Agile Architecture
how much is enough?

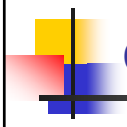
Software Architect 2008

Eoin Woods
Barclays Global Investors

www.barclaysglobal.com/careers
www.eoinwoods.info

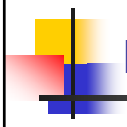
Introductions

- I'm a software architect at Barclays Global Investors
  - responsible for Apex, a new portfolio management system for Active Equity
  - also involved in regional technology architecture, trading systems and global architecture council
- Software architect for ~9 years
  - With enterprise architecture for about 2 years
- Author of "*Software Systems Architecture*" book with Nick Rozanski
- IASA Fellow, BCS and IET member



Content

- Software Architecture and Software Architects
- Agile Principles and Software Architecture
- Practices for Agile Architecture
- Examples of Agile Architecture
- Closing Thoughts

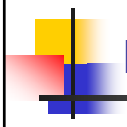


Defining Software Architecture

- A common definition …

> *The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements the externally visible qualities of those elements, and the relationships among them*

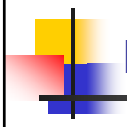
Len Bass, Paul Clements and Rick Kazman (SEI)
Software Architecture in Practice, 2nd Edition



Defining Software Architecture

- A less common definition …

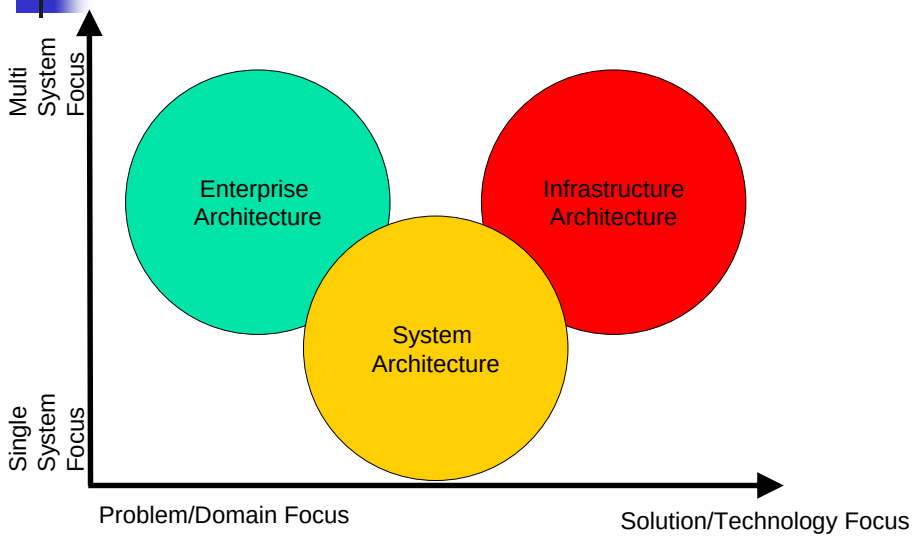
> *The set of design decisions that, if made wrongly, causes your project to be cancelled*
>
> Eoin Woods (at www.sei.cmu.edu/architecture/definitions.html)



Essence of Software Architecture

- Design centric activity
  - designing a system, an infrastructure, a process, … is core to the activity
- Stakeholder focus
  - you serve a wide constituency
  - involves clarifying poorly defined problems
  - results in identification of risks and opportunities
- System-wide concerns
  - e.g. qualities rather than detailed functions
- Balancing of concerns
  - no right answer / least worst option
- Leadership

