Web-ifying Legacy Applications: Welcome to the 21st Century

David Moskowitz
Productivity Solutions, Inc.
Agenda

- The problem
- The target
- The challenges
- The solution
- Questions and answers
# The Problem

- Multiple "systems" to store organization data
  - Data in some systems redundant

- Systems did not talk to each other
  - Data had to be manually moved between them
    - Including: print a report to view/re-enter the data

- Different languages (or dialects)
  - Adabase (Natural), Oracle, DB2, FoxPro, dBase, Access

- Multiple levels and skills for support
The Rest of the Problem

- Each system (and corresponding staff) stove-piped.
- A single requested business change could impact multiple systems and take months to implement and coordinate.
- Cost to maintain the systems: exorbitant!!!
- Information and competitiveness lost
- No real ability to respond to change
The Target

- Integrate the systems
- Clean up the data
  - Identify & reduce redundancy
  - "Active" records versus warehouse
- Add functionality
- Make support (development and help desk) consistent and easy
- Single unified view of their "world"
The Steps — Vowels

- **Account** — explanation of the purposes (how the pieces are supposed to fit)
- **Examine** — what's where, schemas, organization (also identify redundancy)
- **Inventory** — you need to know where you are, before you try to go someplace else
- **Operation** — how does the data get into and out of the system. How will it...?
- **Use** — how is the data used? How will it...?

Also... when, where and expectations
What Doesn't Work…

- Attempt to define (all) requirements
  - Attempt to define requirements too soon
- Failure to prioritize requirements
  - Require everything as a "must-have"
- Assumptions without prototyping
  - Assumptions without known expectations
- No (or minimal) configuration management
  - Even experienced developers need it
What Doesn't Work...

- At the beginning...
  - Failure to obtain a commitment from end-user (customer) management for a real commitment for active involvement
  - Failure to establish a partnership between IT and the "sponsor"
- Failure to validate deliverables against their predecessors whenever there is a change
  - Failure to keep the main thing the main thing
What Doesn't Work...

- Bad idea in the first place:
  - Ambiguous or overambitious
  - No (or questionable) metrics
  - Wrong business drivers

- Overly restrictive or inadequate business cases, requirements, & specs can destroy a good idea

- Poor (sporatic/no) communication with users
  - Vanishing / disappearing users
    - Can we talk about outsourcing...???

- Scope creep

- Agile development when the project manager doesn't understand agile
The Kiss of Death

- No audit trail that demonstrates clear lines of communication with ownership accepted
  - Clear lines of communication enable information to flow effectively and efficiently
  - Ownership prevents confusion or denial over authority and responsibility
  - Both are essential to correct problems

- Missing unambiguous feedback that identifies both successes and misses
Solutions Steps

- Determine the business need...
  - ...and the business drivers

- Craft a conversion strategy
  - Understand the business drivers and the end user expectations
    - Don't assume: only users can supply expectations

- Standard for non-functional requirements

- Do the data model

- Determine the development process model
Determine Business Need

- **When do you do it?**
  - Before, during, ???
    - Baseline requirements at a high level

- **How do you REALLY do this?**
  - Stories, use-cases, formal requirements...??

- **How do you handle changing requirements?**
Craft a Conversion Strategy

- How many old systems will be converted?
- What data is in the old systems?
- How is the old data used?
- How will it be used, what is different in the new system (need, thneed, & expectations)
  - What will change (old data use)?
  - What new data (if any) will be added?
  - What new functionality will be added?
- Anticipate the inevitable
A Word About...

- New functionality
  - Haven't seen a legacy to Web conversion that didn't add capability / functionality
  - If they tell you there isn't any, do NOT believe it!
    - ...or be prepared for, "That's not what I meant."
Craft a Conversion Strategy

- Options: Full parallel operation, partial parallel, no parallel (just do it).
- Drivers: the business need, the data, the user experience
- What type of reliability?
  - Handle 1 million transactions reliably or...
  - Handle 1 transaction accurately and repeat it 1 million times for each "user"
- What type of risk mitigation...?
Conversion Strategy

- **Just do it**
  - Code and fix as you go

- **Plan-driven**
  - Predictive – up-front design
  - Delivers the original functionality
    - Modifications add to the cost
  - Can be CMM (Capability Maturity Model)

- **Agile**
  - Adaptive – evolving design
  - Delivers the needed functionality
Subtle Point

Understand the difference between the **requested** system and the **needed** system.
It's Easy…

- NUTS!

- Stuff they think is easy very often isn't
  - ... and stuff they avoid is actually easy to do.

- Quantum nature of development
  - Once you've made a "path" decision, it dictates what is easy and what is hard
    - Most folks don't see it that way, but...
  - Take it one step further... it's too easy to get into the hammer and the nail syndrome
Parallel Operation?

