BRIEFING PAPER

APRIL 2016

Foundations™ Framework

Maximising efficiency in software engineering to deliver real competitive advantage for your organisation

PDMS

Executive summary

In today's competitive environment, for many organisations software isn't just a utility commodity but a vital strategic asset. Business systems that are developed to help an organisation achieve competitive advantage are often by their very nature complex, and their success or otherwise impacts on performance and the bottom line. Such business-critical projects have many similarities with major engineering projects and benefit from the application of well-established engineering practices and approaches.

This paper sets out PDMS' approach to software development: a combination of sound engineering principles with a robust and proven framework-based approach. Exploring the evolution of software development and current styles and practices, it seeks to explain the reasons why we have adopted both component re-use and a framework-based approach and the benefits that this approach delivers to both us and, more importantly, our clients.

'A framework or architecture is needed to piece the chunks together. These are long established in civil engineering, and IT should be developing such frameworks and architectures and methods of practice.'

Thought Leadership - British Computer Society

Background

Over the past four decades, ICT has established itself as a pillar on which most organisations depend. Aside from supporting utility services such as email, accounting and collaboration, on many occasions it also supports or delivers functions which help a business differentiate its products and services from those of its competitors. This dependence is continually increasing – and so in order to extend or even maintain the ability to compete, organisations need to treat investment in technology as strategic.

The perception of and approach to IT projects, particularly those primarily software related, has changed several times over the lifetime of the industry. Initially software flowed from the requirement to add flexibility to hardware, thus programmers were born out of electronic engineers. Electronics necessitate an engineering approach – since trial and error would be expensive and potentially dangerous – and so electronic engineers make use of calculations, planning and modelling, relying on proven patterns and techniques. Naturally, these principles were carried forward into software development, resulting in a structured and procedural approach to projects. The fact that the estate of computing hardware was arcane, and often shared between multiple users and organisations, reinforced the requirement for such a regimented approach.

Software development quickly branched into a discipline of its own – especially as more specialised languages were developed. This increased the extent of the gap between the statements used to instruct the computer, and the steps the computer took to carry out the instructions.

'The economy of human time is the next advantage of machinery in manufactures.' - Charles Babbage

Software flowed from the requirement to add flexibility to hardware, thus programmers were born out of electronic engineers

In turn, the knowledge of the hardware required by programmers lessened, and the hardware engineers were able to improve their design with minimal impact on the software. As computing resources decentralised, the era of personal computing began. Initially, the development tools, techniques and languages used on the desktop were carried forward from the previous era – the same programmers used the same (or similar) high level languages. Before long, however, the increasingly widespread availability of languages such as BASIC fostered new communities of users – those able to meet their own requirements by writing programmes themselves. In addition home users were able to buy an affordable computer, connect it to a television set and cassette player, and teach themselves programming.

This new generation of programmers cared little for the formal processes and techniques established by their predecessors. They were empowered by programming – there were few barriers and little need for doctrine. The early software industry boomed – and although chaotic by previous measures, there can be no doubt that this was a revolution in the making.

In most engineering disciplines, small-scale endeavours can be achieved without adhering to formal process and standards. A garden shed can be constructed from inspiration, materials and labour - **but** to build a tower block these need to be supplemented by disciplines such as surveying, architecture, and structural engineering. As the reach of IT extends, and the scale and complexity of the underlying software multiplies, the parallels are clear: strategic importance mandates a strategic approach. Globalisation requires interoperability – and interoperability requires standards. Growth requires responsiveness – and responsiveness requires flexibility. To meet these requirements successfully demands an approach which embraces good engineering practice at its core.

Hasty application development

Unfortunately, the continuing evolution in the power afforded by high level languages, programming environments and tools brings risks as well as benefits. The 'drag and drop' style of development, recently in vogue, facilitates rapid application development – undermining or side-stepping engineering principles in the interests of a superficially quick 'win'. This approach can be ideal for a small or temporary project, but few commercial projects are so short-lived that such a tactical approach provides a benefit over the projects' overall lifetime. What is required is an approach which delivers flexibility and development efficiency whilst also upholding engineering principles – and a framework-based development style goes a long way towards achieving this ideal.

