EBR Assessment & The Way Forward

Executive Summary

Business Assessment

- EBR is a very small project by volume and rather small by its functionality.
- It's already not fulfilling its business mission by skipping more records than it actually processes and putting the majority of the processing off to weekends.
- This small processing volume is supported by thousands of technical artifact and up to 15-20 people at any given time.
- The current technical and resource infrastructure is completely disproportionate to its functionality and volume and it's very costly.

Recommendation

- Consolidate and streamline the technical as well human infrastructure. This will enable you to achieve significant cost savings and yet still provide the same or more functionality to the end user.
- Have one EBR subject expert in each functional area like project manager, business analysis, testing, development etc.
- Let the experts do most of or all of the work related to EBR. Don't let dilettantes double in EBR, they will do more harm than good.
- Don't let subject experts leave unless you have a suitable replacement. Understand that turnover is very expensive. It takes 2-3 months or $20-30K to get a good-enough understanding of EBR to be able to contribute.
- On the technical side. Stop piling up poor code on EBR it will eventually break it. You can do this by acquiring a commercial system that will automate the data loading and data updates. This is 80-90% of the development in EBR. Get custom development done only on the core business functionality and get it done by technically highly competent people. Pay more if you have to it will save you money and headaches over the long-term.

Proposals For EBR Improvement

The best way to decrease costs in EBR is to reduce the programming effort required for new functionality as well as to consolidate the current artifacts into as few as possible or get rid of as many of them as possible completely.

You can do this by either

- buying an ETL-type system

  Upside: initially cheaper, faster, easier to implement
  Downside: may still require significant custom programming in the future

- developing a dynamic EBR that will be able to load data streams without additional custom development required most of the time

  Upside: may significantly reduce future development costs
  Downside: Initially more expensive, more difficult and risky, longer to achieve

Buy an ETL

There are many ETL-type commercial systems but they have more negatives than positives. They are expensive, they require expensive custom developers, they produce code that needs to be deployed and maintained.

I do have a powerful non-traditional ETL system that will enable you to load current and future data files with little to no programming effort. Any new data stream can be configured simply by setting up parameters in the system.

This ETL is inexpensive, doesn't require specialty (or any kind of) development and it doesn't produce code, it loads data.

The savings realized by adopting this ETL can be estimated to be (excluding the cost of the license):

- Effort required to load a single source file: 5-10 days of development at $100 per hour -> $4000-8000 (or more, e.g. Margin Lending)
- Effort to load a single source file by ETL: 0.5-1 day to setup the load at $100 per hour -> $400-800

If we take the best-case scenario for custom load which is 5 days and $4000 and the worst case scenario for ETL load of 1 day and $800 the result is **80% saving.**

80% saving is just on development, there are additional savings in testing (simplified) and maintenance (hardly any) that may reduce the overall cost by 90-95% over the life of the system.

Develop Dynamic EBR

In 6 months I can re-factor and/or redesign EBR to become a dynamic data processor:

A) I can develop a generic EBR data loader that will be able to load and extract any kind of text data without any development 95% of the time. There may be cases where some very special data problem may require custom coding which would typically be just a few line shell script that can be plugged into this ETL.

B) I can redactor the core processing GroupApplyPublish so that it is dynamic and doesn't require coding updates for each new stream.

These 2 efforts will result in:

A) The reduction of the future development costs by anywhere from 70-90%. It will also simplify testing and maintenance significantly reducing those costs by as a much as 60%.

B) Increase the performance of EBR so that it can process 100K delta records during weekdays

I can do this in parallel to any new development that will be going on at the same time.

Detailed Business Analysis

Let's start by looking at overall value that EBR offers. Since I have no specific knowledge about the various costs in EBR I will measure value by the ratio of manpower working on EBR to the amount of processing EBR does and compare it some other projects I was involved with.

I have worked with EBR for the past three months and came to understand it in detail what it does and how it does it. I dare to say I have the best technical understanding of anybody working on the project currently onshore or off.

EBR Volume

EBR is a (very) small volume ETL type system. It currently processes 5 data streams, out of those only 2 are daily streams, BCRMS, Global Book. 3 streams are processed on the weekend, 2 once a month, CGID, RBCII, and 1 every week, CNB.

It terms of pure numbers, we are roughly looking at

1. BCRMS – 600K records out of 2.7 million, the rest is discarded
2. Global Book – 100K
3. CGID – 100K
4. RBCII – 30K
5. CNB – 40K out of 373K, the rest is discarded

All in all less than 1 million records on a daily basis comprise the source data set and more on the weekend.

However EBR doesn't process the full data set. It only processes changed records, which is 3-7% of the daily volume or about 3-7K records.

This is a minuscule number yet supported by enormous infrastructure in comparison. Let's look at this infrastructure from 2 different points-of-view: Artifacts produced and Manpower working on EBR.

