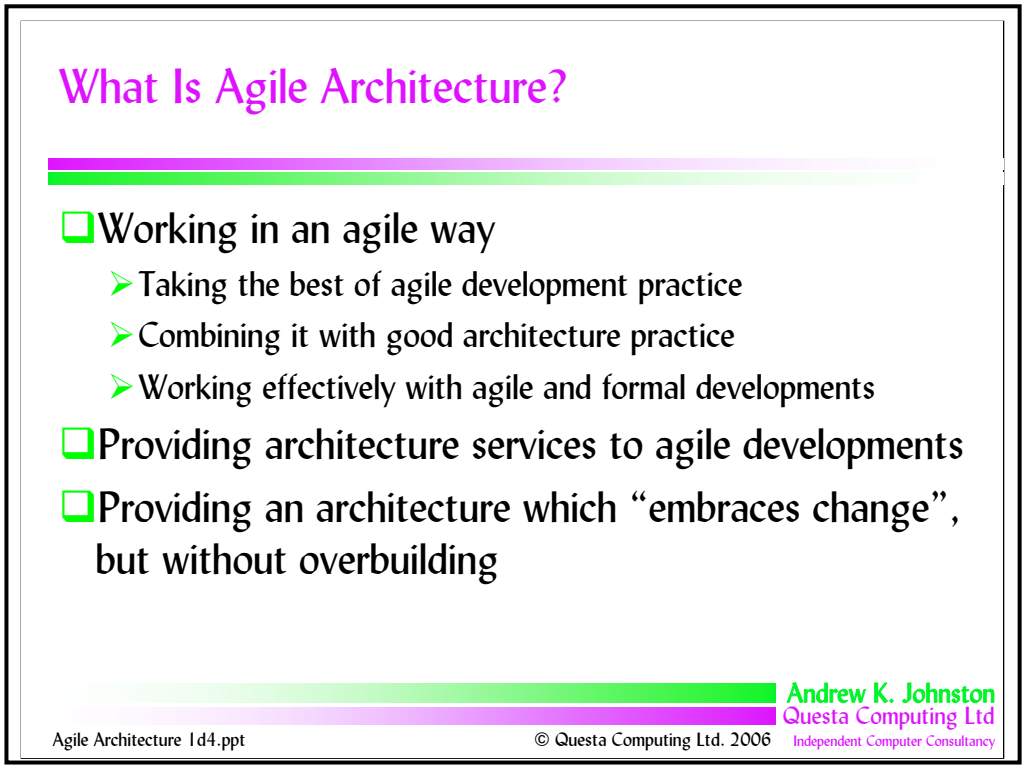
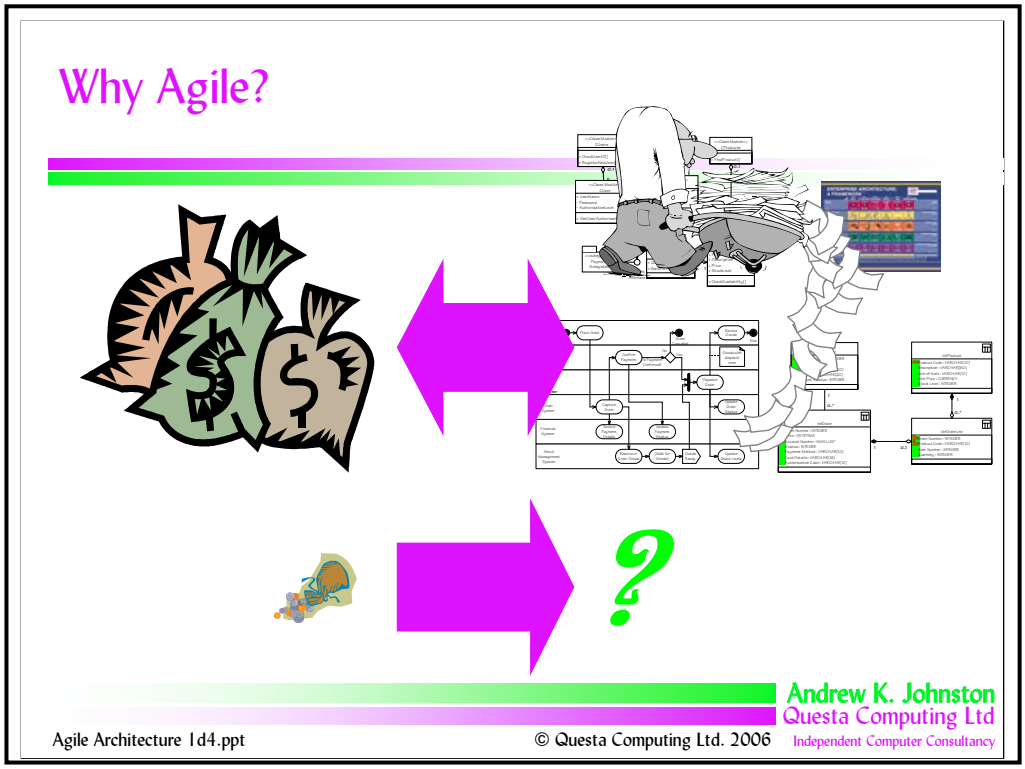
See final page for bio and contact details



Good afternoon, ladies and gentlemen. My job is to keep you awake until the free beer arrives, so hopefully you'll find this interesting.

What's this all about? It's about a couple of different things. Firstly, it's about what we, as architects, can learn from agile development practices, so that we start working in an agile way. Whether your teams follow agile methods themselves or not, I believe you can get a great deal of benefit from agile ideas, and become agile architects.

It's also about making the architecture itself agile, able to "embrace change" as Kent Beck would put it. This doesn't happen by accident, it happens by design, and I'll provide you with some ideas of how.



How did I get started with this? Well, I've been in the business a fair few years, and I've never yet been involved with a successful project which followed a waterfall process and "big analysis up front", as the agile people like to call it.

I have seen several which were cancelled just after finalising a whole bucket of documents. Now I'm not saying that it can't work, but my experience seems to be backed up by those Standish Group statistics we're all tired of hearing, which suggests that no-one else sees such projects succeed very often, either.

But I *have* been on lots of successful projects. And they all shared certain characteristics – an incremental, iterative approach, a belief in the judgement of skilled people over what the method says do next, getting the users and the developers physically close together. Some even used timeboxes and pair programming. So I do have some faith in agile methods.

But at the same time I'm also an architect, and I'm sceptical about the way eXtreme Programming and the others dismiss the value of analysis and design.

<Click>

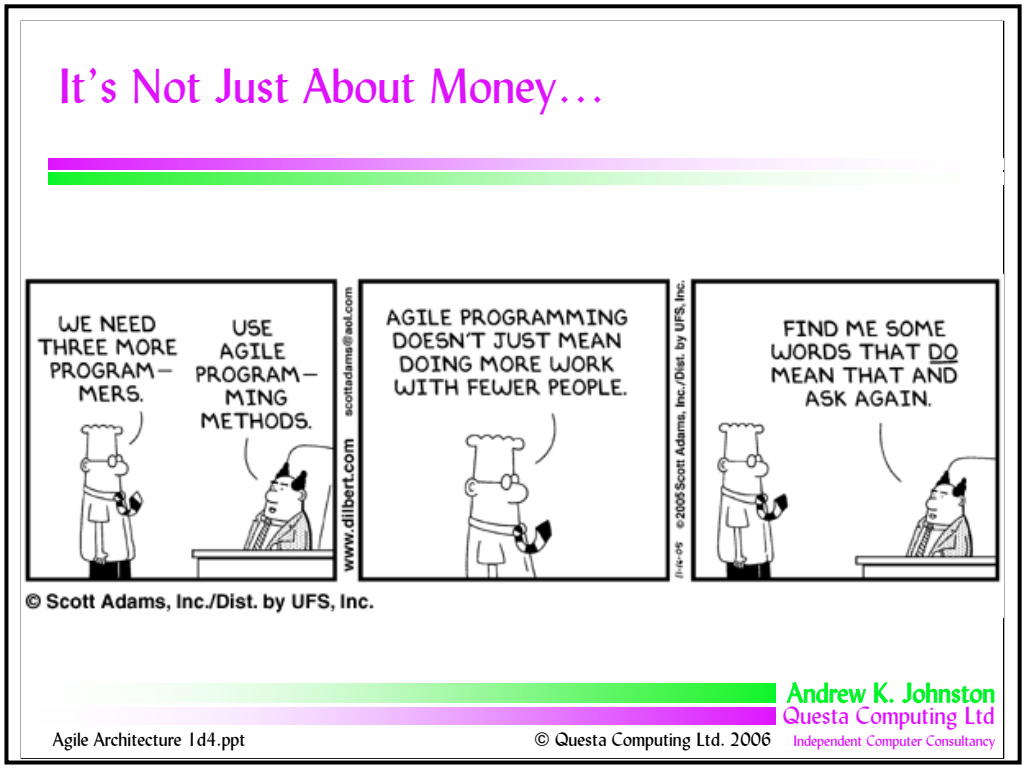
As well as developing software, I also do a lot of work at the enterprise level. But again, I've never seen any organisation really succeed in the "big top down analysis" approach suggested by a lot of Enterprise Architecture Frameworks.

One core problem is limited resources. Lots of models to fill in your framework requires a significant investment, and lots of skilled people, which usually exceeds the available supply.

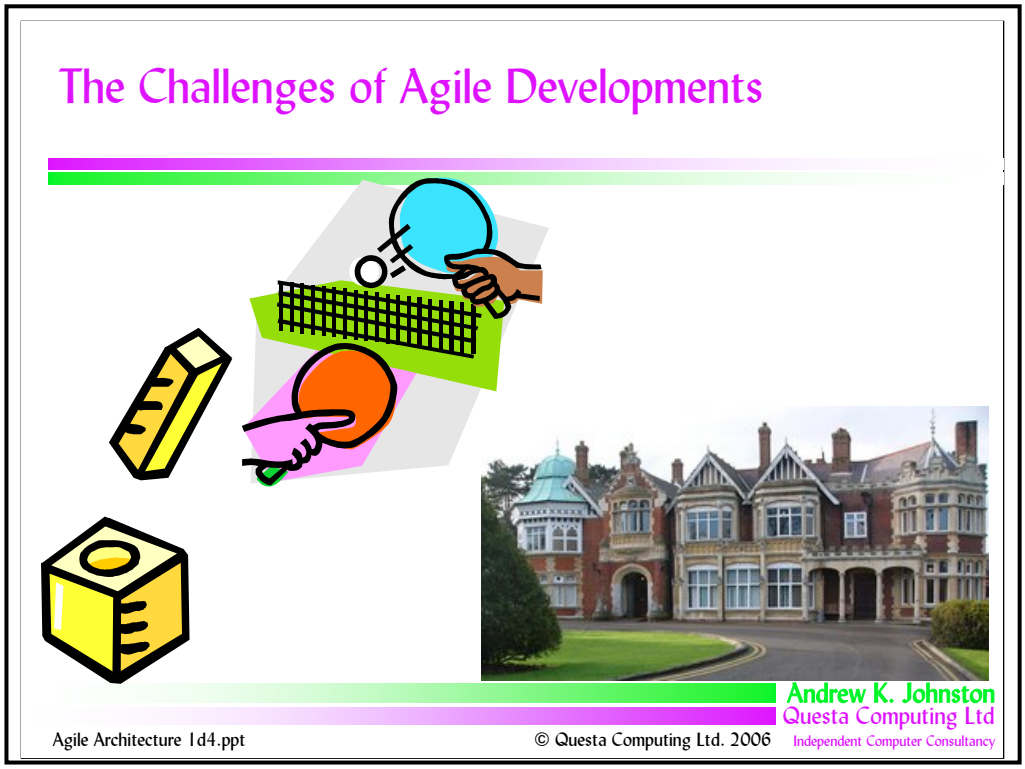
<Click>

The big enterprise approaches don't make it clear how to succeed if you only have a small pot of money and people. Another problem is that such approaches also require a relatively mature organisation and architecture – it's not easy to get started.

I think there's a natural solution, which is to try and apply agile approaches to the work of the architect. That's what I've tried to do.



Contrary to popular opinion, "agile" doesn't just mean doing more work with fewer people. I happen to believe that this can be a result, but that shouldn't be the reason you do it. Hopefully the rest of this talk will explain the other reasons and benefits.



So what about the other side of the coin? What if your developers are already using agile methods? Despite what some of the agile community say, you need a solid architecture for a successful development of any size or style. I don't accept that somehow the architect can be replaced by some sort of "group mind".

In fact XP and SCRUM both have the concept of architecture – XP calls it the "system metaphor" – but both often ignore it in favour of constant refactoring, which may or may not keep the design pure.

<click>

Who recognises this building? It's the manor house at Bletchley Park, where the Nazi Enigma code was broken in World War Two. I could go on about that work as an example of a successful agile project, but that's not why I've included it here.

The building is a wonderful example of the problems of architectures which are allowed to "grow organically". For example look how each section of the front has a different style, and how the round turret at the end has been cut into an old gable, without tidying up the old structure.

