MICRO FOCUS
Leading the Evolution™

WHITE PAPER

# BUSINESS RULES AND GAP ANALYSIS

Discovery and management of business rules avoids business disruptions

WHITE PAPER

# BUSINESS RULES AND GAP ANALYSIS

## Business Situation

More than ever, today's organizations depend on application portfolios to automate core operations. These applications contain sophisticated networks of business logic that govern how business processes behave. As a result, the embedded logic represents competitive differentiators that must be preserved even as your business changes.

But your business doesn't stand still. It is constantly adapting to address new market pressures and opportunities. A merger with another organization may yield overlapping and expensive to maintain systems. Or, a move to a standard ERP system may require that the packaged application be customized to suit the specific needs of the business. Regardless, it is important that the organization recognizes the gaps between the current functionality and desired end state of its portfolio.

An effective means to address this challenge is to generate business intelligence on your application portfolios. This intelligence can help you to locate gaps between the reality of your application portfolio and your business goals. This paper will assess how to locate and account for these functionality gaps. It will also illustrate a methodology for realigning applications with business goals.

## Scenarios

### Mergers and Acquisitions

There is often significant overlap when two companies merge. Large portions of their respective application portfolios may perform duplicate functions. For instance, two merged banks may both have similar corebanking and cash management applications. These systems will have roughly the same functionality. Yet, continuing to manage parallel systems is costly and unnecessary. As a result, management many opt to:

- Retain both applications. This may happen, for instance, if the two cover different geographical areas, client bases, or business areas.
- Completely eliminate one and keep the other (with the option of moving the data from the one that is abandoned to the one that is kept).
- Keep one of the applications, but enhance it, adding some functionality from the one that is abandoned.

To reach a decision, it is important to understand the extent of the three zones: "A only", "B only", and "Common". For instance, if from the point of view of the acquiring company the "A only" zone is larger and more relevant to the business than the "B only" zone, while the "Common" zone is large, the decision would probably be to keep Application A.

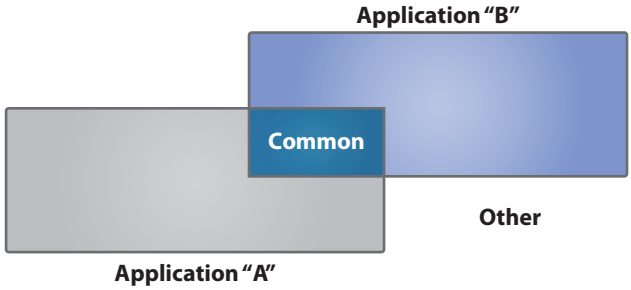



*Figure 1: Business functionality may be shared or distinct, leading to gaps between applications*

If the "B only" zone contains some functionality that cannot be abandoned, a special effort may be needed to port it to Application A. If the "Common" zone is very small in comparison with the "only" zones, a merger of functionality may be infeasible, and it may be advisable to replace both applications A and B with a new one that supersedes both.

### Purchasing an Outside Application Package

In this scenario, a company decides to replace an existing application. The decision may be driven by a number of factors, including the fact that its own application is functionally obsolete. For instance, an insurer may have long relied on an in-house claims processing system. But over time, it realizes that processing claims is not a core differentiator (versus, say, providing low-cost policies). As a result, it may opt to allocate resources away from the development of the in-house system. There are two possibilities:

- Develop a replacement in-house.
- Purchase an application package from an outside vendor.

To choose between the two, management needs to understand what functionality the outside package brings. Performing gap analysis between the legacy application and the outside package may reveal some important facts that may lead to a decision.

| Case | Fact | Decision |
|---|---|---|
| 1 | Outside package contains most, if not all, of the functionality of the legacy application | Adopt outside package |
| 2 | Outside package contains almost none of the legacy application functionality | Maintain / develop application in-house |
| 3 | Outside package contains significant functionality that is in common with the legacy application, and much more. However, there are some functionality aspects in the legacy application that are essential for the business, but are missing in the outside package | Adopt outside package and customize or extend |

In Case 3, the gap analysis is useful not only for the decision, but also as a practical way to measure the degree of customization or extension required. It may result in a list of concrete features that need to be created by customization or enhancement.

