Modernizing RPG & 2E Apps Using Design Recovery

• Background
• Motivation
• Mechanics
• Methodology
• 80’s cross referencing tools
• Mid 90’s Reengineering Automation
  Graphical analysis & Documentation
• Early 00’s Model Based Extractions
  Data Model
• 2005 Model Based Extractions
  Business Rule Logic
• 2005 Model Based Extractions
  UML & DDL
• 2009 Database Modernization & Reengineering

Technological Milestones

Using an Engineered Approach
Passive
- Cross-reference
- Impact Analysis
- Structure Chart Diagrams
- Data Flow Diagrams
- System Diagram
- Data Flow Chart
- Entity Relationship Diagram
- Source Analysis
- Program Structure Chart
- Area Flow Diagram
- Activity Diagram
- Use Case
- Class Diagram
- Business Logic Analysis

Active
- Field Expansion
- Unicode Conversion
- DDS to SQL Conversion
- RPG I/O Externalisation
- RPG Web-service Extraction
- Java Business Logic Classes
- EGL Business Logic Functions
- JSF Generation
- ORM Generation
- Db2 Web Query Meta Data
- Test Data Extraction
- Test Data Cleansing
- Production Data Quality

Reusing Your Assets

500+ Large Scale Reengineering Projects

Concepts Guide
Modernizing System i Applications Using Design Recovery
http://www.databorough.com/designrecovery

Cumulative Knowledge Base
R & D Partners

**Typical System i Application**

- Hundreds to thousands of RPG or COBOL Programs
- Aging Systems – 30 to 40 years old, heavily modified
- Monolithic Programming model
  - Screen
  - Database
  - Business Logic
- Little or no Documentation
- Original Designers no longer available
- Improve the responsiveness of development
- Improve long term maintainability
- Increased access to developers skilled in popular languages
- Increase availability of tools and application packages
- Improve application accessibility through open interfaces
- Improve portability of applications across platforms

**Why Modernize?**

**Developer Population?**
## Modern vs. Traditional Applications

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Modern Application</th>
<th>Traditional Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure</td>
<td>Modular, reusable, component based</td>
<td>Monolithic, intertwined, repetitive code across modules</td>
</tr>
<tr>
<td>Componentization</td>
<td>Functionally cohesive</td>
<td>Arbitrary</td>
</tr>
<tr>
<td>Performance</td>
<td>Highly scalable</td>
<td>Limited scalability</td>
</tr>
<tr>
<td>User Interface</td>
<td>Multiple</td>
<td>5250</td>
</tr>
<tr>
<td>Portability</td>
<td>Diverse platforms</td>
<td>System i and its predecessors</td>
</tr>
</tbody>
</table>

- Inherit Legacy Problems
- Not A Modern Application (Actually Legacy Emulation)
- **No-one recognizes the output**
- Virtually Un-maintainable
- Difficult To Retain Staff
- Poor Performance
- Difficult to Scale
- Not a Long Term Investment

## Code Conversions
Cost & Risk of Modernisation

RPG & COBOL Extraction for 20 years
X-2E: Extraction

Function Defs
- UI Designs
- Fields/Attributes
- Relative layouts
- Flow/Calls
- Data Model/Mapping

Business Logic
- Validation
- Calculations
- Secondary Reads
- Secondary Updates
- Batch Calls

Data Model
- Logical Files/Views
- Key Maps
- Relationships
- Special Fields

Cross-Ref Data
- All Objects
- All Sources
- All Languages
- All Variables

Relational Data Model
- Auto Extracted from Code/CA:2E
- Relational & Physical
- Date Fields

Business Rule Logic
- Validations/Calculations
- Summaries/Narratives/Annotations
- Indexed Database & Mapped to Functions
- CA:2E Action Diagram Logic Extraction

UI/Print Functions
- Screen Formats/Layouts/Fields/Files/Attributes
- Screen Actions Flow
- ORM Mapped to UI
- Conditional Field/UI/Flow Logic

Design Recovery
Design Recovery Documentation

Business Logic Analysis
Data Model Analysis

CA 2E data model as interactive diagram with relationship details.

UML Diagrams & UML Extensions
UML Diagrams S-Shots

- Scientific Forensics
- System wide perspective
- Visualization
- Non Expert Inclusion
- Explicit Information
- Quality of Communication

Summary Benefits of Analysis
Component Generation

What is MVC?

Model-View-Controller (MVC)

- **Model**: Underlying data objects and Business Logic
- **View**: The user interface
- **Controller**: Takes requests from View; passes commands to Model; chooses resulting View

MVC means keeping these functions separate.
Why MVC?

The shift to MVC
Brian Goldfarb, director of developer and user experience platforms at Microsoft.