Technical Artifacts

Database

- Large number of data objects most of them due to poor or non-existing design.
- The infrastructure includes the main processing database EBR, the output database ESN and a staging database.
- EBR alone has over 200 tables and 150 stored procedures. With ESN and the staging database the overall object count comes probably close to 1000.
- All these data objects have had to be developed, tested and they have to be maintained. In

addition all these objects take up enormous amounts of space so much so that all databases were running out of available hardware space for expansion in the last month.
- Disk is cheap nowadays but its expensive to add it to the infrastructure.
- EBR is not only accumulating expenses directly it's actually pushing the cost up at other unrelated business groups at RBC such as networking and storage.

Business Value

Hundreds of unnecessary data objects consuming tens of gigabytes of unnecessary space requiring maintenance.

Code Base

- Hundreds of mostly shell scripts and maybe 100+ java classes. Each stream has 3-10 scripts, they all do very much the same thing the only difference being the file names and the data tables they support. Yet every time there is a new project or new data source the offshore development team copies and pastes a set of previous scripts and just changes the relevant details. As a consequence if there was a bug in one script now there are 2 bugs in 2 separate scripts. If and when the time comes to fix them it's going to be double the work and double the testing.
- There are at least 4 or more environment scripts alone in EBR. This has consequences.
- These scripts are used ad hoc in various places and cause havoc especially in testing. I saw whole days being wasted because 1 or 2 of these were not properly updated and the testing failed. This alone cost thousands of dollars of downtime in the three months I was here.
- In simple terms for 2 active daily data streams, one weekly stream and 2 monthly streams there is no need for hundreds of scripts. It's enormously costly not just in money already spent on developing it but the future maintenance.

Business Value

Hundreds of unnecessary scripts duplicating the same functionality over and over again, mostly poorly developed and hard to maintain.

Manpower

Business Analysis

- In my experience I have never seen this many business analysts on a single small system. Typically a single business analyst should be enough for something like this with perhaps a part-time backup, so perhaps 2 analysts to be on the safe side.
- When I worked at FundSERV there was a single business analyst for the whole FundSERV core processing. FundSERV processes mutual fund trades which are regulated by law and must be perfect according to the law. The trading has complex and lengthy rules and regulations

which the business analyst had to know in detail and he did.

- None of this holds for EBR. The end result is that not a single business analyst at EBR has detailed and thorough understanding of what EBR should do and what EBR actually does. It seemed like business analysts spent most of their time just trying to reconcile their “opinions” of what “EBR” does and what it should do.
- Another anecdote from my experience, in one of the recent EBR sub-projects business analysts produced a whole multi-page document describing what EBR “is doing” and how this should change. My development colleague was put on the project as a developer. His “analysis” showed that 90% of what the document described as “supposedly” happening in EBR actually didn't exist. It simply wasn't there, no functionality, no code, no data.

Business Value

A few hundred thousand dollars a year spent unnecessarily on business analysis in EBR producing misleading or incorrect artifacts/documents that cause more confusion and arguments among the EBR participants.

Testing

- I have never seen this many testers on a single small system before.
- Again let's compare this to FundSERV. FundSERV had 4 full-time testers for their full processing life-cycle, trading 8 billion dollars a day. EBR has this many or more.
- It's not just the numbers; their testing is not that good either. Numerous times a code tested and signed off on by the testing team failed when moved to production. In other words the testing itself was a failure because it didn't catch that.
- Testers should have better understanding than anybody else working on EBR to be able to test it effectively. That's not the case here. Most testers working on EBR testing have a very sketchy understanding of EBR. This is understandable for contractors being on the project for a few weeks but not for full-timers.
- Yet despite the manpower, testing is always behind. I finished my code development in November for performance improvements and I am still waiting in the middle of January for it to be tested. I offered to test it myself but was rebuffed by saying that it's the job of the testing team to test development code. I will long be gone before they get to my code if ever.

Business Value

A few hundred thousand dollars a year spent unnecessarily and ineffectively on testing in EBR that fails often and is always behind.

Development

I understand that the development in EBR is influenced by its past that goes back to 2004. But even taking that into account it still doesn't explain most of the current practices. I will only mention a handful of the most obvious ones.

Bad Code Replication

- It seems EBR project over the years has followed the opposite of what is one the basic

rules/mantras of programming “Don't repeat yourself”.

- The unwritten rule of EBR seems to be “Replicate what is already there as many times as necessary”. The end result is that the EBR code base has been degraded because more bad code was introduced by copying the original bad code.

Dead Code/Data Maintenance

- There are thousands of lines of code, whole scripts, tables and procedures that are no longer used. There are 2 whole data streams that are no longer processed (CIRMS, DEXIA) yet the full code for their support is still in production.

Low Quality Vendor Service

- Vendor offshore development team seems to be under the impression that they were hired as text editors rather than developers. 80-90% of their development is copying existing code and making just enough changes to accommodate their specific task.
- When vendor developers were told to follow certain basic practices, they mostly ignored it.
- When vendor developers were told to change their design and what to change it to, they usually changed a portion to what was asked and left the rest as before.
- There is no effective oversight of the overseas development team. Since the development team is not very good, it needs more than anything an effective management to mitigate its shortcomings. It simply doesn't work.
- Small simple request for code changes take a long time with them and hence are expensive. Case in point, Margin Lending. They were asked to leave out SRF_ID from Margin Lending data source. Their estimate was 3 days. I did it in less than an hour and included it in the testing release the same day.