### Upgrade to a New Release

A company may have purchased and implemented an application package from an outside vendor. At the time of the purchase, the package was on release "N". Because of internal requirements, the package was heavily customized and enhanced. After expending this effort, it did not make sense to take release N+1 or N+2 from the vendor. However, after a number of years, the application fell behind in technology and functionality and a major upgrade was needed.

The outside vendor is now on release N+2, which seems very attractive. The problem, however, is how to upgrade without losing the highly valuable and proven customization and enhancements. The situation is illustrated in Figure 2.

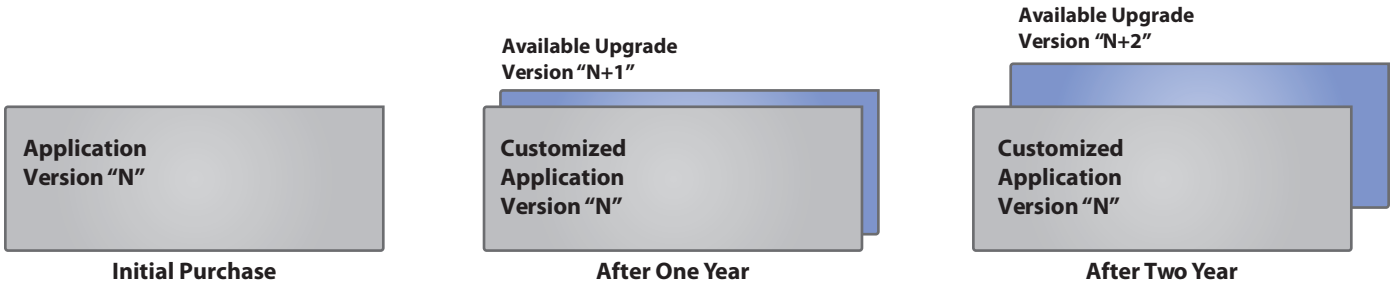



*Figure 2: Over time, customization to a packaged application will move it increasingly out of alignment with upgrade versions.*

In each of the scenarios we have described, it is important to understand the gaps between our current and desired realities. Now let's look at how gap analysis can be deployed to uncover and bridge the differences.

## Gap Analysis Aids Decision Making

To avoid the 'analysis paralysis' that can come from comparing gaps, we will want to automate as much of the process as possible. This requires that we form clear and measurable criteria. While applications have many aspects in many dimensions, we are particularly interested in comparing their functionality. More precisely, as we saw in Figure 1, we are interested in finding what is common between A and B – what is in A but not in B, and what is in B but not in A. This is what we call 'gap analysis'.

### Measurements

The demarcation between the three zones "A", "B", and "Common" may not always be so clear. It is possible, for instance, that two tables belonging to the two applications are almost identical. Perhaps they differ slightly by name and by one or two columns. Where should we then place them: in the "Common" or in the "Only" zones? The answer depends on criteria that could differ from project to project. These criteria may be specified in terms of thresholds. For instance, one may declare that two tables are matched if they differ in less than 20 percent of their columns. In general, a matching index may be computed as:

> Table Matching Index = Number of matched columns / Number of unique columns in both tables

For instance, if the table CUSTOMER exists in both applications and in the first one it has the columns (FIRST_NAME, LAST_NAME, ADDRESS, DOB) and in the second (FIRST_NAME, LAST_NAME, ADDRESS, PHONE), then the matching index will be 4 non-matched columns / 5 unique columns = 0.8.

If the threshold for matching is selected as 0.5, then the two tables are matched. It is obvious that one may build other matching schemes and calculate the matching index in other ways.

