

**Building Enterprise Architecture**  
The Endava Approach



## ▶▶▶ HIGH LIGHTS



“The architect of a system, like the architect of a building, is the sponsor’s agent. It is his job to bring professional and technical knowledge to bear in the unalloyed interest of the buyer, as opposed to the interests of the salesman, the fabricator, etc.” - WWISA



Architecture comprises the critical technical decisions that will shape the IT and business solution; architecture is therefore central to the sponsoring business realising the benefits that justified the original IT investment



Architecture provides a direction and shared metaphor to inform detailed design decisions, but as architecture is an emergent construction it cannot be articulated fully in advance of actually constructing the solution. We advocate developing the architecture alongside the working code, refining and reflecting the solution to complex implementation problems in the architecture by solving the problems in the actual design



“Conceptual integrity is the most important consideration in system design. ... Simplicity and straightforwardness proceed from conceptual integrity. Every part must reflect the same philosophies and the same balancing of desiderata. ... If a system is to have conceptual integrity, someone must control the concepts.” – Fred Brooks



Keep iterations small with clear & achievable goals addressing key architectural risks early: “The primary reason [project success rates have improved] is that projects have gotten a lot smaller. Doing projects with iterative processing as opposed to the waterfall method, which called for all project requirements to be defined up front, is a major step forward.”  
– Jim Johnson (Standish Group)

# CONTENTS

## **INTRODUCING THE SERIES OF ENDAVA WHITEPAPERS 5**

### **INTRODUCTION 6**

- WHAT IS ARCHITECTURE? 6
- THE ENDAVA APPROACH 7

## **COMMON PROBLEMS IN SYSTEMS ARCHITECTURE 7**

## **MOVING FROM WATERFALL TO AN ITERATIVE APPROACH 11**

- LIMITATIONS OF THE WATERFALL APPROACH 11
- THE ITERATIVE APPROACH 12

## **WHAT IS ARCHITECTURE & THE ROLE OF AN ARCHITECT? 13**

- ARCHITECTURE 13
- SUCCESSFUL ARCHITECTURE 14
- THE ARCHITECT ROLE (WITHIN AN ITERATIVE PROJECT) 15
- THE SUCCESSFUL ARCHITECT 16
- SPLITTING THE ARCHITECT ROLE 16

## **COMMUNICATING THE ARCHITECTURE 18**

- USE-CASE VIEW 19
- LOGICAL VIEW 19
- IMPLEMENTATION VIEW 19
- PROCESS VIEW 19
- DEPLOYMENT VIEW 20
- COMBINING THE VIEWS 20

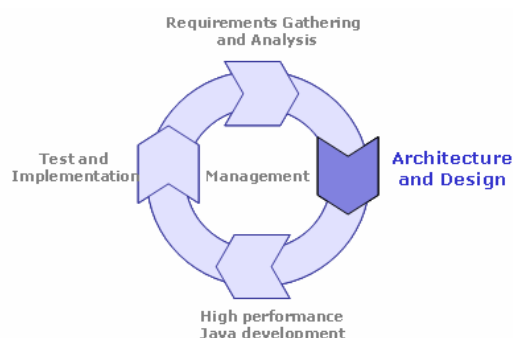
## **ESTIMATION 20**

## **INTRODUCING BEST PRACTICE 21**

## **APPENDIX: ARCHITECTURE CHECKLIST 22**

## **ABOUT ENDAVA 27**

Introducing the series  
of Endava  
Whitepapers



With targets to reduce operational costs, reduce errors in processing, increase settlement throughput and meet the demands for greater transparency from the regulators, the pressure to streamline processes and integrate technology in the back office is greater than ever. However, these programmes of change impact numerous areas within the organisation and therefore can add a significant amount of risk and complexity. To ensure these programmes deliver real business benefit best practice must be applied to -

- Modelling current business and system processes and defining target solution requirements. Consolidating business processes and increasing STP can enhance productivity and drive down costs
- Creating designs and architectures that cater for the new services and requirements, managing legacy interfaces, data migration, and legacy retirement, that allow business volumes and requirements to grow and change
- Managing a modern development project and catering for both the technology and the iterative practices that enable programmers to be productive, engage with the business needs, and demonstrate that architectural goals are met
- Passing key quality gates and performing comprehensive testing that traces from requirements and the architectural constraints and ensures the solution is validated at a unit, integration, system and user level to meet best-practice standards and reduce commercial risk delivering a low defect level in the production environment

This paper is the second in a series of four Endava white papers, drawing on the experience gained on some of Europe's largest programmes of this type. They will consider the end to end software development process and impart useful knowledge and points of view about best practice across the IT programme lifecycle, from requirements management, architecture and design, through to development and testing.

While each paper focuses on a specific topic, it also considers the relationships and communication flows across the key activities. Good relationships and communication throughout the development lifecycle bridge the gaps that can emerge between the people and artifacts involved at each stage. In covering the full development lifecycle, we discuss the benefits of adopting an iterative approach to system

development, and demonstrate how this can work for complex technology integration projects.

