

Design & Development Considerations



Introduction

When considering a new system design and ultimately a new development, there are many things to think about. Some will be high level - to do with requirements and technical architecture, some lower down - linked to the design at the interface and application level, and some will be purely technical.

Whether you're working on an in-house development, outsourcing to a 3rd party, or considering a pre-built packaged solution, many of the same considerations apply.

The notes below are by no means exhaustive, but they do provide a starting point for the discussion of a number of topics and an insight into the areas that typically need to be at least considered as part of a new project & ongoing development.

These are also some of the same methods used by our own consultants & developers when working on or discussing a development project or purchase.



Get all the requirements up-front

It's amazing how often shortcuts are taken here but even if the project is to be phased in, it's best to get a list of firm requirements at the start. By this we mean, *all* requirements - including those that will not necessarily be implemented yet. The design may be impacted or even compromised if something major comes along later, so it's worth knowing at the start.

This can be difficult, especially as some users will only tend to answer the questions you ask. But sometimes, it pays to ask more "off the wall" questions just to test the water. For example, discuss areas you think you know the answer to or areas that they couldn't possibly want to include – just in case they do!

You won't necessarily be able to include everything, but having all the requirements does at least mean they're out in the open from the start. Only then can you move on to determining *how* to deliver a suitable solution and what can and can't be included.



Involve the users

Usually the ultimate customer, it's worth involving key users, or their representatives at an early stage in the design as well as on-going throughout the development. They will often want to be involved and ultimately it is these people that will determine the success or failure of the deployed system when it goes live.

They will hopefully allow any issues that arise to be pinpointed & resolved more quickly and also highlight any issues that may not have been picked up earlier.



Match up *all* the requirements

Of course, the customer's requirements are key, but so are the IT departments. After all, it's probably your environment the system will be running on so consider anything you wish to add as well.

Do the customer's requirements ...

- Make sense ?
- Fit in with your own short & longer term strategy ?

If they have any specific proposals, are they best for performing the results they wish to achieve ? If not, now's the time to say.



Implement the base design

Having discussed & documented broadly what the system will do *and agreed it with the customer*, the next step from an IT perspective is to consider the design itself. How will you achieve the desired result in terms of an IT solution ? Don't think directly about tools at this stage, but what the result will look like once you've finished. Initially you could view this as an "ideal" solution, and then possibly compromise later if required.

Considerations here include ...

- How will the user interface with the system ?
- Where will the data be stored & will it require secure or shared access ?
- Is there any shared logic to consider ?
- Are there any tasks that require special consideration – e.g. interfaces, CPU-intensive processing etc. ?

Some of these are discussed in the next few sections but it's a good idea to consider them all together at the start.



Think about the data

If the data is sensitive, or important to the user's work or organisation, consider where it is being, or will be held – even if it's only accessed by one or two users.

The backup & security strategy of your SQL Server cluster, for example, is probably (hopefully!) far better than a Microsoft Access or text file placed locally on a workstation. However, a database solution such as this is more costly if you don't already have the environment set up, so alternatives might also need to be considered.