A final matching index may be used to summarize the entire gap analysis between two applications, along the same path. In this case, we may define:

> Application Matching Index = Number of matched objects / Number of unique objects in both applications

Given that in many cases we have matched objects that fall in the Common zone but are not really identical, it is useful for the final report of the gap analysis to indicate the average degree of matching. One may state for instance that two applications have 80 percent common functionality, but this 'common' functionality has an average matching index of 70 percent.

### What to Compare

It is apparent that a functionality comparison between two applications cannot generally be accomplished at the code level. We say 'generally' because in some special cases it may be possible, specifically when the two applications represent a split on two separate branches of the same base code, as in the 'Upgrade to a new release' case above. In this simple case, one may proceed with the following steps:

1. Take an inventory of all objects that could be expressed in source code in both applications (programs, screens, table descriptions, etc.).
2. Compare using just the names and types of objects and create the initial 'Only A' and 'Only B' inventories.

3. For the 'Common' zone, compare the objects of same name and same type to determine if they are really identical. If not identical, compute the matching indexes of the matched objects. (For instance, the same program P may have 1,800 lines of code in Application A and 2,000 in Application B. The matching index cannot be more then 0.9.)

This methodology would give satisfactory results, provided that the branching of the two applications did not happen too far back in time. If they split long ago, it is possible that the differences grew so much that a code-to-code comparison is not relevant, as in the case where developers change program names.

Code comparisons are entirely impossible if the two applications are from two entirely different code bases. The only way to perform gap analysis is to move to a higher abstraction level. Choosing the right level of abstraction is key to a successful gap analysis.

Let's consider the diagram in Figure 3, which shows various levels of abstraction, and of course, one may refine it by adding new levels. Each level presents different opportunities for gap analysis.

### Code level

Code level consists of the actual artifacts of the application – programs, screen definitions, and table definitions. It includes their full specification in the form of program code, data definitions, screen definitions, etc. It could be used only for small variations of the application over relatively short periods of time (for instance, in the case of successive releases).

### Technical level

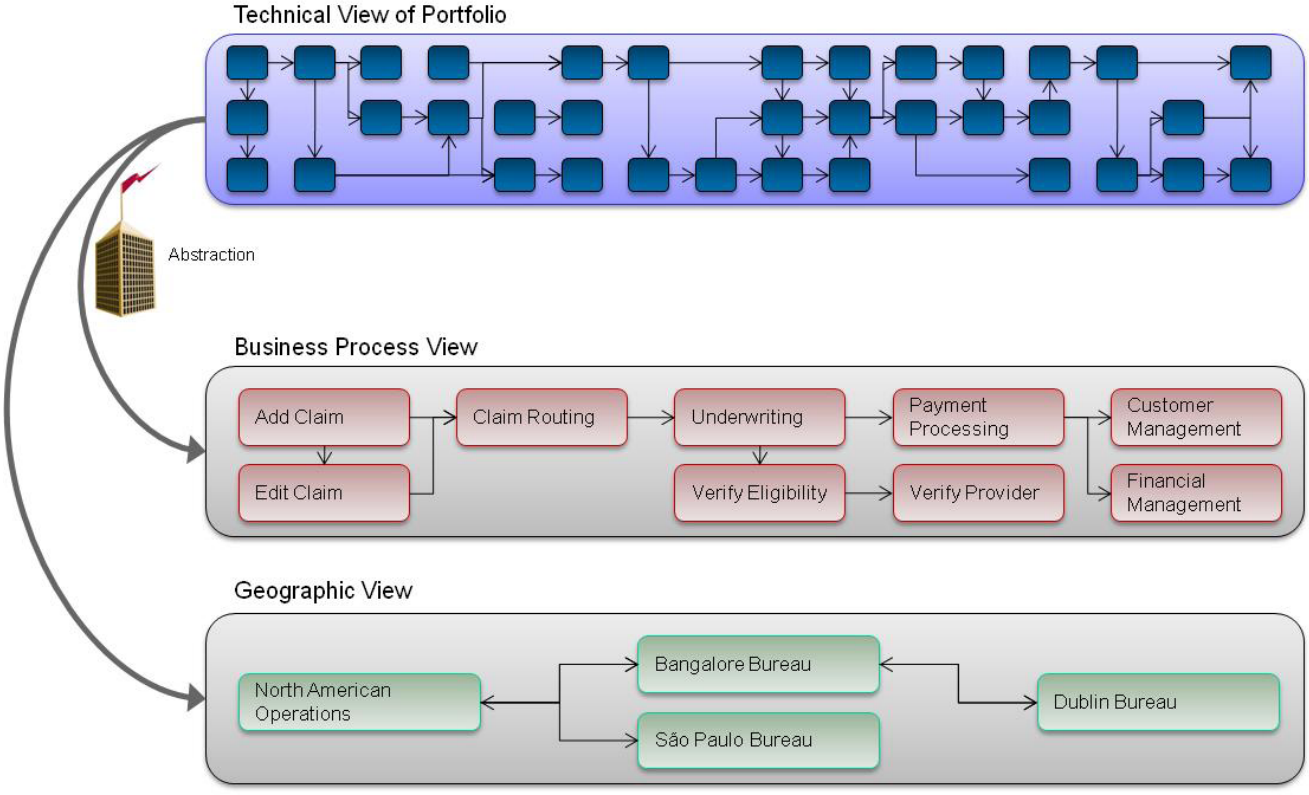
Technical level consists of the same artifacts, but the technical implementation details are abstracted. One may know the name of a program and its relationships with other programs or tables or screens. However, no actual code is used in comparisons. This level could be used for larger variations of the application, as it evolves over longer periods of time or it branches into separate instances, on independent paths.

