IT 4.0 – The Challenge and The Opportunity

Value and Data Driven, Query Centric, Functional, Self Service ...

Dave Thomas

www.davethomas.net

Bedarra Research Labs  YOW! / GOTO Conferences  DepthFirst Carleton University
Canada  Queensland University of Technology Australia

Lodestone Foundation
IT 4.0 Outline

Challenges
- IT 1.0 - 3.0 Rigidity and Lost Opportunity
- Business 4.0
- Technology 4.0

Opportunities
- Simplified Flow - Value Driven
- Accelerated Delivery
- Data Driven - Query Oriented
- New Modularity - Microservices and Actors
- Self Service - Ultimate Pairs
Jurassic Software? – Life in The Tar Bits

1.0 Mainframe with COBOL and 4GLs, Waterfall

2. PC and Unix – Desktop GUI, Client–Server, C/C++, Relational and SQL Stored Procedures

3. OO, 3 Tier, Java/C#, Middleware, App Server, Agile, Web 1.0
Escalating Costs of Ownership

- Recruiting Retaining Talent
- License Complexity Costs
- API Instability
- MORE Accidental Complexity
- License Expense Complexity
- MORE SW HW Platforms
- Increased Code Bloat
- Certification vs Development
- Maintenance vs Development
- Open Source Management
- MORE APIs
- MORE Languages
- MORE Tools
- Platform Framework Tool Churn
Business 4.0

- Innovation delivered through Agility (beyond Rigidity)
- Business Model Value Driven
- Data Driven Real-time Business
- Continuous Adaptation and Delivery
Better, Faster, Cheaper – A New Road?

We must improve our way software delivery to meet the challenge.

- **Focus on Value and Flow** – Target resources and innovations to where they will make a difference.

- **Refactor our organization** - to enable more concurrency and reduce cycle time without reducing quality. Leverage what works and not be constrained by current best practices. If it is slow it has to go!

- **Explore and Innovate** – we need to envision alternatives and evaluate them quickly before betting too much on any approach. We need to fail vast to maximize ROI and time.

- **Advance and Automate Development** – use alternative techniques to communicate, design, estimate, build, test and deploy.
Technology 4.0
Let the Hardware Do The Work!

$25,000 buys a computer 1 TB RAM with 40 TB disk and 32 cores!
$200 buys 1000 4 GB cpus on Amazon for 1 hour!

• Automated Build and Test is mandatory
• All interesting data is in memory! DB is an oxymoron
• Inexpensive Data Conversion/Translation
• Data Compression and Encryption is “free” on multi-core
• Speed and Memory enable Simpler Algorithms
• Enable End User Computing at Scale

If only the Software will let us have mechanical sympathy?
### So Many Languages to Choose From?

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td>C</td>
<td>17.246%</td>
<td>-2.58%</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
<td>Java</td>
<td>16.107%</td>
<td>-1.09%</td>
<td>A</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
<td>Objective-C</td>
<td>8.992%</td>
<td>-0.49%</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td></td>
<td>C++</td>
<td>8.664%</td>
<td>-0.60%</td>
<td>A</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>↑</td>
<td>PHP</td>
<td>6.094%</td>
<td>+0.43%</td>
<td>A</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>↓</td>
<td>C#</td>
<td>5.718%</td>
<td>-0.81%</td>
<td>A</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td></td>
<td>(Visual) Basic</td>
<td>4.819%</td>
<td>-0.30%</td>
<td>A</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td></td>
<td>Python</td>
<td>3.107%</td>
<td>-0.79%</td>
<td>A</td>
</tr>
<tr>
<td>9</td>
<td>23</td>
<td>↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑</td>
<td>Transact-SQL</td>
<td>2.621%</td>
<td>+2.13%</td>
<td>A</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>↑</td>
<td>JavaScript</td>
<td>2.038%</td>
<td>+0.78%</td>
<td>A</td>
</tr>
<tr>
<td>11</td>
<td>18</td>
<td>↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑</td>
<td>Visual Basic .NET</td>
<td>1.933%</td>
<td>+1.33%</td>
<td>A</td>
</tr>
<tr>
<td>12</td>
<td>9</td>
<td>↓</td>
<td>Perl</td>
<td>1.607%</td>
<td>-0.52%</td>
<td>A</td>
</tr>
<tr>
<td>13</td>
<td>10</td>
<td>↓</td>
<td>Ruby</td>
<td>1.246%</td>
<td>-0.56%</td>
<td>A</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td></td>
<td>Pascal</td>
<td>0.753%</td>
<td>-0.09%</td>
<td>A</td>
</tr>
<tr>
<td>15</td>
<td>17</td>
<td>↑</td>
<td>PL/SQL</td>
<td>0.730%</td>
<td>+0.10%</td>
<td>A</td>
</tr>
<tr>
<td>16</td>
<td>13</td>
<td>↓</td>
<td>Lisp</td>
<td>0.725%</td>
<td>-0.22%</td>
<td>A</td>
</tr>
<tr>
<td>17</td>
<td>12</td>
<td>↓</td>
<td>Delphi/Object Pascal</td>
<td>0.701%</td>
<td>-0.40%</td>
<td>A</td>
</tr>
<tr>
<td>18</td>
<td>53</td>
<td>↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑</td>
<td>Groovy</td>
<td>0.658%</td>
<td>+0.53%</td>
<td>B</td>
</tr>
<tr>
<td>19</td>
<td>19</td>
<td></td>
<td>MATLAB</td>
<td>0.614%</td>
<td>+0.02%</td>
<td>B</td>
</tr>
<tr>
<td>20</td>
<td>26</td>
<td>↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑</td>
<td>COBOL</td>
<td>0.599%</td>
<td>+0.15%</td>
<td>B</td>
</tr>
</tbody>
</table>
How many Classes & Packages in Java?