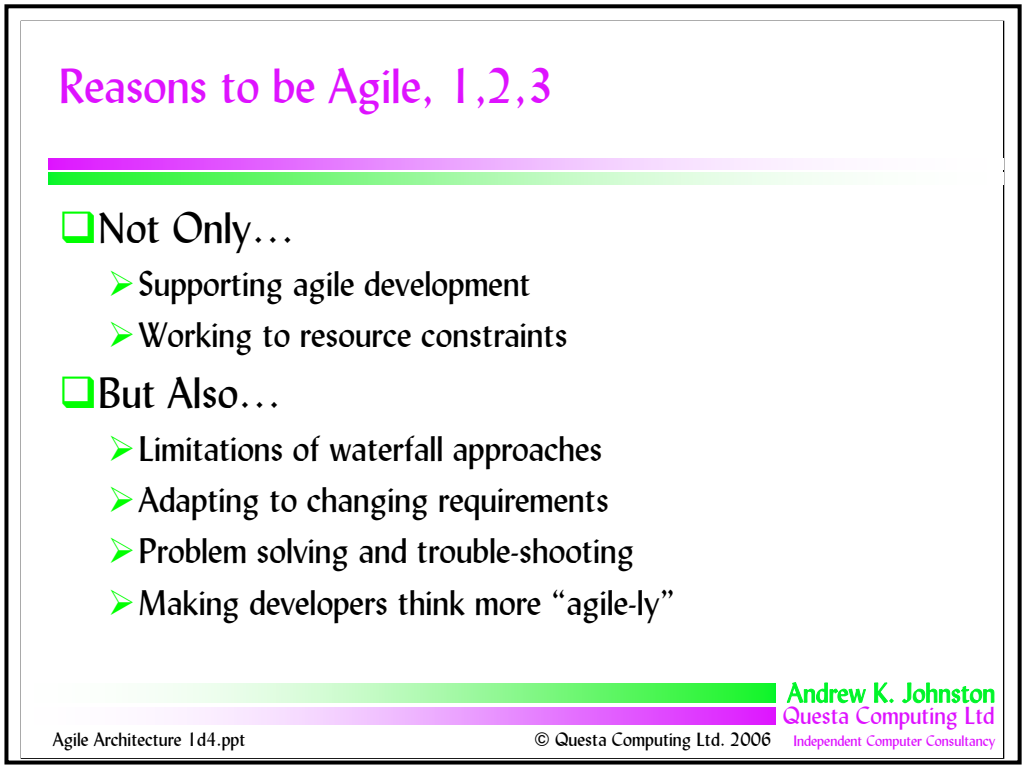
If you don't control them, it's easy for agile developments to produce the software equivalent. I'm sure we've all seen systems which have grown through cut and paste code manipulation, with results not unlike this.

<click>

Although agile methods tell you to "embrace change", change is not necessarily free – it depends on what changes. And if the developers are agile-ly reacting to every user request then you can easily get a very high rate of "ping pong" churn in some requirements. I worked on one system where the marketing department was constantly feeding in changes, many of which went against common sense and ultimately had to be reversed out. In such circumstances a strong architecture does three things: it reduces the cost of change, it provides a reference against which the sillier requirements can be challenged, and it can provide mechanisms which allow incorrect requirements to be rapidly reversed out.

<click>

So any development needs an architect. The problem is that the architect may be a "square peg in a round hole" unless he or she adjusts to fit in with the project structure and approach. And the architect's role may change with the style of project, maybe to coordination rather than sole ownership of design. The truly agile architect can deliver value to any sort of project.



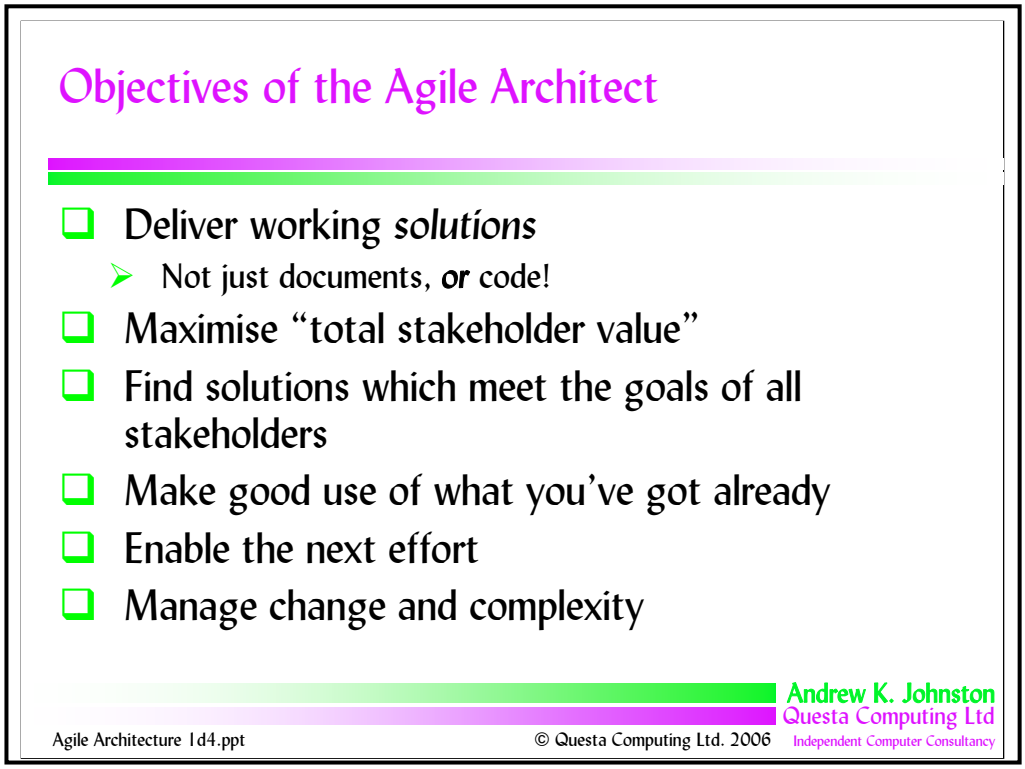
Even if you're working in more formal developments, there are lots of reasons why you might want to be more agile.

First and foremost, the budget or staffing may not allow for a "do everything" approach. If so, you'll have to choose what to do and what not to do. We can learn a lot about how to make these choices from agile approaches.

Just "trying harder" with architecture doesn't address the classic problems of waterfall developments. Technical and commercial decisions have to be made without adequate information, and the technical risk is deferred to late in the development, where it has greatest impact. Also, whatever we like to think, requirements *do* change, in unpredictable ways, and we need agile ways of responding to them.

Finally, agile is a way of thinking. It's about taking the shackles off, and letting yourself range over possible solutions. This is good news for both the initial design, and for when you have to solve problems. It's a skill every architect needs to learn, and then needs to teach to other developers.

However, being agile is not a "licence to hack". You still need to control key aspects of any development. But if you're agile, you'll start to think more constructively about which steps to follow, and which to avoid.



The objective of the agile architect is a solution, or solutions, which meets the business need, balanced with other constraints and goals

This contrasts sharply with other processes:

•Agile development methods like XP are all about delivering code, which may not be necessary

•Some formal methods seem to be mainly about delivering documents, and any software which emerges is treated as a sort of unpleasant by-product

•Processes promoted by software or service suppliers are often focused (sometimes covertly) on making you choose a solution which sells more of their product

The best agile solution may be one where we don't deliver any new software at all, making effective and innovative use of what we have. This is one area where an agile architect may take a different view to processes, either agile or traditional, focused on "doing something new".

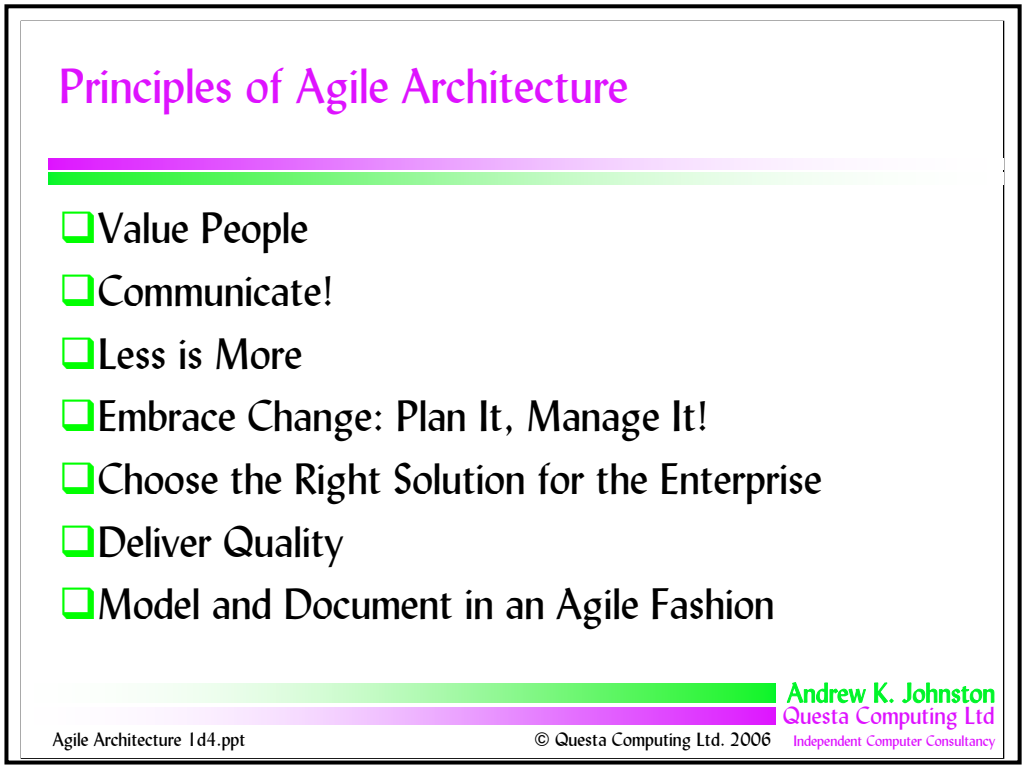
The agile architect aims to optimise "total stakeholder value", a "sweet spot" balancing all the stakeholder requirements. You have to consider a mixture of quantitative, approximate, qualitative and personal, human factors. As a minimum, you have to meet the minimum goals of all stakeholders, or they will block the project or ensure its failure.

To achieve this the agile architect has to become an "independent broker", and may have to follow a line which diverges from the IT or business "party line". Effectively the architect becomes the "agent of the client", but recognises that there are many clients, with many different objectives.

Note that maximising "total stakeholder value" is not the same as minimising TCO, unless the business has a very broad view of "costs". Any good architect has to address himself or herself to both the cost and value sides of the balance.

You do have to think about the future, but avoid getting trapped in "analysis paralysis" - thinking about the future so much that you make no progress in the present. At the other extreme some agile methods only think about today, and ignore the real cost of change and complexity over time.

The agile architect has to strike a balance between these extremes, making sure that real progress can be made, but also trying to avoid short-termist decisions which just store up trouble for the future.



All agile approaches are founded on a set of key principles, and Agile Architecture is no different.

When I started thinking about how to apply agile ideas to architecture, I looked at the principles of the Agile Alliance, Extreme Programming, DSDM, Agile Modeling, and others. I found several common principles, some of which could be adopted immediately by the agile architect, but some needed a bit of modification to fit the architect's viewpoint.

This is the set I finally came up with.

•"Value People". If there's one core theme in all agile approaches, it's that people, applying their brains, matter more than following processes.

•"Communicate!". Success in architecture is all about communication. I wouldn't expect any decent architect, agile or not, to argue with this one. Hopefully I can provide a few pointers on how to achieve it.

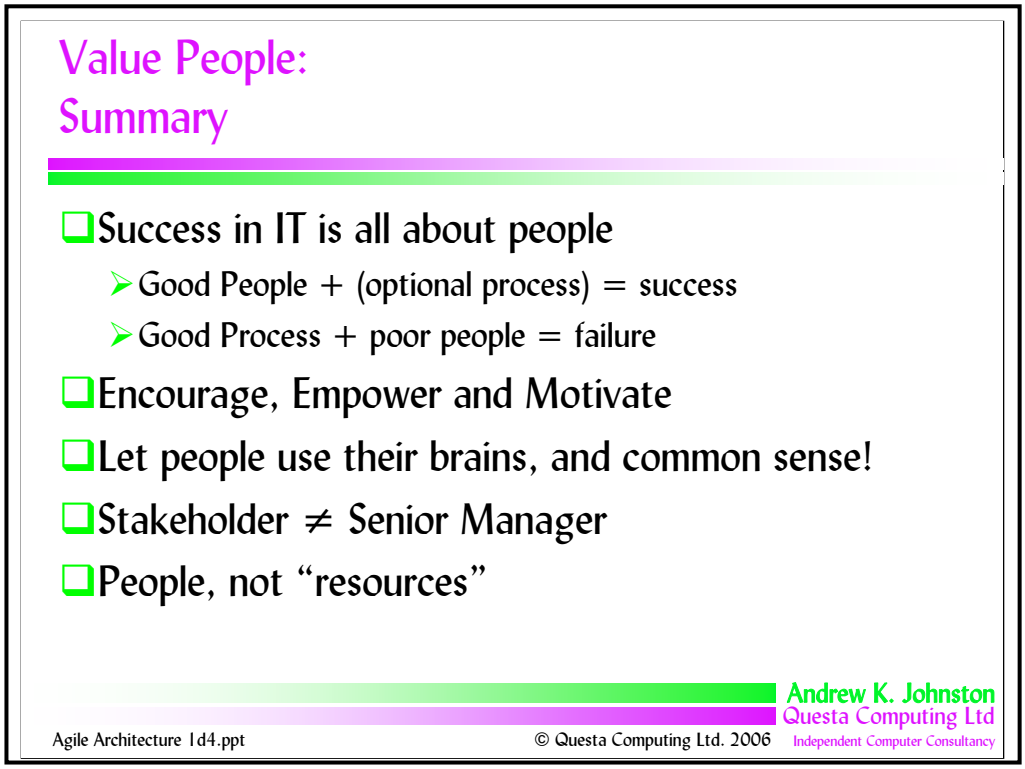
•"Less is More". A lot of agile approaches talk about "keeping it simple" and "just good enough" and "valuing simplicity". These are all good principles, but need some interpretation to deliver an agile architecture. I wanted to boil them down into a single core concept, and "less is more" is what I came up with.

•"Embrace Change!" is the war cry of eXtreme Programming, and others. The trouble is that they work on the assumption that "change is easy and cheap". In real, complex, enterprise-scale systems it won't be, unless you've planned and designed for it. So the agile architect's version is slightly different.

•"Choose the right solution for the enterprise". This is the "balancing goals" thing again. The best solution isn't necessarily the cheapest, or the simplest, or the best technical answer, or the best fit to the users' requirements. It could be none of these. This is about finding that optimum solution. For an agile enterprise architect, it's about finding an optimum set of solutions rather than the most elegant architecture.

•"Deliver quality". Agile methods make up for their lack of formality by embedding techniques, like pair programming and test-driven development, which increase the quality of the result. In the agile architect's world things are a bit different, but this principle tries to capture the same spirit.