## Functional model

Functional model level refers to models that describe the functionality of the application, regardless of the particular technical implementation. One may know that a customer applies for a loan, what information the customer provides and which are the steps to obtain the loan, but does not care which programs support these activities.

Abstracting allows an analyst to compare applications built on totally separate code bases. In principle, one could compare the functionality of two applications built twenty years apart and based on technologies belonging to different generations. In order to make a comparison, one needs to have functional models of both applications, preferably built on the same metamodels.

As seen in the diagram below, abstraction allows us to avoid complexity and gain a 'business intelligence' approach to viewing the portfolio.



Today, there are models that enjoy a large acceptance (for instance, UML or SBVR) and are adapted and supported by international standards organizations; e.g. OMG. The challenge is how to abstract from the code level to higher levels. There is also technology available that allows users to overlay user-defined semantic structures onto application code in order to reduce complexity and gain a model view.

Unfortunately the use of functional models is not enough for an automated gap analysis. Consider the following case, in which two car rental models express the same relationship between two main classes. In the first it appears as Customer – Rents – Car, in the second as Client – Rents – Automobile.

Although they have the same meaning for a human, a gap analysis program cannot match them unless it understands that in this context Customer and Client are synonyms, just as Car and Automobile are synonyms. Again, technology can help to support this challenge. Automated glossaries exist that link technical terms with their business synonyms, ensuring one to one comparisons.

## Normalized functional model

We use this term to designate functional models built on standard vocabularies. There are two sources of normalized models:

- Industry framework models have been built that describe typical business processes within a vertical. These are generally built by industry consortia or by large system integrators. These entities have typically acquired significant experience or insight regarding a particular industry; e.g., banking or insurance.

If two banking models are developed from the same industry framework, it is very probable that they use the same vocabulary. This fact makes automatic comparisons possible and practical.

- Ontologies are data models that represent sets of concepts within a domain and the relationships between those concepts. Unlike common data models, ontologies are created and exist with the declared purpose of becoming universally accepted standards for their domains. There are thus ontologies for a great variety of domains, such as publishing, wine, plants, or anatomy, and various domain organizations continue to develop ontologies.