**Total No of Classes**

Java1.02 → 250  Java1.1 → 500 Java(2-4) → 2300  Java5 → 3500

Java6 → 3793  Java7 → 4024  Java8 → ???

**Total No of Packages**

Java6 → 203  Java7 → 209  Java8 → ???

How many frameworks? ....

How many serialization formats? ...
Of Course you know 4.0 Alien Languages!

Languages

Racket Event Actors JRuby

Programming Languages

Java

Concurrent Collections Dart Declarative

ClojureScript CoffeeScript JavaScript Processing C++11 Clojure Agents Scheme Immutable Vector Array

LGuAMLIN Functional LINQ LIVELY

Rails Scala LINQ
Lost in the Big Data Galaxy

Big Data

Visualization RDF Column Store
DFS Triple Store MapR
Big Graph Statistical Sampling
Unstructured Hive
Hadoop Streaming
Zettabytes
Collaborative Analytics
Machine Learning
Semantic Web
SPARQL Data Scientist
Hadoop
Web Next? Open Force versus Walled App Empires
API “Field of Dreams”

Give them APIs and pray the Applications will...

ProgrammableWeb API Growth 2005 - 2013

Complexity of Technical Alternatives!
Simplicity! - The Road Not Taken?

If you can't explain it **simply**, you don't understand it well enough.

– Albert Einstein
Lean – Ouch! Thinking and Leadership?

Value Driven Flow

The Principles of Product Development
FLOW
Second Generation Lean Product Development

DONALD G. REINERTSEN
Lean Software Organization
Technical Ladders, Playing Coaches and Communities

Management Ladder
- Executive
- Technical Director
- Team Leader
- Individual Contributors...

Technical Ladder
- Outstanding Contributor
- Principal Engineer
- Distinguished Engineer

Align Compensation with Work Products and Goals

©2011 Bedarra Research Labs. All rights reserved.
Projects are from Mars - Products are from Venus

Projects are about apps and resource; Products are about Features

IT sees feature teams == project teams; Products require component and feature teams;

Projects App is all we need, reuse is optional and unlikely; Product Architecture and Reuse is Essential

Project focus is my App ; Products all about Interfaces, Dependencies

Project apps can evolve : Products need Design to Last

Projects resources pooled ; Products own their $, backlogs, teams ...

Projects delivered incrementally ; Products need a schedule and deliverables for major functionality
## Eliminate Projects! – Manage to Your Capacity

<table>
<thead>
<tr>
<th>Program</th>
<th>Feature (F1 to F18)</th>
<th>Team (Blue, Red, White, Orange, Yellow, Green, Purple)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>F1, F2</td>
<td>Blue</td>
</tr>
<tr>
<td></td>
<td>F3, F4, F5, F6</td>
<td>Red</td>
</tr>
<tr>
<td>P2</td>
<td>F7</td>
<td>Yellow</td>
</tr>
<tr>
<td></td>
<td>F8, F9, F10, F11</td>
<td>Green</td>
</tr>
<tr>
<td>P3</td>
<td>F12, F13, F14, F15</td>
<td>White</td>
</tr>
<tr>
<td></td>
<td>F16, F17, F18</td>
<td>Orange</td>
</tr>
</tbody>
</table>

### Company Backlog
- Program Backlogs
- Team Backlogs

### Feature Backlog
- **Program**: Medical Imaging
- **Feature**: MRI Mechanical Control
- **Epic**: Table Movement
- **Story**: Horizontal Movement
- **Task**: Position w/ Joy Sticks
Rapid Application Development Process
The More Things Change? ... IT Stays the Same!

