SQL injection.

Our development team are highly conscious of web application security and use experience as well as tried and tested Microsoft technology at the core of each solution to mitigate against common attacks. In the past, we have found that use of these tools have uncovered some common vulnerabilities detailed above and once they have been identified we are able to take the best course of action to remove the security flaw. When the solution is in place, we share details of the newly uncovered exploit and how to guard against it with the entire team.

ENCRYPTION

Encryption is the process of rendering information unintelligible to unauthorised parties. The unencrypted information, or "plaintext", is encrypted by means of an encryption algorithm using a secret key. To a third party, the encrypted information is indistinguishable from random noise but to a recipient with knowledge of the algorithm and key the information is easily decrypted to reveal the original plaintext. When an effective algorithm and key are used it is only feasible for a third party to defeat the encryption by leveraging significant computational resources over often unrealistic timescales.

Information can be encrypted to prevent unauthorised use whether at rest or in transit. Data is encrypted when at rest to provide protection in the event that an attacker gains access to files either physically or over a network. Examples of this include encryption features available in some relational database systems or full-disk encryption used by the operating systems of portable devices. Encrypting data in transit involves encrypting information temporarily while it is transmitted across a network such as the Internet, a mobile phone network or Bluetooth, in order to prevent eavesdropping.

Enable routinely uses encryption techniques to protect data at rest and also to protect users' confidentiality when interacting with web applications. The most common form of encryption employed by Enable's systems is HTTPS, ensuring that aspects of web application communication, including the downloading of application code and the exchange of user information, are encrypted. An unauthorised third party observing traffic between the application and user would see a stream of apparently random noise.

HASHING

In contrast to two-way encryption functions - which render information unreadable until decrypted by a recipient in possession of the secret key - hashing algorithms are designed to be one-way and irreversible. An input of any length is processed by the hash function to produce an output (or "digest") of fixed length. The output is indistinguishable from random noise and bears no apparent relation to the input with the smallest change in input altering the output to the degree that it has no similarity to its previous value.

When an effective hash is used, it is infeasible for a third party to determine the original input without trying all possible inputs until a match is found. This, as with encryption, is likely to take an unrealistic amount of time to achieve. It is also infeasible to find two inputs that result in the same output – an occurrence known as a collision.

Uses for hashing include verifying the integrity of information (meaning whether the information has been altered) and verifying passwords — Enable uses hashes for both of these purposes.

The standardised SAML single-sign-on mechanism, employed in a number of Enable's applications, uses an XML signature — which contains a digest produced using a hashing function - to ensure messages are not altered. The integrity of the message can be verified by checking that the signature contains a valid digest. Many of Enable's applications authenticate users by requiring entry of a correct username and password combination. Storing passwords in plaintext is bad practice because a malicious third party that gained access to such a store would immediately have knowledge of all users' credentials and could then authenticate with the system as any user. It is far better to store the hashes of passwords because an attacker that gained access to them would have great difficulty determining the original passwords while the system can still easily perform authentication by comparing the hash of an entered password against the stored hash.

PUBLIC-KEY CRYPTOGRAPHY

Public-key cryptography involves pairs of one public and one private key. The public key can be shared widely with no adverse effect on security while the private key must be kept private by its owner. Information can be encrypted using the public key and only decrypted using the private key, allowing the integrity of a message to be verified by encrypting a hash digest using the private key and verifying the hash after decryption using the public key. By using this technique a message can be known to have originated from the holder of the private key, and unmodified while in transit, assuming the secrecy of private key has not been compromised.

Performance testing

DATA VOLUME LOAD TESTING

Data volume load testing is the process of putting demand on application databases in order to simulate anticipated peak load conditions and identify potential performance bottlenecks. Databases will be populated with large volumes of data in order to simulate the expected data usage.

It can be very difficult to simulate how a system will perform in production without realistic data. In the past, performance issues could often go unnoticed until a lot of data was added to the database at which point certain areas of a system would become virtually unusable. By proactively performing data volume load testing we can identify performance issues during development and immediately resolve them, ensuring that the system can handle these anticipated data volumes after a release to production.