Development Manager Role

- I have tried to stay away from touching on specific people in this report but in this case I have to address this specific role because of the obvious implication, development effort is poor because the manager is not good.
- I believe the development manager is the most competent person on the EBR project but is constantly constrained and undermined by the infrastructure, management, vendor and above all by not having the power and authority to enforce rules and practices as she sees fit.

Business Value

Good value in Toronto with competent developers. This is more than offset by barely competent offshore development. The offshore team produces bad code which the local team than manages. It should be the other way round.

Support

- I don't know how many people do production support for EBR but I do know there are several

and I know that they are not tasked with doing anything more than first-level support.
- While I worked at Morgan Stanley I did 24 by 7 support level 2 and 3 for 30 systems (yes 30!) most of them 5 to 100 times bigger in terms of code base and processing volume. I was part of a team of 8 and that was just my part-time role, I was also part-time developer.

Business Value

Hundreds of thousands of dollars a year spent unnecessarily on support that doesn't seem to do much else or be any good (lacking in basic command line skills).

Management

A couple of most obvious and important observations.

- The volume of the development being pushed on EBR in the last 3 months by management is out of whack with EBR infrastructure. EBR is at the breaking point, it's already burdened with a lot of bad code, yet more bad code is being pushed on it constantly. The current infrastructure (people, databases, hardware) cannot deal effectively with the volume of new request coming down.
- The management doesn't seem to recognize where the competence and value lie. They just terminated 2 developers in Toronto (myself included) who were “fixing” EBR while keeping vendor team which creates most of the most serious technical problems in EBR e.g. is continuously degrading EBR.

Technical Assessment of EBR

Infrastructure

- EBR infrastructure is by-and-large OK.
- However I would like to point out that it the fact that it uses Sybase as its main data-store causes a number of technical issues. Since this is a standalone DB with no external dependencies it would be worthwhile to consider switching to one of of the open source databases such as MySQL or PostgreSQL. In my experience MySQL is superior to Sybase in any terms I can think of including cost, performance, maintenance etc.

Tools

- Basic tools are lacking such as professional grade terminal; Putty is OK but there are better options out there that are quite inexpensive.
- There is no proper Sybase client software, not for developers. There are decent open source alternatives but they are not good enough for development to work efficiently with many Sybase servers and dozens of databases.

Architecture

There are 4 main functional components/stages in EBR:

1) Data entry into the system
2) Core business processing
3) Data output out of the system
4) Job scheduler directing and running the data streams.

This architecture is often not respected as illustrated below.

1) **All data streams bringing in data should go into data staging and they should be separate**. This should include primary data streams BCRMS, CNB as well as secondary, Newton, CDR.
That's not the case: NEWTON for example is inside BCRMS (already fixed in development) and CDR is in the output stage.

2) Core business processing. **All business logic should be at this step.**
Not the case as a lot of the business updates are done in the 3rd stage. The GroupApplyPublish process could use a rewrite but just fine-tuning 'Publish' might be enough.

3) **No business logic should be done here only output creation & formatting.**
Not the case; a lot of business updates done in the ESN population stage as well as classification, CDR LEI and others.

4) **No enterprise scheduler.**
Scheduling functionality is scattered throughout the code between shell and database and is nothing more than rudimentary. Initially the scheduling functionality was small so it might have made sense in not having a separate scheduler. However as the data streams multiply scheduling is becoming a must.

The reason why it's important to stick to the processing segregation as outlined above is that it's then easier to modularize and share functionality and code.

The current practice of ad-hoc tacking on various services at different stages of processing requires expensive custom programming and just as expensive maintenance.

Stage 1 and 3 are generic data processing tasks that should only require generic possibly commercial solutions.

Only stage 2 should require (expensive) custom programming.

Development

I will just mention the most basic and glaring issues from the perspective of programming practices and methods. I will not go into detail.

- Ancient technology including Java 1.6, EJB, SOAP, ksh.
- Inappropriate technology use.
  Shell scripting shouldn't be used as the main programming tool in EBR. Perl should be used instead because it is easier to debug, modularize, develop code in and with a plethora of free modules that provide any functionality one can possibly need, business or technical. Because it's a scripting language it doesn't require compiling and packaging which are expensive, time-consuming and error-prone.
- No programming standards.
- When basic programming standards are introduced they are not followed.
- Basic programming principles are completely ignored: modularization, code sharing, simple, maintainable design, effective simple coding
- Instead anti-programming principles are followed: code duplication, complex design for basic tasks, spaghetti code everywhere (shell scripts, Java files, stored procedures), proliferation of environment scripts at all the wrong places.
- No longer used code and data are still in production years after they stopped being used.
- Production data unavailable. This is a delta system that works by accumulating data changes over time. Not having data available every day makes it impossible to properly develop and test this system.