When Good Architecture Goes Bad

Mark Dalgarno

Software Acumen

Blog: *The Variation Point*
(http://blog.software-acumen.com/)

Email: mark@software-acumen.com

(Approximate) Agenda

- Introduction
- Architectural smells
- The cost of decay
- Causes of decay
- Preventing decay
- The value of architectural integrity
- Close

Introduction - Software architectural decay

- Architecture **as-is** diverges from architecture **as-intended**.

- Results in a decrease in the ability of a system's software architecture to meet its stakeholder requirements.

Introduction - Example of architectural decay

- Researchers compared two versions of ANT
  - System built in three layers *taskdefs, ant, utils*.
- V1.4.1 (11 October 2001)
  - Layers well-separated.
  - *ant* layer monolithic but small.
- V1.6.1 (12 February 2004)
  - *ant* layer dependent on *taskdefs* (upward dependencies).
  - *ant* layer now very large but still monolithic.

See http://www.stsc.hill.af.mil/crosstalk/2005/11/0511SangalWaldman.html for more information on
The study and http://codefeed.com/blog/?p=98 for a brief early Ant project history. Accessed 19/1/08

Architectural Smells - Structural Smells

- Code in the *wrong* place
- Problems in class, package, sub-system and layer relationships
- Insufficient decomposition
- Too much decomposition
- Obsolescence
- Overgeneralization

Architectural Smells - Whiffs (or subtle smells)

- No one on the team can tell you (or agree on) what the **as-intended** architecture is.
- The time, effort and risk in implementing further changes increases – productivity and quality decrease.
- It becomes harder to predict the effect of further changes on cost, schedule and quality.
- Further changes typically cause the **as-is** architecture to deviate further from the **as-intended** architecture – the situation becomes worse.

Architectural Smells - Exercise 1

- In groups, identify one or more examples of architectural decay from your own experience.
- Were any smells (or whiffs) associated with these examples?

The cost of architectural decay – an experiment

- STSC conducted a study with two variants of a mature software system (50k LOC).
- Variant 1 – existing system with structural defects.
- Variant 2 – system with architecture restructured to remove defects.
- Both teams given same maintenance task (adding approx. 3k of code).
- Team 1 needed over twice as long as team 2 to complete the task. Team 1's results contained more than 8 times the number of errors than the work submitted by team 2.

The cost of architectural decay

- Lowering quality lengthens development time – but is business aware of this?
  - Do they care or will they worry about getting out of 'debt' later?
    - Beat competitor to market.
    - Grab market share.
    - Win contract on the cheap & charge more later.
  - Can be hard to communicate state of architecture to business
    - Hard to understand architectural issues.
    - Blame culture – how did it get that bad?

The cost of architectural decay – Exercise 2

- Read Case Study 1.
- In groups discuss whether it is credible that architectural decay led to this significant decrease in productivity?
- What do you think of the company's proposed solution?

Causes of decay

- Change brings decay
  - Functional / non-functional changes.
  - Environmental changes (inc. team, tools).
  - Worse if architecture doesn't support change.
- Ignorance, misunderstandings, mistakes
- Hard to visualize **as-is** architecture to see if it matches the **as-intended** architecture
- Insufficient value placed on evolvability and ongoing architectural integrity

Causes of decay - Exercise 3

- Read case study 2.
- In groups, discuss whether the architecture will decay when the system is maintained.
- Justify your answer.

Preventing decay – a skeleton process

- Start out with a sustainable architecture.
  - Assess it using change scenarios.
- Visualize the architecture as the software evolves.
  - Compare **as-is** to **as-intended**
- Use metrics to highlight architectural smells.
- Refactor to maintain integrity.

Preventing decay - Exercise 4

- In groups, list things that could have been done to slow or prevent architectural decay in one of your own examples from Exercise 1.

- Include anything you tried that did or didn't work at the time.

Slowing decay – what other people said

- 10 experienced architects / developers completed a small survey for Software Acumen
- Most had not heard of tools to help visualize software architecture
- Desired features of such tools were:

4. Visibility of software architecture **as-is**
5. Interrelationship comprehension
6. Ability to check and enforce architectural integrity
7. Advance visibility of the effects of refactorings
8. Identification of components to enable re-use
9. Identification of opportunities for refactoring
10. Elimination of cyclic dependencies to improve code quality

The value of integrity - the problem

- It's hard to measure the (money, time, organisational, personal) benefit of architectural maintenance activities.
- Architectural integrity pays off over the long term in many cases.
- You may get a quicker return if you spend money elsewhere.

The value of architectural integrity – Exercise 5

- Revisit one or more of your earlier examples of architectural decay.
- What was the (money, time, organisational, personal) cost of letting the architecture decay?
- If you could go back in time what steps would you take to reduce these costs? How effective do you think these steps would be?

Things to ponder…

- *"The average developer is too stupid to use architectural analysis techniques"* – unnamed CTO, October 2006
- Who's responsible for architectural integrity?
- When you specify an architecture do you spend enough time considering how it would be affected by change?
- *"It's all about communication"* – SPA 2008 participant
- Does it matter if architecture decays as long as the tests pass?
- How bad should a software system's architecture be before you scrap it?
- Architectural decay is depreciation of the software owner's assets – should this be reflected on the owner's balance sheet?

Recommended Reading

- *Refactoring in Large Software Projects: Performing Complex Restructurings Successfully*, Martin Lippert, Stephen Roock, Wiley 2006
- Lehman's laws of software evolution

  M M Lehman, J F Ramil, P D Wernick, D E Perry, W M Turski, *"Metrics and Laws of Software Evolution – The Nineties View,"* metrics, p. 20, Fourth International Software Metrics Symposium (METRICS'97), 1997

Summary

- Any successful software system is likely to evolve.
- Unless preventative work is undertaken the architecture of the system will decay.
- As the architecture decays the cost and risk of further development rises.
- There are lots of different things that can be done to slow architectural decay – you (just) need to work out what the best value approach is.

Thanks go to…

- Rob Machin, UBS Investment Bank
- Thomas Eisenbarth, Axivion GmbH
- Klaus Marquardt, Drager Medical Systems GmbH
- Paul Clements, SEI
- SPA 2008 workshop participants

- Email mark@software-acumen.com for these slides. Visit **http://blog.software-acumen.com/ for session write up.**