Outline

- A story
- Agile processes and software/system architecture
- Value and cost
  - Cost
  - Value
- Value and cost of architecture
- A proposed simple strategy to put value on architecture

Story of a failure

- Large re-engineering of a complex distributed world-wide system; 2 millions LOC in C, C++, Cobol and VB
- Multiple sites, dozens of data repositories, hundreds of users, 24 hours operation, mission-critical ($billions)
- xP+Scrum, 1-week iterations, 30 then up to 50 developers
- Rapid progress, early success, features are demo-able
- Direct access to “customer”, etc.
- A poster project for scalable agile development
Hitting the wall

- After 4 ½ months, difficulties to keep with the 1-week iterations
- Refactoring takes longer than one iteration
- Scrap and rework ratio increases dramatically
- No externally visible progress anymore
- Iterations stretched to 3 weeks
- Staff turnover increases; Project comes to a halt
- Lots of code, no clear architecture, no obvious way forward

Agile Processes & Architecture

- No BUFD (no Big Up-Front Design)
- Incrementally develop (& deliver) value
- xP: Metaphor
- FDD: Features
- Earned-value system -> burn-down charts
- Very short iterations (a.k.a. sprints)
- Refactoring
- Gradual emergence of the design…
Context does Matter

- For medium to large software-intensive systems, or in novel systems, an architecture will NOT gradually emerge as the result of constant refactoring.
- The Wall
- Architecture lacks sex appeal

Value and Cost

- Value: to the business (the users, the customers, the public, etc.)
- Cost: to design, develop, manufacture, deploy, maintain

- Simple system, stable architecture, many small features:
  - statistically value aligns to cost
- Large, complex, novel systems?
**Cost**

- **The old:**
  - Function points, SLOC
    => time
    => $$$

- **The “new”:**
  - Story points, ideal days, velocity
  - Backlog (Scrum) used to drive increment content
  - Self-tuning process

---

**Cost: the classical view**

- **Productivity** $SLOC/\text{person-month}$
- **Size** $F.P.$, $SLOC$
- **Effort** $\text{person-month}$
- **Staff**
- **Duration**
- **Cost**
Cost: the agile view

- Size
- Effort
- Staff
- Duration
- Cost

- Velocity
- Productivity
- Story point per ideal day
- Ideal days
- Actual days

Value?

- Traditionally in dollars
- Decomposition, interdependencies
- Priorities (used in XP’s planning game)

- Not very successful

- Earned Value System muddies the water even more
  - Are we speaking about value?
  - Cost? Both?

- What is the value of architecture? About nil
  - Seen only as an additional cost

Cohn 2006

Beck 2001
A Proposed Strategy

- Cost in points
- Value in units

- Get valuation done in relative units (what a unit mean is irrelevant)
- Try to break-down big value elements

- Keep value independent from any notion of cost

Relate to: 100-point method, Karl Wiegers’ prioritization scheme, …. AHP, Theory W (?)

(cont.) Valuing architectural design

- Value architecture by taking units from top level, user-visible features, and flowing them down to non-visible development elements

- How?
  - The “revenue taxation” model: 12 % across
  - The “head tax” model: collect fixed amount of units
  - The “pay-per-use” model: pay a percentage only if it makes use of it
  - The “auction” model ??
(cont.) Flowing-down value

- Need a rich dialog between
  - Developers
    - Architecture, design
    - Dependencies between design “chunks”
    - Costing development in points
  - Business representatives
    - Features, prioritization
    - Valuation in units

  ... during early phases to jointly “flow down” value to development elements
    - ICM: valuation and architecting phases

Boehm & Lane 2007

Points (cost) and Units (value)
Some rules for the game

- Total value is constant, through flowdown, hence throughout architectural design
- Adding requirements adds value (using relative units to evaluate)
- Total cost evolves with architectural design (it should go down, or maybe not)
- Costs re-evaluated as development progress (agile concept of velocity)
- *Value cannot be changed after implementation to change priorities*
- Keep costs and values well separated
- Can't deliver architectural bits without user visible bits (and vice versa)
- MMF

Key points

- Value often not correlated to cost
- Express value in relative terms, not absolute $$$
- Proceed to architectural design
- Re-allocate some of the user-visible value to non-visible element, with constant sum
- MMF = minimum marketable features
- Schedule iteration sequence based on fully valued development elements + dependencies
- Better fit to a *revised* Earned-Value System
- Many benefits in the dialog itself (value is in the journey)
Value (units), Costs (points), and real $$$

\[
\sum_{\text{units}} \text{Rev. $$$} \quad \sum_{\text{points}} \text{Dev. $$$}
\]

Time

Alternative approaches

- CBAM = Cost Benefit Analysis Method
  - Chap 12 in Bass, Clements, Kazman 2003
- IMF: Incremental Funding Method
  - Denne & Cleland-Huang, 2004
- Analytic Hierarchy Process
- Evolve* - Hybrid
  - Günther Ruhe & D. Greer 2003, etc…
### CBAM: Cost Benefit Analysis Method

- **Concept: Utility**
  - \(= \text{Value (?)}\)
  - Utility-response curves: linear, steps, exp,…
- **Concept: Scenario**
  - And priorities
- **Architectural strategies**
  - Their value, and utility
  - Their cost
- **Benefit and ROI (Return on Investment)**

### IFM: Incremental Funding Method

- **MMF** = Minimum marketable Features
- **AE** = Architectural elements
  - Cost
  - MMF depends on AE
- **Time and NPV** = Net Present Value
- **Strands** = Sequences of dependent MMFs
… but the same issues

- How to assign realistic
  - Value
  - Cost
  - Priority
- to each chunk of software?
- And how to make it appealing to the agile projects?
  - Separation between the visible (feature) and the invisible (architectural element)
  - Make it practical for small and medium teams

Conclusion

- There is both value and cost in Software Architecture
- They may be articulated in simple, non financial terms
- to assist planning iterative development
- and avoid “hitting a wall”.

- Start small and simple.
- Get fancy later.
References

- Boehm, B. and Lane, J.A. (2007) Using the incremental commitment model to integrate system acquisition, system engineering, and software engineering, University of Southern California, Los Angeles, September 2007.

References (cont.)