## Introduction

Designing and implementing an effective solution architecture to support the delivery of a business application is perhaps the hardest challenge facing organisations, and specifically Programme Directors, charged with delivering major IT change programmes. The solution architecture must be performant, scalable and capable of meeting business requirements that are often largely unknown at the start of the programme. It must do this without the need for fundamental overhaul or incurring significant additional cost.

Once defined, the architecture must be communicated and understood by the entire delivery team and the principles must be consistently adopted across every aspect of the programme if major delivery problems are to be avoided. The business sponsor may also need or want to have a perspective on the “what and why” questions surrounding the solution provided by the architecture.

This paper reflects and reports on the Endava experience of architecture in iterative projects, gleaned from practical work in the field, of designing, implementing and evolving enterprise solution architectures. It also provides some pointers and help as to how to avoid the many dangers that are waiting out there for the unwary. We also introduce representations of the architecture that we believe are crucial for defining and communicating priorities and ensuring a common understanding across all stakeholders on the programme.

This paper will also provide an insight into the Endava approach to architecture, and comments on our experience and learning in this field. We also discuss opportunities to improve communication and understanding between team members within a project and, in a wider context, between the business customer and the project team. It is our view that you can dramatically increase the chance of successfully completing a project on time, on budget and to meet customer expectation using these techniques. At Endava, we have practical experience of seeing this deliver results time and time again on large programmes of software development, system renewal and technology implementation.

### What is Architecture?

The solution architecture provides an overall framework on which the major components, interfaces, and technical approaches, that will constitute the significant technical decisions in that system, will be based, and the IT and business solution built. The solution architecture is therefore central to the sponsoring business realising the benefit that justified the original IT investment.

There is however no accepted single agreed definition of software architecture.

According to the IEEE, "An architecture is the highest-level concept of a system in its environment." This leaves us with the problem of what "highest-level" means, for example the system's users will have a different highest-level concept of the system than the developers.

Other commentators prefer to define architecture as the shared understanding of a system, and yet others as the decisions that must be made early in a project.

While it is typically necessary to tackle major architectural decisions early in the development life-cycle the systems architecture will continue to evolve throughout the life of the programme. During the delivery project the architecture remains an evolving entity defined in architectural models through various views and viewpoints, in the systems documentation and the working software.

### The Endava Approach

At Endava we use an approach to architecture description that calls for the creation of discrete views of different aspects of the architecture which address the needs of the key audiences, all built around a central core described by use cases as described in our first whitepaper concerning requirements management. By having separate views of the different aspects of the architecture, the developers, for example, can more easily see how the design has been derived from the requirements to support the business goal.

This paper discusses the best ways of creating and integrating these views into the development cycle. It also looks at team structures and processes that help define and improve effectiveness of the role of the architect as well as improve productivity and communication flows between the business and IT (analysts, and developers and testers) stakeholders.

### Common Problems in Systems Architecture

A key reason our clients turn to us for help on projects is our ability to mitigate the problems associated with poorly formed architectures. If not addressed, these problems will at the very least lead to the project running late or over budget, and in more serious cases to the complete failure of the delivered system to satisfy the needs of the customer.

Endava has found the most common architectural problems faced by organisations setting out to build major new business applications to be:

1. **Water Torture:** Blind adherence to waterfall techniques
2. **Why? What? Who? How?:** Debilitating ambiguity and/or confusion surrounding what the systems architecture is and/or what is expected from the Systems Architect

3. **Blissful Ignorance:** The complete absence of a cohesive systems architecture that can be used to support system design and delivery activity
4. **Groundhog Day:** Failure to define architectural principles or complete architectural tasks that deliver artifacts in time to allow the system design and delivery activity to start/finish
5. **Architecture versus Design versus Development:** Ambiguous interfaces and hand-offs between the key delivery disciplines and architectural "usability" – in this case by the delivery team.
6. **Non Functional Requirements:** Establishing and satisfying the NFR – early enough in the life-cycle.
7. **Communication and Engagement**
8. **Change Management**

This is a long list but anyone who has spent time on software development projects will be familiar with the range of problems that commonly occur. Often these can be directly linked to the use of a broken approach to structuring projects, the lack of role definition, poor communication and definition of interfaces between project roles, and of course poor change management. Here is a brief synopsis of the problems identified above:

**1. Water Torture:** The ingrained use of the 'waterfall' approach to software delivery poses one of the greatest difficulties for those trying to deliver projects.

Under the waterfall approach, a project proceeds according to rigidly defined phases. Each software development activity is performed in sequence and follows the general flow from requirements to analysis, design, code and test. A preceding phase is completed before the next starts; and phase completion is judged on whether it has matched the previously defined requirements. Interfaces between phases are fractious and typically rely heavily on documentation. Some of the major problems identified over the years with this approach include:

- Inflexibility to feedback and change. Independently executed phases mean that delivery teams can be shut down after milestones are reached, with little overlap between phases, if any at all. Consequences are therefore postponed often until the final testing phase, and then require complex areas of the system – right back to base architectures to be re-worked. This is expensive and time consuming.
- Lack of communication or lack of clarity in communication. It is very difficult to complete all tasks or clearly write everything learned in a phase in a document before moving on to the next phase
- Poor traceability back and forward through the different project stages;

- Change is regarded as the enemy, and reject by the process rather than regarded as a natural part of delivering great software.