Create – Read – Update – Delete (CRUD)

Business Analytics, Mobile, Cloud Computing, Multi-player Games ...
IT is still! = CRUD + Workflow + Data

Input and Outputs - Forms, Reports and Data Transformations

Workflow – Form Flow and Transformation (Boxes and Arrows)

Event/Actions – Event-State Tables/Sourcing

Rules/Actions – Decision Tables (Rule Engines)

Complex Calculations – Constraints/Dataflow (Spreadsheets)

Data and Relationships – Data Models (ER= OMT)

Objects are in the technology not in the domain!
Data Intensive Computing

All roads lead to some form of Functional CRUD!

Applied Functional Programming (aka Super CRUD)

SQL + Functions + Streams – e.g. Greenplum ...

NoSQL Databases – Dictionaries on Steroids (Big Table, CouchDB...)

Map Reduce Data Parallel

Hybrid JVM, CLR/LINQ functional languages F#, Scala, Clojure

Vector Functional Programming

Reactive Programming Rx...
Faster, Better, Cheaper

How can we reduce the software cycle time?

Needs

Solution

Development
Answer - Ship Less Code! Make it Easier to Change!

KLOCS (1000 lines of code) Kill! => Be More Expressive

Dependencies Strangle => Micro Service Architecture

Avoid Frameworks inject dependencies => Less Objects, more FP?

Data Driven Always More Flexible Than Code Driven => Descriptions, Tables and Queries

Automate Everything!

Enable DIY Programming => Let the business think and compute
Automate Everything!

- Simplified Requirements Capture - Delta analysis
- Automated Testing
- Automated Build
- Automated Deploy
- Automated Test Construction?
  - Randomized Testing
- Automated Program Construction?
  - Programming By Example
  - Machine Learning
Use Less Objects and Less Code!

Object Refactoring harder than Changing Data and Functions
Many Apps have few if any domain objects!

Table Driven Programming

<table>
<thead>
<tr>
<th>Rules</th>
<th>Decision Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calculation</td>
<td>Spreadsheet</td>
</tr>
<tr>
<td>Data Validation</td>
<td>Domain and Range Table</td>
</tr>
<tr>
<td>Mapping</td>
<td>Lookup Table</td>
</tr>
<tr>
<td>Flow</td>
<td>Data, Work Flow, Message</td>
</tr>
<tr>
<td>Events, Matches</td>
<td>State Table</td>
</tr>
<tr>
<td>Process, Reports</td>
<td>Input-Output Table</td>
</tr>
<tr>
<td>Acceptance Criteria</td>
<td>BDD</td>
</tr>
<tr>
<td>Domain Models</td>
<td>Entity-Attribute Dictionary</td>
</tr>
</tbody>
</table>
Table Oriented Programming

A picture is 1000 words, a table 200 and a diagram 50

Advantages

• Easily understood by Business, BA, Dev and QA
• Easy to create, refactor and extend using Excel
• Modularity through structured tables
• Consistency /Completeness Checking
• Easy to version and Diff
• Efficient Automated Data Driven implementation
• Data Driven means changes can be “hot deployed” to a running application

Applications

Insurance, Banking, Taxation, Healthcare, ATC, Real-time...

TOP Programmers Wanted!

©2009 Bedarra Research Labs. All rights reserved.
Simplify - Reduce Integration Time and $$$

• ATOM/RSS feeds on our legacy/partner systems – journal files, events ...

• Use ODBC as a simple interface to complex server systems

• REST and JSONify your services, Provide a scriptable service, Use Self Described Data e.g. LinkedData

• Use a simple MashUp tool to deliver a integrated application view
Script to Save Time and $$$

• More and more applications are disposable at least in
  ▪ make it work and get it out there,
  ▪ scale it later if you need to
• Script Softly for productivity
  ▪ Ruby, Python, PHP, Groovy, JavaScript, Clojure...
• Leverage cloud services (map reduce, cloud DB..CRM)
• Leverage core internal and external services via REST/ODBC
The Emergence of A Simpler Application Infrastructure

- Examples - On Demand, Software As A Service such as Amazon S3, EC2, SimpleDB, Google App Engine, Sales Force ...
- Simpler limited “thin” service API (< 50 ) closer to underlying platform which provides support for scalable, distributed, secure computing
- Independence on mainstream vendor Underware and Middleware
- Google Linux, VMware Virtual Machine, MS Azure Hypervisor V

Application Development Benefits

- Small Service API (thin to none class library & frameworks)
- Limited Choice Reduces Decisions and Support
- Leverages Other Apps through Services
- Total App Responsibility from envisioning to production i.e. App Team caries the beeper
We all know that testing costs a lot and takes time, mocking is hard especially when working in a changing complex environment. Let's not bother!