•"Model and document in an agile fashion". The agile architect's deliverables include documents and models. The principles of agile modelling maximise the value of the project's modelling and documentation work.



Success in IT is all about people. With good people the process becomes largely irrelevant – good, motivated, empowered people will find a way of getting a good result regardless. Conversely, if you don't have good people, or they are not motivated, or they don't feel empowered, or the process gets in the way then you're doomed to failure, no matter how good your process is.

If there's a single idea at the root of all agile approaches, it's this one. In my Eurotunnel days I had two bosses who really understood this, and were both very inspiring. Remember this was in the days when I still believed in the waterfall, and before "agile" or even "iterative" were common words. The then IT boss summed it up something like this. "I don't want my guys advancing in a neat line, one step at a time, into the German machine guns. I want a team of crack commandos, who get in, do the job, and get out quickly."

He had two signs on his wall. One said "It is easier to ask for forgiveness than permission." The other said "Lead, follow, or get out of the way. But do *something*." *That* is empowerment.

This is about people using their brains. It's about making the best possible use of the good, bright, people, and that means not letting the process and the bureaucracy get in the way. But it's also about being alert to the limitations of individuals. If they don't have the right abilities then all the process in the world won't solve the problem, and you need to work with the managers to find a distribution of work which plays to their strengths and weaknesses.

This principle is also about valuing the views and the needs of all stakeholders, including the users, developers, operational staff and so on.

You can quickly spot an organisation which doesn't do this. Just ask them to list the stakeholders. If their list is just directors and the top levels of the management hierarchy, then you've got a problem.

Other common signs of not valuing people include management by dictat, with no real consultations, or ascribing no value to efficiency improvements. In such a situation, the agile architect should make it a primary objective to try and address this imbalance, promoting the needs of the other stakeholders.

And remember, they're people, not resources. If you treat them like anonymous things, they'll behave accordingly. Reap as you sew!

Let me tell you a story…

In the last year I've had a modest building project which tells an interesting story about how different people contribute to "the architecture".

In any area with high property prices it's common for buildings to expand over time, progressively extracting more "value" from a given site. Over 50 years something like our bungalow can easily double in size. If you're familiar with Stuart Brand's model of building evolution then these are major changes at the structural layer. Buildings change just as radically as software, only usually over a much longer time.

So we had our loft converted into new rooms. We did it in two stages - the front a few years ago, followed by the rear and new staircase.

The front was designed by a professional architect (with letters after his name). He produced the design, got planning approval, contracted a structural engineer, got building regulations approval and wrote the formal spec for the builders. We found a builder and project managed the build. The result was fine, and does a great job of hiding a growing house behind a deceptively small and quaint front aspect.

However, for the second stage, we took a different route. I didn't want another small room full of odd angles, as proposed by the original architect, and I realised that a small change to the roofline would give us a large clear room, and about another 100 square feet.

Having sold the idea to the sanctioning authority (that's my wife), I then prepared new high level plans, mainly by Photoshopping the old ones, and these were adequate to get planning approval.

Building regulations approval requires more formality, so we engaged the structural engineer directly to calculate things like beam sizes, and create a set of formal, detailed plans. The structural engineer made his own architectural suggestions. One, to move the new internal walls slightly and create more space, we happily accepted. Another, repositioning the bathroom, we rejected as it would have spoiled the architectural integrity of our new space.

The build mainly followed the plans. However the builder persuaded us to change the materials for one wall, and increased the specification of some beams. An intrusive buttress from the first stage was a problem, and he and I jointly designed a way to redirect the loads so it could be removed. Thus refactoring happens in building construction just as it does in iterative software development.

(Notes for slide 10 continued)

So what does this mean for IT architecture? Good ideas come from many different players, and the true architect must combine these into an attractive, coherent vision. Parts of the architecture may be delegated to others, particularly specialists in a particular technology or process. And others will contribute to the architecture's evolution over time.

Who are the architects? Many people contribute to an architecture. You could say they are all architects. "The Architect", if there is one, is the person who guides these contributions into the whole.

OK, OK, why have I got these pictures on this slide? Last year we went to Venice. Now, as you all know, Venice is an old city subject to constant reconstruction, renovation and preservation activity. And as we wandered around, we kept on seeing similar groups of four people, earnestly gathered around one old building or another:

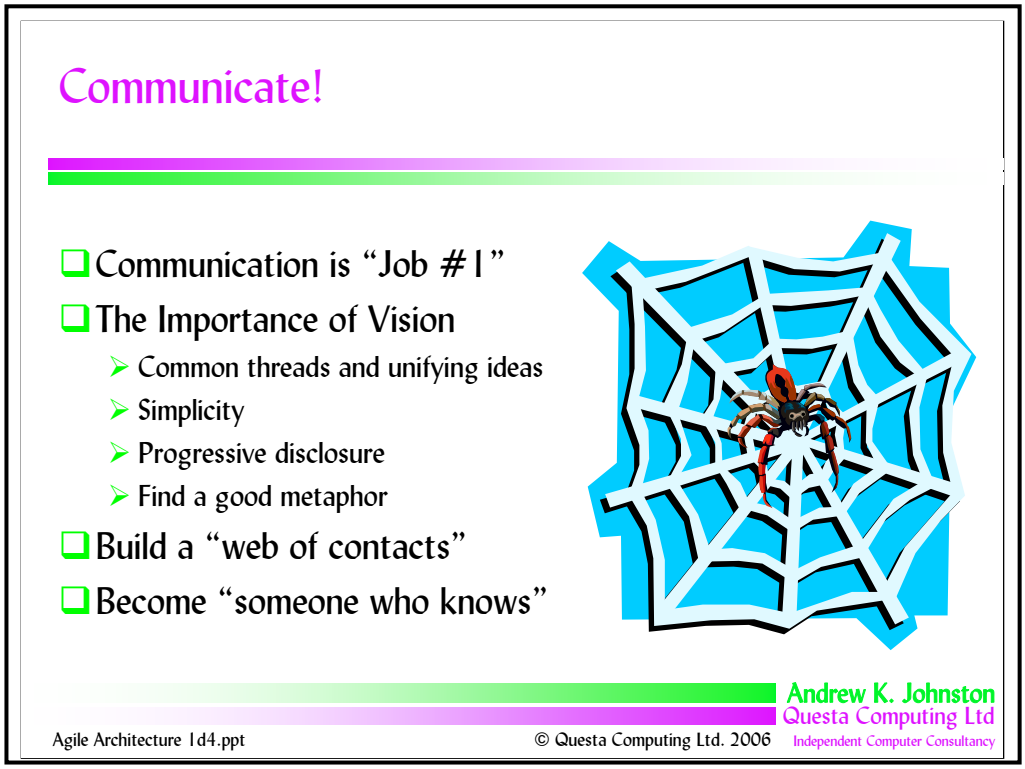
•One had a bow tie, a natty jacket, and, more often than not, a pair of red trousers. The architect. I've got a pair myself.

•Then there was the scruffy-looking bloke in overalls. The builder.

•The project manager looked a bit more formal. He was usually wearing a suit, or at least a jacket and tie, but nothing as flamboyant as the architect.

•And finally, there'd be a fourth person, or maybe a couple or small group. They'd usually be dressed casually, with concerned looks on their faces and hands on their wallets. The clients.

But as I said a minute ago, the architect is not just the guy in the red trousers. In a very real sense, they are all architects.



Communication is at the heart of being a good architect, especially an agile one. A key part of your job is to communicate, clearly and regularly, with all your stakeholders.

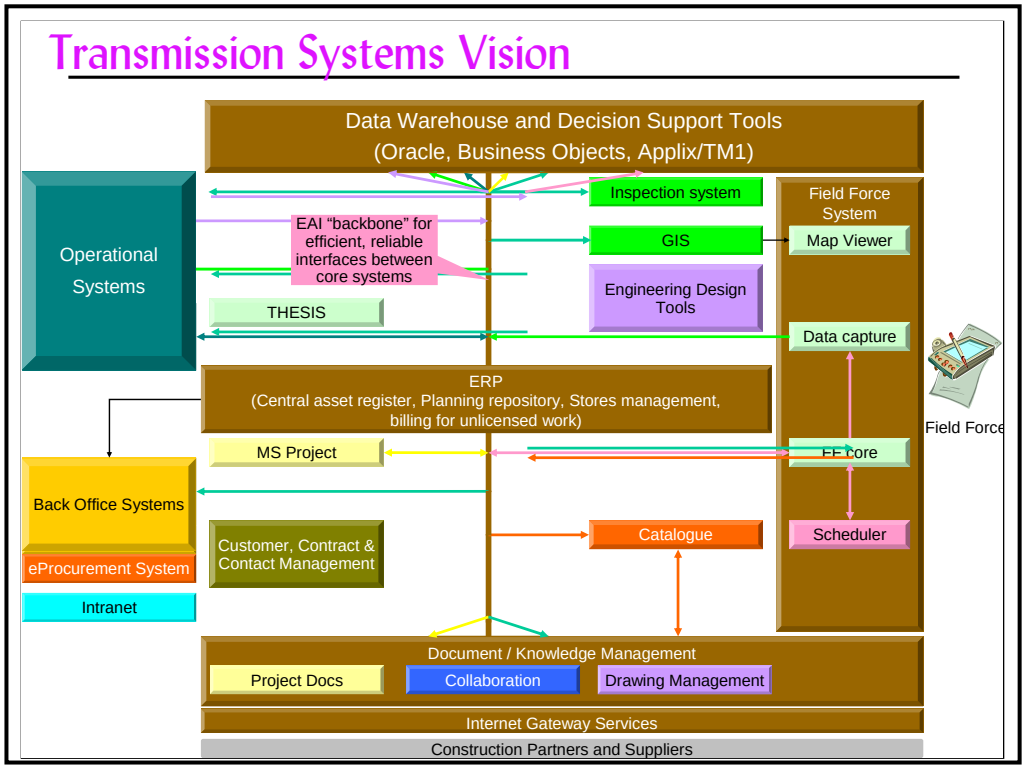
Only by communicating will you understand the goals, constraints and requirements. Only by communicating will you persuade others to adopt and maintain the right, high-quality, consistent solution. Only by communicating will you get others to buy into your vision for the solution.

This vision of the overall solution, how it fits together, and what it will do for the users is arguably the true architect's single most important contribution. People need to "get" the concept, in terms they can understand. This is one of those places where UML doesn't help you, but some good PowerPoint slides will do.

The vision has to communicate both what the system will do – its key functionality – and how it will do it – the key structural elements of the solution architecture. If the two line up, then you win in two ways: this confirms that you've probably got a good architecture, and you will be able to spread understanding of both a bit more widely, so that developers understand the aims, and users have some sympathy for the technical approach. At the enterprise level you may be talking about a set of solutions, but the same is still true.

Extreme Programming doesn't really have the concept of architecture, but instead it has the concept of the system's "metaphor". This is an idea worth stealing – find yourself a good metaphor from everyday life, and people will find it much easier to relate to your concepts.

For example, I have a metaphor for the agile architect. A good agile architect is like a spider, sitting at the centre of a web of contacts, feeling for vibrations which might suggest something happening which needs his attention. One of the challenges is that you don't have enough time to talk to everyone all of the time. But you should try and get yourself into a position where someone you don't know personally is told "you should make sure Andrew knows about that", or "Ask Andrew, he may know the answer".



The trick with the vision is to concentrate on the common threads and unifying ideas. Keep it simple.

Here's one I prepared earlier, as they used to say on Blue Peter. This diagram communicates the enterprise vision for a group of asset and work management systems. The "Big ideas" were:

Key data held in central repositories, not end-user systems

A single view of assets, provided by a single core repository and synchronised satellite systems, rationalising a number of smaller systems in the process.

A single view of work, with consolidated planning and work management using the same core system

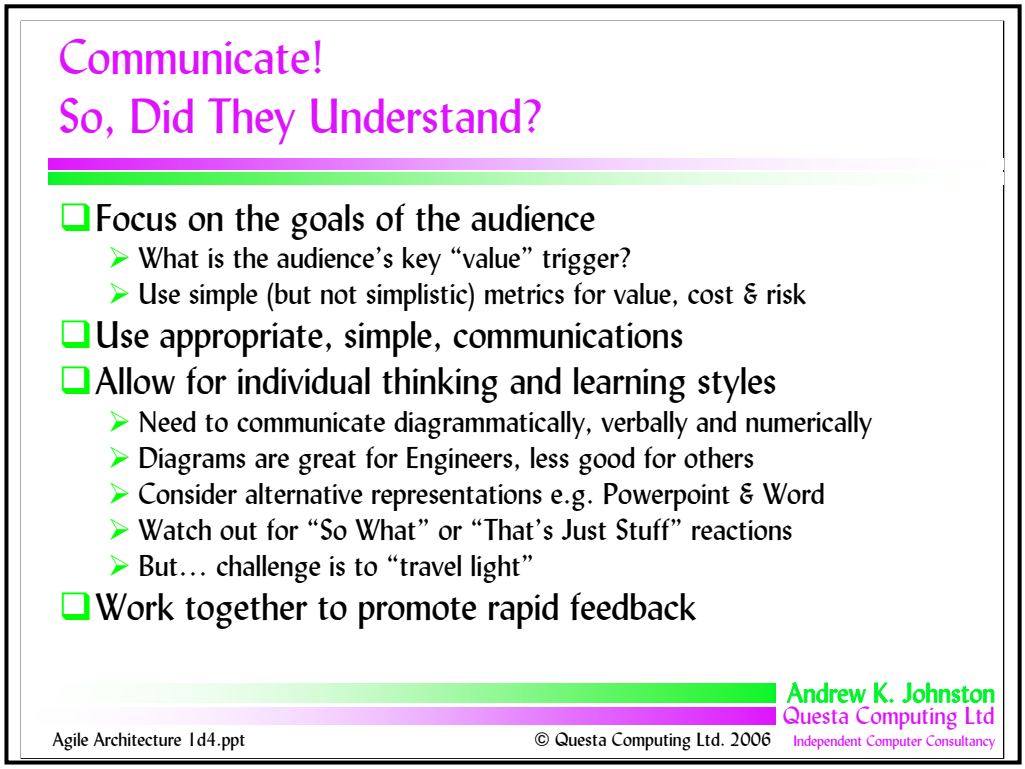
A field force solution to despatch work to the engineers, and capture data in the field

A data warehouse and business intelligence tools to provide consolidated long-range planning

An Integrated Document Management System provides the basis for all document-related solutions, establishing a structured home for things like engineering drawings, and also acting as the centre for collaboration with engineering suppliers

Key systems are linked by an EAI backbone providing for data to be generated once, used many times

Once you have such a vision you can use it and enhance it in many different ways. One very good trick is "progressive disclosure", where you use a sequence of diagrams, maybe even animations, to communicate the highest-level picture first, and then reveal more detail behind the big blocks. Alternatively these can show how things will change over time, which is a key part of your agility strategy.