**2. Why? What? Who? How?:** Few organisations are familiar with (i.e. have adequate capability for) what is involved in the development and deployment of major new business systems and the major business upheaval that this almost inevitably brings. Consequently great care is required when designing the programme organisation to ensure that ALL STAKEHOLDERS and the project team(s) – including the Architect - completely understand the role of the architecture and the architect.

This must be defined in terms of the:

- Deliverables
- Operating Model / methods / processes:
- Interfaces with the business community, programme management, business analysts, designers, developers and testers
- Key Performance Indicators (KPI) i.e. how will the performance of the architect be measured

**3. Blissful Ignorance:** Organisations still attempt to build large high performing systems without any explicit attempt at first defining the architecture.

Consequently each delivery team can end up deciding for themselves the software environment and tools most suited to their needs (they think) and the development tools, frameworks and patterns they should use.

As they say ... “you get the architecture you deserve!” and things will go wrong – if you are lucky you will discover this problem sooner rather than later.

**4. Groundhog Day:** This is essentially the opposite of not realising a systems architecture is required at all and manifests itself as an over-emphasis on the architectural dimension of the system to the extent that nothing moves forward until the architecture is (deemed to be) complete.

The trouble is it never is complete and so things do not move forward, analysis paralysis has struck!

Due to ever emerging and changing business requirements, and an endeavour to resolve all the architectural questions before starting development, things keep looping around, never reaching a point at which it is deemed safe to start. Ultimately things simply time-out amid much acrimony.

**5. Architecture versus Design versus Development:** Very similar or even a sub-set of “Why? What? Who? How?” but worthy of a mention as we see it so often.

This can be articulated as:

**System Architect:** the architect has exclusive responsibility for major architectural decisions, and enterprise level technical co-ordination, and does this largely in isolation from the rest of the team

**System Designer:** the designer isn't getting much access to or direction from the Architect .... but must get started so makes some assumptions!

**System Developer/Tester:** running hard at a delivery date and unable to wait any longer for the Architect or Designer to tell them about the architecture, design, frameworks, patterns etc they just get on with the job best as they can!

The key message here is that there is inadequate definition of the interfaces defined between these roles. Conclusion: There will be collateral damage!

**6. Non Functional Requirements:** This is a problem area because it is often difficult to define the NFR of the business application during the early stages in anything but the most general terms. It is then even more difficult to demonstrate that the proposed architecture will support the specified business needs prior to deployment in an operational environment. Consequently this is often left very late in the life-cycle yet almost invariably causes problems and anxiety.

There is nothing to be gained from delaying consequences in this way... and there is much to be gained from identifying a problem while there is still time to address it.

Get started early.

**7. Communication and Engagement:** As discussed in the first paper in this series, Endava firmly believes it is key for an analyst to be able to communicate effectively to the architect and developers as well as understand the business needs and goals. If the requirements aren't made clear to the architect, the resulting system will not meet specified objectives. The customer might get system elements they don't need, or are detrimental to the desired goal. Alternatively, they might get some of what they want, but the system doesn't work well or proves impossible to implement fully.

The communication needs to flow in both directions. Often the root of project success or failure rests with the architect. If the architect is seen as residing in an ivory tower, putting out a SAD that is too unwieldy for developers to read, and not communicating or engaging effectively, the project will lose its focus. Developers are likely to interpret the designs in different ways because they do not understand why the architect has made a particular decision. Or they might even ignore the architect and interpret the requirements in their own way to fit their own agenda – in the absence of clear direction.

Good communication and the adoption of iterative methods and supporting tool-sets can help materially here.

**8. Change Management:** Communication failures can occur under an iterative approach as well as a waterfall approach. And while the 'Chinese whispers' syndrome is exacerbated by the lack of team continuity, which is often the case in major programmes or those structured in a waterfall manner, this is not the biggest problem with the waterfall approach.

The biggest problem is that sound change management becomes all but impossible. If a requirement changes at a later stage, such as coding or testing, the project teams must go right back to the beginning because the implications for other parts of the project, particularly its underlying architecture, can be massive. Such bad change procedures are a nightmare for financial and commercial management, and are a major reason why projects often go over budget. It is also why teams of lawyers are often involved, drafting contracts that anticipate downstream conflict over delivery and payment.

Varying requirements are not the only sources of change in a project. When programmers try to build a design, they may find that it doesn't work and needs to change (big programmes are invariably pushing boundaries and things do not always work as expected). When testers check a system is finished, they may find that although it fulfils the requirements, it is unusable in practice. All software development projects experience change, for the simple reason that people are not able to comprehend all the details of a new system until it is actually released and can be tested and seen operating in practice.

Communication and change are linked. As a project changes, those changes have to be communicated to the people that need to know about them. Without good communication between everyone on a project, there is a high chance that someone will make a decision without knowing all the material facts, leading to that decision having to be changed later.

### **Moving from Waterfall to an Iterative Approach**

#### **Limitations of the Waterfall Approach**

Endava advocates adoption of iterative development approach to all significant projects. Here is some rationale to support this position.

