Software Evolution in the Financial Industry - Practitioner Report

Carl Worms
Guest Lecture 2016
IAM Bern
Who I am

- Studied Computer Science in Karlsruhe, Germany
- Lived from programming for 15 years
- Walter Masing awardee (DGQ)
- IT Architect/SWE Process Architect at a major Swiss bank for 16 years
- PC member of IEEE conferences since 2007
IT in Large Banks?
IT in large banks

• Huge global infrastructure
  – >100,000 PC/Laptops
  – ~30,000 Servers and (>0) big mainframe
  – A multitude of technologies (HW, OS, DB, ...)
  – ~5,000 Applications, >500m SLOC
  – ~2,000 SOA like Services
  – ~50,000 Data Bases, >1,000,000 data attributes
IT in large banks

• Huge global IT organization
  – ~25-30% of the bank’s staff is in IT
  – The big Swiss banks are the major IT employers in Switzerland
  – Global distributed SWE
  – ~50 high level processes areas (one is SW design)
  – A multitude of SWE methodologies
  – A multitude of cultures
IT in large banks

- High performance
  - >100m transactions/month
  - >10 tons of prints (accounts, tax reports)/year
- Strictest security requirements
- High regulatory constraints

- Extreme change rate (monthly/quarterly)
- >20.000 bug fixes/year in a major hub
Question 1

If you are the CIO (Chief Information Officer) of a large bank - what’s your major concern?
Question 2

If you are the Chief Architect of a large bank - what’s your long-term vision?
Answer from Practice:

Managed Evolution

From: Murer/Bonati/Furrer, Managed Evolution; Springer 2010
Managed Evolution

Managed Evolution: Balanced Development of IT Efficiency and Business Value

Driver: Business requirements, Time to market

From: Murer/Bonati/Furrer, Managed Evolution; Springer 2010
All systems have an architecture, be it an implicit or an explicit one.

If the architecture is implicit, we have no way to control, analyze, reason about, evolve, and communicate it.

It is the role of the architect to establish an explicit architecture of the system.
Software Evolution Management Principles

• **Flexibility** with regard to business organization and expansion
  • applications tend to live longer than the organization .... Thus, we keep the architecture flexible (-> Multiple Channels, Multiple Countries, .....)

• **Componentization**
  • The whole platform is decomposed into components with well-defined interfaces among each other.
  • Components encapsulate data and related functionality
  • Application domains serve as high-level components
  • The decomposition into components is supported by an adequate piece of integration infrastructure
Software Evolution Management Principles - Componentization

Number of Components to Integrate

- small
- large

Complexity of Interfaces

- low
- high

Cost
Software Evolution Management

Principles

• Accept and handle **trade-offs**
  • Architecture is not an exact science
  • There are conflicting goals and interests
    => necessity of architecture process to define standards…and to handle exceptions

• The **importance of standards**
  • The main property of a standard is its widespread acceptance and support in the market
  • The discussion of architecture will generally focus on standards rather than products
  • It is an important duty of IT Architecture to enforce guidelines that restrict the use of the product to standard features

• **Avoid unnecessary technology diversity**
  • each infrastructure functionality is covered by exactly one product
  • … but temporary overlaps when old standard is being replaced by new one
  • … exception: exert price pressure on vendors
Software Evolution Management Principles

• Be properly positioned in the technology life cycle
  • there is a lot of hype around life cycle
  • … a careful technology portfolio management is important
  • Managed Evolution makes it possible to predict the replacement of a specific technology and to allow for an adequate transition phase

• Stay in the mainstream
  • Generally a bank’s technology strategy will be an ‘early follower strategy’ of mainstream products
  • exceptions: Security, Channel technology, Large system integration
Software Evolution Management Principles – Lifecycle Management

Phase 1: New Ideas
Phase 2: Technology
Phase 3: Products
Phase 4: Market
Phase 5: Domination
Phase 6: Phaseout
Software Evolution Management Principles

- **Adequate make or buy decisions**
  - Purchase Non bank-specific applications and technology infrastructure
  - Development activity should be concentrated on core business functions
  - ERP systems/Standard Software will pick up bank functionality

- Business processes have to be adapted to given processes of bought applications
- A highly developed integration architecture is key for successful standard software projects
Software Evolution Management Principles – Focus on Core Business

**Block A**

**Infrastructure**
- only buy for new components
- forced replacement of in-house infrastructure components
- focus on integration of best-of-breed market standards
- extend role of infrastructure into applications

**Block B**

**Non bank-specific applications**
- extend role of ERP packages
- concentrate on few software vendors for smooth integration
- follower strategy, low risk
- buy-only for new applications
- insulate bank-specific part