There are two ways in which ontologies can be used. If from the very beginning the models are built on the vocabulary of a standard ontology, then they could easily be compared. However, this is not realistic at this time, as most existing functional models predate universally accepted ontologies.

A second solution is to build for each model a standard translation table, relating its vocabulary to the vocabulary of the ontology. This is the standard solution for the N-to-N relationship problem, in which a central hub is used (in this case, the ontology vocabulary) to reduce the number of possible relationships from N2 to N. If each model owner provides a mapping to a standard vocabulary, then every pair of two models could be compared.

# The Case of Business Rules

Sometimes objects are difficult to match and compare. This is the case with business rules, which do not have short and convenient names to match, but appear as English language statements. (Short names or identifiers may be used internally, but they would be meaningless and this would not help.)

How then can an analyst match rules that appear as English language statements, such as "To rent a car, the customer must present a valid driver license"? A fully automated matching is perhaps impossible, and a manual matching is not practical. If two applications each have 1,000 rules, one needs to make a million comparisons, which is infeasible.

A practical, semi-automated approach could, however, work very well. The approach is based on the idea that in a business rules model rules are related to terms. In the example above, the terms would be "rent," "customer," and "driver's license." Supposing that the rules use a common vocabulary – as explained in the section above – a computer program can create clusters of rules based on the same terms. The clusters of the two applications can be presented side-by-side, for a manual or visual comparison.

Supposing that a cluster has an average of five rules, an analyst could easily compare five rules for the three terms above with five rules in the second application. To compare these two clusters of rules, one would make 25 comparisons. Counting the 200 clusters resulting from the 1,000 rules, one comes up with about 5,000 comparisons, much less than the original one million. As the human eye and brain could easily compare five objects against another five objects, a business rule approach to gap analysis is entirely feasible.

The approach suggested here assumes that one has access to the collection of business rules implemented in an application. Unfortunately this is not the case for most existing applications – unless we are talking of those which from the outset are designed to work with business rule engines. As business rule engine technologies are still in their early phases of adoption, from a practical point of view one needs some way of identifying and classifying the business rules looking at the actual program code. This could be done manually or with the aid of some specialized software tools.

How could business rules be identified in the code? There are various methodologies that attempt to do it. In most cases there is some heuristic approach that renders a number of business rule candidates, later to be examined and either accepted and documented or discarded.

As an example, one may look for IF blocks that test the value of a variable coming from a user interface. Any such IF block has the potential to represent a validation rule ("customer must be at least 18" or "policy start date cannot precede approval date"). Another possibility is to look for certain computations resulting in values for data that is saved in tables or files ("a 10% discount is applied for orders over $100").

Once the business rules are identified, they need to be properly documented and – at least for the gap analysis approach suggested here – related to the terms to which they refer. The availability of rule repositories for two applications will open the way for practical and efficient comparisons between them.

## Expressing Gap Analysis Results

Depending on the goals of the gap analysis, one may be interested in the simple, high-level conclusion ("the two applications have 70 percent in common, but the second one is 30 percent richer in functionality"), or in a detailed report on the similarities and differences between the applications. The detailed report could indicate the matches that were found, the degree of matching between individual objects, and what objects are in the non-common zones.

## Conclusion

Gap analysis is a powerful tool for making strategic decisions regarding the merger or retirement of applications, as well as for obtaining detailed information to be used in future customization projects. To accelerate the process, users must take advantage of technology that automates the process. Software that allows users both to discover business intelligence and 'top-down' abstractions of existing applications, as well as 'bottom-up' rules documentation can address this need.

## About Micro Focus

Micro Focus, a member of the FTSE 250, provides innovative software that allows companies to dramatically improve the business value of their enterprise applications. Micro Focus Enterprise Application Modernization and Management software enables customers' business applications to respond rapidly to market changes and embrace modern architectures with reduced cost and risk.

**For additional information please visit: www.microfocus.com**