Types of Software Architect

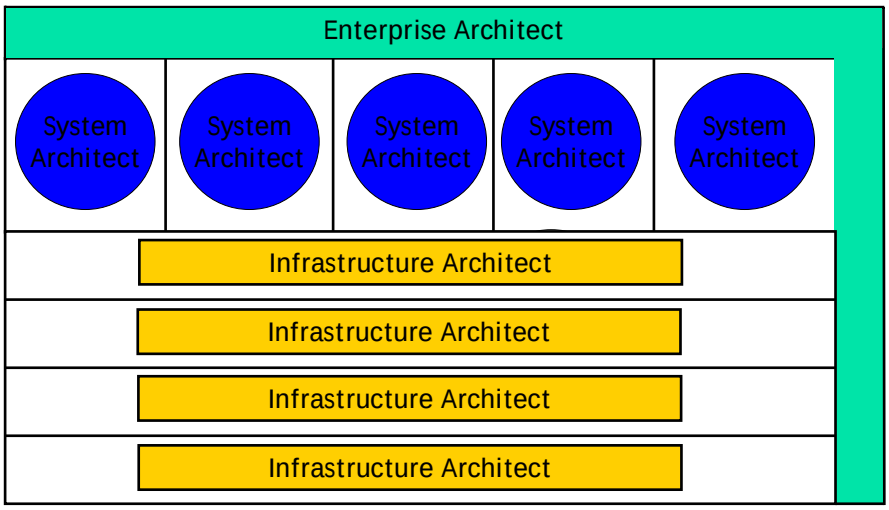


Synonyms

- Enterprise Architecture
  - enterprise architect, functional architect, business architect, strategic architect, domain architect, stream architect, …
- System Architecture
  - software architect, solutions architect, application architect, systems architect, technical architect, …
- Infrastructure Architecture
  - infrastructure architect, technical architect, technology architect, database architect, middleware architect, network architect, storage architect, …

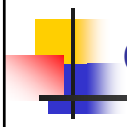
This presentation is about enterprise and systems architects

Architect Relationships



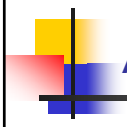
What Architects Aren't

- Managers
  - CIO
  - CTO
  - Development manager
- Technology consultants
  - Oracle technologist
  - Java technical lead
  - BEA product specialist
  - ...



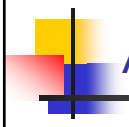
Content

- Software Architecture and Software Architects
- Agile Principles and Software Architecture
- Practices for Agile Architecture
- Examples of Agile Architecture
- Closing Thoughts



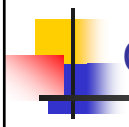
Agile Principles

- The agile manifesto values …
  - **Individuals** and **interactions** over processes & tools
  - **Working software** over comprehensive documentation
  - **Customer collaboration** over contract negotiation
  - **Responding to change** over following a plan
- Key motivation is to facilitate efficient delivery and change



Agile Practices

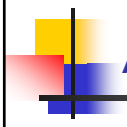
Release frequently	Integrate changes often
Iterative delivery	Collective ownership
Customer is available	Customer prioritises
Simplest thing possible	Collaborative design
Specify via “stories”	Small teams
Test driven	Automate routine tasks
Refactor when needed	Shared workspace



Common Priorities

Architects and agile teams share many priorities:

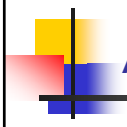
- Focus on the consumers of the systems
- Efficient delivery of valuable software
- Simplification and reduction of cost
- Quality & reliability of delivered software
- Supporting efficient change
- Effectiveness of communication



Architecture / Agile Conflicts

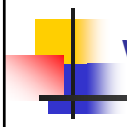
So with shared priorities, why can there be conflict?

- "Big Up Front Design" (BUFD - large documents)
- Document centric vs. delivery centric
- Separation of decision making from delivery responsibility
- Different views on processes and controls
- Differing time horizons for return on investment
- Architectural priorities vs. "customer" (user) priorities
- Agile deliveries in larger change programmes



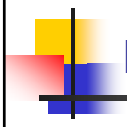
Architecture Effectiveness Index

- Level 0 – no architectural impact
  - architecture is largely ignored, seen as irrelevant
- Level 1 – Stopping things getting worse
  - essential decisions coordinated
- Level 2 – Stable and organised
  - architecture understood, shared, aids change
- Level 3 – Architecture centric
  - architecture is the point of reference for change



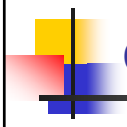
Why Align Architecture with Agile?