→ basically same application useful e.g. to Nestlé

**Block C**

**Bank-specific applications**
- build on higher level infrastructure
- componentize system
- individual make-buy decisions when renewing parts of application portfolio
- be launch customer if beneficial to overall strategy
Software Evolution Management Principles

- **Make it as simple as possible but not simpler**
  - Choose the simplest solution that fulfills the requirements
  - No “overengineering”

- **Design for low-cost production and simple maintainability**
  - the cumulated maintenance and production cost of an application by far exceed its development cost
Software Evolution Management Principles – Design for Maintenance

Total cost: development + production

- Development cost of good design
- Development cost of bad design or overengineering
- Production cost of good design
- Production cost of bad design or overengineering

Initial Development

Lifespan of application / infrastructure

Total cost of good design

Total cost of bad design or overengineering
Software Evolution Management
Questions & Answers
Practical application:

The Cloud
Cloud Computing –
Typical Platform Offerings

Applications

Application Platform (AP)

DB Hosting Platform

Compute Hosting Platform (CHP)

Hardware

Software as a Service (SaaS)

Platform as a Service (PaaS)

Infrastructure as a Service (IaaS)

Practical Experience:

SharePoint
Microsoft

JAP*
Java EE
WebLogic

DWH*
Oracle

DHP*
Oracle, SQLServer, Sybase

Mainframe*
IBM

CHP*
Linux (RedHat)
Windows

* Example Platforms
Which proportion of the applications in a large bank is easily «cloudable»?
What properties of software are required for migration into the cloud?
Cloud Design Principles 1

• **Location independence.** Images and application components within images must be independent of the location (region, site, network zone, test/prod environment) they are deployed in

• **Placement independence.** Placement of images guided by capacity management under the condition that affinities and anti-affinities are adhered to; images must be independent of the placement on actual servers

• **Cloning.** Images created via cloning of templates as much as possible: CHP* build with optional layer 2 components and application components

• **Dynamic in-place configuration.** Images' final setup (standalone and within blueprint) happens in chosen environment dynamically, e.g. at first boot time or at runtime when first used. Use as few central, active administration tools as possible

• Only *horizontal scalability* supported for simplified capacity management and elasticity

• **Capacity is allocated in fixed-sized chunks** on shared servers based on memory requirements. No over-provisioning. Use inactivity timeout for single image passivation and entire blueprint deletion in development/test environments.

* CHP = Compute Hosting Platform (see p. 9)
Cloud Design Principles 2

• **Eco System Automation with Service APIs.** Eco Systems must be configured automatically (CMDB, monitoring, authentication/authorization, network, load-balancers, firewalls, etc.)

• **High degree of automation** of operation processes

• **No client login** on provided images; necessary functionality supported via service calls

• **Infrastructure Service APIs.** Infrastructure implementation (eco systems or platform services) is hidden behind well defined infrastructure service APIs towards application developers as well as between infrastructure components. Provide for abstraction (allow implementation to change independently) and for automation (programmability, e.g. for automated CMDB integration in platform services or for automated testing and continuous integration in development)

• **Avoid vendor dependencies** as much as possible; if necessary hide behind abstraction layer
Cloud is *more than virtualization!*

Impact for Application Development

<table>
<thead>
<tr>
<th>Cloud Aspect</th>
<th>Paradigm Shift for AD</th>
</tr>
</thead>
</table>
| Resource abstraction towards clients (compute, storage, and network resources) | - Order capacity instead of HW  
- Choose from simple, standard options  
- Make no assumptions about placement (e.g. host names)                                                                                     |
| Rapid provisioning with self-service                      | - Test early, test often, explore  
- Test individually in entire application context  
- Rapid prototyping early business feedback                                                                                                    |
| Reproducible provisioning, configuration & deployment (persistent specifications with infrastructure service APIs) | - fully automatable test cycles (provision & build test env., run tests, decommission test env.)  
- quickly reproduce production problems in test environments                                                                                   |
| Rental model (pay as you go)                              | - Significantly lower entry cost (start small and quick)  
- Order and pay only what you need  
- Return what you do not need anymore                                                                                                          |
| Elasticity (grow and shrink capacity on demand)           | - Horizontal scalability  
- Statelessness  
- Fast startup, graceful shutdown of components                                                                                                  |
Software for the cloud
Questions & Answers
Appendix

Literature

S. Murer/B. Bonati/F. Furrer;
Managed Evolution: A Strategy for Very Large Information Systems,
Springer 2010

C. Worms, P. Schnorf, C. Hagen;
SOA and Cloud – Experiences from a large Enterprise
Key note of MESOCA 2012