Focus on your audience. Write every document, and build every maintained model, with a defined purpose for a defined audience. Recognise that different stakeholders need different views, and support them with appropriate information.

Think about the key value of the system, or systems, to the audience. For managers you'll need to explain how the system will make their business easier to manage, or more profitable. For users you'll need to explain how the system will make their jobs easier or more enjoyable. For those who'll run the system you'll need to explain how it will be reliable and easy to run…. And so on.

If you need to talk about things like value, cost and risk then don't be afraid to use simple metrics, but watch out for the "simplistic metric". Don't boil everything down to one financial number, for example.

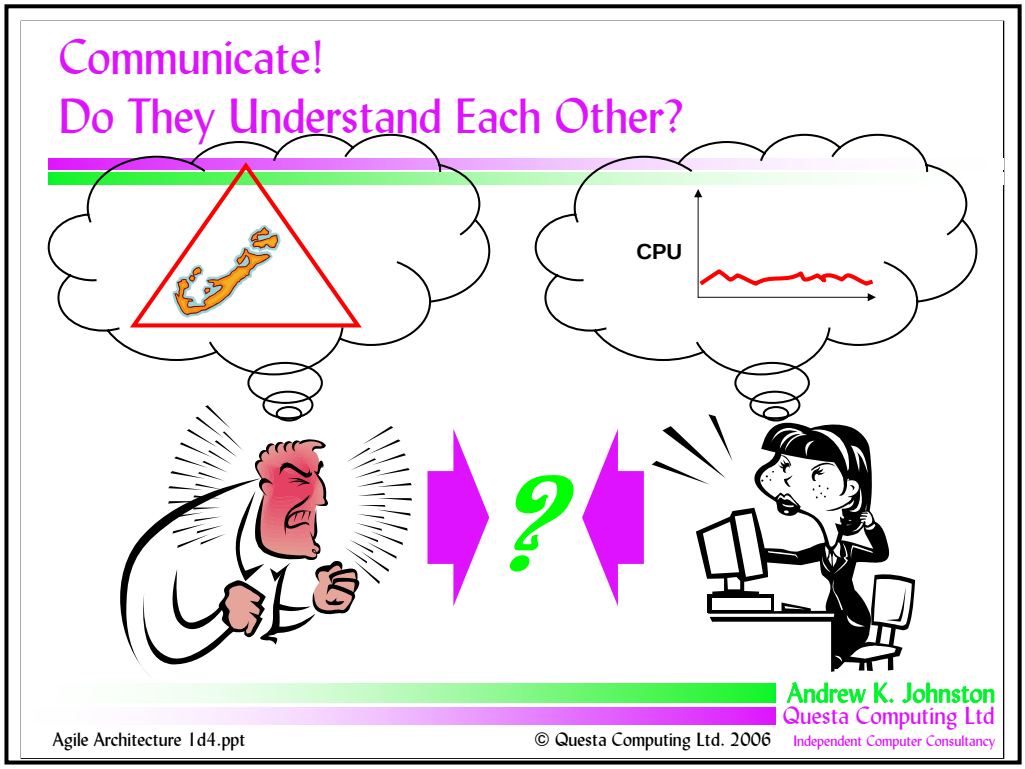
Use appropriate communications for each audience, matched to their skills, goals and needs. Keep the content and style of communications as simple and brief as possible. Keep documents short and models simple. Focus on content, not representation or "polish".

You need to understand that different people think, learn and understand in different ways. Some relate best to pictures, and that's often the mode of choice in the PowerPoint world. But others don't. I had one manager a few years ago who just didn't get the big picture diagram on the previous slide, which everyone else loved. I happened to have a four page Word document which I'd prepared earlier. Bingo! He loved it, said it was made everything much clearer, just by turning the pictures into sentences.

If you're a competent architect then you can probably understand two or more styles of thinking. So you need to constantly remember that some of your stakeholders won't. If you get a "so what" reaction, then you need to refocus, either with a different goal, or using a different style of communication.

Now the only trouble is that one of the principles of agile documentation is to "travel light". You don't want to maintain very much documentation. So your challenge is to support just enough documentation to communicate to everyone who needs it, but not too much.

Remember that the best way of conveying information is by a face-to-face conversation, supported by other materials. Shared work at a whiteboard will generate excellent feedback and buy-in. Work as closely as you can with all the stakeholders, including your customers and other developers. Encourage comments and suggestions, and try not to be protective or defensive of your ideas. Remember that anyone may have a good idea or valuable insight, and that open and honest communication helps people to make better decisions based on better information.



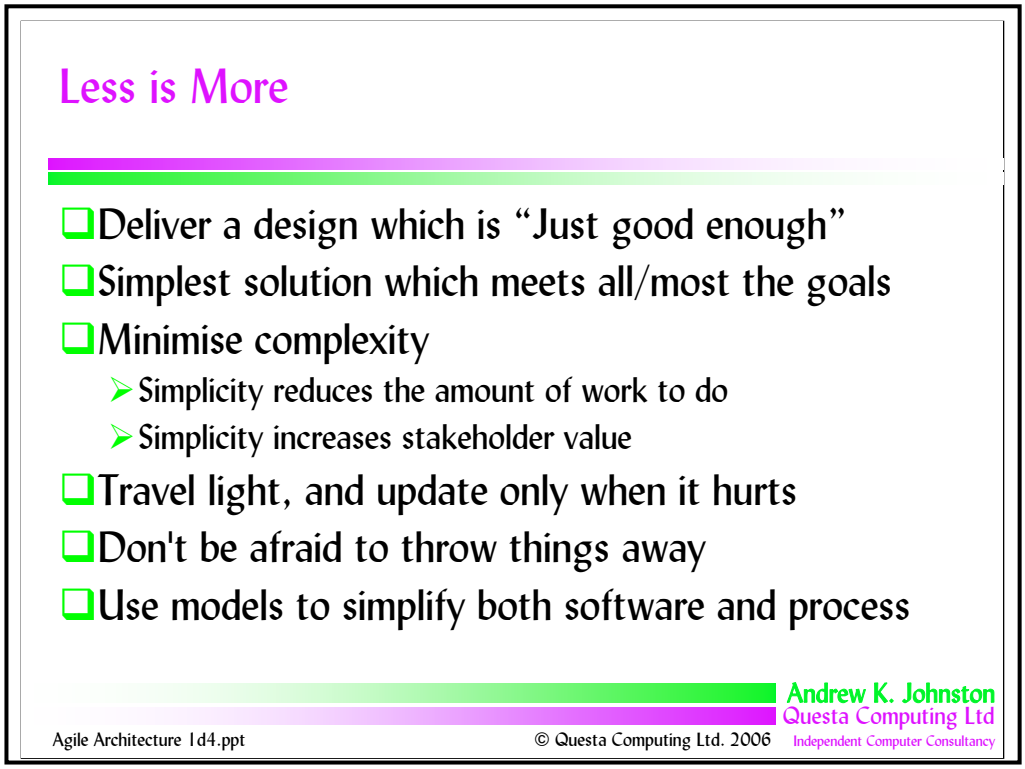
Part of the architect's job is to make sure that not only do people understand you, but they also understand each other. You may be the only person who can translate between the business view and the technical, and if so it's a very important role.

This picture shows a situation I was in a couple of years ago. We had a new system which wasn't performing very well. It had a complex integration scheme, in which messages flowed from one application to another. The business people were making a change in one place, then waiting for a long time, and sometimes the job would turn up promptly at the next point, but sometimes it took a very long time, or even disappeared completely. That's why I chose the icon. The island, in case you don't recognise it, is Bermuda, and some of the business people viewed the system as a sort of Bermuda triangle.

On the other side I talked to the developers. They couldn't understand the problem. They were trying to measure performance using traditional measures like CPU and IO load on the servers, which usually looked fine. And before I got involved they were trying to explain this to the business, and everyone was getting very frustrated.

The solution in this case was to get the developers measuring things with real business meaning – the time taken for a message to go from one person pushing the OK button in the first system to showing up ready for the next person to do something with it.

As soon as the developers starting working with this model they managed two things. They started to talk in a language the business could relate to, and they started to spot all sorts of bugs and bottlenecks to fix, so over time the system has become highly reliable and performant.



Agile programming methods focus on delivering software which is "just good enough". For the agile architect the focus is slightly different – you know you have to deliver solutions which can accommodate known change, and known growth, and fit in with the wider goals of the enterprise. But you shouldn't over-build. Your aim is "just good enough for the enterprise".

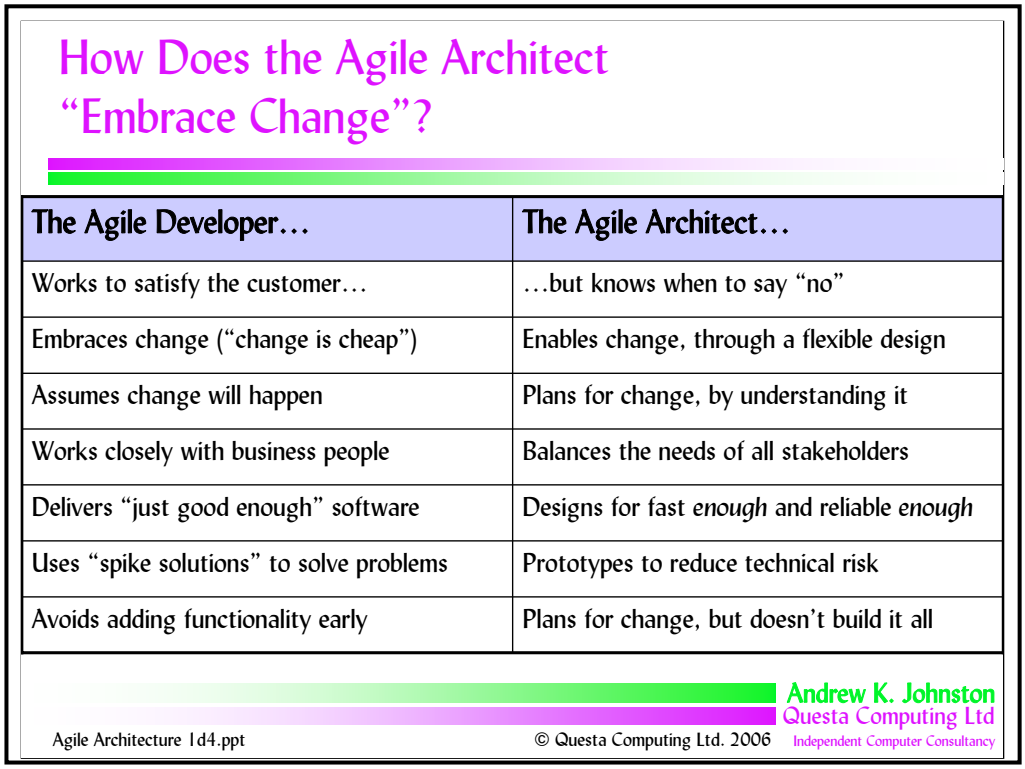
This solution will meet most, ideally all, of the stakeholder goals, including strategic fit and ability to change. It probably won't be absolutely the simplest, crudest solution, but as a general rule, a simpler solution will be better than a more complex one.

However, you do need to take a holistic view of complexity, and apply the same thinking at the enterprise level. For example, creating lots of separate little systems or data sources increases total complexity. So the simplest solution overall might be one larger enterprise-scale database.

One of the architect's key responsibilities is to manage complexity. The more complex the solution the more it will cost to deliver, change and run, and the more things can go wrong. Minimising complexity will deliver stakeholder value. One of the key causes of eroded value is the overhead and extra costs from complexity.

Everything you create: whether document, model, code, test or any other artefact has a cost. If you decide to keep it, each will need to be maintained over time, increasing the cost of change, and what people must know and understand. The less you keep and maintain, the more agile you are. Don't be afraid to throw things away when they have served their purpose.

And remember, the right amount of modelling makes your life simpler. It's much easier to explore an idea by drawing a diagram or two instead of writing code.



The difference between the agile developer and the agile architect is most graphically shown by their different attitudes to change. The agile developer assumes that "change is cheap, almost free", so doesn't worry about tomorrow. When the customer says "Jump!", they ask "How High?" The Agile Architect knows this knee-jerk approach is wrong.