The load testing will usually be done throughout the project build and, after finalising an initial schema, we will often pre-populate the application databases with realistic test data. The test data is generated using automated tooling (an example being Red Gate's SQL Data Generator) and allows us to ensure that the data generated is realistic and relevant to the various entities being populated. By using automated tooling we are able to quickly recreate test data if required - an action that would be unachievable in large scale systems if we were generating data manually.

Load testing throughout the project build allows us to design the system functionality to work with these large data volumes from the outset rather than having to potentially rebuild aspects of an existing system due to unforeseen issues.

USER VOLUME LOAD TESTING

User volume load testing is the process of putting demand on a software system by simulating large volumes of concurrent user actions in order to determine how the performance of a system is affected when it is under peak user load. By performing user volume load testing we can identify and resolve areas of potential poor performance early in the development process.

User volume load testing is generally carried out by a number of users performing a pre-determined routine of system actions and measuring the system response times. While it is possible to perform this process manually with real users, this is error prone and time intensive so we currently make use of automated tools (examples being West Wind WebSurge Load Tester and Apache JMeter) in order to carry out our user volume load testing. This tooling provides the development team with the ability to perform actions within the system and automate the steps before replaying them on their own or concurrently. These tools also allow us to precisely measure the response times which we can then use to identify and improve any areas of the application which are underperforming.

User volume testing will often be combined with data

volume testing in order to get an accurate idea of how a system will perform in a production environment. By having a full set of realistic data and numerous users we can stress test the system to get a more authentic idea of how it is likely to perform in a production environment.

APPLICATION PROFILING

Performance is an important factor of any application as it should be as responsive and reliable as possible to ensure a high level of usability.

Application profiling is the process of using tools to monitor an application in order to identify any areas that require optimisation so that performance and reliability can be improved wherever possible. Examples of areas that need to be analysed include memory usage, CPU usage and duration and frequency of function calls.

There are several application profiling tools that we use during development to monitor and identify any places where optimisation would improve performance. These allow us to target different areas of software execution such as .NET code execution running on the server, SQL queries running on the application database and JavaScript code running on the client.

MICROSOFT SQL SERVER PROFILER

SQL Server Profiler is a profiling tool that we use to monitor SQL Server databases and capture data about events that can later be analysed. Examples of these events include connections, disconnections, the start and end of SQL statements and the execution of stored procedures.

SQL Server Profiler allows 'traces' to be created to capture the types of events we are interested in while filter criteria can be applied to restrict the event data that is collected when the trace is run. For example, a filter could be used to capture only events applicable to a known database and where the duration of any queries run exceeds a specified time limit. Trace data can be examined immediately or saved and replayed for later analysis.

We regularly use SQL Server Profiler to identify any long-running database queries and stored procedures which can then be examined and optimised appropriately. This may involve modifying a SQL query or batch of queries to make them more efficient or using other tools such a Database Engine Tuning Advisor to identify any database indexes that could be added to improve query performance.

In future, our use of SQL Server Profiler will likely decrease in favour of the new Extended Events graphical interface that is provided with SQL Server 2016.

VISUAL STUDIO PROFILER

In addition to diagnosing performance issues relating to application databases we also use profiling tools included in Microsoft Visual Studio to locate areas of the application code where most work is being performed. These areas can then be investigated and optimised.

The profiler built into Visual Studio allows the developer to create and run a performance session which samples the functions that are executed in the application at regular intervals and then produces profiling data that indicates where in the application code most of the time was spent. Information such as CPU and memory usage can be monitored and recorded.

Following investigation and optimisation of the code, we run the Visual Studio profiler again to capture the output of the performance session and compare this to the original output to identify the level of improvement that has been achieved.

PROFILING ON THE CLIENT

We use other profiling tools such as Chrome Developer Tools to allow us to identify areas of the application that run on the client. For example, the amount of CPU time spent processing JavaScript and CSS selectors or the amount of memory being used by JavaScript objects. These tools allow us to identify and optimise the code that runs in the browser.

Other testing techniques

END-TO-END TESTING

End-to-end testing, also known as system testing, is the practice of testing the flow of an application from start to finish to ensure that it is behaving as expected throughout. These tests typically simulate real-world behaviour, exercise the interaction between components of a system and ensure data integrity.