A commonly held view is that veering away from drag-and-drop development towards a more structured, architectural approach is a retrograde step. Opponents argue that agility must be pervasive throughout the project lifecycle and time spent planning is development time wasted - all tools available should be used. This encapsulates the age-old tension between tactics and strategy, but framework-based development is a viable and proven middle ground.

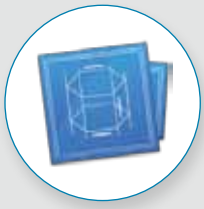
Millau Viaduct, Southern France

'A framework or architecture is needed to piece the chunks together. These are long established in civil engineering, and IT should be developing such frameworks and architectures and methods of practice.'

Thought Leadership - British Computer Society

foundations

Foundations™ is the PDMS framework for enterprise and Internet systems based on Web technology which brings the following contributions to a software project:



Architecture

The 'scaffolding' of a project, which uses a number of key architectural elements to frame the scope and functionality and to promote architectural consistency within and between projects.

Elements

A repertoire of reusable components providing a head-start with common requirements.

Lifecycle

A set of processes which can be incorporated into a project to ensure that the maximum value is derived from the Architecture and Elements.

Framework Development

Architecture

The Architecture is key to our approach, since it defines the technical structure of the project. It is based on principles from a number of contemporary development paradigms, and combines them to deliver development efficiency, reuse, reliability, scalability and extensibility.

The Foundations™ architecture PDMS uses is a variant of an architectural principle known as Hierarchical Model View Controller (HMVC), a development of one of the most enduring patterns in software engineering – MVC. The standard MVC model clearly delineates responsibility into tiers (model, view, controller), but falls short in defining how each tier can be built-up of reusable and loosely coupled components. This (the H in HMVC) is the vital ingredient required to ensure that the benefits of the architecture are realised across larger projects and teams while deriving maximum benefit from the suite of existing components.

Reducing technical overhead

The second principle of the Foundations™ Architecture is 'convention over configuration' (also known as 'coding by convention'). This well known approach reduces technical overhead and promotes consistency. To paraphrase, "where there is no reason to make something different, ensure it is the same". By encompassing this assumption into the framework, default behaviour can remove the requirement for time consuming and error prone configuration.

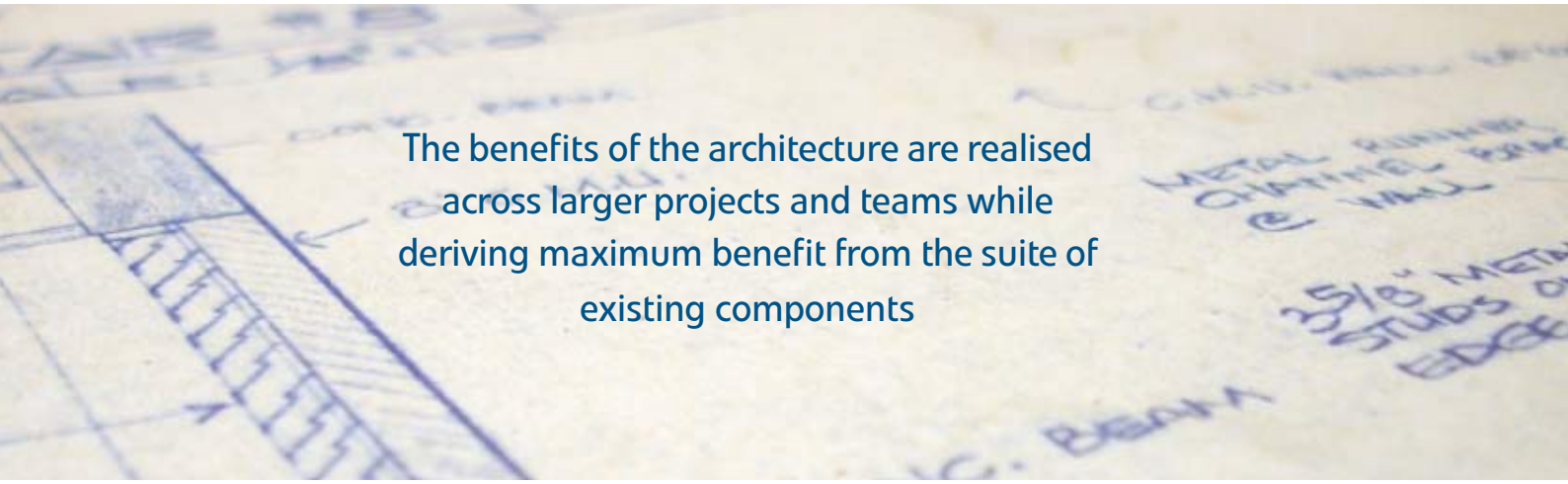
An example of this principle adopted by the Foundations™ Architecture GPL (Generic Persistence Layer) is that the name of a class (the blueprint for an object) representing a database table should be the same as the name of the table – thus allowing the class to be generated automatically. There are, of course, situations where this is not appropriate (for example, accessing a legacy database) – so a facility is provided to override this behaviour – but in the vast majority of cases, this simple convention cuts out significant overhead.