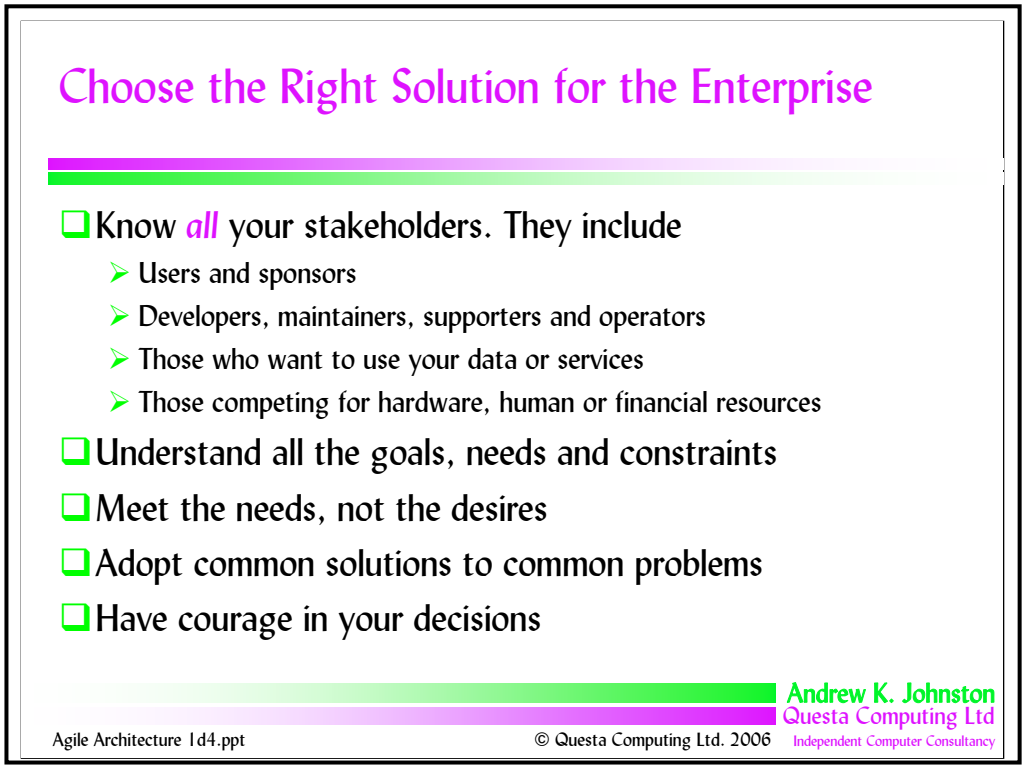
The cost of change in a real-world enterprise system is never that small. You must plan for change, and understand its costs. You must deliver an architecture which can accommodate likely change in the best way for the enterprise. However "agile" your process, if your architecture cannot handle change then it is fragile, not agile. True agility is the ability to undergo change quickly and easily without degrading the architecture, and with as small as possible an impact elsewhere.

Challenge users whose requirements are not stable enough. Remember I talked about ping-ponging requirements? This is where an agile programming approach can get very expensive.

Manage change. Make changes in a controlled way. Don't accept any change regardless of its cost and impact. Know when to say "no". If a proposed change creates a problem, then look for alternatives whose impact is smaller. Common sense and a consideration for other stakeholders are your key tools.

Travel light, so the cost of change is minimised, and communicate so that everyone understands what is changing, why, and how it will affect them.

Manage and plan for change, but don't resist it, and don't anticipate it too far ahead. There's nothing agile about being paralysed by analysis, or overburdening your design with features you don't need. Accept that things grow and evolve, but give that evolution structure, and support, and room.



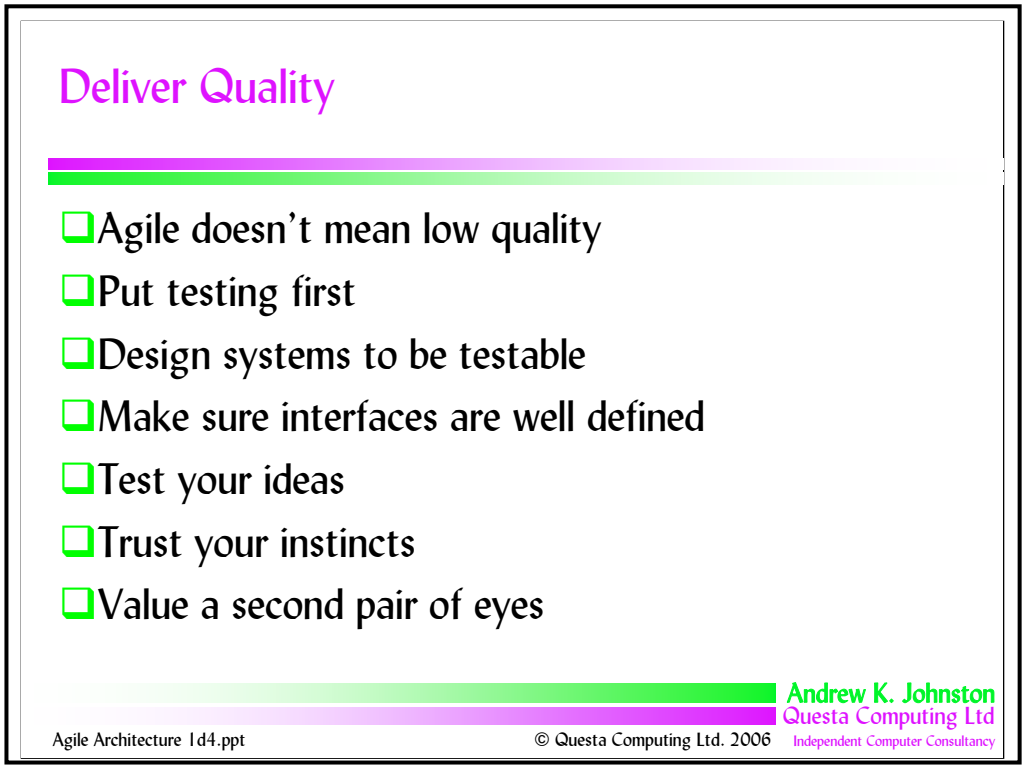
You have to know your stakeholders, and understand their goals, their needs, and the constraints these may place on the solution. As well as the users, and those paying for the system, stakeholders include those who develop, maintain, support and operate the systems.

Others who want to use your data, build an interface or compete for limited hardware resources are stakeholders too.

Actively try to find the best balance of goals and constraints for the overall situation. The best solution for the enterprise might not be exactly what a particular stakeholder wants, but you should strive to meet his business needs, if not his desires.

Don't re-invent the wheel. Build and use common core capabilities instead. Avoid duplication, which simply builds complexity. This applies at all levels – within a single system, and also at the enterprise level. Nurture enterprise assets and support groups who want to build or use them.

Have courage in your decisions. Be prepared to defend them. Listen to the arguments, but avoid letting one stakeholder or concern dominate the decision making process.



Being agile in your choice of what to model, document and build is not the same as reducing quality. Poor quality serves no-one: neither you, your current stakeholders nor future ones. So you need to focus on quality work, but measure quality by the strength and usefulness of the deliverables.

Put testing first. Some agile methods develop tests before code. This is a good practice to emulate. If you can't write the actual test scripts, you should still verify your understanding of any requirement by defining how it can be tested.

You need to design the architecture to support testing: ensure each system is ***controllable***, so that tests can be performed easily, and ***observable***, so you can verify the test, or find out what has gone wrong.

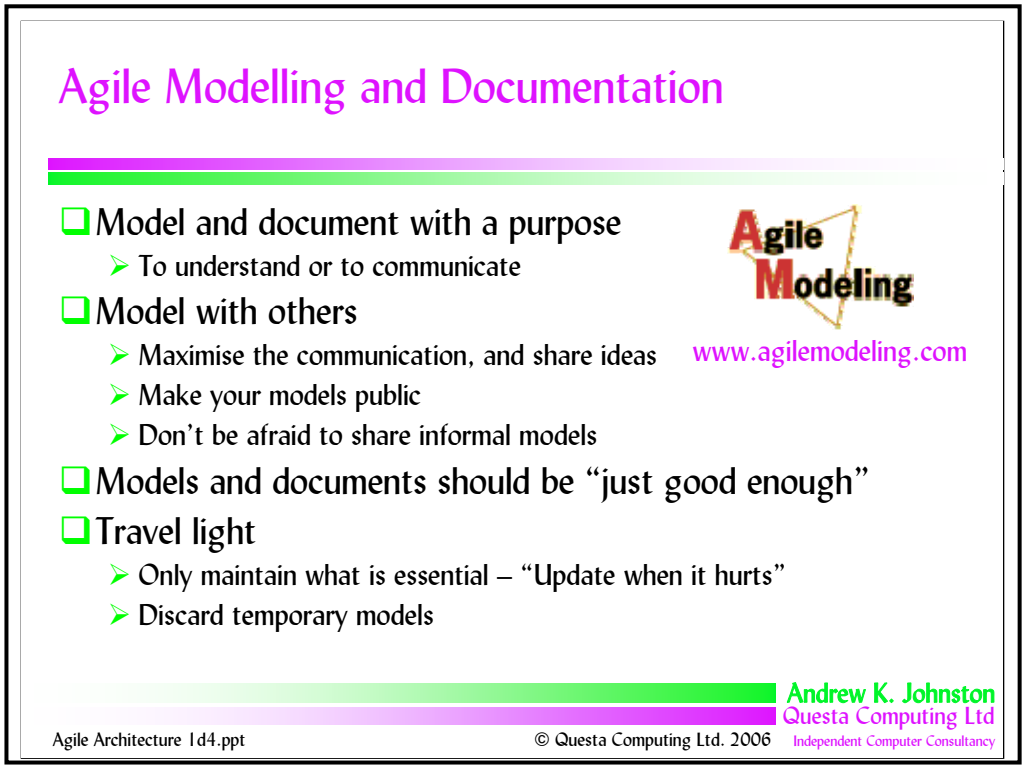
Make sure interfaces are clear and well defined. This applies to both physical interfaces between systems, and human interfaces between groups. Define physical interfaces using formal models, or take advantage of component and service interface definitions. These need to be formally controlled and maintained – they are the contract between the developers and users of different systems.

You may not need to be so formal about human interfaces, but make sure you understand who is responsible for what. Don't be afraid to write this down and get agreement, especially in more difficult situations.

Test your ideas. In some cases the best way is to write some code, and develop a prototype to reduce the technical risk. If this is not practical, think of other ways in which you can "test" the architecture, without committing a lot of effort to an unproven design.

Trust your instincts, and those of other people. If something "feels" inconsistent, or unworkable, then there may be a good reason. Use your instincts to guide your modelling and testing efforts, and challenge requirements which don't pass common-sense checks. If two alternatives cannot easily be rationally separated, but one "feels" better, go with that choice.

But don't underestimate the value of a second pair of eyes. I'm actually a great believer in the eXtreme Programming practice of "pair programming". If this works, then why shouldn't a similar approach to design? Sometimes the right approach is to work with a colleague. But even if I develop something alone, I always try and get almost everything peer reviewed.



A lot of agile methods leap straight into code, and try and get away without any modelling or documentation. This ignores the great value of modelling, as a way of understanding the problem and shaping the solution at very low cost.

On the other hand, many formal methods seem to produce megabytes of models and shelf feet of documentation, but nothing of enduring value. That can't be right, either.

The best balance seems to be the one invented by the Canadian agilist Scott Ambler. He invented the idea of "agile modelling" – getting the value from modelling, but working in an agile way to minimise bureaucracy. There's a great web site and book if you want to find out more, but note that being from the other side of the pond Scott spells modelling with one "L", not two.

The basic idea is that you only model, or create documents, with a clear purpose in mind. There are only two real purposes to modelling, either to understand the problem, or to communicate with others. Framing and exploring a possible solution are special cases of these two.

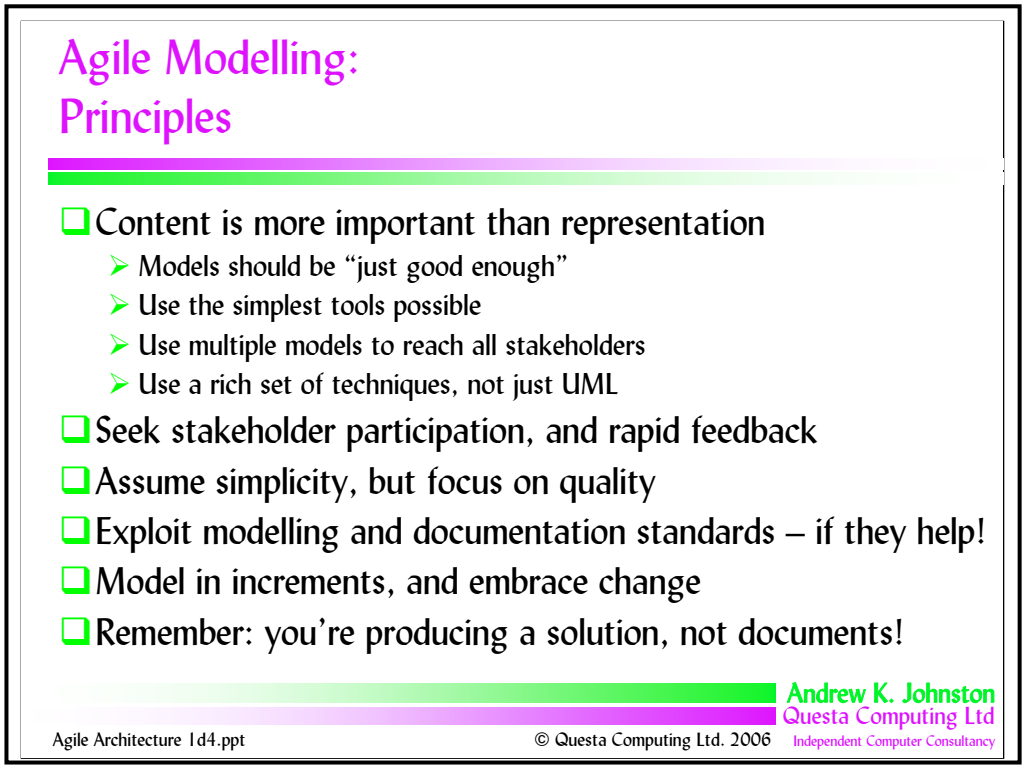
Just like any other agile activity, modelling is best done with other people. So to the agile architect, modelling is ideally about sitting or standing around a whiteboard, talking and using the whiteboard to share and record ideas. That maximises the communication potential, and anything else is probably inferior.

Sitting in a pub drawing pictures on the back of an envelope might well be agile modelling. Sitting alone for many hours with an expensive CASE tool probably isn't.

And when you've created your models, make them public. This may be as simple as keeping the whiteboard visible, it may posting a digital photo on a web site. But it shouldn't usually mean hours of work converting a model which "works" into a different form.

The aim, like other agile things, is to produce "just good enough" models and documents for the job in hand.