- When does it make sense to use parallel?
- When does it make sense to run in full parallel (complete old and complete new)?
- When does it make sense to run some processes in parallel operation?
- When does it make sense to, "just do it?"
Parallel Operation

- Two types of conversions
  - Single stand-alone system
  - Integrating multiple systems
- Easier to do full parallel operation for the stand-alone system
  - That doesn't mean parallel operation should be ignored for integration
- Big-bang (just do it) isn't the best alternative
Non-functional Requirements

- Replacing existing systems sets a standard
- Reliability
  - 1 million transactions or 1 transaction that can be processed 1 million times.
    - This isn't a "load" issue – at least not yet
    - It is a quality metric!!! How?
- Throughput (and response time)
  - Is the old standard sufficient
Crafting the Data Model

- It's in the business drivers
- Talk to customers!!!
  - Determine how they use the current system
    - Have developers sit with users and actually watch
  - Determine what / how they expect to change
    - What do they expect will change: process???
- The data model is based upon user experience and user expectations
  - The real driver is: Business Need!!!
Crafting the Data Model

- Ultimately, this controls everything else
  - How the users use the system
  - How the developers build the system
- Entity-relationship or Object Model
  - Most legacy conversion should use ER
- Input derived from planning and analysis
- Outputs: ER diagram & data document
  - Data doc describes the details of data objects, relationships, and rules required by the database
The Data Model

- Most labor intensive / time consuming part of the development / conversion process
- Goal: assure data objects are completely & accurately represented
  - Data model expressed in easily understood notations and natural language
  - Reviewed and validated by end-users
  - Sufficient detail to be usable by database developers as a blueprint to define schema
Agile or Traditional (Process)

- If it's mission critical, this might not be the time to "bet the farm" on agile
  - Need a supportive organization
  - Agile is difficult if traditional reporting required
  - Agile can work, provided everyone buys in

- Quality metrics
  - Defect count
  - Specification & process compliance
  - Customer satisfaction
Agile and Legacy Conversion

- There usually will be conflicts between agile and traditional project management.
  - **Agile**: adaptive scheduling
  - **Traditional**: predictive scheduling (plan-driven)
- Plan-driven development typically focuses on both fully developed architecture and design
- Agile development focuses on a framework architecture and simple design that emerges during development
Technical Approaches

- Two conversions: "Systems" & data
- Keep old data for some period
  - Use JCA for non-relational systems
  - JDBC for relational data
  - Screen scrape???
  - Plug-able wrappers
- On-the-fly conversion as needed
  - With some parallel operation
  - Seamless access to old and new required!
People Approaches

- If all you have is legacy developers...
  - Everything in `main(...)` isn't a good idea
  - Teach them about objects
    - Use Java in-context
    - Don't start with EJB
    - Consider appropriate frameworks (hibernate, spring...)
      - Remember the driver is business need, not tools

- You need an object-oriented architect
  - Architecture should capture the big picture vision
    - Defines a thought frame(work)
Don't Assume

- Yes, CICS is event driven...
  - ...that doesn't mean CICS developers really understand objects, events, and behavior
- Just because they work on a system, don't assume they understand enterprise data
- Mainframe folks aren't going to be comfortable with toys
Best Practices

- Constant prototyping and testing
  - Prototype everything
  - Waiting to the end to test doesn't work
    - Teach JUnit and friends

- Get and keep users involved
  - Developers and users should be joined at the hip

- Don't define all requirements up front; allow them to evolve
  - Needed system versus requested system
Technical Best Practices

- Don't use SOAP over HTTP or XML between application layers
  - Do use SOAP at the edges – works best for integration
- Use accessor (get / set) methods
  - But not with distributed objects
  - Create a method that returns an entire object (or object set) don't use multiple gets
Best Practices

- You need to show business value to get approval for the project
  - Focus on business needs first, technology should be a distant second
- Most common business need addressed by Web services is **integration**
  - If you haven't considered it to this point, think about it now: a service-oriented approach works
  - But...
Web Services

- Don't do Web services for the sake of doing Web services
  - Needlessly adds costs for a distributed system
- If you don't need to externalize a specific interface, don't do it
  - If you don't have a need, turning it into a service is a waste of time (and money)
Web Services

- Point-to-point works best
  - But... with lots of interactions, the Java client gets very complex
  - Solution – refactor the code and separate the code that figures out how to use the plumbing
    - Consider an enterprise service bus
      - Similar in concept to the mediator design pattern

- Avoid spending time writing plumbing code
  - Most of it is going to be (should be) similar
Web Services

- Design the Web service to look like the underlying transaction
  - Don't use a "conversation" or "interactive" model
  - Use a forms-based approach

- SOA design patterns – determine granularity of services & business requirements
  - Adapters (for legacy code)
  - Component replacement (change internals)
  - Composition (BPEL et al)
  - Unified logical view (façade)
Summary Checklist

- If it connects IT more closely to the business, do it!
  - IT work in/with the business unit

- If it improves communication about IT, continue it!
  - Tough sell, internal marketing

- If it encourages good IT management practices, try it!
  - Hands-off style with smart, motivated staff
  - Small rapid deliverables; prototyping; constant user involvement
Questions ???

If you don’t ask, who will?

If not now, when?

There aren’t any dumb questions.

The only dumb question is the one not asked!
Thank You

For more information:

David Moskowitz
Productivity Solutions, Inc.
147 Ashland Avenue
Bala Cynwyd, PA 19004
+1-610-664-7726
davidm2@usa.net