Don't repeat yourself

A related principle is 'Don't Repeat Yourself'. If something needs to be specified more than once, then there is an opportunity for inconsistency and an overhead in implementation and maintenance.

An example of this principle from Foundations™ relates to the size of data fields. If a Company Name field is configured in the database as having a maximum length of fifty characters, then this need not be configured elsewhere. Validation rules, user interface elements, warning messages and any other elements relying on this specification defer to the definition in the database. If the database field is extended to contain sixty characters, then by default everything else changes automatically (unless configured otherwise).

The third overall principle adopted by Foundations™ is refinement by derivation. Object Orientation is a well-known programming paradigm which many development approaches embrace to a greater or lesser extent. Using Foundations™, it is a basic principle adhered to throughout development – it is the principle which allows reuse of our existing component suite with minimal compromise over functionality.



Derived benefits

A criticism of legacy modular development styles is that a component either needs to fit your requirement or your requirement needs to fit with the component. This reduces the extent to which reusable components can be used to cover the overall project scope: unless you are prepared to compromise significantly; the requirement is very standardized; or you are very lucky. By definition, it is likely that that any strategic (rather than utility) requirement is intended to be a commercial differentiator – so for key projects, any requirement to comply rigidly with the norm is unwelcome. Typically, this mandates a largely bespoke development project, with the associated elevated cost, risk and delivery timeline.

Using object derivation allows off-the-shelf components to be used even when the fit with requirements is loose. The developer can use the existing component to the extent that it fits, and then meet the requirement more closely by extending or tailoring the functionality without changing the standard component. In practice, this derivation can be extended to multiple steps, and in multiple directions – resulting in a rich set of variants. For example, a fundamental component provided by Foundations™ is a 'Role' – which represents a function of an Individual or Organisation. Derived from Role is Client – which provides various standard facilities such as recording of payment terms, credit limit, billing address and so on. It is unlikely that on any particular project, the set of attributes and functionality we have assumed are

The developer can use the existing component to the extent that it fits, and then meet the requirement more closely by extending or tailoring the functionality without changing the standard component

relevant to a Client exactly match the requirements, but using derivation, we can vary the definition of a Client in the context of the particular project without being precluded from taking advantage of the other Foundations™ components which understand how to interact with Clients. This approach extends out to more complex, specialised and sector-specific components – so even when the overall project requirement is unique, our standard components can contribute significant functionality.

Elements

This principle of derivation is fundamental to Foundations™ Elements – the suite of components which fit into the Foundations™ Architecture to provide a head-start on common functionality in business systems. Foundations™ Elements are split into five categories:

1 User Interface Components which make-up the Web-based interface with which end-users interact, such as a results pager, data entry form, or tree. During the course of a project, we typically create derived components to meet particular requirements, or even implement entirely new components to meet any particularly novel design features.

In order to provide a rich and effective user experience, these components include client-side elements which run in the user's browser. However, in order to deliver systems practical for all users, accessibility is an important concern – so our interfaces are designed taking into account best practice in this area.