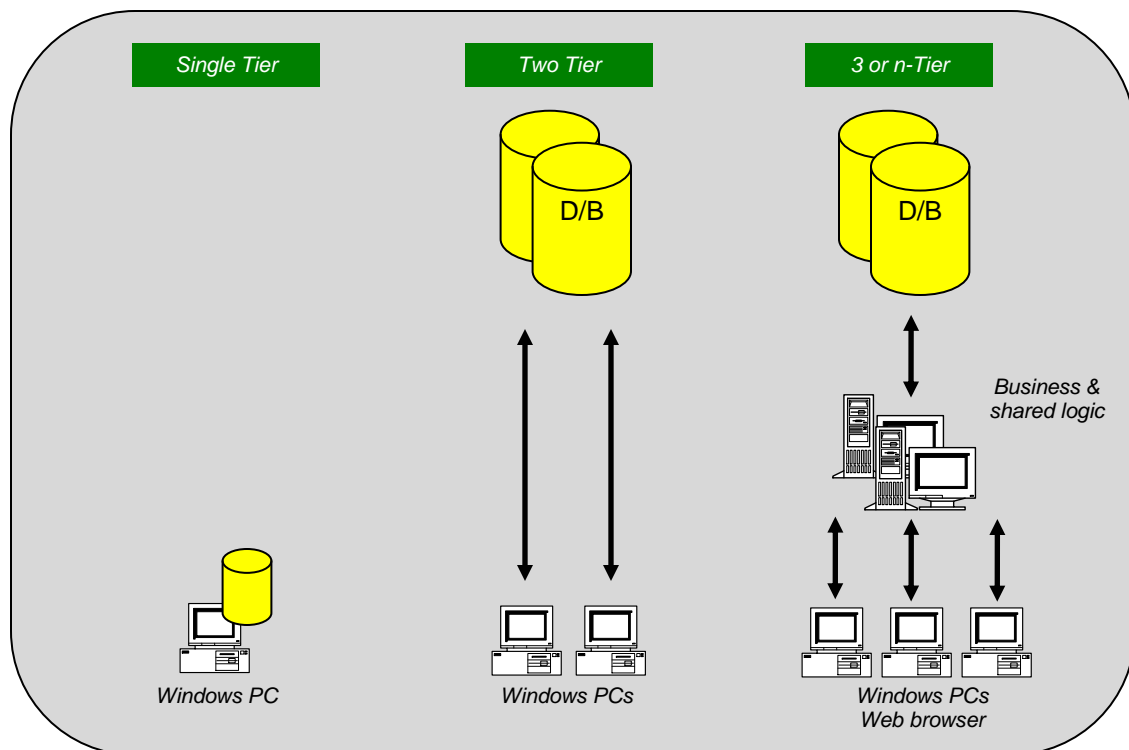
Tiers not tears!

Though not necessarily applicable to smaller systems, larger solutions can often benefit from the splitting of logic into separate components, located on one or more so-called “tiers”. Typically, a tiered approach is split into 2 or 3 logical layers as shown below ...

- **Front end.** Front end logic is located here. This could be a web browser, a “thin Windows client” or a “fatter”, client whereby business and even data logic is located on the local PC.
- **Business logic.** A middle-tier is often responsible for business logic. This may be remote COM, DCOM or COM+ components which implement more complex and CPU-intensive logic.
- **Data logic.** This layer holds the data store – be it directory services, file/directory, a mainframe file or a database such as SQL Server or Oracle. Business logic, in the form of database stored procedures can also be placed here.

The advantages of this approach, especially for larger systems include the ability to ...

- Use less powerful desktop PCs & remote laptops, as more work is done centrally.
- Place complex & CPU-intensive logic on more powerful server-based hardware.
- Share resources amongst all users from a central source.
- Locate business logic near to the data “network-wise”.
- Centrally manage key components & business rules.
- Allow business logic to be placed within the database itself for maximum network efficiency with stored procedures.





What interface should you use?

What does the user want to see, how do you want it to look and how does this fit in with your overall IT strategy ? Sometimes this seems obvious at first glance, but even then, it's worth considering all the primary options, if only to rule them out ...

- Web front ends are very popular and have a number of clear advantages. They can provide a good looking user experience, is easy to customise and are easy to deploy to the desktop (browser).

However, they can be slower than more traditional approaches so are less good for data entry or high volumes of data. Reporting, or more complex displays are also often more tricky from a web browser than a dedicated reporting tool or PC client application.

For example, let's say you're a supermarket considering a new system ...

- If your system is selling to customers at home, a web-based interface is an obvious choice. It's good for deployment, doesn't require them to have any special software and gives you scope for a colourful & pleasant interface.
- If, however, it's for a depot, supplier or clerk performing data entry, would it still be as suitable ? After all, they probably want a fast interface with maybe reporting options – they certainly don't want to wait to see a picture of the beans they're sending out.
- Rich Windows front end applications provide the functionality up front, but may require more powerful hardware on the client PC level itself. They must also be deployed, along with any associated tools & licences.

Also, in this case, most if not all the processing is typically performed by the client, meaning that any data that comes from the network must be pulled down before being processed. This can increase network traffic quite dramatically if you're not careful.

However, they provide for a rich user experience which may be important and may be more suitable for in-house systems, or smaller local applications.

- In a well designed system, a multi-tiered approach can often compensate for the above deficiencies, but requires careful planning, especially when determining where to place the individual units of work. This in turn adds complexity so it's important to get it right.