End-to-end testing is carried out in the context of the system's specified functional requirements in order to ensure it meets or exceeds necessary targets and the customer's expectations.

During the build phase of a project, the development team routinely carries out manual testing of the application, exercising the system as a whole while focusing on specific features that are undergoing development. This may involve signing in to the application and performing a number of tasks before verifying the outcome and signing out again.

For some systems with particularly complex business rules and calculations, we set up realistic end-to-end scenarios that can be executed automatically. These routines simulate usage of the system and then verify that expected outcomes are produced.

REGRESSION TESTING

Regression testing is the process of verifying that an existing piece of software continues to work as previously expected after updates or modifications. The idea is to detect and resolve any issues with existing functionality that may have arisen due to changes in the software as part of a new phase of work.

At Enable, our main method of regression testing is the use of automated testing such as unit and integration tests — these tests are covered in more detail in the automated testing section of this document. The automated nature of these tests allows us to quickly run any modified software through an established suite of tests that can prove the existing functionality of the software works as expected.

Any issues identified can be quickly investigated and resolved at the development stage rather than later on in the process.

PERFORMANCE TESTING

As changes to a software application have the potential to affect the performance of that application, it is standard practice for Enable to rerun load and stress testing on software where modifications are likely to have affected performance — this type of testing is covered in more detail in the Performance section.

This testing can identify areas of unexpected performance loss for further investigation and resolution and can also be used to highlight the improvements in performance and scalability that have been achieved by the modifications carried out.

USABILITY TESTING

Enable integrate usability testing as part of the application development and maintenance processes. During the normal development phase of work, a solution will go through multiple usability reviews, allowing Enable to improve the usability of an application at the time of development. This ensures a better result than carrying out usability tests after development is complete and making changes to the solution retrospectively.

The development team is also trained to proactively raise usability issues which they encounter while working on a system, whether during the initial development or support work. These issues are added to a backlog for which dedicated time is scheduled for members of the team to rectify. This established process means that any issues encountered will be logged and addressed, even if there is not time to address them at the time of noticing an issue.

ACCESSIBILITY TESTING

A number of techniques are used at Enable to ensure a consistent approach to accessibility. These include:

— Regular code reviews and design sessions;
— Manual development time allocated to testing of new and modified features;
— Automated accessibility testing as part of a continuous integration (CI) process;
— Regular training and knowledge sharing sessions.

Of these approaches, developer training and automated test processes are key to maintaining high levels of accessibility and usability across our applications.

Designing for web accessibility is the process of making content that is available on the internet accessible to all users regardless of ability or how that content is consumed. While typically described as making web sites available to people with disabilities, developing for accessibility in fact makes the web more accessible and more usable to all users.

Enable believe that all users, regardless of disability, should be able to consume web content and, as a result, we develop applications with a strong focus on usability and accessibility in mind. This ensures that our applications provide a high-quality experience for as broad a range of users as possible.

Maintaining an accessible application provides a number of key benefits over inaccessible sites, including:

— Reaching a larger audience;
— Better usability for all users;
— Easier and cheaper application maintenance.

Enable develops using established industry standards for accessibility, including the W3C Web Content Accessibility Guidelines (WCAG) 2.0. These standards provide a number of key requirements that web applications must meet in order to meet accessibility needs. These include:

— Ensuring that user input controls are operable using a wide range of input devices;
— Ensuring that content can be presented in different ways;
— Ensuring that suitable alternative formats are available for content (for example, providing textual alternatives to images);
— Maintaining a clear and logical outline and navigation across an application;
— Making it easier for users who may have a visual impairment or colour blindness to access content.

Designing with accessibility in mind allows developers to make decisions early on that maximise usability and meet these accessibility standards as a matter of routine.

This mindset also allows developers to maintain a clear separation between the structure and content of each page and the presentation of the page. This separation of concerns allows us to easily change the layout or visual appearance of an application without affecting the meaning of the page. This makes maintaining and adding new features easier and cheaper as the separation of content and presentation allows for the same content to be easily consumed in different formats, such as when a later phase of work adds a print view of a particular page.