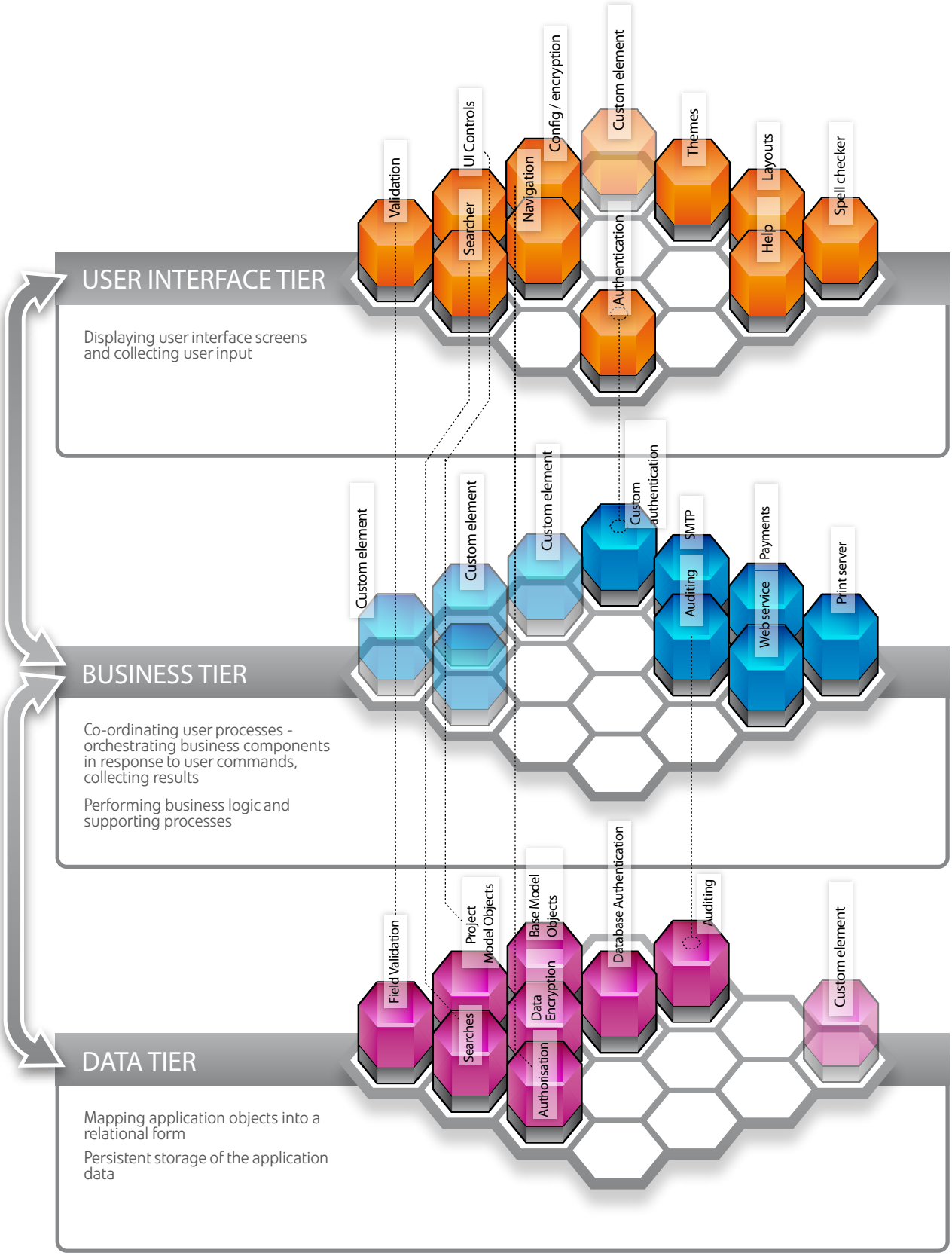
We are mindful that even within a single system, different users and communities of users have different requirements. Therefore many of our user interface components are configurable by end users to meet their needs. For example, the columns shown in a result list, and the order of the results, can be changed to suit the task being undertaken.

2 Themes and Layouts which control the 'look and feel' of the user interface, including layout, fonts, colours and branding. Foundations™ provides a library of standard off-the-shelf themes and layouts, and custom implementations can be created where required – for example to meet organizational brand guidelines.

Where appropriate, themes and layouts can be selected by the end-user to suit their needs and the requirements of the devices they are using. For example, a high contrast theme can be selected for visually impaired users, or a kiosk theme used where touch-screen kiosks will be used.