- Avoid retreat to the "ivory tower"
  - ensure relevance of what we do
- Focus on stakeholders
  - make sure we deliver value
- Support change
  - stay relevant as the situation evolves
- Work well with development teams
  - shared values and priorities
- Focus on delivery
  - ensures the right priorities



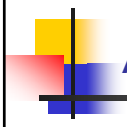
Making Architecture Agile

- Expect change, allow for it
  - work incrementally
- Be demand driven
  - prioritise by customer needs
- Be delivery (not document) focused
- Make information relevant and accessible
- Produce working "software" regularly
- Simple, simple, simple (but not naïve)
  - process, artefacts, solutions
  - simplicity is a precursor for agility



Content

- Software Architecture and Software Architects
- Agile Principles and Software Architecture
- Practices for Agile Architecture
- Examples of Agile Architecture
- Closing Thoughts



Agile Architecture Practices

Allow for Change

- Deliver Incrementally
- Allow reprioritisation
- Identify clear component responsibilities

People not Process

- Work with development teams
- Keep backlog visible

Software not Documents

- Good enough models & docs
- Regularly deliver something that "runs"
- Have solutions for security/DR/HA/…
- Build PoCs for credibility

Collaboration

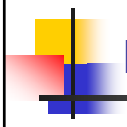
- Identify *minimal* principles and share them
- Share information online
- Focus on x-cutting concerns

Practices – Allow for Change

- Deliver Incrementally
  - Visible progress, early feedback
  - Allow others to be involved in decisions
  - Deliver something useful regularly
- Allow reprioritisation
  - As your customer priorities change
- Identify clear component responsibilities
  - Allow confident extension
  - System architects define system modules
  - Enterprise architects define systems

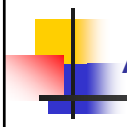
Practices – People not Process

- Work with (and in) development teams
  - don't drop documents and walk away, talk to people
  - develop things jointly
  - developers are a great source of knowledge and experience
- Keep you backlog visible
  - let people know what you're planning
  - get prioritisation and feedback from the team and managers
  - allow others to pick things off the backlog if they have time



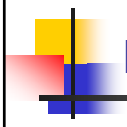
Practices – Software not Docs

- Good enough models & docs
  - but make sure they *are* good enough!
- Regularly deliver something that "runs"
  - if not raw code, something else that works
  - may be useful directly or for research purposes
  - conventional code or a service or a spreadsheet or …
- Have solutions for security/DR/HA/…
  - rarely solved well by the individual teams
  - typically need to work across systems
- Build PoCs for credibility
  - yours *and* the solution's!



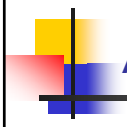
Aside: Good Enough Modelling

- What is good enough?
  - consider currency, precision, detail, completeness
- Experience suggests
  - focus on models at the component/interface/connection level
  - prefer models & databases to pictures
  - be precise, even when abstract
  - ensure models can be updated easily
  - model with a purpose (audience and use)
- Areas to consider
  - system architect: functional structure, deployment, information
  - enterprise architect: system of systems, shared domain models



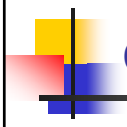
Practices - Collaboration

- Define minimal principles & share them
  - small set easily understood & accepted
  - choose your battles (high value decisions)
- Share information online (e.g. Wikis, SourceForge)
  - allow easy access, comment and update
- Focus on cross-cutting concerns
  - avoid other people's areas of responsibility
  - these cross-cutting areas are often neglected otherwise



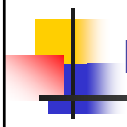
Aside: How to Work Across Teams

- Solve their problems
  - Have proven solutions for integration, security, DR, …
  - Immediate value to development teams
- Jointly develop your architectural principles
  - Ensure they are understood and agreed
- Collaborate rather than police
  - Review to share & improve, not to govern
- Stay out of internal decisions
  - Unless invited or you need to avert catastrophe
  - Collaboration will mean that you have input anyway



Content

- Software Architecture and Software Architects
- Agile Principles and Software Architecture
- Practices for Agile Architecture
- Examples of Agile Architecture
- Closing Thoughts