The waterfall process was based on an empirical observation of 30 years ago that the cost of change within IT projects rises exponentially with progression through development activities. For example, a change that occurs during design work is much more expensive than one occurring in the previous analysis phase. To address this, projects were structured such that decisions were made as early as possible in the process, as it would be more expensive to make them later. Today, this rationale does not make sense. In the last 30 years, progress in programming languages, tools and techniques mean that change in later phases is not as expensive as it once was. Large projects now also tend to be more

complex than those of 30 years ago, making decisions even harder to get right early in the project, and making change inevitable.

Iterative development approaches have emerged and matured over the past decade and address most of the weaknesses inherent in the waterfall approach. The Endava experience in delivering large, complex projects for the financial services industry has consistently proven their effectiveness.

And this is not just our view. Industry analysts widely credit iterative methods with increasing the percentage of development projects that succeed. In 2004, the 10th edition of the annual CHAOS report from The Standish Group, which researches the reasons for IT project failure in the United States, indicated that project success rates had increased to 34 percent of all projects. That's more than a 100-percent improvement from the success rate found in the first study in 1994.

Asked for the chief reasons project success rates have improved, Standish Chairman Jim Johnson says:

**"The primary reason [project success rates have improved] is that projects have gotten a lot smaller. Doing projects with iterative processing as opposed to the waterfall method, which called for all project requirements to be defined up front, is a major step forward."**

### **The Iterative Approach**

Like phases in the waterfall model, each project iteration in an iterative development approach is designed to reduce the project risk. But unlike waterfall phases, iterations can consist of any development activity, with particular value given to producing working software. Typically, iterations reduce risk by focussing on the critical requirements or on the most complex parts of the system early in the development process. Even the first iteration in a project should produce software, even if it may not do much that is useful to the business yet.

This approach allows software to be built all the way through the project and avoids deferring nasty surprises until the end. It recognises that change is the norm, not the exception, so it allows the different development activities to be performed throughout the project.

Of the iterative processes, the Rational Unified Process (RUP) has been successful on the largest of projects and is widely used in the industry. Endava advocates and uses many of the recommended processes in RUP, but uses its experience of RUP and the Agile processes such as DSDM, Scrum and XP to select the most appropriate practices for the project in hand and tailor where necessary.

## What is Architecture & the role of an Architect?

Architecture affects everyone involved in a software development project, yet team members or stakeholders often do not understand enough about what it is or what the architect does.

In this whitepaper we set out to clarify:

- What architecture is
- The role and responsibilities of the Architect within an iterative project

### Architecture

The definitions of architecture mentioned at the start of this paper describe architecture as a subset of the design. The difficulty is working out what elements of the design are part of the architecture. Architecture is the information about the system that many people need to know about. It encapsulates the significant decisions that shape how the developers will build the system, how it will present itself to the users and how it will operate and perform in a live environment. It is the system blueprint which emerges throughout the project life-cycle only to be considered complete as the system enters live operation.

**Architecture comprises the critical technical decisions that will shape the IT and business solution; architecture is therefore central to the sponsoring business realising the benefits that justified the original IT investment.**

Architecture is defined as a subset of the design. While it is difficult to define which subset, the following definitions help<sup>1</sup>:

- “The highest-level concept of a system in its environment.” (IEEE)
- The organization or structure of significant components, and their interactions.
- An abstraction of a system’s implementation.
- The parts or aspects of the design that are considered important.
- The parts or aspects of the design that need to be understood by many people, the shared understanding of the system design<sup>2</sup>.
- The decisions that are (or are thought to be) hard to change.

<sup>1</sup> For more definitions, see: <http://www.sei.cmu.edu/architecture/definitions.html>

<sup>2</sup> See: <http://martinfowler.com/ieeeSoftware/whoNeedsArchitect.pdf> (There are many other relevant articles on this website)

While these definitions all present helpful ways to think about architecture, architecture is emergent (as is design in general) throughout the life of the project.

**Architecture provides a direction and shared metaphor to inform detailed design decisions, but as architecture is an emergent construction it cannot be articulated fully in advance of actually constructing the solution. We advocate developing the architecture alongside the working code refining and reflecting the solution to complex implementation problems in the architecture by solving the problems in the actual design.**

It is important also to realise that architecture may be represented or described in many ways, but that these are not themselves the architecture but a means of communicating what the architecture is. For example, a software architecture document and the finished system both represent the architecture, but neither can be said to be the architecture.

The word 'architecture' is sometimes used to describe the process of determining the architecture. Some development processes describe an architecture phase, during which effort is focused on the architecture. Do not make the mistake of thinking that the completion of the architecture **phase** indicates the completion of work on the architecture.

### **Successful Architecture**

Successful architecture has the following characteristics:

- All parts of the design considered important are recognised and present within the architecture. In other words, the architecture is complete and has no omissions. E.g. if security is important to the system, then a successful architecture will address it.
- The architecture is modular and has conceptual integrity and consistency. Every part reflects the same principles, and the same balancing of priorities. "Good features and ideas that do not integrate with a system's basic concepts are best left out."
- Everyone who needs to understand a part of the architecture does so. e.g. developers coding a part of the system understand all parts of the architecture relevant to their work.
- All representations of the architecture are consistent with each other. e.g. a document describing the architecture is consistent with the code in the finished system.
- The architecture is of good design and fit for purpose. What makes design good is an important and complex area, but is outside the scope of this document.

