Introductions

- I’m a stream technical architect for UBS Investment Bank
  - Our organisation includes technology architects, software (system) architects and stream architects
  - I’m one of the architects responsible for the ETD area
- Software architect for ~9 years
  - Plus enterprise architecture for ~2 years
- Author of “Software Systems Architecture” book with Nick Rozanski
- IASA Fellow
Enterprise Architecture

- IT strategy / business alignment
- Technology strategy and standards
- Functional architecture
  - System responsibilities
- Integration architecture
  - System interfaces
  - Inter-system flow
- Cross system common design

Our focus as stream architects

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

<table>
<thead>
<tr>
<th>Practice</th>
<th>Principle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Release frequently</td>
<td>Integrate changes often</td>
</tr>
<tr>
<td>Iterative delivery</td>
<td>Collective ownership</td>
</tr>
<tr>
<td>Customer is available</td>
<td>Customer prioritises</td>
</tr>
<tr>
<td>Simplest thing possible</td>
<td>Collaborative design</td>
</tr>
<tr>
<td>Specify via “stories”</td>
<td>Small teams</td>
</tr>
<tr>
<td>Test driven</td>
<td>Automate routine tasks</td>
</tr>
<tr>
<td>Refactor when needed</td>
<td>Shared workspace</td>
</tr>
</tbody>
</table>

Common Priorities

- Architects and agile teams have many of the same 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

- 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

Why Agile Architecture?

- Avoid retreat to the “ivory tower”
  - Ensure relevance of what we do
- Focus on stakeholders
  - Make sure we deliver value
- Support change
  - So we deal with the realities of our environment
- 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

Organisational Environment

- **External Stakeholders**
  - Central technology teams
  - Legal, Audit, Compliance, ...
  - Organisation wide initiatives
- **Business Management**
- **Architects**
- **Development Teams**
  - Technology constraints and direction
  - Validation and review
- **End Users / SMEs**
- **Change Teams**
  - Functional needs
Architecture Effectiveness Index

- Level 0 – no architectural impact
  - Architecture 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

Agile Architecture Practices

**Allow for Change**
- Deliver Incrementally
- Allow reprioritisation
- Identify clear system responsibilities
- Design a catalogue of integration options

**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-system concerns
Practices – Allow for Change

- Deliver Incrementally
  - Visible progress, early feedback
- Allow reprioritisation
  - As your customer priorities change
- Identify clear system responsibilities
  - Allow confident extension
- Design a catalogue of proven integration options
  - Limit the choice while solving the problem
  - Provide clear rationale for making choices
  - Allow for the special cases as they’re inevitable

Practices – People not Process

- Work with development teams
  - Don’t drop documents, talk to the teams
  - Develop things jointly
  - Great source of knowledge and resources
- Keep backlog visible
  - Let people know what you’re planning
**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
  - Something useful directly or for research purposes
- 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
- Our experience suggests
  - Focus on models where systems are components
  - 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
  - Systems, responsibilities, inter-system connections
  - Shared domain model (so allowing integration)
Practices - Collaboration

- Define minimal principles & share them
  - Small set easily understood & accepted
  - Choose your battles (high value decisions)
- Share information online (e.g. Wikis)
  - Allow easy access, comment and update
- Focus on cross-system concerns
  - Avoid the system’s areas of responsibility

Aside: Engaging 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
Examples

1. Defining a catalogue of integration options
2. Agile modelling
3. Common solutions for family of systems

Example 1 – Integration Options

- Improving an environment with “light touch” EA
- Context of many systems linked in many ways
- Integration had evolved organically over years
  - Largely predated messaging
  - Perl scripts, shell scripts, direct database access, stored procedures, pub/sub, file unload/load, …
- Each feed worked perfectly well in its own way
  - But little or no commonality
  - Difficult to understand, monitor, debug
  - Very brittle making change difficult
Example 1 – Solution

- Apply *minimal architecture principles* practice to define how to integrate systems
  - Producers independent of consumers
  - Publish once in neutral form
  - Identify each interface as bulk or event oriented
- Apply *catalogue of integration options* practice to give set to choose from, with rationale for each
  - **Bulk**: db to file / file to db via ETL tool in CSV form
  - **Event**: messaging via pub/sub product, XML or 3rd party formats used (e.g. FIX based)
  - **Other** options on a case-by-case basis

Example 1 - Result

- Given a set of options and strong rationale, people have reacted positively
  - Previous situation due to lack of existing principles
  - Never had time (or motivation) for standardising
- Still have tactical work going on but the standard options are considered first
  - Choose a standard option if possible
  - If not, then allow a custom approach but with a plan for future rectification
- As feeds are rebuilt, we are moving towards our desired standardised future state
Example 2 – Agile Modelling

- Context is UBS acquisition of a competitor FCM
  - UBS ETD 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

![Diagram showing tool support](image)

- ClearCase Model Repository
- XML Output
- Custom Plugins
- Custom Data Extracts
- CSV Output
- Custom XSLT
- Web Doc Generation
- Custom Report Generation

Example 2 - Example Model View

![Example model view](image)
Example 2 – Generation to Web

Example 2 – Progress & Results

- The model web site is becoming a reference point for change teams
  - Provides database of system & connection definitions
- Our RSM extensions allow data to be extracted
  - So providing motivation for maintenance and use
- Use of model allows many views to be quickly created from the content
  - Although Powerpoint/Visio integration still problematic
- Keeping currency is a constant challenge
Example 3 – Common Solutions

- Example of focused 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

[Diagram showing Front Office Systems, Middle Office System, Trade Store, Operations Users, Payments System, Settlement Processing, Confirmations Processing]
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
Summary

- 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

Questions and Comments?

- Now or to Eoin.Woods@ubs.com