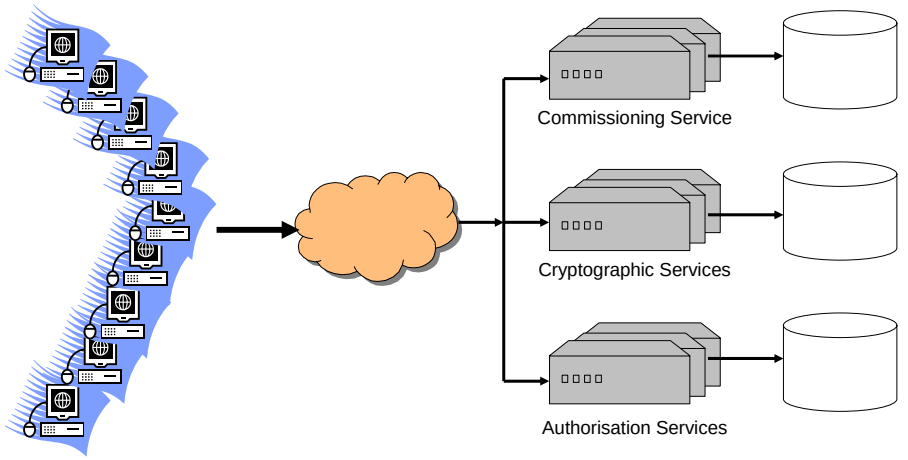
Examples

1. Building a new family of servers
2. Agile modelling
3. Common solutions for a family of systems

Example 1 – A New Server Family

- A startup company needed to create a new generation of its product very quickly (to market in < 9 months)
- The architecture needed 5 new servers from scratch
- The product must be capable of supporting Internet scale loads
- The previous product had a bad reputation for reliability, supportability, scalability and performance
  - this has to be sorted out this time around
- The server engineers were 400 miles away from the rest of the engineering team (and the server architect)
  - Oh, and they'd just been acquired from another company!

Example 1- A New Server Family



Example 1 – A New Server Family

- Some key risks became clear early on
  - Tight timescales mean that people "just want to do it". This can lead to localised solutions, a fragmented approach and so an unsupportable product.
  - There obviously wasn't time for a lot of BUFD and risk mitigation
  - Not enough time to build a neat generic reuse library to force commonality across the servers
  - People focusing on key requirements (e.g. performance) could well lead to others being forgotten (e.g. reliability)
  - Short term focus could well lead to a product we couldn't maintain and enhance – this wasn't acceptable
  - Geographical split made communication difficult
  - Engineers were new to the company so we didn't have prior experience and relationships to fall back on

Example 1 – Solution

- Applied the *identify clear component responsibilities* practice
  - prevented overlap or confusion within the architecture
- Applied the *good enough modelling* practice
  - allowed a sharp focus on what was actually needed
- Applied the *minimal principles* practice
  - made sure the key decisions were agreed up front
  - only bothered with principles that really mattered
- Applied the *standard architectural solutions* practice
  - made sure that security and HA in particular were standard
  - saved a lot of redundant design time across the servers