"In the same vein, Microsoft has recognized the importance of the MVC pattern to developers, and it views it as a proven, robust way of making applications, The needs of customers have changed now."

- Reduce Maintainable Code Base
- Agile Maintenance
- Consistent Architecture
- Lower Training Costs
- Optimise Performance
- Portability & Scalability
- Modular separation of function
- Easier to maintain
- Data Integrity

Generating Application
• Each Screen Format has business logic indexed
• UI Events Removed from code
• Indicators and UI messaging converted to standard messaging
• I/O Standardised into generic components in Java/C#
• Procedural Code is converted to event driven code
• Standardised Parameters and API’s
• Variables Rationalised back to DB Fields if Possible
• Each Screen Format converted to JSF
• Java/EGL Controller Bean/Handler per screen
• Uses a open source/Net application framework as necessary or useful

How is design Re-factored?

Web - JSF JEE MVC
Project Methodology

- Project Discovery
- Proof of Concept
  - Migration of a defined subset of functionality to primary desired variant. Customer can then inspect and benchmark actual migrated code.
- Model Preparation
  - Complete documentation
  - Application Subdivision
  - Model/Application clean-up/refactor
- Test Case Development
- Migration
- Training
- Global & Specific Customization

On Going Changes during Project

- High Level Automation minimises impact of change
- Precise impact analysis upfront in tooling
- Last minute input
Testing

- Distributed and so focussed and more manageable
- Consistency – code is consistent so results are more predictable
- Semi automated – OO and Event code model allows for automated unit testing

Future Maintenance

- Potential for Automation – Consistent structure
- Distributed Architecture/Modular/Known (you know where to make the change or enhancement)
- Object Oriented - reuse
- Semi automated – generation from models
Stability & performance

- Distributed Architecture
- Modular Scalability
- Modern frameworks – Hibernate/Spring/nHibernate
- Automated benchmarking

Design Reuse Options

Recovered Design

- UML
- XML
- DDL

- Documentation
  - X-Analysis
  - Word/PDF/EXCEL

- New IDE as MVC
  - Java/C#/EGL
Collaborative Modernization ...

- Inherit Valuable Assets
- Exact and Accurate Knowledge Transfer
- Lower Risk
- Re-Use Existing Skills
- Configurable
- Gradual, Granular & Scalable
- Long term result - **maintainable**

Why Use Design Recovery?
Questions & Answers

Thank You

Thank you for viewing our presentation
For more information or to book a demo or trial, please contact us below:
info@databorough.com
www.databorough.com
North America: 705 458-8672
Europe: +44 1932 848564

Danke

Obrigado

Grazie

Merci

Grazias

Gracias

Thank you
Refactoring Legacy Applications

Introduction

Defining the Problem

Resolving the problem

Case Studies

Summary

Agenda
Concepts Guide
Modernizing System i Applications Using Design Recovery
http://www.databorough.com/designrecovery

Cumulative Knowledge Base

Mainstream
• 80’s cross referencing tools

• Mid 90’s Reengineering Automation
  Graphical analysis & Documentation

• Early 00’s Model Based Extractions
  Data Model

• 2005 Model Based Extractions
  Business Rule Logic

• 2005 Model Based Extractions
  UML & DDL

• 2009 Database Modernization & Reengineering

Technological Milestones

• Application Architecture

• Code Complexity & Growth

• Technology Evolution & Standards

• Knowledge Dispersion & Loss

Defining The problem
• Monolithic
  - Screen handling
  - Database I/O
  - Business Logic/Conditional Flow

• Inheritance & Erosion
  - Legacy - S36/S38 int/multi-frm/mbr files
  - ERP
  - Acquisition & growth
  - Regulatory

**Application Architecture**

• In-source, Outsource, Offshore
• Interfacing & Integration
• Regulatory Demands
• Competitive Business Pressures

**Code Complexity & Growth**

Based on complexity measurements such as the number of highly independent paths through a program's source code, do you consider your organization's code to be more complex than in the past few years?
Technology Evolution & Standards

Typical System i Application

- Hundreds to thousands of RPG or COBOL Programs
- Aging Systems – 30 to 40 years old, heavily modified
- Monolithic Programming model
  - Screen
  - Database
  - Business Logic
- Little or no Documentation
- Original Designers no longer available
• Application Discovery
• Refactor
• Automation
• Quick Wins & Rewrites

Resolving The problem

• Documentation
• Visual Analysis
• Functional Subdivision
• Communication
• Validation

Application Discovery
Documentation

Documentation – Business Rules
Visual Analysis – Data Model, OWU and field analysis