The other fundamental idea is to "travel light" – to keep the amount of maintained documentation to an absolute minimum. Not every model becomes a document. You only maintain the documents and models which are really essential, and then only when there's a real reason to do so, not just because the process says it's the next thing to do. And throw documents and models away once they've served their purpose.



Agile Modelling is based on a number of key values, principles, and practices. I've only got time now to share a very high level overview, so for more detail go to Scott's web site.

Firstly, content is more important than representation. If your model captures the right ideas and information, it doesn't matter if it's a bit informal. Only "tidy it up" if it's absolutely necessary.

Use the simplest tools possible – quite often these will be pencil and paper or whiteboard. Sometimes the "right tool for the job" is PowerPoint, or a drawing tool, or a fully-fledged CASE tool. That's fine too.

Don't be afraid of using lots of different models. To fully understand a problem you may have to create several different models in parallel. You may have to model the same ideas in different ways for different audiences. That's the concept of "viewpoints" and is an important part of good architecture, agile or not.

Don't trap yourself in UML. There are lots of other good modelling techniques, and the right one for the job may be something more unusual, or maybe even more old-fashioned.

Most of the time, you're modelling to communicate and understand. So get your stakeholders involved in modelling, and use to get feedback. Expect this to generate comments, and counter-ideas, and changes. That's good, because the cost of handling a change on a whiteboard model is cheapest of all - even cheaper than the XP guys tell you it is to change the code! Open and honest communication is your goal.

Keep it simple. As well as using the simplest tools, this means keeping the content simple. Resist the temptation to "dress models up" with decoration which doesn't add any real information. If you want to learn about how to keep visual representations simple, read one of Edward Tufte's books.

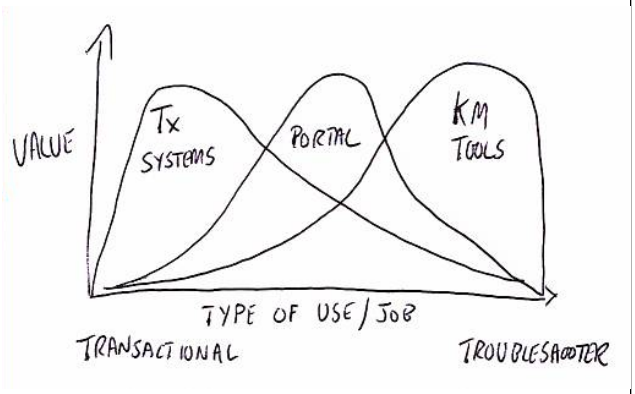
Having said that, your aim is quality work, so don't omit things which really are necessary, and follow appropriate standards, particularly when your model is going to be taken forward and worked on, or maintained, or transformed by someone else. The agile enterprise architect may be trying to work across several federated groups, and following basic documentation standards may be a good idea. The important thing is to make them your servant, not your master.

Don't try and model everything at once. Remember, you're trying to avoid "big analysis and design up front"– that's not agile. So expect to develop your models incrementally, and change as you do so.

And finally, remember your aim is to effectively implement requirements, not to document them. Find ways to test your modelling ideas, and focus on using them to get a working solution as early as possible.

Agile Modelling: An Example

- Don't be afraid of hand-drawn models, and new representations



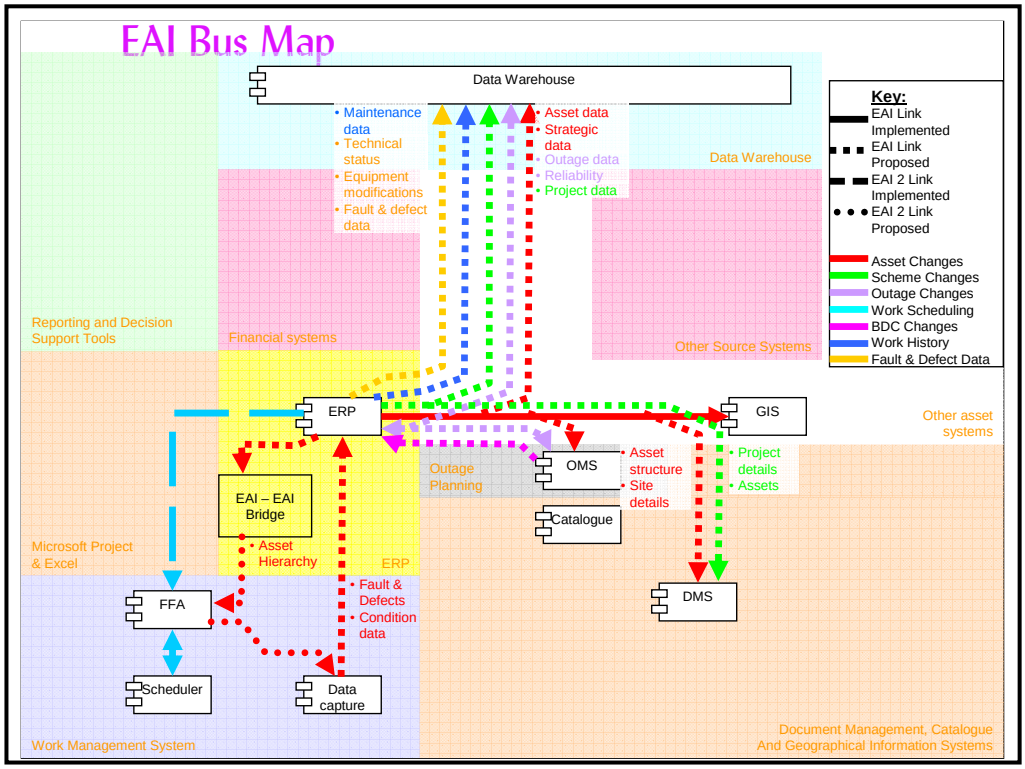
Andrew K. Johnston
Questa Computing Ltd
Independent Computer Consultancy

Agile Architecture 1d4.ppt © Questa Computing Ltd. 2006

Here's a real example of an agile Enterprise Architecture model. We were discussing the possible role of portal technology when we developed this model of where it delivers best value. The y axis shows business value, and the x axis positions the nature of work, from the mainly transactional to very variable troubleshooting.

A given worker's role can be modelled as a profile curve. Basically a profile which peaks near the left-hand end relates to users who work mainly with transactional systems. Workers at the right-hand end need knowledge management systems, to find information in unpredictable ways. The portal delivers most value to those in the centre space.

The things which make this an agile model? It's "good enough" in its current form. It helped communicate these ideas to quite a broad audience. And therefore it was the right model for the job, although not a standard representation.

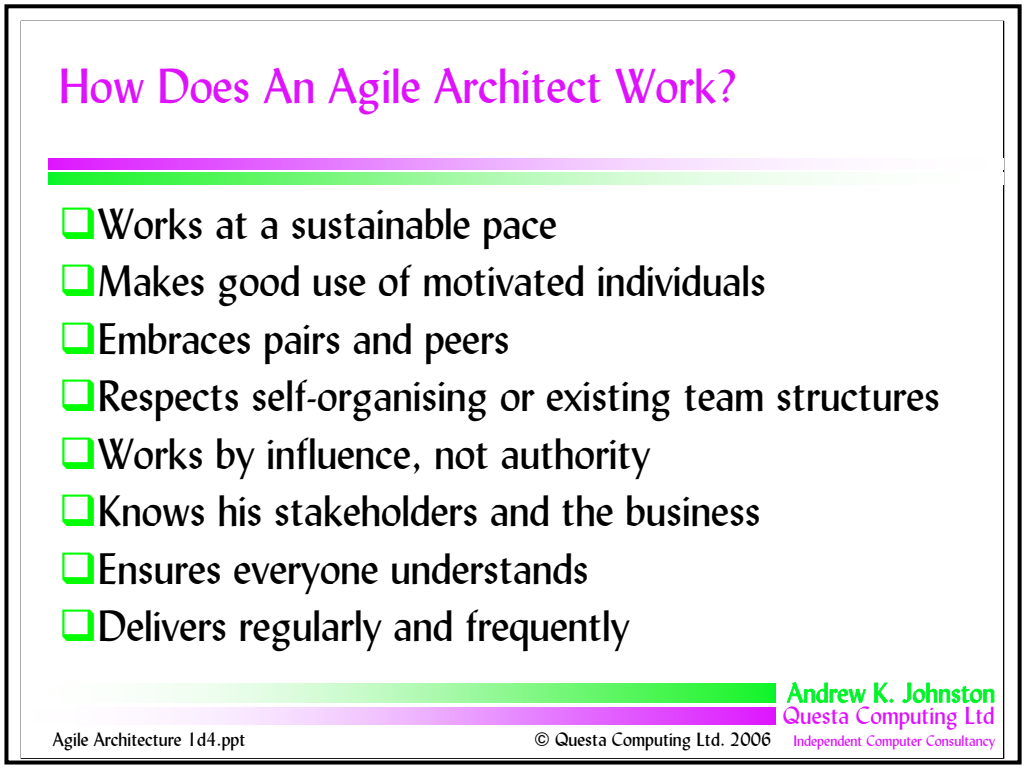


If you're serious about agile modelling, don't Be afraid to experiment, or to "Borrow".

Guess where I got this diagram from? It's actually based loosely on the Paris bus map, modified to show how data flows around a moderately complex EAI scheme.

I got the idea because we always referred to our EAI scheme as a "bus", and I started thinking of it like a set of bus routes, where a given type of data gets on at one stop, the source system, and gets off at another, the destination. And just as sometimes on a bus map the routes split, something similar happens on our bus, where data goes from one source to many destination systems, to keep them all in step.

Admittedly this isn't the world's most agile model, but it has proven a very useful communication tool. You don't always know whether something like this will work or not, but it's usually worth trying.



There are a few tricks to working as an agile architect, and most of them come from the world of agile development.

Firstly, work at a pace you can sustain. It's very easy if a few architects are providing support to a number of projects to get overwhelmed. If you do so the quality will suffer, and you certainly won't be "agile". You need to consciously limit yourself, and to say "no" when you can't commit to something. Even in an organisation with a culture of overwork, if you are consistent in your working pace, but deliver fairly reliably, most people will respect an honest answer.

A good agile architect makes the best use he can of the people around him. There's an old saying that one volunteer is worth ten pressed men, and that's certainly true. Try and use anyone who's keen. Don't be afraid to delegate, but try and understand their strengths and weaknesses.

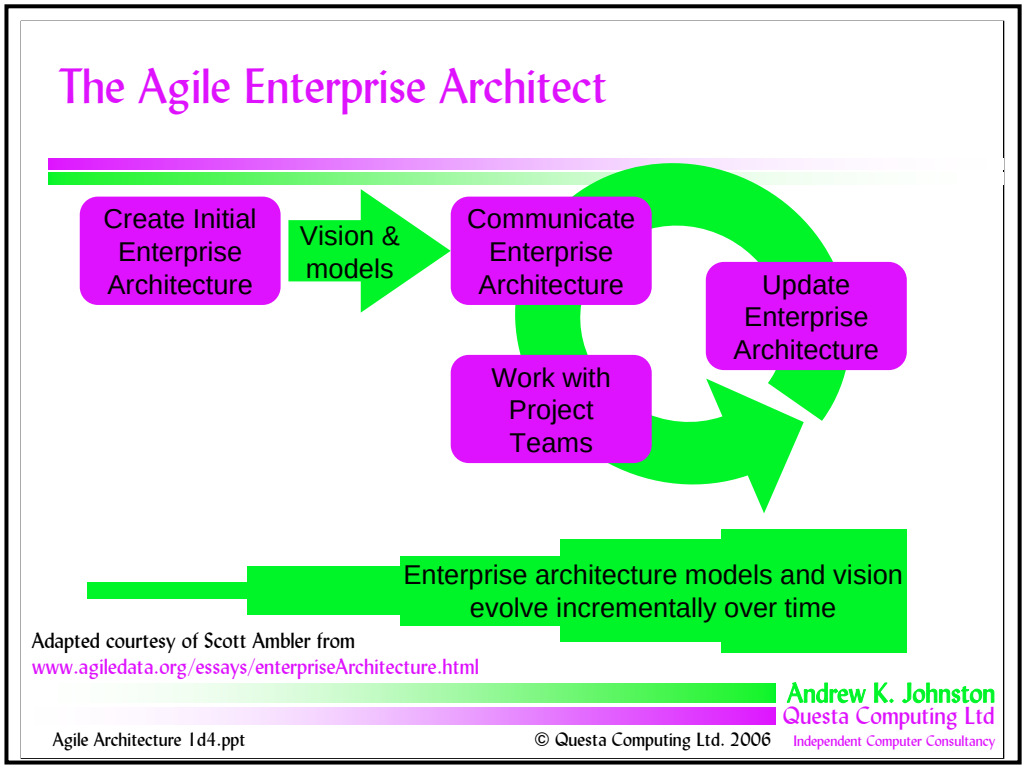
Use your peers as a key resource. If you can, find a partner to share ideas with, and to work with on difficult problems. At least try and get a second pair of eyes on all your work. It will always benefit from some form of review, and most people are only too pleased to be asked.

The best agile teams organise themselves, and often exploit existing team structures and relationships. If you've got existing relationships try and work with them. If the team organises itself a certain way, try and go with the flow. This will usually be more effective than trying to fight it, although you may sometimes have to make difficult decisions or recommendations if there's a genuine problem.

It's often the case that an agile architect has relatively little formal authority. If you can be seen as a reliable, unbiased source of information there's some hope that you'll be listened to. If you have to resort to enforcing things managerially you'll fail.

Work hard to make sure your stakeholders all understand key issues. A good architect, agile or not, spends a lot of time working as translator between different parties.

Deliver your products regularly and, if possible, frequently. A regular delivery cycle makes it easier for people to plan for and adopt each release. Frequent deliveries progressively increase the value to the users, and prove that the project is working. However, be pragmatic – each delivery has a cost, and you need to find a sensible balance.



If you're still awake, you'll notice that I haven't said much specifically about *enterprise* architecture and enterprise architects. This is quite deliberate, and I haven't forgotten the title of the conference.

I suspect I distinguish less between software and enterprise architecture than some people do. For me enterprise architecture is a question of "level" rather than something fundamentally different. The processes, and how the agile architect can apply agile ideas, still apply. It's just that they apply to groups of systems rather than single solutions.