The architecture and the supporting documentation must address the needs of all the stakeholders and interested parties and provide the context and content that they require to allow them to discharge their responsibilities.

It is ultimately the blueprint that defines the key design components such that the integrity of the overall design is maintained both during and after the development of the system.

### **The Architect Role (within an iterative project)**

In the waterfall process, the architecture is typically completed in an early phase, while in an iterative project the architecture is constantly refined. This makes the task of recognising what parts of the design are architectural, and communicating them, much more challenging.

An architecture is successful if it leads to a successful software system. It must be of good design for it to result in a successful system and must be communicated effectively. The person responsible for these tasks is often called the architect.

On most projects, especially iterative ones, many people contribute to the architecture. However, with many players involved in defining the architecture, there is an increased risk that it will not be consistent. In his book, *The Mythical Man Month*, Fred Brooks calls this consistency conceptual integrity.

**“Conceptual integrity is the most important consideration in system design. ... Simplicity and straightforwardness proceed from conceptual integrity. Every part must reflect the same philosophies and the same balancing of desiderata. ... If a system is to have conceptual integrity, someone must control the concepts.”**

From this standpoint, and from practical experience, Endava advocates that one person has an overall ownership and accountability for all aspects of the architecture. For the same reason, it is highly preferable that this role to be held by the same person throughout a project life-cycle.

The architect must be part of the overall programme management team providing technical direction for the Programme Director, Project Board and the delivery team. The role will include:

- Selecting and qualifying the high-level architectural components including all design aspects relating to the use of frameworks and patterns
- Ensuring that methods, tools, and working practices, as they relate to architecture and design, are defined and adopted across the programme in order to assure design integrity, good communications and high levels of productivity
- Ensuring that the architecture is fit for purpose
- Resolving technical disputes and bottlenecks

- Ensuring that interfaces (in and out) are identified and specified including related responsibilities
- Testing the architecture early and often – particularly as it relates to the NFR
- Winning and maintaining the support and confidence of the programme leadership and business sponsors
- Communicating with all stakeholders in appropriate language and terms to ensure full understanding and compliance

### **The Successful Architect**

This document has defined an architect by what they are responsible for, not by what they should spend their time doing. The architect is likely to be the most technical person the client is able to communicate with. So in discussions with more technical team members, the architect must act as the client's advocate in order that everyone understands the client's concerns and wishes.

Based on hard-won experience Endava firmly holds the view that the on a major development project the architect must take a strong leadership role and display a number of core competencies and characteristics. These include:

- Deep technical knowledge, competence and confidence with an ability to influence and convince others of the merits of particular technical and/or design choices
- Strong leadership and communication skills in terms of both the delivery (technical) team and the business sponsor(s) and other stakeholders
- Consultative style capable of leveraging latent team knowledge and capability and carrying the opinion and support of the delivery team
- Business empathy with the ability to establish and maintain strong business relationships

Endava believes that correctly defining the role of the architect and their relationship and engagement with other teams and business stakeholders will greatly reduce project risk and increase the likelihood of success.

### **Splitting the Architect Role**

Some systems can nonetheless be too large and/or complex for a single person to maintain a detailed knowledge of the entire systems architecture. In these cases several architects or an architecture team may be unavoidable. However, if they are to operate effectively there must be clear demarcation lines. In order to keep things as simple as possible responsibilities should be allocated to each architect for a specific part of the architecture and these should be largely independent of, separate from, or irrelevant

to each other. A change in one should not require a change in the other.

In reality, parts of an architecture are seldom totally orthogonal, so there must be a way to resolve conflict between architects. If an issue affects several architects, there must be only one that has responsibility for the decision.

### **Oranges and Apples**

Architecture addresses both the interface and the implementation of a system. The architect role can be split along this division into an "orange" architect role and an "apple" architect role.

The **orange architect** is responsible for the architecture of all aspects of the system perceivable by the user, everything that would be in the user manual, i.e. the user interface in the broadest sense or what the system looks like on the outside.

The **apple architect** is concerned with the architecture of the system implementation, the technologies, processes and key mechanisms internal to the working of the system.

The orange architect determines what the system does, while the apple architect is interested in how it does it. Clearly the apple architecture is dependant on the orange architecture, but they inform and refine each other. If an issue affects both orange and apple parts of an architecture, the orange architect makes the decision.

### **System Orthogonality**

If a project needs more than two architect roles, consideration should be given to whether the system being built can be split orthogonally. For example, some systems have reference data or reporting parts that are not directly involved with the core operation of the system. In this case consider the merits of having one architect role for each key element, complemented by another across the entire system who can resolve conflicts.