Ideal for large-scale systems, a multi-tiered approach overcomes some of the limitations above, allowing thinner clients (web or Windows) but still allow for heavier processing. However, for smaller systems, it's can become over complex in terms of the development, management & later support.

Taking all of the above into account, there's nothing to say you shouldn't implement more than one interface. Maybe your customer should use the web (with associated graphics etc.) whereas the data entry clerk could have more direct access to the data ?

If this is the case, the rest of your design will probably want to be adjusted accordingly, so that common code can be shared. Again, a multi-tiered solution is ideal if you have more than one interface as you can write your business logic once, and be able to access it from different front ends. But again, this needs to be planned carefully, to ensure these central components meet all or most client needs, and don't instead compromise them.



Phase the implementation or “big bang” ?

How do you want to implement the solution ? Is it something that can be built on in a phased approach or is it an “all or nothing” solution ?

A phased approach can often have a number of advantages ...

- It gives the customer something more quickly
- It allows “issues” to be highlighted early in the development as the user will get to see *and use* the system at first hand earlier.
- It allows later phases to correct or adjust earlier issues or avoid them as new features come along.
- Fewer developers are typically needed. This in turn lessens some of the problems of large development teams and adherence to common standards etc.
- It may also mean that the system can be funded over time, as opposed to one huge development.

... but is not always suitable and may not always meet the end-users needs.



Hardware considerations

If your environment is well defined and you have a set of standard server builds & architectures, this may influence, or dictate the approach you wish to take. If not, take a moment to consider your options.

One solution may require a dedicated server whereas another is solely client-based – and therefore doesn't need the expense. That in itself is great, but check the client PC will be up to the task. Even if it is, does that approach fit in with your longer term IT strategy ? Also, deploying a system to many users individually may be more costly and cause more design issues than a centrally managed service.

Above all, is the hardware choice, and implicitly your choice of environment going to compromise the way the system can evolve in the future. If it is, should you look at this now ?



Single or multi-user, does it matter ?

Is the system going to be ...

- A standalone solution ?
- Used by a small group of people ?
- Used by a larger number all accessing the same information ?

The answer to this is important when it comes to the correct choice of data store – and in turn may lead you to consider different access methods.

For example ...

- A local system, accessing its own data can write to a file or a small Microsoft Access database without too much fuss. But place the data on a network and get a few people accessing it and concurrency issues might – and probably will, arise.
- Larger SQL-based systems can handle literally hundreds or even thousands of users, all concurrently accessing the same tables without any problem, as long as your design *and the logic within the application code* takes this into account.

Having avoided locking problems, there's one more question to ask. Is it possible or likely that two users could access the same information at the same time. If the answer is Yes, consider whether this itself matters ? Should the second user be allowed to overwrite the first user's changes without knowing they were there in the first place ? If not, concurrency needs to be designed in, preferably from the start.



Do you have existing systems to consider ?

There are three reasons for thinking about this ...

- Firstly, does the new system need to interface with any existing solution ?

If it does, then this may guide (or restrict) you to a particular set of technologies. For example, if everything's already in an Oracle database, it doesn't make much sense to create a new SQL Server installation, unless your strategy is moving that way.

Interfaces too may be more efficient or readily available between some systems than others.

- Secondly, it's worth bearing in mind that your developers and/or support team will be much more familiar with the installations you already have.
- Lastly, but by no means least, there's the cost implication and the possible added expense of purchasing new software as well as hardware to run it.

If your existing infrastructure will cope, then maybe that's a good way forward unless again, your strategy is moving away from that approach.

Of course, we're not saying that you shouldn't consider new alternatives. After all, technology moves on every day and it may be cheaper in the long term to utilise newer options! But the three points above are at least worth considering.



Off the shelf or bespoke?

Depending on who you ask, you'll get various answers to this. Anyone who's brought in a product, realised it doesn't do what they thought and then failed to coerce it into doing what they originally wanted will naturally shy away from that approach again – and sometimes, with good reason. Likewise, the thought of managing a team of developers and waiting 6 months for them to deliver may put you off that method. Both are valid, but equally both are not necessarily good arguments if you do your homework.

The first step is to see if there are products on the market that might do what you want and have the features you need. By at least looking, you'll certainly get some ideas as to what features you ought to consider. At the same time, you'll get an idea of cost and what you're paying for. For example, paying for a solution that can run on multiple platforms when you only have one is certainly worth knowing.