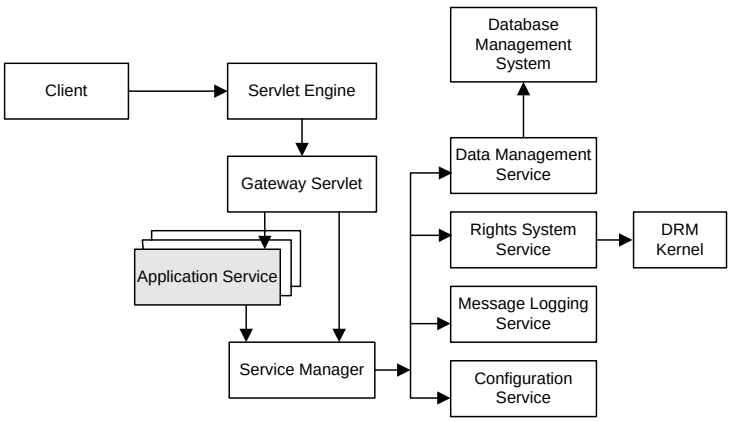
And we worked very hard

Example 1 – Architecture Deliverables

- A server product line architecture
  - 4+1 views to define the essential commonality *with rationale*
  - defined standard technology
  - defined cross cutting mechanisms and conventions
  - defined a standard design pattern for the servers
- Development standards
  - essentially the "development view" of the architecture
  - defined i18n, logging, request handling, technology to use, test approach, exception handling, product configuration, …
  - Anything not here or in the architecture was up for grabs
- The implementation of one of the servers
  - architect suffered his own decisions
  - resulted in rapid refinement of the product line architecture!

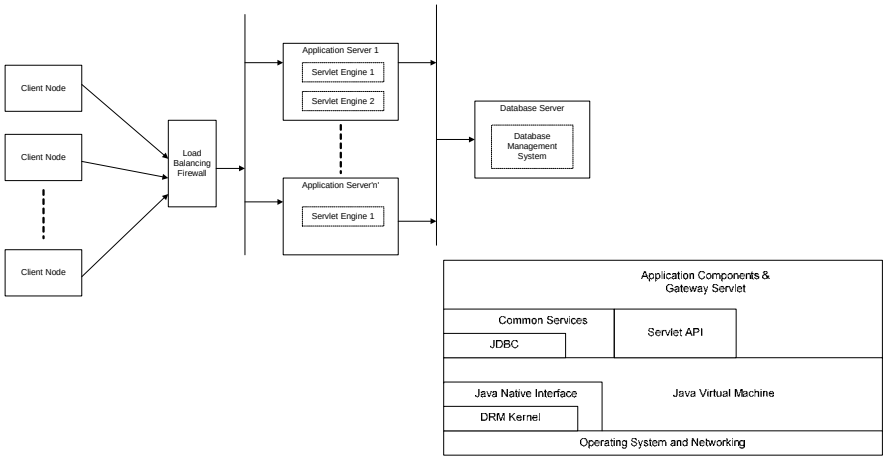
Example 1 – Architecture Fragments

- Generic functional view of server



Example 1 – Architecture Fragments

- Deployment and Layering (Development) views



Example 1 - Results

- Ultimately successful because product was delivered
  - first version ready for beta testing within 9 months
  - deployed at major mobile manufacturer test labs in this time
  - judged to have met all of its significant requirements
- The architecture was definitely used
  - the server developers all used it
  - high degree of commonality achieved (without reuse)
  - parts of it were revised right through the development cycle
- Server team became much more integrated
  - challenge of problem forced communication
  - architecture provided context for decisions and discussions
  - something to "rally around" when dealing with other teams
  - architect working in the team had a lot of beneficial results