### **Focus on the client**

Software development projects exist for the benefit of the sponsoring business, so everyone on the project team should appreciate how their work is affected by the client's wishes. More than any other member of the team, the successful architect must try to understand the client's interests, intentions and motivations, for these reasons:

- The architecture guides the detailed work that derives the business system. If the architecture is wrong, the finished system will also be wrong. So it is essential that the architecture reflects the wishes of the client;
- The architect will ensure shared metaphors are used to communicate the key business and technical concepts in the

system, and that these concepts are sufficiently detailed, clear, precise, and systematically record using standard architectural techniques such as view-based visual modelling

- The architect will act as a client advocate in order that everyone understands the client's concerns and wishes;
- Conversely, the architect will also be responsible for communicating the importance of the technology choices being made back to the sponsor

**"The architect of a system, like the architect of a building, is the sponsor's agent. It is his job to bring professional and technical knowledge to bear in the unalloyed interest of the buyer, as opposed to the interests of the salesman, the fabricator, etc." - WWISA**

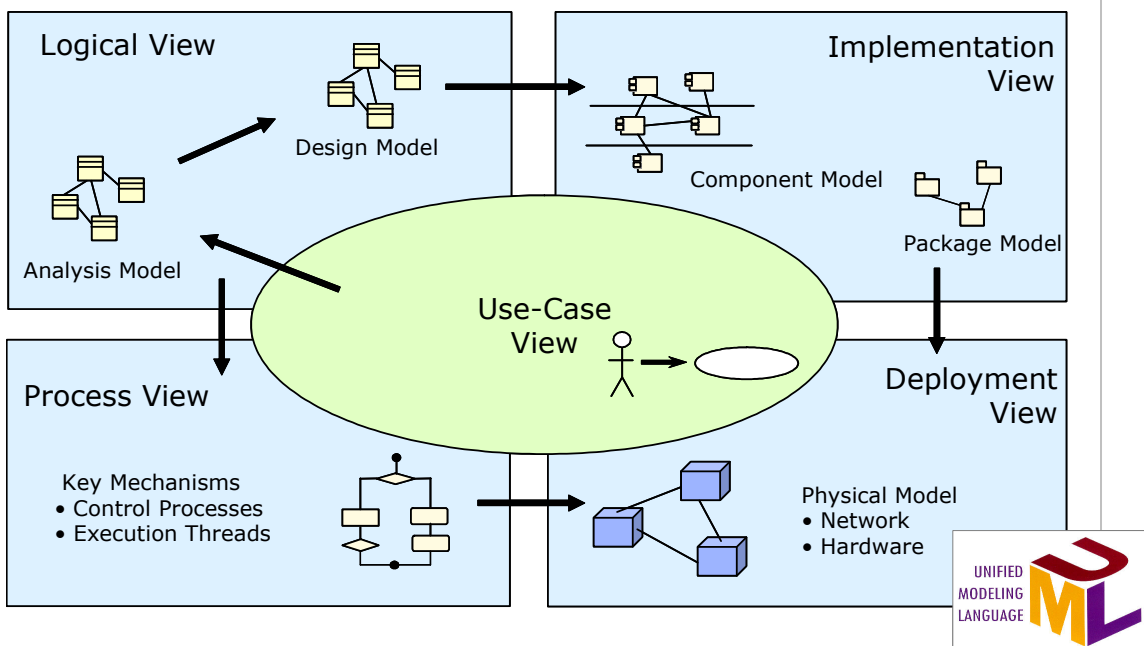
### Communicating the Architecture

Effective communication within a software project relies on communicating the right amount of information. Experience tells us that architecture must be communicated much more widely and effectively than many other aspects of the design.

One of the most common communication gaps associated with documenting software architecture is when authors try to represent more than can reasonably be expressed in a single diagram. Logic breaks down and this often leads to a lack of clarity, which in extreme cases undermines the architect's role. As part of a strategy for improving clarity and communication, Endava uses the 4+1 View model, using architectural views and viewpoints, developed by Philippe Kruchten at Rational as the basic foundation of communication with all the key audiences.

The approach has had much written about it in the software engineering world. At its most basic, it is a way of classifying different perspectives of the overall architecture, with a focus on reflecting as many of the functional and non-functional requirements as possible. The intention is to address the interests of different stakeholder groups separately, so they find it easier to understand the particular aspects of the system they are concerned with. Additionally, by describing the architecture in multidimensional terms, the team has a common vocabulary.

The four different views are organised around and based on use cases. The four views are:



**Use-Case View**

This view addresses the functional requirements of the system in Use-Cases (or User stories), i.e. what the system will do for the users in terms they will understand using activity diagrams to represent key business workflows and process.

**Logical view**

This view addresses the functional requirements of the system in objects and the behaviour of the objects in the system. It is where the business objects are held, which merge closely with the use cases in the centre. In RUP, design and analysis models are created in this view.

**Implementation view**

Sometimes also called the component view, this is where the packages (code organisation) and components are mapped out.

**Process view**

This is where key processes and interactions between components and objects are illustrated. It addresses issues such as concurrency, scalability, system start-up and shutdown.

### **Deployment view**

This view shows how the software elements are mapped to the underlying hardware platform. It addresses deployment and performance.

