Descending from the Architect's Ivory Tower

Andrew Rendell, Principal Consultant
Valtech UK
Agile 2009
Chicago

August 24, 2009

delivering business agility
Introduction

- Many organisations have long history of waterfall development
- Years of refinement have produced techniques for technical architecture which contradict Agile ideals
- How do you fulfil the role of technical architect in such an organisation and maintain the agility which allows you to deliver software?
- Examined through the experience of a real world project
Organisational context

- T-Mobile International Mobile Portals and Content Delivery Group
- Delivers mobile applications which drive data usage across the T-Mobile network
- Developed their expectations of the technical architect role over several years
Broad definition of Architect

- In the context of this group
- Wide remit across several projects
- End to end technical responsibility (i.e. if it does not work, you have failed)
- Delivery of consistently efficient solution which is fit for purpose
My experience on waterfall projects

- Technical architect is discouraged from taking hands on approach
- Role closely coupled with review of design documentation
- In fact – becomes primary tool for architectural quality control
- Development process structured around review quality gates
Waterfall makes *you* ineffective

**Conventional document review based approach:**
- Isolates the architect from the technical implementation they are responsible for
- Reduces their ability to succeed by taking them away from the technology which made them great candidates for the architect role in the first place
- Supports the fallacy that the technical architect is the all-knowing centre of the technical universe
Tower syndrome

- Poor visibility
- Safety
- What does he want?

- Poor visibility
- Architects ivory tower
- Developers
What does this lead to?

- **Architect disenfranchised**
  - A waste of years of learning and experience

- **Poorly regarded by implementers**
  - A dabbler
  - Poorly informed
  - An impediment to progress
  - Avoid their attention at all costs!
Do Architects have a role in agile?

- Yes!
- Organisations still have the same requirement
- Logical career progression for technicians
- How can the role be delivered without ascending the tower?
Background to project

- Summer 2007 new Portal application developed
- Condensed time scales
- High profile
- Unusually high risk for the organisation
- Tactical agile adoption required to deliver
- Project succeeds and delivers on time
- Further adoption not only possible but highly desirable
Post launch hangover

- Doubts that too much of the old processes had been discarded
- Was the team creating a maintenance nightmare through lack of documentation?
- How to keep delivering at rapid pace with increasing scope?
Scaling the team

- Increase headcount
- Two teams
- Different managers
- Different locations
- Little interaction at individual level
- Silos form
- Began to deliver rapidly
- But...
Architectural bad smells

- Diverging code styles
- Previously commonly applied patterns overlooked
- Unpredictable and inconsistent behaviour
- Technical debt rising dangerously
Increasing code base and complexity
Rate of change

Release

NCSS increase
Complexity increase
Why was the architect ineffective?

- Architect remained on the critical path for one team's development
  - Same expected velocity as any other developer
  - Deliverables actually *increased*
- Remit well understood
- Mechanisms for fulfilling that remit not well understood
Redefining the approach

- Problems recognised
- Opportunity presents itself to make changes
- Reorganisation of technical governance
- Need to scale effectively
Scale development capacity

Empowerment

Provide direction

Empirical analysis

Enable communication

Maintain and improve quality

Delegation
Techniques employed

- A selection of the techniques employed
  - Structuring the application to promote good governance
  - Documentation that is 'good enough'
  - Key tool: Testing
  - Reflecting on code structure
  - Providing direction and rigorous checkpoints
  - Delegation
Structuring Application

- Promote good governance through physical architecture
- Analogous to TDDs 'detestable code'
  - Worth restructuring code only to make it easy to test
  - Worth restructuring application only to improve technical governance
- Contentious
  - Seemed like the wrong motivation
  - Emotional response: Costs too much
Structuring Application

- Why increase complexity / size of monolith?
- Cost to restructure low
- Smaller, more manageable deployment units
Structuring Application

- Teams decoupled structure mirrored application
- Silos recreated, but
  - Clean interfaces
  - Easy to police
  - Freedom to innovate
  - Not license to disrupt
- Operations team supportive
- Massive saving in test
- Probably saved the project
- Simpler structure was easier to govern
Documentation that is 'Good Enough'

- Determined not to expend valuable effort on documentation which has no clear use
- Where documentation is required it must be fit for purpose, not a tick in box
- Valuable documents for this project
  - Web service API specification
  - Use case documentation
Documentation that is 'Good Enough'

- Discarded detailed design doc
- Previous approach
  - Mandated detailed design document per module
  - Significant overhead
  - How was it actually used?
- New approach
  - Key areas identified by architect and team
  - White board sessions
  - Captured with digital cameras and uploaded to wiki
  - Wiki pages on specific topics
  - Gives architect the required visibility
Documentation that is 'Good Enough'