What it takes

1. Modular micro service architecture
   - instant deployment and tear down
   - loose data coupling
   - well defined SLA

2. Simple Functionality in each Deployment

3. Stringent SLA Monitoring for Deviant

4. Let it fail architecture (Erlang versus exceptions)

5. Replication

Applications – Telecom, Finance, eCommerce …
What Is a Micro Service?

A service provides an interface to a specific subset of functionality/data in an enterprise.

- Versioned Services can be concurrently deployed to enable new and older apps.
- Services are not frameworks; they are components with value only interfaces.
- Services are realized as components with consumable APIs.
- One or more services can often be supported by a service team.
Technical Value Proposition

Services increase modularity, reduce coupling, increase technology and delivery choices.

▪ Services reduce large monolithic applications to a set of single function technology independent APIs which can be composed into business applications
▪ Services are loosely coupled hence can be incrementally be developed and deployed
▪ Services are easier to distribute, provision and monitor
▪ Services expose the tangible business architecture versus the internal technical applications architecture
▪ Services enable parallel Business App development and service definition and development (feature and component teams)
▪ Services are realized as components with consumable APIs.
▪ Services easier to deploy and test.
▪ One or more services and be owned and implemented by a team.
Enabling Loose Coupling

- All APIs are value based and where possible stateless
- Isolation of services in separate processes/machines
- Simple Pipes and Filters when possible
- RSS/ATOM feeds from events/updates/logs
- Occasionally Disconnected – replication and sync; event source..
- Simple efficient implementations using co-routines..
- Orchestration/Composition using Scripting Process
  - Messaging.. Node.js, Erlang, Actors
- FP thinking encourages value orientation and composition
Service Architecture and Design

- Occasionally-Disconnected
- Micro-Services
- Interfaces
- Self-Described
- ServiceBus
- REST
- JSON
- Node.js
- Separation-of-concerns
- Functional-Web
- Actors
- Pico-container
- QuickCheck

Isolation

- Radomized-Testing
- SLA
- Orchestration
- Scripting
- HA
- PBE
- Let-It-fail
- Table-driven
- Messaging
- Eventual-Consistency
- SBE
- Replication
- ODBC
- Protocols
- Integration-patterns
- Strong-generic
- Coroutines
- Flow-Oriented
- API
- SOLID-principles
- Provisioning

- Erlang
- Lock-Free
- Component
- Data-driven
- RSS
- Service-Test
- Reactive
- Pipes
- Code-Deploy-Monitor

Micro Service - The Business Value Proposition

Business Apps can be more easily configured from Services

- Large Monolithic Systems are decomposed into simpler services which enable
- Services and their SLA published in a web catalog
- One or more services can be composed into an App with a tailored interface for a given market
- App delivery can be supported by simpler less technical end user tooling such as Wikis, Visual Connections
- Apps can combine services from insurance and banking etc.
- Bus Apps = Services (Features) + Services (Components)
DIY - Bus App Development

Do It Ourselves Programming – The Empowerment

Business Driven Development

- Enterprise Mashups – The Real SOA?
- Applications Assembled from Feeds and Services
Loose Coupling – Let’s Hope It Sticks This Time

Data Flow – Data Flow Computing ... Maxor FPGA DF
Structured Analysis and Design (SADT)
Unix Pipes and Filters
Flow Base Programming – J. Paul Morrison
Hewitt Actors
Spreadsheets
Visual Programming - Labview
Actors - Erlang
Query/Collection Oriented Programming
202x?

- Happy end users with DIY computing
- Best Practice = Lean + Real Models to Code By Business
We can learn from our Children!
“The future has already arrived. It's just not evenly distributed yet.”

William Gibson
Summary

- Focus on continuous delivery of value
- Maximize Flow
- Leadership and Skills Matter
- Favor targeted high value change over systemic change
- Build Products not Projects
- Respect the Individual and Organizational APIs
- Just Enough Design and Architecture
- Features and Components both essential
- Ensure every feature has an associated acceptance criteria
- Acceptance Tests >> API Test >> Unit Test
- Automate everything
- Use the right tool/practice for the right job

© 2012 Bedarra Research Labs. All rights reserved.