> In designing and maintaining components, our goal is to minimise a significant technical overhead - the effort involved in creating any functionality which is not unique to the project. Instead, developers' skills can be used on the unique aspects of a project such as the critical business requirements

Architecture and Elements



Our themes and layouts are designed to provide a consistent experience across all modern browsers, and it is our policy to keep the list of supported browsers under constant review.

3 **Base Model Objects** such as Individual, Organization, Role, User and standing-data objects such as Country, Language, and Currency. As well as a set of generic objects, basic objects for particular domains – such as healthcare, criminal justice, social networking and telecoms – provide an additional head-start for appropriate projects.

4 **Applets** are common islands of end-user functionality such as a user manager, data maintenance applet or audit search. Applets are self-contained (in that they are usable 'out of the box'), but are typically integrated seamlessly into the applications which will use them.

Where similar functionality is identified in various areas of a system, we will typically create an applet. This reduces risk and work, and means that as enhancements are made to the applet they are available in all areas where the applet is used.

5 **Framework Elements** – components providing generic services such as authentication, audit, configuration and navigation. These services are provided implicitly by Foundations™, not requiring any additional effort by the developer, and so provide consistent and dependable behaviour across the system.

Before routing the request, the Dispatcher determines whether the currently authenticated user can legitimately access the function, and if not, the request is rejected before even passing it to the function itself

Security

Internet facing applications in particular are under constant attack from many diverse threats. By their very nature, such applications often need to be accessible from anywhere – and a layered security model is required to mitigate threats. The outer layer is provided by the network infrastructure – firewalls, screening routers and similar devices. The next level of protection is provided by the operating system. Foundations™ uses the Microsoft Windows suite of products and the closely integrated IIS web server software. These are well proven products, constantly updated and refined to meet ever-changing threats. The third level of security is provided by the application itself. In the case of Foundations™, the component of the architecture concerned with application edge security is the Foundations™ Dispatcher.

Each request from a user is supplied to the Dispatcher from the web-server software. The request is inspected for validity and to determine which function within the application the request should be routed to. Before routing the request, the Dispatcher determines whether the currently authenticated user can legitimately access the function, and if not, the request is rejected before even passing it to the function itself. Security is therefore implicit – the project developers do not need to concern themselves with these checks – so there is little opportunity

for oversight or inconsistent security policies across the application.

If, as part of this process, the Dispatcher determines that the user needs to be authenticated, then this authentication process can be performed using the inbuilt Foundations™ authentication model (for example, based on username, password, location), or can delegate authentication responsibility to a separate service – such as Microsoft Active Directory, OpenID or a proprietary mechanism. Authentication regimes can be mixed within a single application – and if the user is already authenticated by virtue of their environment, then this authentication information can be trusted by the dispatcher where appropriate.

Authorisation is the process of differentially controlling access to functionality or data based on the authenticated identity of the user. Foundations™ controls access to functionality based on a rich policy framework. Each function supported by an application is linked to a policy, and policies are assigned to individual users or groups. A user can be a member of multiple groups, and users, groups and policies can be managed by appropriately authorised system administrators. If a user does not have access to a particular function, then the relevant links and menu entries are automatically hidden with no effort from the developer or administrator.

Data authorisation (row and column level security) has been a thorny issue for web-based applications since few underlying database engines support the granularity of access control required. The Foundations™ GPL (Generic Persistence Layer), which sits between the application and the database, provides unrivalled flexibility and control in this area – allowing fine control over which users can access what data with little effort from the developer.

Objects in the database can be assigned a security level using a scale appropriate to the application. At the most basic level, users and groups can then be granted access based on the type of the object and the security level. At a second level, this security can be further qualified based on the operation being attempted. Using these three dimensions – the target object type, the identity and group membership of the user, and the nature of the operation – a wide variety of security models can be created.

This data authorisation model is implemented as a GPL filter. As with all other aspects of the framework, the behaviour can be tailored through derivation – or replaced/supplemented by project-specific rules as where required.

Auditing is the final security aspect provided implicitly by the framework. Foundations™ can audit two aspects of application use implicitly – application access audit, and data audit. Application access audit captures each user session, which functions were accessed, and what parameters (form entries) were provided for each function. Data auditing is integrated into the GPL, recording which database records were accessed in the context of which function. If the function resulted in any changes, auditing can record which fields were changed.