User experience quality

UX DESIGN

Despite being taken for granted by some, user experience (UX) is one of the most important features of a system; While there are many ways in which a client's requirements could be achieved through software, if what is developed is unusable or confusing this can result in user mistakes, reduced productivity or even the total failure of the project to meet its objectives.

Enable takes UX very seriously and we consider it to be an important selling point of our software. To meet the high standards we set ourselves, we conduct a UX planning exercise for each area of work. The key point of this exercise is that, before any code is written, the developer has considered the different possibilities for the look of what is to be created and how the user will interact with it.

During UX planning, most developers produce pen and paper sketches showing the possible alternatives while others are more comfortable using drawing packages. This is treated as a creative process and we do not standardise the way the designs are created and allow our team the freedom to express themselves. We simply require that developers produce a set of clear designs that can be considered and critiqued before the best ideas are chosen. The following is a sample of the kind of questions to consider during the planning process:

— What are the most important features of this functionality and are they at the forefront of the page?

— What are the secondary features of this functionality? Should they be displayed in a panel or presented on a separate page?

— Typography: How could different font sizes and weights be used to convey meaning to the user?

— Animation: Would animation provide clarity to changes of state on the page?

— Grouping of information: Is all the data on the page grouped sensibly?

— Use of images: Could images be used to convey meaning either alongside or in place of text?

— Consistency: Is the design consistent with other areas of the system? Things that look the same should work the same, and vice versa.

Once all of the designs have been produced they will be reviewed and critiqued. This is often carried out during a one-to-one meeting with the project lead but can also be done in a group session with some or all of the project team. Following this, a design is agreed upon and the developer begins working towards it.

UX REVIEWS

Planning is one part of the UX process but reviews are required to help consolidate it. During the build phase, UX reviews are held periodically to consider what has been developed and if it is the best that it

can be for the client. The typical approach to the UX review process is that a developer evaluates what work is complete or what work needs extra attention and presents it to other members of the team.
Those present in the review then critique the work for:

— Compatibility with the rest of the system: Not all pages should be exactly the same but everything that is produced needs to be consistent with the overall design;
— Usability;
— Appearance;
— Performance: although performance bench-marking would not be carried out as part of a UX review, any perceived slowness of the user interface would be flagged for later attention

A list of feedback will then be produced and added to the project team's backlog.

In addition, it is possible, although rare, that the design which was correct at the time is no longer valid in light of new information and in this case the user interface can be entirely redesigned and rebuilt.

Once the feedback is added to the backlog the aim is for the team to address the feedback prior to the next scheduled review so that, in the next review, the results of the feedback can be assessed.

CROSS BROWSER COMPATIBILITY TESTING

As there are some slight variations between how different browsers process and display the same code, our software must be compatible with all of our clients' preferred choices. Enable conducts cross browser capability testing with all the known major browsers including Google Chrome, Internet Explorer (or its current incarnation Microsoft Edge), Firefox, Safari and Opera to ensure that the end product functions consistently across multiple browsers, operating systems and devices.

Usually the browser compatibility is done against those browsers the software is expected to be used on. For instance developing for IE8 can be quite awkward, and we will only optimise performance/ ensure it works on this browser if the client expects the software to be used on IE8.

Code quality & collaboration

CODE ANALYSIS

Code analysis is a term which can be applied to several practices used at Enable to help ensure code quality. We use a number of tools across our chosen languages and, although the exact nature of the analysis varies by language and tool, at a high level the process and goals are the same. Code analysis is a form of static analysis performed at compile time rather than at run time. Compilers naturally report on syntax errors which prevent the source code from being compiled into a running program and code analysis makes further checks to ensure that the code satisfies several additional rules.

The analysis rules can cover a wide range of areas such as design, performance, maintainability and security. In many cases code analysis rules are based on industry standards or best practices outlined by language designers. Analysis rules are not as absolute as the rules of the language syntax and, in some cases, it's not possible for the analysis tooling to conclusively determine if a rule has been violated. The tooling normally outputs a warning which will be investigated by a developer and if the developer confirms the issue it can be resolved. If the tooling is picking up a false positive the developer annotates the code to suppress the specific warning being raised.