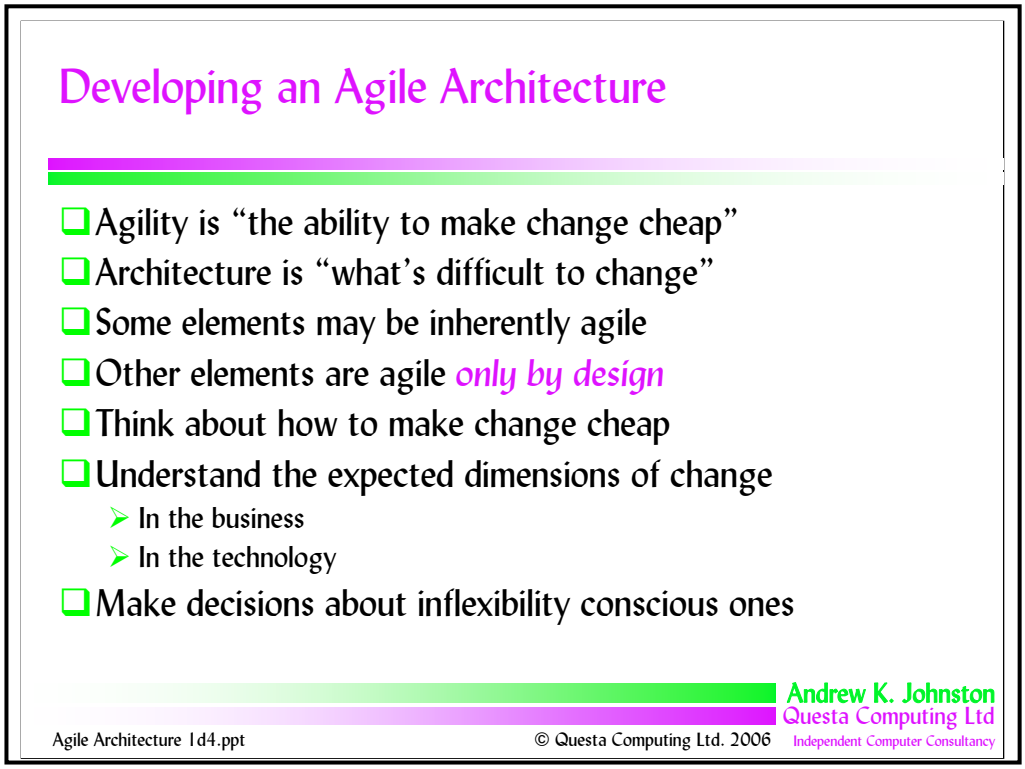
Enterprise architects do have a specific position, spanning the boundaries between different groups, particularly between different project teams. This makes communication an even more essential part of the role. The whole point of enterprise architecture is to have systems that join up better. So you need to focus on models which share information across projects. The EAI bus map is an example of this.

Modelling and documentation should not be the agile enterprise architect's primary focus. That should be supporting the architecture within project teams, coaching developers in the architecture and architecture skills. You also need to verify that the proposed enterprise architecture works in practice, and change it if it doesn't. The best way to achieve both of these is to get involved with project teams, to work with them to understand, test and evolve the enterprise architecture.

The figure shows how the enterprise architecture evolves in an agile way. The enterprise architecture team will probably create the initial architectural vision and models. This shouldn't be a long process – maybe a few weeks work. Any longer and you run the risk of developing models that don't reflect something that will work in practice. Remember, your models need to be just good enough, not perfect. An agile enterprise architect may also ensure the technical ideas work by developing small prototypes.

The models are then developed by working with the project teams and inviting feedback. They start small and are fleshed out incrementally over time based on the feedback you receive from the business and IT communities.

For the agile enterprise architect, it's even more important to work by influence rather than authority. You may be working in some sort of federated environment, where a number of semi-autonomous groups don't come under common management. Or you may find that teams don't really want to fall in with what they see as "external" guidance. Only by delivering real value and gaining their trust will you have any chance of success.



So far, we've talked mainly about the agility of the architect. For the last few slides I'm going to talk a bit more about the agility of the architecture itself.

Agility can be defined as "the ability to make change cheap". Agile developers assume that change is cheap, that we don't have to think about tomorrow. The agile architect knows that this is not always true.

Yes, some aspects of change might be very cheap – tweaks to the user interface of a system, for example. But other things are fundamentally more difficult to change – an underlying data model shared by several programs is a good example, and I'm sure you can think of many others.

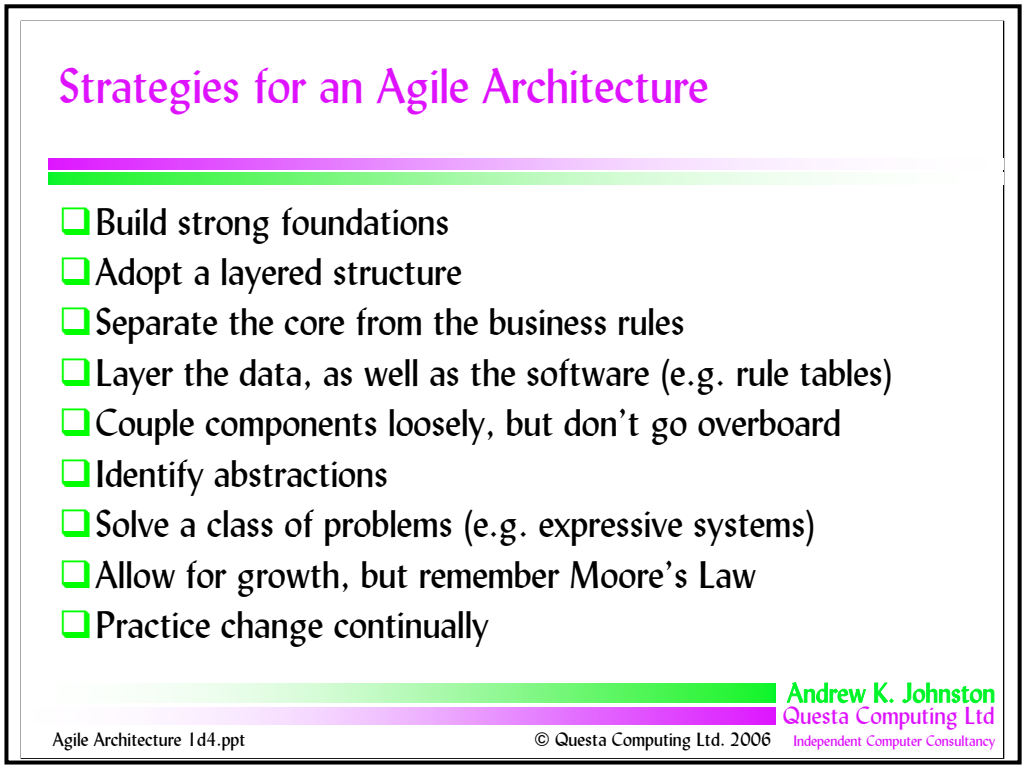
I recently had the pleasure of hearing Martin Fowler talk on architectural issues, and he had a great definition of architecture – "Architecture is the stuff which is difficult to change". I'd just add "unless you design it that way".

The agile architect has to think about how to "make change cheap". But it's not at all agile to overbuild a system accounting for every possible change. That way lies the madness of analysis paralysis and massive over-complication.

So just like an agile programmer designs for "the requirements I know now", the agile architect has to design for "the changes we can reasonably expect". Some might be obvious – for example if the business is growing then you clearly need to allow for scalability. Others might be less obvious.

If a particular change is repeated, or contradicts another, or contradicts your common sense, then try to find a way in which you can make and undo that sort of change more easily, without having to change artefacts like documents, source code or data structures.

If you're good, your architecture will easily absorb predicted changes. But you won't predict everything. So you also need to understand which aspects of an architecture introduce inflexibility. Ask the question: "are we happy to make this decision in the knowledge that it will be difficult to change?" Make such decisions conscious, not accidental.



We're actually quite good at making systems agile. We know how to make almost any sort of change cheap if we anticipate it.

This slide is only a very quick introduction to some of the ideas. There's more on my web site, and in the general literature. A good source of ideas, completely outside IT, is "How Buildings Learn" by Stewart Brand – it's required reading for any agile architect.

One of his most interesting observations is that flexible buildings tend to be built on strong foundations, and the same is true for IT. You'll never be agile if you spend all your time fixing problems with the core software and infrastructure.

An agile architecture always has some sort of layered structure, separating out things which have specific purposes, or which change at different rates to the others. In particular business rules will tend to be in separate layers from the core capabilities, especially if you're building software which may be used by more than one business. The principle of layering has almost nothing to do with your choice of technology – it's possible and good practice with C, or with the latest SOA technologies.

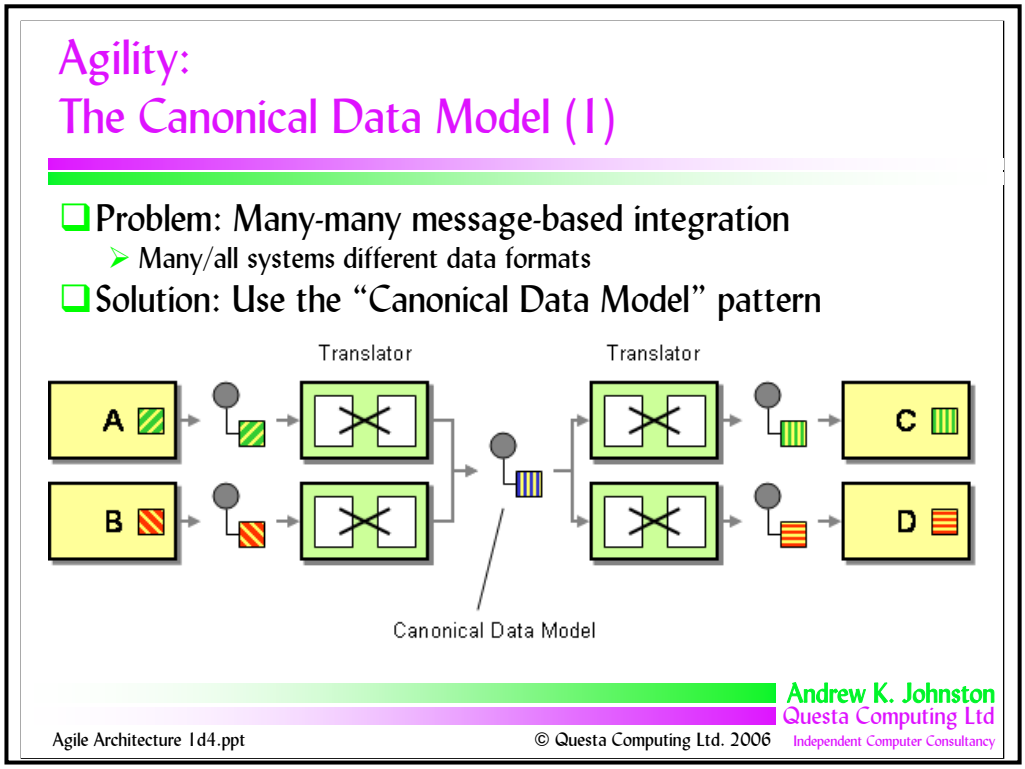
Look for opportunities to layer the data as well as the software. One common example is a rule table, which controls the use of data, separate from the "content" data it controls.

Couple the components in your design loosely, but don't overdo it. I've heard of architectures in which every component interaction uses service technology, or EAI, and these always sound like the architect is pursuing an ideal whether it has real benefits or not. Religious fervour isn't very agile, and neither are the architectures it produces.

Constantly look out for abstractions which allow you to solve a group of problems with the same solution. At the software structure level, you need to specifically watch out for cut and paste duplication – a sure sign that there's a set of problems with a lower-maintenance solution. But think about the users' problem space as well. They may be asking you to deliver specific business functions, when they might be better off with a tool which solves a whole class of similar problems.

Make sure your system has some room for physical growth. It's not agile to have to get new hardware if usage increases by a few percent. But don't overdo this one. Remember Moore's Law. Buying new hardware later will usually be cheaper than buying it now.

And remember that being good at anything requires practice. This is equally true of change. Use techniques which support constant change, such as continuous integration and refactoring, and get good at them.



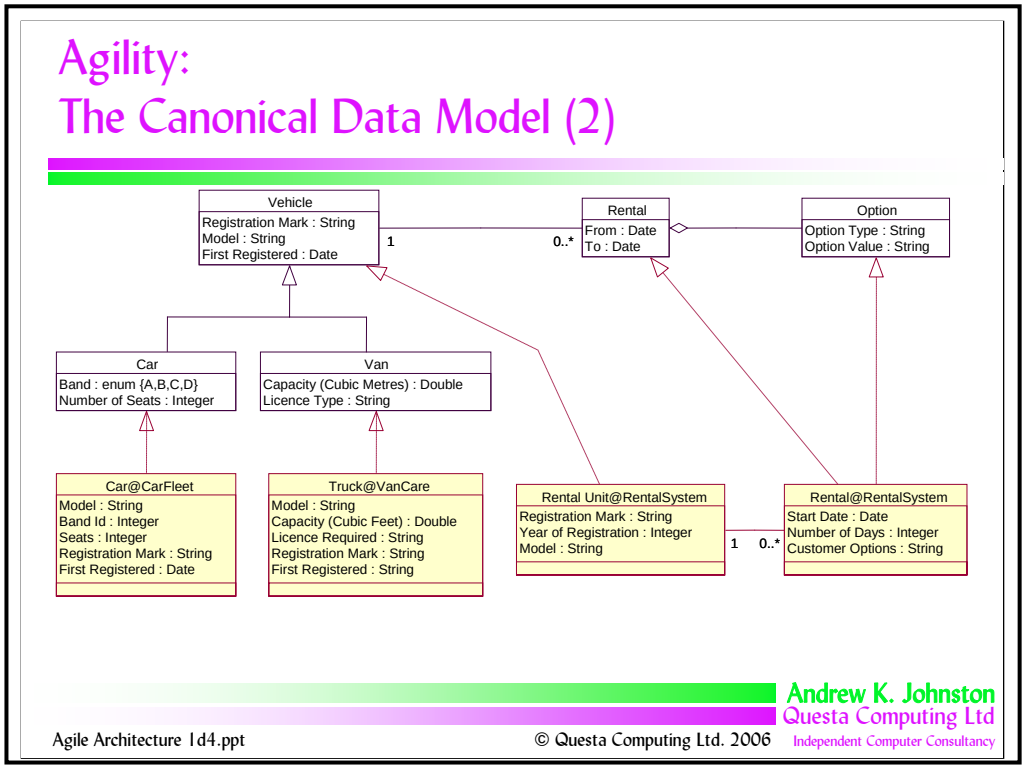
Here's an excellent example of how to build considerable agility into a group of systems for relatively little extra effort. It's also an example of applying a pattern, and of the power of a domain-specific modelling language.