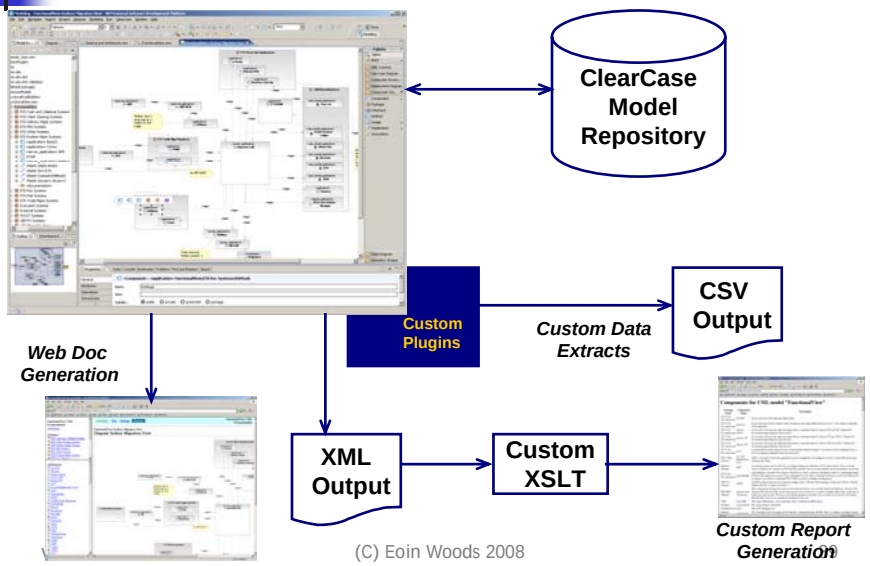
Example 2 – Agile Modelling

- Context is capital markets acquisition of a competitor
  - exchange traded derivatives (ETD) domain
  - existing bank's futures clearing merchant (broker) acquiring the business of a competitor bank
  - acquirer had ~50 systems
  - acquired organisation had ~50 systems
  - both complex individually and no one with deep knowledge of both
  - lots of change needed to achieve integration
  - difficult to understand existing and future state systems landscapes

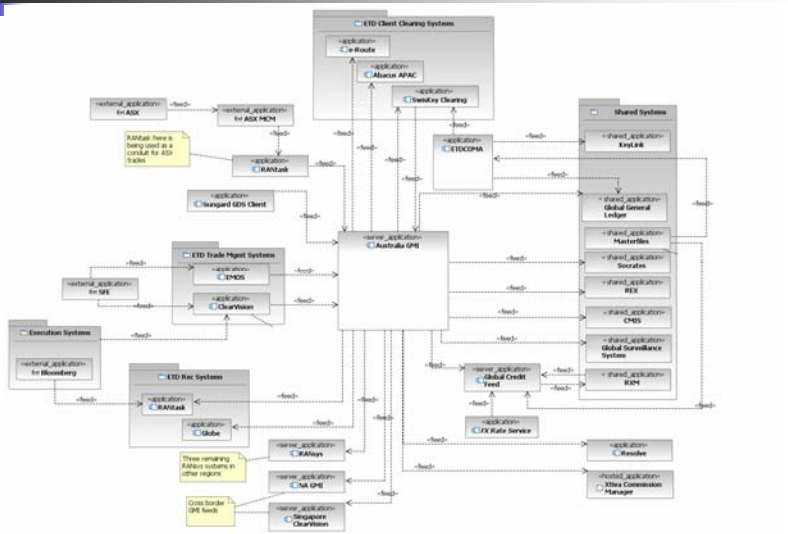
Example 2 - Agile Modelling

- Applied the *good enough modelling* practice
  - not enough time for exhaustive modelling exercise
  - identified what would deliver value quickly
    - system level future state (systems and flows)
    - just built a model to meet this need
  - created a precise, but abstract systems and interconnections model in UML using IBM RSM
  - created a model (not a picture) to allow multiple uses and motivate updates
    - never published as a document, just model outputs