Since the analysis process is automated it can produce results quickly and accurately. However, while there are rules that can be enforced via code analysis that would be difficult or impractical to enforce manually, code analysis is merely an aid to code quality not a complete solution. It does not replace other quality assurance processes such as code reviews, and analysers are only capable of inferring so much about the execution of a program at compile time.

Code analysis forms a minimum standard for code quality and can be used to identify trivial quality issues early and accurately. By running this analysis while building the solution we can be assured the source code never falls below these minimum standards of quality. This allows us to maximise the value of our manual analysis processes by focusing on the more complicated aspects of code quality and building upon the base line provided by code analysis.

Examples of tools used to analyse server-side C# code include MSBuild code analysis, StyleCop and Roslyn code analysers. Example of tools used to analyse client-side code include gulp-sass-lint and gulp-tslint.

CODE REVIEWS

Code reviews are sessions in which two or more developers discuss and examine one another's code. These occur during the development phase so the benefits are incorporated into the final product.

The intention here is to highlight any mistakes, improve the quality of the written code and provide more team visibility on a development technique, a given technology or the implementation of a specific area of functionality.

Frequently reviewing new code promotes well-designed and efficient code and keeps developers up to date with how a project is progressing. It also provides a great platform for developers to learn skills from other members of the team.

Providing developers with time dedicated to explaining their approach to a given implementation is invaluable in highlighting ways in which their written code could be improved and also showcasing techniques and approaches which could be reused by other developers in the future. This wider discussion on development techniques and approaches promotes the spread of knowledge throughout the team and encourages an iterative approach to development where each implementation is better than the previous one. As our software solutions are usually built over a number of different phases with a number of different project teams, it is important for a wide variety of developers to have knowledge of different areas of the system. Code reviews provide an ideal platform to increase developers' knowledge of a system or area of functionality.

During phases of project work, weekly code review sessions are arranged in which two developers will pair up to discuss and review each other's code. Notes of any areas that have been discussed and opportunities for improvement that have been identified are then written up and made available to the wider team. Having dedicated time to review code allows problems to be flagged and resolved early and keeps team members up to date with the progress and implementation of the wider project.

PAIR PROGRAMMING

Pair programming is a development practise whereby code is written by two developers working together at a single workstation. At any given time, one programmer will be the driver whilst the other will be the navigator; while the driver writes code, the navigator will read and check the code while considering problems and where to go next. While pair programming, the roles of driver and navigator will frequently alternate.

Pair programming generally increases the overall quality of the resulting code and reduces the number of bugs. It encourages the driver to explain the code as it is being written and compels the navigator to focus on the overall direction of the code. This reduces the chance of bugs and leads to a clearer articulation of the hidden complexities in coding tasks. Additionally, should the driver comes across a problem, there are two people who can work together to find a solution. Pair programming also promotes the diffusion of knowledge throughout our team and ensures that more than one developer has intimate knowledge of the written code at all times. A developer who is unfamiliar with a particular component can learn a lot about it in a short amount of time when paired with a developer who knows it a lot better. This also applies to general programming knowledge as junior developers are able to pick up techniques and broader skills from more experienced team members while pair programming.

PROJECT TEAM FEEDBACK

At the completion of every project each team member is encouraged to give feedback on how they felt the project went and is asked if there are any things that they would like to see done differently on their next project. This process is important to us as it allows individual team members to voice their opinions and also can help bring up issues that may have been missed by their team leader. At Enable, we pride ourselves on our staff retention rate and we believe that allowing team members to have their feedback heard and recorded is a critical contributor to this.

PROJECT DELIVERY MEETINGS

When a solution is ready to be delivered, Enable's Client Services team will familiarise themselves with the solution, and then attend a presentation meeting with the client. This meeting will involve demonstrating the software that has been created, talking through release notes, and starting the UAT process. During UAT, the team will provide support to the client, by email, over the telephone and face to face. In a fast-paced cyclical process, the client will review the work completed and provide feedback, which will then be addressed by developers, allowing the client to conduct further reviews. After six weeks, the client should have been able to verify that the individual statements in the specification documents have been fulfilled in the software solution they have received.