Using these three dimensions - the target object/type, the identity and group membership of the user, and the nature of the operation - a wide variety of security models can be created

Benefits

To the project owner

Reduced overall project time and cost – achieved through a streamlined design process and reuse of existing components.

Improved application security – achieved by virtue of the Foundations™ architecture enforcing good architectural practice, blocking basic vulnerabilities (such as SQL injection and XSS), and reusing components which have been previously security tested.

Improved runtime performance – achieved through Foundations™ caching, and reuse of components which have been optimised and streamlined to an extent not normally practical during bespoke developments.

Sooner 'first light' – spikes of functionality can be delivered quickly (with limited business functionality) to promote early feedback.

Interface design guidance – the wide range of Foundations™ User Interface Controls have been designed to take into account usability, accessibility and best practice.

Continual involvement – we like our clients to understand our approach, and the Foundations™ process encourages on-going interaction between the client and the PDMS team throughout development, particularly during design iterations.

Scalability – the Foundations™ Architecture is designed to scale smoothly as capacity requirements increase.

To the project team

Design process – we have standard documents and templates to ensure that systems are designed to make best use of Foundations™ components, and to ensure that the information required is captured in a manner that meets the requirements of both the project team and client.

Fast start-up – our development teams are fully conversant with the framework and how best to utilise it, minimising time spent getting started on new projects. Teams can be scaled up - and down - as required, and new team members drafted onto a project are quickly productive.

Efficient recruitment – our 'boot camp' process allows new developers to quickly get-to-grips with the framework, and then transition onto client projects.

Ownership – Foundations™ is fully under our control, including release cycles and functionality. We are not beholden to third parties regarding roadmap, changes and bug-fixes.

To the end user

Consistency – both within and between Foundations™ applications. Knowledge gained from using one system area can be extrapolated to other areas.

Configurability – Foundations™ applications are more configurable to an individual user's requirements than can be warranted in most bespoke developments.

Usability – the Foundations™ user interface components have been developed and refined to provide a responsive and flexible experience without compromise to accessibility and browser support.

To the IT Department

Low footprint – Foundations™ applications make efficient use of infrastructure.

Flexible deployment – a number of deployment models are supported to suit varied network topologies and availability/security requirements.

Standardised – a Foundations™ application requires no special hardware or system software, and runs on standard Microsoft operating systems and database platforms.

Reliable – a number of high availability technologies (such as load balancing, clustering, and replication) are supported to provide continuous service in light of hardware or network failures.

Quick to deploy - Foundations™ is browser-based, and so requires no roll-out of client-side software

Conclusion

The Foundations™ approach to software development offers an architecture and a process which provide security and reliability yet with the flexibility to meet bespoke bsusiness system requirements.

All too often when embarking on a new software project, there is a propensity within the industry to write new code to perform well established tasks, and this can be to the detriment of reliability in many applications. Foundations™ encapsulates our accumulated knowledge of nearly 20 years in the IT industry and hundreds of successful software projects.

Whereas requirements may be unique from project-to-project, we meet these requirements through another iteration of an established process which is both repeatable and dependable. Reuse aids productivity, speeding up the development process, but more importantly it contributes significantly to quality - you are reusing something that is tried, tested, proven and trusted.

To meet market, competitive or regulatory demands, systems need to be able to accommodate change. The Foundations™ architecture and development approach aids longevity: systems developed in this way can survive major changes in functional requirements, organisational structure or scale.

Foundations™ helps to deliver software that is a real strategic asset to any organisation, using an architecture, components and processes that have been proven to overcome the challenges of complex software projects.

About PDMS

PDMS provides software engineering and related IT services including hosting and on-going support. The company has been in business for 19 years and has offices in the Isle of Man and central London. PDMS specialises in developing web-based business systems using the Foundations™ platform and components. PDMS works with a wide variety of clients in both the public and private sector and has developed a reputation, not only for delivering innovative solutions, but also for working closely with customers to really understand the needs of their business.

Contact us

PDMS Ltd.

Global House

Isle of Man Business Park

Cooil Road

Douglas

Isle of Man

IM2 2QZ

Tel: +44 (0) 1624 664000

Fax: +44 (0) 1624 678787

www.pdms.com