### **Combining the Views**

The amount of work done on each view will depend on the complexity and type of project. But when combined with a focus on delivering working code and iterative development, this 4+1 breakdown of different elements of the architecture can help the entire team work to deliver high quality software. This is particularly the case when an object-oriented, component-based development approach is embraced.

The 4+1 approach also helps stakeholders to confirm that their concerns have been addressed. For example, if their concern is in one view, they don't need to look at the others. It also makes the design easier to understand, because it becomes, in a sense, multidimensional.

## **Estimation**

Estimation is an area where organisations can benefit from much greater accuracy if they adopt an iterative development approach. With the right tools, organisations can produce relatively refined estimates from the use cases, as discussed in the first whitepaper in this series (Requirements Gathering and Analysis). However the most value from estimation comes when it is applied to the design artifacts, as a greater level of refinement can be achieved.

The Endava approach to estimation is based on experience and metrics accumulated from many large payments and financial services programmes and projects that have been managed, at least in part, to principles outlined in RUP. These metrics have proved incredibly consistent across all these projects.

Endava has developed an estimating tool which we employ on large programmes to provide accurate estimates as early as possible in the process.

The estimation tool imports the system use cases or design model classes. Attributes relating to complexity, type, and requirements volatility, can be ascribed to each class or use case. Using historic metrics, the tool generates a man-day estimate for the implementation discipline. This is then factored up to include all project disciplines and activities, and is triangulated against:

- An estimated resource profile (i.e. an estimate staff numbers of each discipline per month). This will help show if the man-day estimate is realistic in the desired timescales, and provides an input to the project planning and resourcing work.
- Actual man-day numbers on the same project. If a part of the project has already been developed, or a phase has completed, the actual man-days spent can provide an invaluable benchmark to

compare with the estimated man-days for the same piece of work. This may challenge, or increase confidence in, the decisions made in the estimate.

Often an estimate will not be for one monolithic system, but for a project of several different parts that relate to each other but need to be estimated separately. Using the Endava estimation tool, a class or use case can be allocated entirely or partially to a part of the project or phase.

The Endava estimate calculation tool integrates closely with design tools such as Rational Rose and/or XDE, with automated data transfer. All the data resides in the UML modelling tool, and after the calculations, application of complexity and prioritisation, and sanity checks have been performed, enriched data can be written back to the models.

In an iterative project, progress in the current iteration sets expectations for the future ones. Some iterative processes describe calculating the project velocity, which is a measure of time taken to develop business functionality. As the Endava estimation approach integrates with design tools, it is simple to revise the estimate at every iteration, incorporating new knowledge about how fast the project is progressing, as well as impacts from changes to the requirements and design.

## Introducing Best Practice

Communication is a critical factor in the success of every software project. Communicating the architecture gives the team access to the information they need to know, and keeps the architecture up-to-date and relevant. Everyone needs to understand what the architecture is, and how their responsibilities relate to it.

Endava has found that the role of architecture and the architect on projects is often not well understood. One might have assumed that architects create the architecture, and that someone who calls themselves an architect knows what their role on a project is. But this is not necessarily the case. For this reason, Endava ensures all interested parties and stakeholders are actively involved in creating the architecture, and that the role of architecture and the architect are both explicitly set out and commonly understood.

It can be hard to change the way projects work. Engaging with an external partner such as Endava to assist projects, define roles, mentor architects, or evaluate the effectiveness of architecture, can be a valuable move that reduces project risk and ensures projects are delivered on time and on budget ... and that business objectives are met.

## Appendix: Architecture Checklist

Aspects of the architecture are often ignored accidentally. If a team is working hard on one area, they may overlook another. To try and ensure the architecture addresses everything important, it can be useful to refer to a checklist.

It is the role of the architect to ensure the architectural solution addresses each of these critical implementation issues, and resolves any requirements for non-functional attributes, such as resilience, scalability, security and operational management. For each item in this list, the architect must ensure the design addresses it to an adequate level.

### **Operational management**

- How will the operators monitor, shutdown and restart the system or subsystems?
- Can subsystems be restarted without affecting other subsystems?

### **Legacy interfaces**

- Which external systems does the new system need to communicate with? Remember these systems may not be visible to the users, for example they may be used by the support department to monitor applications.
- What data will be passed across each interface? How much data will be passed across? How often?
- How will each interface work? Consider all aspects, including data format and character set, network protocols, security between systems, failure detection and recovery.

### **Reconciliation**

- What aspects of the system require reconciliation? This may include information flowing between systems or subsystems (i.e. check one system actually received what another system sent), and information flowing in and out of a subsystem or component (e.g. check all tasks given to a component to process have been processed).
- How often does reconciliation need to take place? It needs to be often enough to give sufficient time to handle a reconciliation failure, should one arise.
- What information is needed for the reconciliation? If this information is not available already, how will it be gathered and made available?

### **Migration**

- If the system is replacing an existing system, how will the transition to the new system happen? Will all users move to the new system at the same time, or will migration be slow?
- Which data needs to be migrated from the old system to the new system? How will this be done? When will the actual migration happen (e.g. it may need to be done when data is not being changed)?
- If the old and new systems will be running in parallel for a period, to what degree do the two systems need to be kept in step?
- Is there a need to be able to switch back to the old system if a major fault is found in the new system once it is live? How will this be achieved?