Technical design principles

Several widely-recognised design principles are employed at Enable to deliver strong software architecture solutions. These primarily consist of DRY and SOLID principles promoting a focus on separation of concerns, components with a single responsibility and minimising upfront design by only developing what is necessary.

Training is provided in-house to ensure that all developers abide by these principles when architecting systems. By doing so, we are able to deliver a consistent level of quality without duplicating functionality or over-engineering solutions while producing highly maintainable and extensible solutions in an agile and flexible manner.

D R Y

DRY (or "Don't Repeat Yourself") is a principle of software development aimed at reducing repetition of all kinds. It fundamentally encourages code reuse to make code easier to maintain and reduce the likelihood of bugs.

The DRY principle is stated as "Every piece of knowledge must have single, unambiguous, authoritative representation within a system". When it is applied successfully, modifications to any single element within a system should not require changes to be made to other logically unrelated elements. Elements of the system which are logically related will all change predictably and uniformly, allowing changes to be kept in sync and to be predictable in scope.

By employing this principle at Enable, our teams are able to make code changes with confidence and knowledge of the scope of those changes. This eases the maintainability of large codebases across our team and reduces the likelihood of making a change in one area and inadvertently introducing a bug elsewhere.

S O L I D

SOLID is a mnemonic acronym used to represent five basic principles of object-oriented programming and design as part of agile and adaptive software development.

These principles represent guidelines which should be followed when developing software in order to ensure that the code base is both legible and extensible. By following these principles, we are able to ensure that our software is robust, reusable and easy to maintain going forwards.

Each of the principles are described below.

S I N G L E R E S P O N S I B I L I T Y P R I N C I P L E

The single responsibility principle states that every component should have responsibility over a single part of the functionality provided by the software and that responsibility should be entirely encapsulated by the component, with all its services narrowly aligned with that responsibility. A responsibility is defined as

a “reason to change” and the principle ensures the system remains robust as changes are made as they are isolated to single components and reduce the risk of impacting on unrelated functionality.

OPEN/CLOSED PRINCIPLE

The open/closed principle states “software entities should be open for extension, but closed for modification”. It refers to the use of abstracted interfaces where implementations can be changed and substituted for each other. These interfaces can have partial implementations and can be extended to more complete implementations. This can promote multiple implementations with common code re-use while allowing each implementation to be independent of each other.

LISKOV SUBSTITUTION PRINCIPLE

Liskov’s substitution principle states that if a program module is using a base class, the reference to the base class can be replaced with a derived class without affecting the functionality of the program module. The intent of this is for derived types to be completely substitutable for their base types, ensuring that these classes extend their base implementations without changing their behaviour.

INTERFACE SEGREGATION PRINCIPLE

The interface segregation principle states that no client should be forced to depend on methods it does not use. It splits interfaces that are very large into smaller ones which are more specific to the roles they perform so that the clients that use them are only aware of the methods that are of interest to them. It is intended to keep a system decoupled by reducing the concerns of an interface and thus makes software easier to refactor, change and redeploy.

DEPENDENCY INVERSION PRINCIPLE

The dependency inversion principle refers to a specific form of decoupling software components. Conventionally, dependency relationships are established from high-level components to low-level dependencies. This limits the re-use of the higher level components as they require specific implementations of the lower level ones. The dependency inversion principle reverses this dependency, causing both high-level and low-level components to depend on abstractions instead of concrete implementations.

When this principle is applied it means the high-level components are no longer working directly with the lower-level implementations and cannot be responsible for creating them. Instead, a creational design pattern will be used such as a factory, container or prototype, to manage the objects created.

CODE COMMENTING

Enable aim to produce self-documenting code wherever possible. This practise benefits our developers significantly by allowing them to easily read and understand the original source code. If a particular piece of code is not intuitive, we may add documenting comments at the appropriate level.
We carry out peer code reviews and pair programming exercises during project work to help identify unintuitive code.

FIND US ONLINE

ENABLE.COM

HELLO@ENABLE.COM

UK OFFICE

+44 203 998 7470

10-12 THE COURTYARD
STRATFORD-UPON-AVON
CV37 9NP, UK

USA OFFICE

+1 628 251 1057

535 MISSION STREET
14TH FLOOR, SAN FRANCISCO
CA 94105, UNITED STATES