Most of us have heard the 80/20 rule ... “if it does 80% of what I want it’s the better approach to take”. In our experience, however, that rather depends on the 20% that’s missing. Here are some guidelines to consider ...

Products ...

- Don’t have to be designed or built directly
- Are often supported by the vendor
- Can often be implemented more quickly
- “May” be industry standard

But a bespoke solution ...

- Gives you control over the features you implement
- Allows you to decide which features to include, and which you don’t need
- Means you don’t pay for features you don’t want, if your design is correct
- Can be extended or updated in your time-scales, not the vendors
- Can be managed in-house

So, if you’re looking at products, we recommend you consider the following ...

- Make sure they do what you want them to
- Don’t assume they can be forced to do something they’re not originally designed for. The vendor may say it can, but consider the facts first. Often the product can’t be customised that much and even if it can, will it do a job it wasn’t designed for well ? Also of course, do you want a customised version ?
- *Don’t assume they can be forced to do something they’re not originally designed for (it’s worth repeating that one)*
- Be wary of customisations. Who will perform them and what will they cost ?
- If you do need to customise ...
 - Check what will happen to any custom options when the vendor upgrades their product ? Will they upgrade yours, or will it cost you more to implement them again ?
 - Check the support arrangements. “Special builds” may come with additional costs.



What do you gain if you start again?

If your new system involves replacing existing solutions, or overlaps with them, consider what you gain by starting the design again.

- You may gain a lot. A clean slate means you can design from the ground up, with less constraints than if modifying an existing solution.
- But to reinvent & rewrite what you already have could also be costly, in terms of time and money, and bring little benefit if the existing solution fundamentally works well.

Don't be persuaded simply on the grounds of technology. We all want the latest version or the newest tool, but check what you're actually gaining. A lot of IT is actually very similar, even if it's implemented slightly differently. Maybe this year's solution brings a new, faster development environment, allowing the solution to be written in half the time. But if you've already developed the same solution using last year's tool, what's the benefit in creating the same thing again – even if it is faster this time round ?!



Do you have a longer term vision ?

Most organisations have a longer term plan, covering the next few years and with IT this might also be updated annually. If you do, then clearly any solution being considered now should fit into the “new world” ... or the new world needs to be updated. Either way, the two should be in-line and you may want to discuss the solution you're considering with your technical architect.

Also, bear in mind that a well designed system is often more than capable of working in tomorrow's world, even if it's designed for today. For example, if you have a scalable system designed around COM+, simply porting it to .NET will bring little benefit unless you wish to take advantage of other features. Microsoft tends to give developers many options for performing the same thing, such as data access, yet when you look more closely, they tend to boil down to the same principles in the end.



What's architecture would work best ?

We've come a long way already and now got a good idea of not only what we want, but also the overall approach we want to take. The next step then is to consider a little more of the detail – not in terms of tools, but how the final system will fit together.

What we need is an application or system that ...

- Does what the user wants.
- Does what you want it to.
- Fits in with your overall IT strategy.
- Fits in with any hardware or interface constraints etc.

A block diagram is a good starting point, noting the interface(s), the data store(s) and the main processes that will be required. Linking the ones that need to talk together gives you an overall view of what's required. If you have multiple interfaces, consider whether some of them actually require the same processes ? If they do, then the process itself might be considered as middle tier/shared logic unless it can be placed directly in the database.

The chosen architecture needs to support and should encompass *all* the components we need to use. It will probably be made up of a number of separate pieces, especially for larger systems in terms of the front end solution, any background services, components and data stores.

By now, some of these will be obvious – the front end interface (e.g. web vs Windows GUI) and the data store (such as SQL Server). But here are another few worth thinking about ...

- Background services.

The use of Windows services is often overlooked, but these are extremely useful and can be run in the background - even when no user is physically logged on, making them ideal for server-based roles. Many functions of Windows & the software it supports are implemented using Windows services and there's nothing to stop your system using them too, if you have this sort of logic to perform.

- Business logic.

Even if you've opted for a single or two-tier system, you might still want to consider the business logic – logic that retrieves or updates data, or performs processing on it, separately from the data or the interface itself. This can be useful in breaking down the requirements & design, and highlighting specific areas that may need more careful planning and thought.