Visual Analysis – Deconstructing UI Layer
Basic Code Refactor
- Create Subroutines and in the process, elimination of GOTO / CAB / TAG statements.
- Replacement of left hand indicators used to control program logic flow with IF, THEN, ELSE type logic statements.
- Removal of primary / secondary file structures – eliminate use of RPG cycle
- Inclusion of D-Spec statements as created by ILE conversion process
- Extraction of messages to a message file format
- Externalization of inline tables (compile time tables/arrays)
- Replacement of ADD, SUB, MULT, DIV with EVAL whenever practical
- Removal (and archival) of commented out code

Database Refactor
- DDS to SQL
- Field Expansions
- UNICODE

Advanced
- I/O Externalization
- Subroutine to Procedures
- Business Logic Externalization

Refactor

Manual Approach
- Human Error
- Freeze Development
- Single Shot
- More Testing
- Labor Intensive

Engineered Approach
- One small change per pass
- Design Flexibility
- Repeatable
- Last Minute Changes
- Re-Usable

Automation
### Passive
- Cross-reference
- Impact Analysis
- Structure Chart Diagrams
- Data Flow Diagrams
- System Diagram
- Data Flow Chart
- Entity Relationship Diagram
- Source Analysis
- Program Structure Chart
- Area Flow Diagram
- Activity Diagram
- Use Case
- Class Diagram
- Business Logic Analysis

### Active
- Field Expansion
- Unicode Conversion
- DDS to SQL Conversion
- RPG I/O Externalisation
- RPG Web-service Extraction
- Java Business Logic Classes
- EGL Business Logic Functions
- JSF Generation
- ORM Generation
- Db2 Web Query Meta Data
- Test Data Extraction
- Test Data Cleansing
- Production Data Quality

---

### Extracted Repository

#### UI Designs & Logic
- Business Rule Logic
- Data Model
- Cross-Reference Data

---

### Reusing Your Assets

---

### Quick Wins
- Documentation
- DDS – SQL
- Field Expansion
- UNICODE
- RPGIV
- I/O Externalization

### Rewrites
- ILE
- SOA
- Web

---

4/28/2010
- 600 Plus projects
- Multi-Level Impact Analysis
- Case/Type Search Algorithms
- Case/Type Replacement Algorithms
- Exception Reporting

**Case Studies — Y2K/Field Expansion/Euro/UNICODE**

- Worldwide Insurance Company
- Million plus lines of code
- Integration with new package solution
- Need to use/update two sets of masters/transactions
- Automatically externalized RPG i/o into procedures/modules
- Change RLA into Callp’s

**Case Studies — Externalize i/o**
- Large ERP system
- 55 Million Lines of code
- CA 2E & RPG
- Transforming into new application framework
- Auto Extract Business Logic and Data Model
- Generate documentation and XMI & DDL
- Import into new frameworks/tooling (Borland)

**Case Studies - Design Extraction**

- Large Financial Institution (Texas)
- Large database with heavy batch I/O processes
- DDS database
- Rewriting core processes
- Auto-Converted DD S to DDL and added security/constraint logic to DB

**Case Studies - Database Modernization**
• Document
• Analyze
• Design & Plan
• Automation
• Rewrites

Summary

• Accurately Measuring Impact of change
  accurate, current detailed, inferred and abstracted

• Consistent & Current Information Sharing
  “even the most experienced developers need to communicate effectively”

• Controlling Knowledge Loss
  systems become increasingly riskier and difficult to maintain

• Education & Inclusion
  visualizing application design broadens user base(QA/Testing/non-400)

• Design Recovery
  business rules, data models, process models, meta-data models

• Automated System Re-engineering
  Field expansion, UNICODE, refactoring, I/O Externalization, Java, .Net

Improving Development Quality

AND Reducing Cost
Collaborative Evolution...

- 24 Years i series experience
- Skills in RPG/Java/COBOL/.Net/EGL/PHP
- Experts in legacy application designs
- Large Scale Reengineering
- Large Projects
- Global presence and workload distribution
  - Asia Pacific
  - Europe
  - Americas
- Certified Consultants
- System i Specialization

Databorough
- “We reduced our impact analysis from 4 weeks to 11 seconds.” Mark Rinker – Mapics
- “We saved close to a million dollars on the first projects” Chris Nickchen – New Penn
- “the key to unlocking the promise of both [web 2.0 and SOA] lies in leveraging of legacy business logic.” Ted Tritchew – Infor
- “[this] has provided a fundamental capability to increase quality and timeliness in development and support.” Mike Quested - Fiserv
- “This helped us take complete control of our legacy applications, and has improved the quality of the development process through understanding and knowledge.” Julie Hollis – Davies Turner

**Customer comments:**

**Questions & Answers**