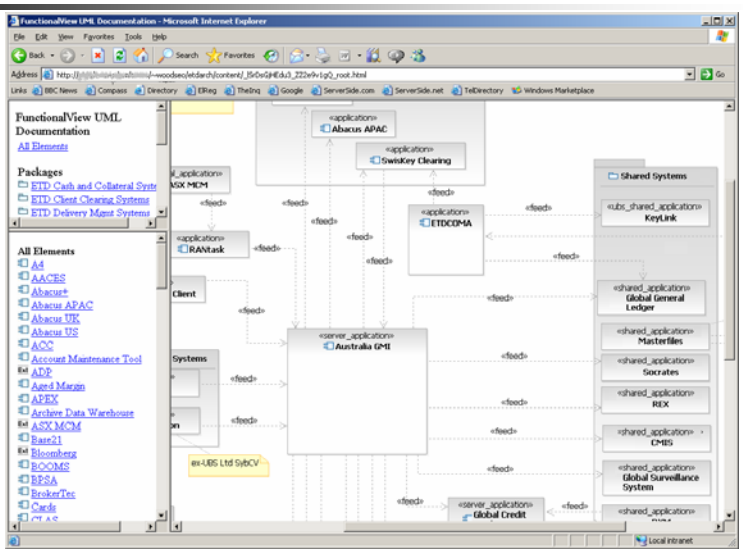
Example 2 - Tool Support



Example 2 - Example Model View



Example 2 – Generation to Web



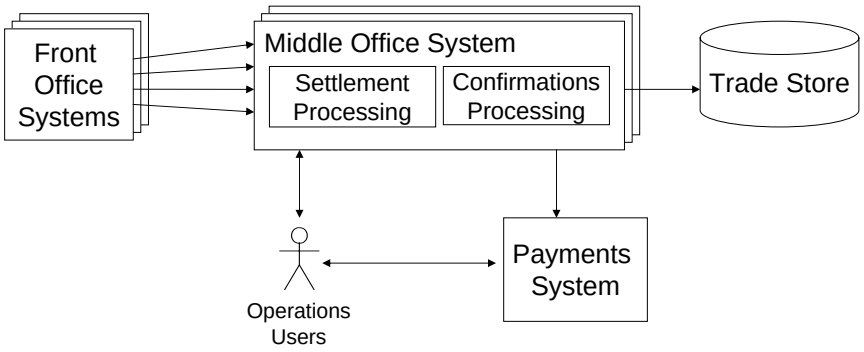
Example 2 – Results

- The model web site formed a useful a reference point for change teams
  - provides database of system & connection definitions
  - more technical change managers found it accessible
- The RSM extensions allowed data to be extracted
  - providing motivation for maintenance and use
  - Excel was a useful "lingua franca" for distributing and checking
- Use of model allows many views to be quickly created from the content
  - although Powerpoint/Visio integration still problematic
- Keeping models current remained a constant challenge

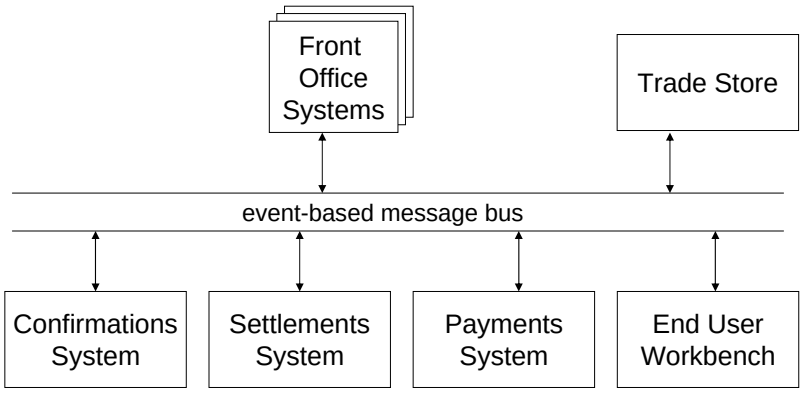
Example 3 – Common Solutions

- Example of focused x-system architecture making agile development more effective
- Context was large tightly coupled systems
- Applied focused set of architecture practices
  - Defining clear system responsibilities
  - Defining clear, minimal architectural principles
  - Small amount of common design (messaging)
  - Provided solution patterns for DR, integration, security
  - Worked directly with (in) the first teams

Example 3 – Starting Point

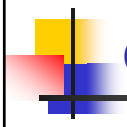


Example 3 – Result



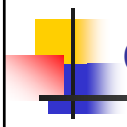
Example 3 - Results

- The architecture work resulted in
  - Smaller systems, better defined responsibilities
  - Looser coupling via well defined neutral event model
  - Development team independence
- Has allowed much easier change
  - Systems can have own development & release cycles
  - The architecture has enabled overall agility
  - Overall result is more effective delivery and change



Content

- Software Architecture and Software Architects
- Agile Principles and Software Architecture
- Practices for Agile Architecture
- Examples of Agile Architecture
- Closing Thoughts



Closing Thoughts

- Making architecture more agile benefits everyone (architects, users, dev teams)
- Architects share many priorities with the agile movement
- Small number of effective practices improve agility immensely
- Agile architecture has worked well in practice and made architecture relevant
- It's all about delivering valuable working systems

The Ultimate Proof …

Eoin Woods
Barclays Global Investors
eoin.woods@barclaysglobal.com
www.eoinwoods.info