- Some areas justified more formal capture
  - Added to Sprint Backlog
  - Keeps cost visible
- Used UML tool
- Centralised, version controlled
- Generated documentation sets from tool
Contestation

- Wider organisation still document centric
- New approach contentious, especially when interacting with that wider organisation
- Security audit performed on project
- Negative preconceptions about agile
- Architect able to supply documents generated from repository
- At least as rigorous as conventional peers
Testing: The key tool

- TDD key developer tool
  - Subject of Agile 2008 presentation
- Automated tests main mechanism for ensuring quality
- Empirical evidence
  - As opposed to abstract view of design to be realised
- Justify their high cost
Test consumers and classification

Developers
- Unit tests
- Load tests
- Acceptance tests

Architect
- Load tests
- Acceptance tests

Business stakeholders
- Use case tests
Testing

- Cost of TDD is initially high
- Cost of writing Use Case tests is high and impacts some key individuals
  - We failed to create them quickly enough
  - Delay diminished value
- Cost of being involved in load testing high and created scheduling conflicts
- Nothing else can provide same confidence levels
- Just remember: Nothing is foolproof!
Reflecting on code structure

- Development priorities and patterns had been communicated
- Not always followed!
- Use early code review to catch problems
- Driven by Architect and project technical lead, not a team activity initially

Practical exercise
- Used Eclipse code navigation facilities
- Used UML reverse engineering
- Unit tests exercised code
Reflecting on code structure

- Exercise produced list of corrections
- Architect annotated code with FIXME and TODO
- UML diagrams from reverse engineering export to wiki
- Having a formal diagram somehow made the process 'more significant'!
- Fed into retrospective
Positive outcomes

- Architect gains confidence in team
- Issues corrected early
- Developers confidence boosted
  - Positive feedback
  - Know are interpreting guidelines as intended
  - Reported in retrospective
- Architect becomes familiar with code
- The source is no longer an opaque artefact
Practical issues

- Prohibitive cost
- Only executed handful of times per project
- Identifying where to invest time in review
  - Allow developers to nominate 'cleverest code'
- More cost effective solutions
  - Developer driven white board sessions
  - Same objectives
  - Cost distributed amongst team
  - Less useful – more abstraction, erode common ownership
Direction and checkpoints

- **Direction**
  - Developer principles on wiki

- **Checkpoint**
  - Architecture check list

- **Not generic – very application specific**
Architecture checkpoint

- Unlike conventional checkpoints I have experienced
- Similar to code review in that based around workstation
- Question and Answer style
  - Not all simple yes / no
  - Aim to stimulate further investigation
Logging and monitoring

- List all the components in the application whose failure would result in an error state which affects the customer experience and requires operator intervention to correct.
  - DLS failure
  - SDP failure
  - NGCS failure
  - Tomcat dies
- Are all errors that could cause the above scenario going to produce a FATAL error log message?
  - Tomcat looks after itself - FATAL.
  - Looks pretty good for DLS etc. get an ERROR for bad HTTP status code.
  - NGSC looks okay as well, catches all exceptions and reports ERROR.
- Do all errors that relate to an unexpected problem within the application result in an ERROR log message and a single stack dump (per occurrence)? Almost, FIXME added to code. Double stack dump.
- Do all errors that indicate a transient or customer specific issue with a partner system result in a single WARN log message with no stack dump? No, but cannot identify this problem. Could try and handle HTTP connection error or HTTP 500 but probably overkill.
Cost and benefits

- Exposed some show stoppers
- Demonstrated rigour to external observers
- Documented empirical evidence
- Brought architect and project lead closer to code
- High cost in time from key team members
- Scheduling difficult, often delayed which increased risk
- Not fool proof tools, but very powerful
Architects value Horizontal over Vertical Scalability

Same pattern can be applied to development teams

Architect must delegate

Quality of team is paramount
Delegation: Building the Team

- **Anti-pattern**
  - Interviews where selection is based on emotional response
  - Interviewer post rationalises decision
- **Use empirical evidence instead**
- **Case study driven interviews developed**
- **Specific, not generic, to role and project**
- **White board sessions to talk through a design**
- **Use Eclipse and Maven to demonstrate TDD etc.**
Delegation

- Ivory Tower architects seem to concentrate on technology not people
- This impedes their ability to scale their development capacity
Conclusions

- Documentation as a tool for technical governance is not an effective strategy
- Architect must reduce their isolation through involvement in the high value technical activities
- Automated, accessible tests are the Architect's best tool
- Architect cannot be a full time developer
- Activities that make the developers aware that the Architect is close to the source increases common ownership and invigorates high standards
- Soft skills can be as important as technical acumen
- The best techniques cannot always be applied, target the most important areas and apply cheaper methods elsewhere