Technologies such as COM+ and component services can also be considered to help provide & centrally administer a middle-tier solution.



What development tool would work best ?

This is much more easily answered now that you have all the requirements & an approach documented up-front. You may think it a rather daft question, but it's easy to get side-tracked down a particular technical path which, on day one, seemed great but if you look now, knowing what you do, might not fit the bill.

Your developers probably want to use the latest version of Visual Studio or create a “whizzy” Flash presentation, but is this the best solution for what's needed ?

- Which development tools can deliver the approach you're thinking of taking ?

Remember, you can use different tools for different components, as long as they can communicate and interact together.

- What's your standard development environment now ?
- What's your IT development strategy for the future ?
- Who will support it long term ?
- What resources do you have available, in-house & externally ?

Again, we don't necessarily want to limit the design to this, but it's an obvious consideration, especially where a number of suitable options are valid.

The outcome of this may mean that some parts of the design need to be considered again, or re-designed to fit in better with the tools chosen, but its better this way round, than constraining the initial design on day one with the use of tools that have yet to be decided.



Use experts where you will benefit

IT is a big area and covers many different techniques & technologies. With that in mind, it's quite likely that you'll have expertise in some, but not all areas covered by any medium or larger system. To compensate, consider bringing in this expertise when required to back up your development team and advise the project as and when needed. Use these resources wisely – remember they may also be able to advise at a higher level as well as providing coding resource.

Examples might be database administrators (for design, SQL, tuning & performance monitoring), COM+ specialists – for middle-tier component logic design & interfaces, GUI design, web design & graphics etc.



Review as you go

Post implementation reviews are great, but by then it's a question of learning from the lessons that have already happened. A better approach would be to pick up on issues as they arise and the best projects tend to do this. If possible, changes can be incorporated to handle them or they can be noted for a later phase or update.

Try and involve representatives from all areas in the meetings.

- End users like to be kept informed of progress and are actually quite likely to spot issues as the discussions progress as they know the underlying business more than the IT team.

They also have tendency to remember things as you go along – maybe things they take for granted but which you, and the developers, not surprisingly didn't realise.

- Developers too are key, as they are the ones interpreting the design and so may also realise or raise problems/questions.

As the design & build progresses, continue with these reviews – preferably at regular intervals. It's not all about problems – you might also discover functionality that's no longer required, potential shortcuts that can be made, or even new features that could be implemented at a later date.

As with any team-based activity, involving these people keeps everyone in the loop and reduces surprises further down the line.



Keep an "Issues List"

Create & maintain an issues list. This holds details of both major stumbling blocks & minor issues. As each is researched, keep the list – which should be a single, easy to use document that's kept updated. Resolving an issue also means it's moved to a "Resolved list" at the bottom of the same document, along with the agreed resolution.

It's worth bearing in mind that you'll almost certainly have some, if not many issues. Hopefully not too many, but you'll have some, even with the smallest of systems. As the project gains momentum, the issues list will be longer than the resolutions. But as it progresses, you should find fewer issues and more resolutions.

This is one of the ways you can keep track of the development and ensure things continue to progress well.



Start the user guide early

At first this might seem a strange one but there's more than one good reason for starting the system's user or support guide early. Not only does it get the documentation out of the way and not left to the last minute, but it also forces the team to write down & explain what the application is doing and how it does it.

Something that's over-complex is always tricky to describe! If you find it tricky to explain in the user guide, the design itself is probably difficult & confusing for the end user and probably needs to be tweaked.



Manage Expectations

This should be automatic if everyone is being kept in the loop but the main idea is that there shouldn't be any unexpected surprises, for anyone.

Customers will be happy if they get what they expect, so they - and you, should be aware of any limitations or features not yet included as the project progresses.



If developing, adhere to standards

Firstly ensure you have some standards – it's surprising how many teams don't! They don't have to (and shouldn't) run to volumes, but they should enable everyone in the team to follow similar rules. They shouldn't be constraining, but they should ensure that no matter who the developer is, the approach taken is broadly similar.

This is important, both during the development and subsequently for supportability as it is not only neater, but allows future amendments and enhancements to be implemented more easily.

Perform spot checks too, even if you're not a developer yourself. For example ...

- Read the comments – do they actually help the understanding of the logic, especially in more complex areas ?
- If you took out all the code and left only the comments, what would it look like ? Is the answer a blank screen ?