The problem occurs when you have multiple systems which need to be integrated. We've all seen the pictures which show how an integration hub reduces the problem, by replacing N-to-N connection spaghetti with a hub which communicates with each system. The trouble is that sometimes the hub is only a physical one, and each pair of systems still has its own message format. So you've still got the problem at the software level.

The solution is simple enough – you apply the hub pattern at the logical, data format level as well. Each integrated system uses data in its native format – whatever comes out of the messaging adapter – but then each format is translated into a common or *canonical* data structure, and that's what flows on the message bus or through the message hub.

Without this, if one of the systems changes, you've potentially got to change several translators. With this, you only have to change the translator closest to that system. So you enable the real power of the EAI hub model.

The picture is taken from Gregor Hohpe and Bobby Woolfe's excellent book on "Enterprise Integration Patterns". This is an excellent reference for anyone trying to understand or use message- or service-based integration. In it they invented a true "pattern language" – creating icons for simple patterns, and then graphically representing the more complicated patterns using combinations of the simple icons. It's a great design language, and shows the power of using the right modelling tools in the right place.



How do you design the canonical data model? There's no magic wand, but here's a modelling technique which may be useful. I invented this a couple of years ago with my colleague Richard Wiggins.

A real enterprise has a very complicated data architecture. Most data will be held in large legacy or package systems, about whose data structure you know very little. Some key data may reside in external systems maintained by service providers or business partners.

<Click>

Here's just a tiny part of such a scheme, showing a couple of data entities as they are realised in three separate systems for a car rental company. You can see how they have different structures and formats, but there are some obvious common concepts (which is encouraging, because they are describing parts of the same business).

<Click>

What you can then do, just using standard UML, is to model the "core concepts" in a "high level data model". Basically you need to capture each underlying concept in a flexible form.

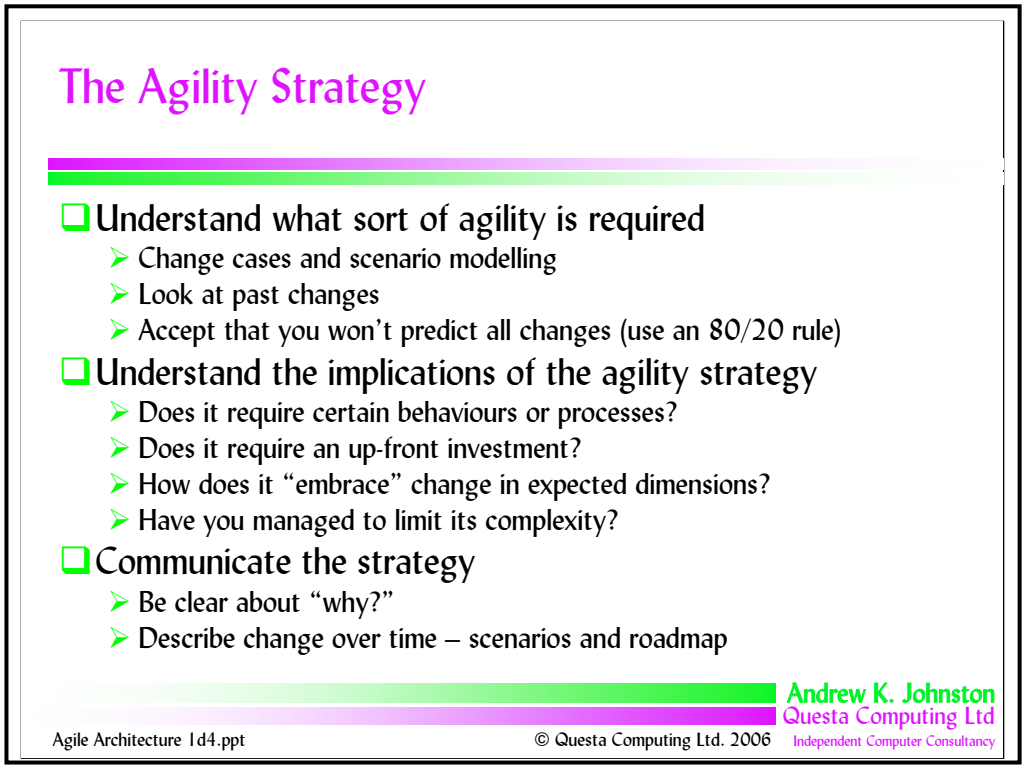
<Click>

Finally, you can indicate the way in which the physical systems realise the High Level Data Model, again using standard UML notation.

Your canonical message format will have a bit about message numbers, dates and so on, but will probably just be an XML representation of a branch of the High Level Data Model.

Now, just in case you think I've forgotten the title of my talk, you do need to guard against this becoming a "big analysis up front" exercise. You can build the model one "leg" at a time, and expand it incrementally, following refactoring principles. You can develop the model top-down, or bottom-up. Personally I'd start by trying to capture the essence of one major entity, and work out from there.

If you want more details, download the paper from my web site.



Agility doesn't happen by accident. You have to design your systems and development processes to deliver it. And you need to understand what sort of agility you are trying to deliver.

Do you want to scale quickly because the business is growing? You need to design an architecture which can "scale out" at all levels. Do you want to deliver new functions quickly? You may need support for a scripting language. Do you want the users to modify details of business rules themselves? If so, you'll need a rule table or similar.

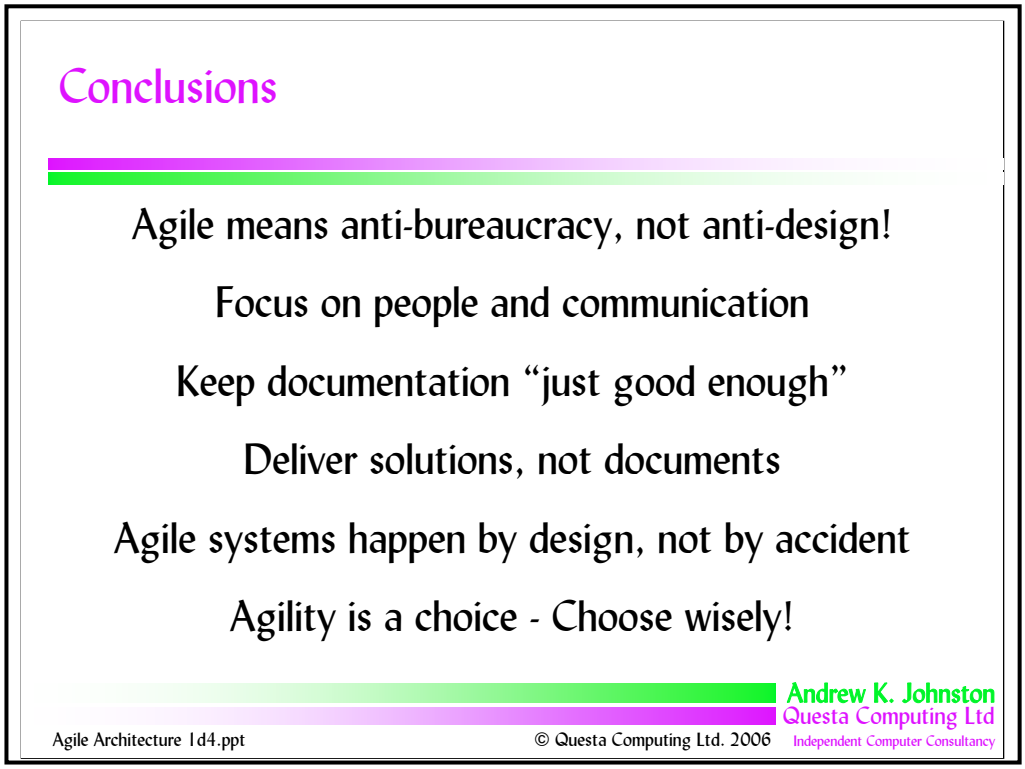
The agile architect will establish a strategy that identifies which aspects of the system or systems are flexible, and which are not. In an ideal world this will derive from a business strategy, but even without one you can still be explicit about the architectural dimensions of agility.

Techniques like change cases and scenario modelling can help understand the required dimensions of agility. But they're not rocket science – fundamentally they all boil down to "think what might have to change, and how we'll react if and when it does". Look at past behaviour – "ping pong"-ing requirements are an obvious indicator of where flexibility is required. But accept that you won't predict every change, and there's a definite 80/20 rule.

You need to understand how the strategy will work, and how you can make sure that the architecture does not grow too complex, with a large up-front or ongoing cost.

You need to communicate the agility strategy and its implications to all your stakeholders. It may require certain behaviours. It may need an up-front investment, like a canonical message model. So you need to make sure everyone understands and buys into it. You need to document the strategy, but this can be an agile document - it might be a formal paper, it might be a slide deck, it might be a set of sketches on a whiteboard.

Be clear about why the architecture is agile in some ways, and not in others. And communicate a picture of change over time, and possible alternative outcomes. The best tools for this are your scenarios for future development, ideally in both business and system terms, and a "roadmap" showing how today's systems will evolve over time to those of the future.



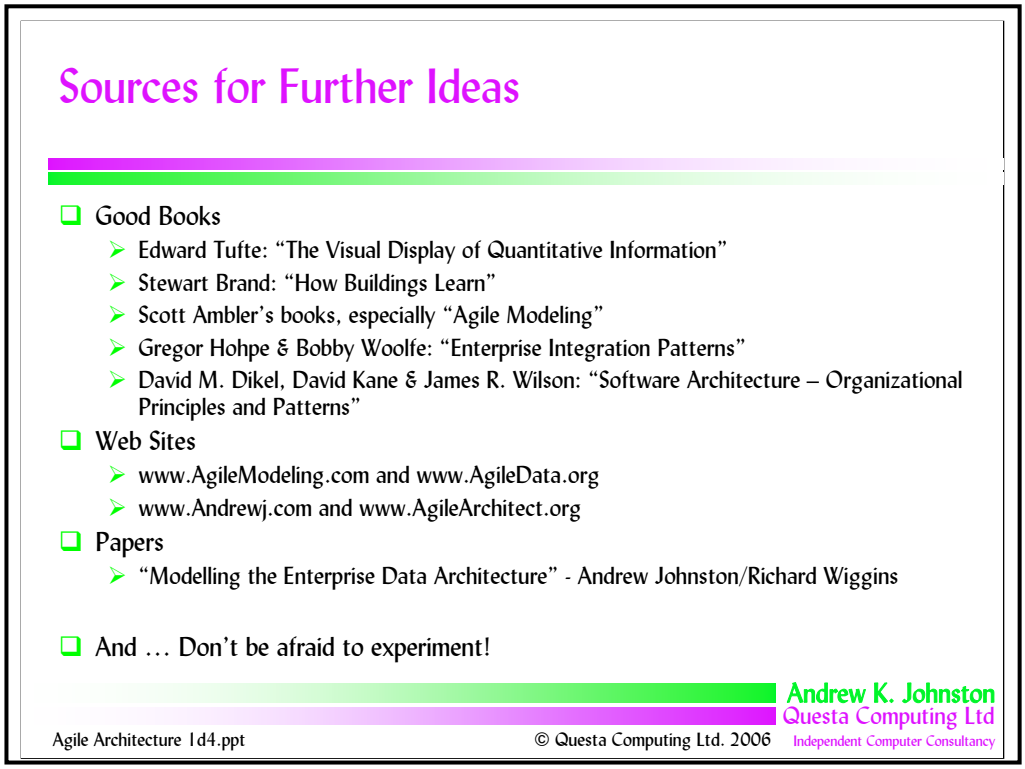
Hopefully I've given you some idea of what an agile architect is, and does, and how an architecture can itself be agile.

Contrary to the ideas of some agile developers, I'm a strong believer in the importance of architecture and design. However, contrary to some formal approaches I think that we have to focus on people and their understanding, not on shelf-feet of documentation.

Documentation and models are important, but they need to be just good enough to serve their purpose, which is to help deliver working solutions, not bits of paper.

The agile mantra is that "change is cheap". But if that's true, it's by design, not by accident. As an agile architect, you need to try and "make change cheap", but in specific ways designed to meet the needs of the stakeholders.

Understand what agility they need, understand what helps you do the best job, and choose wisely!



<The following does not form part of the presentation>

Initially an engineer and then a developer, since the late 1980s I have focused on developing, specifying and validating the structure for IT solutions. Under a variety of titles I have reconciled solutions with a wide range of constraints, dealing with quality, process and business as well as technical issues.

Now an independent consultant, I specialise in strategies and technical architectures allowing businesses to get maximum benefit from IT resources. Deploying unique managerial, commercial and technical skills, I help deliver high quality, cost-effective solutions whether developed, procured or using existing assets, either by developing architectures or as an architecture trouble-shooter. My track record includes several instances where my innovative designs have delivered significant business benefit or cost savings.

Over 20 years I have built experience of systems ranging from enterprise asset management to field force automation, shipping to retail back-office. More recently I have focused on systems integration, data and knowledge management.

I run the AgileArchitect.org website, and am also the author of "A Hacker's Guide to Project Management" and a number of published papers on architecture issues.

For more details, please visit www.andrewj.com

To contact me by email, please use eac@andrewj.com

Thanks, and I hope you enjoyed the presentation

Andrew