### **Resilience**

- How will the system handle failure?
- What are the different failure scenarios, and how will the system respond to each?
- Will information be lost during failure, will data integrity be affected? Is this understood/accepted by the business?
- How will the system function in a failed state, and how will it recover (or be recovered)?
- Does anyone want to know if the system fails? What do they need to know?
- Does the system need functionality to handle non-fatal defects that emerge in production, e.g. the ability to regress the system to an earlier point of processing?

### **Security**

- Can all users of the system perform all functions? If not, what is the conceptual model that defines who can do what? For example, are users categorised in some way?
- What factors are needed to determine if a user can perform an action? These may include user status (e.g. a test user can't create any urgent documents) or relationship to the target of the action (e.g. a user can only edit an employee's data if the user is the employee's manager).
- How will user security privileges be maintained? E.g. will there be screens to do it?
- Is there data that has to be protected against interference from everyone, including technical support staff? How will this be achieved?

### **Scalability**

- In which dimensions does the system need to scale? How will it be achieved?

### **Flexibility**

- Are some areas of the requirements likely to change? Maybe the areas are known to change regularly, e.g. tax rules, or maybe the users have in mind features they may want the system to include in later releases.
- Can the system accommodate changes in these areas more easily than other changes? There may be standard ways of addressing this problem, e.g. frameworks for internationalisation.

### **Off-the-shelf products**

- Have products been considered for all or parts of the system? Products may be available to implement domain specific functionality or framework services (e.g. persistence, security).

### **Reuse**

- If there are systems in place already that perform similar functions to the new system, can the new system reuse some parts of the existing systems?
- What will the interfaces between the systems be? The new system may want to use functionality that was not previously exposed to an external system, which may require a change to the existing system.
- It is important for the team developing the new system to have a good relationship with the team maintaining the existing system. For example, changes to the existing system may impact the design of the new system, so they must be communicated appropriately.

### **Modular design**

- Do key subsystems have clear responsibilities and interfaces?

There are many different reasons for putting effort into architecture. It is beneficial to understand the reasons that team members and stakeholders hold in order to address their expectations about architecture and the architect's work.

For each item in this list, ask yourself: how important is this to the system's stakeholders? does the architecture adequately address it?

- Enabling decomposition and downwards management of plans and teams. As mentioned previously, the design of a system determines the team structure (and visa versa).

- Cost control (and scope control). Architectural decisions typically impact cost more than design decisions, so controlling these decisions more carefully gives more control over costs. Additionally, the architect may be one of the only people on the project able to take a balanced view on scope decisions, as they understand both the business and technical factors involved.
- Communication in all directions. Architecture can be communicated upwards to the project managers, sideways to other development teams, downwards to the programmers, externally to customers, externally to shareholders, internally to other parts of the business.

## 5 Key Steps to ensuring Successful Architectures



**“The architect of a system, like the architect of a building, is the sponsor’s agent. It is his job to bring professional and technical knowledge to bear in the unalloyed interest of the buyer, as opposed to the interests of the salesman, the fabricator, etc.” - WWISA**

**Architecture comprises the critical technical decisions that will shape the IT and business solution; architecture is therefore central to the sponsoring business realising the benefits that justified the original IT investment**

**Architecture provides a direction and shared metaphor to inform detailed design decisions, but as architecture is an emergent construction it cannot be articulated fully in advance of actually constructing the solution. We advocate developing the architecture alongside the working code, refining and reflecting the solution to complex implementation problems in the architecture by solving the problems in the actual design**

**“Conceptual integrity is the most important consideration in system design. ... Simplicity and straightforwardness proceed from conceptual integrity. Every part must reflect the same philosophies and the same balancing of desiderata. ... If a system is to have conceptual integrity, someone must control the concepts.” – Fred Brooks**

**Keep iterations small with clear & achievable goals addressing key architectural risks early: “The primary reason [project success rates have improved] is that projects have gotten a lot smaller. Doing projects with iterative processing as opposed to the waterfall method, which called for all project requirements to be defined up front, is a major step forward.” – Jim Johnson (Standish Group)**

## About Endava

Following the merger between Brains Direct, a leading supplier of Eastern European offshore software development and IT services, and Concise Group, a City-based IT consulting and services organisation, Endava was established in 2006 to create a focused IT services business to provide the highest quality IT consulting, software, sourcing and support solutions using the very best West and East European talent. Endava is aiming to be the *premier provider* of IT talent and advice from Emerging Europe; bringing sustainable transformation to the locations where they work, delivering domain-focused technology consulting, systems implementation and managed service solutions predominantly to the Financial, Technology, Media and Telecom sectors.

Endava provides expert domain knowledge and implementation experience in IT architecture, integration, IT security, and infrastructure managed services, with a highly skilled and cost effective software development capability from Eastern Europe, delivered via our blended sourcing approach. Endava has delivery centres in Chisinau, Moldova; Bucharest and Cluj, Romania; with clients across the UK, Europe and the USA.