Review outsourced designs & logic

If a 3rd party is developing the system for you, or you have 3rd party consultants/developers in house, do the same checks as if they were your own personnel. After all, you're paying for expertise so you should ensure you're getting it.

Ensure they're adhering to their own standards (for outsourced developments), and that those standards are up to the task.



How will you measure success ?

Although not directly linked to the design, this is still worth considering as part of the overall strategy. What are the markers that will indicate that the project or system is working well ?

Consider what will determine success or failure.



A good design is flexible, but ...

If you've got the design right and it meets your user's (and your) requirements, then it should be flexible enough to allow for certain changes that will inevitably arise. However, it's also important not to lose sight of the initial goal, which is surprisingly easy to do.

"Scope creep" can delay a project to such an extent that (i) it never completes and (ii) if it does, the design is compromised to such an extent that the end result is a poor imitation of the original.

The issues list will help here too.



Get an independent view

For larger projects, you might also want to consider getting a 3rd party review as the design & development work progresses – maybe at strategic points in the project's lifecycle.

If you have little experience in the area yourself, this also allows you to get an expert opinion on what's going on and also to ensure what you're being told is actually what's happening, in an unbiased way.



Supportability

Something else to be consider quite early on is the supportability of the system. Remember, you don't want anyone getting any surprises, so it pays to get whoever it is involved early. If support will be provided by an external company, involve them too, so they can determine how things will work and what the possible costs might be.

Either way, your support costs may be reduced if you talk to the team or company early on and get their views. Maybe avoiding certain technologies or options, for example, could help supportability in the future without compromising functionality.



How can 3Ds help ?

We hope the above sections have given you at least an insight into some of things we would recommend you do as part of a new design or development. Like we said at the start, it's by no means an exhaustive list – after all, most projects have their unique characteristics (and characters!) but it gets the ball rolling.

“Consider” is a word used throughout this paper and that really says it all. *Consider* the options put forward. Some will be very important and you'll want to extend while others may not affect you or be as relevant.

Our own consultants, development & support teams have years of experience using similar techniques. We are happy to discuss or help you with any part of your design or development, or with system reviews. A brief description of our company is given below and our contact details shown at the end of this paper.

For more information, please visit our web site – <http://www.3Ds.co.uk>



About 3Ds (UK) Limited

3Ds (UK) Limited is a privately owned UK-based software house & consultancy specialising primarily in Windows-based client, server, database and web-based solutions. Established in 1998, we have many years of experience in project development, consulting, support and training.

We have extensive knowledge of both Microsoft development platforms & environments as well as live user-driven environments and we use this to support our clients, be it for targeted assistance with a specific technical or design issue, longer term strategy and technical support, or development resource. The 3Ds name itself stands for "*Design, Develop, Deliver, Solutions*".

In addition to creating & supporting newer systems, we also fully appreciate your commitment to existing systems & solutions. With this in mind, we are also able to integrate robust, flexible and efficient solutions with your existing technologies. Using this knowledge, we can help your business grow and take advantage of technology through your own IT solutions, as well as provide technical assistance with your existing systems.

Whether you have legacy systems, use the latest Windows development tools, or wish to link the two worlds together, our consultants maintain the highest standards to deliver the results you want - be it through consulting, training or software development.

As a software house, we also develop & support a full range of monitoring solutions (<http://www.Sentry-go.com>) which provide our customers worldwide with affordable, automated monitoring solutions that are both quick & easy to use, whilst providing support teams & management with the information they need access to.

For more details please visit our web site, <http://www.3Ds.co.uk>.



Contacting Us

If you would like further information on any of the above, any of the support or development services or solutions we offer, however large or small, or our automated monitoring solutions, we'd be happy to talk to you.

- ✓ E-mail Contact@3Ds.co.uk
We aim to respond to all e-mail enquiries within one working day
- ✓ Post
3Ds (UK) Limited,
69, Esher Road,
East Molesey,
Surrey.
KT8 0AQ
United Kingdom.
- ✓ Telephone (+44) (0) 208 144 4141
- ✓ Fax (+44) (0) 1932 225349



3Ds (UK) Limited
Design, Develop, Deliver Solutions!

69, Esher Road,
East Molesey,
Surrey.
KT8 0AQ
<http://www.3Ds.co.uk>