Oracle Upgrade Gap Analysis Pointers on Focus areas for effective Remediation

Rajalakshmi Thiruthuraipondi Natarajan

rajalan11@gmail.com

Abstract

Every new version of Oracle Applications comes loaded with new features and capabilities to improve business operations. The scalability feature of Oracle EBS lets the implementing organization to customize the product to best suit their needs and it is the responsibility of the upgrade team to read the existing landscape to identify the changes the new version comes with, how it fits the current business operations and what changes are to be made in order for the new application to fall in line with the existing operations, and even better, how does it improve the business. This can be excruciatingly manual and a tedious activity, with lots of changes for oversight and human errors. This article is aimed to ease some of these tasks by identifying the focus area that both the impact analysis team and the tools tend to ignore as part of thei,r gap analysis based on various types of customizations that can fully benefit the company and make best use of the upgrade.

Keywords: Oracle upgrade, Gap analysis, Oracle Customizations, Upgrade Impact Analysis, Custom PL/SQL, Oracle EBS, Oracle implementation approach

Introduction

In a constant effort improve and provide additional value, Oracle Corporation releases a new version pretty much every year for its products. These new versions are their commitment to evolve along with the new technologies and provide a value add to their customers. From one of it earliest version of 10.X back in 1990s to the most recent 12.2.12 version, the application has undergone a significant change and has adapted with the fast changing IT environment and meeting demands of its customers and as years pass by, the customer need to upgrade their application to a newer version for various reasons, either to keep themselves up to date with the most recent technologies or forced to do so , since the version that they are in might soon go out of support. Anyone working in a company that has Oracle EBS would know that there is always custom objects created on top of Oracle standard components to suit the company specific need. This can range from a simple field name change to introduction of a custom module altogether and with Oracle upgrade, in any case, it is the responsibility of the upgrade team to analyze the system and strategize to make the best use of the latest version (or any version the company deems fit).

During upgrade, every piece of the architecture might have a gap and needs to be remediated for the new function to perform well, such as the Server capacity, Operating System, the Java version or the Database version and even the Network and each team is responsible for performing their own gap analysis to come up with a plan to fill the gap. The key difference between these gaps and the application gap, which we will be seeing in detail in this article is that the above-mentioned gaps need to be remediated even before ethe version is upgrade, else the new version cannot be installed successfully, even if it does, will perform poorly and inconsistently. The good thing about this gap is that this can be caught fairly easily and early, whereas the other kind, the functional gap is hard to find and it pops up only when the end users start using

it, and at this it might take a long time before the issue even surfaces, which makes it dangerous and the inherent need to identify and address the same.

Considerations for Gap Analysis

Apart from the hardware and software requirements, the gap in Oracle Application itself can be broadly classified into technical gap and functional gap. These deviations are based on the how the application is designed to behave in the newer versions, which would affect the custom code on top of it.

Technical Gap

Technical gap is the situation where a piece of code or a component gets impacted because of the changes introduced in the new versions. These are particularly dangerous, since they can remain dormant for a long time and show-up when a particular situation is hit. Even worse is a possibility of a rogue code, which can create significant damage to the business. The ease of finding these types of gaps depends largely on how the component is designed and coded and not to mention the accuracy of the technical documentation. Some of them are fairly easy to identify and capture, since they might fail at the time of compilation, but others not so much and need extensive knowledge in the existing and new versions to catch it.

To quote an example, with Oracle 12.X, there were several tables that were replaced, such as RA_ADDRESSES were replaced by a bunch of HZ tables, and the customer code using the former would have failed compilation and easily caught. On the other hand, the multi-org concept was introduced, and the row level triggers was put in place to restrict access even at the table level. The custom codes, using these tables would have no problem getting compiled and the issue will not show-up unless the component is in actual use.

Also, the coding practices like using dynamic queries can escape the compilation errors and are very hard to catch unless its is operational. There are several such cases, and each can behave and create issues in its own way, causing unsavory results and damage before it is caught and fixed.

Functional Gap

Functional gap is when the new application flow or capability is different from the existing one, resulting in the custom pieces not working as expected. Predominantly, the custom codes act as an add on or based on the standard code and hence any changes to the underlying standard code might cause the customer code to behave erratic. Though it might sound a lot like a technical gap, the key difference here is that the component, by itself is fine, but when fit in a flow, do not have the right or enough input to produce the desired result, or the downstream might be altered to accept in a new format. Similar to technical gap, a detailed and accurate document can go a long way in identifying the gap, but in reality, the accuracy of such functional document remains unreliable as there is high chance that it is not up to date with the operational changes over the years.

While this might not be as dangerous as the rogue code in technical gap, it can still create subsequent damage. It can result in operational inefficiency and confusion, leading users scrambling with the result not knowing what might have go ne wrong wasting value time and resource, and even might damage the company’s reputation.

For instance, again with Oracle 12.X.X, the sub-ledger accounting concept was introduced, which was additional step in the accounting process. With this process, the necessary steps need to take in to account the intermediate step in the period close and financial reporting process and have the custom

program or process built around it altered accordingly. Failing to do so will result in reports not returning the right data causing delays in such critical process, and the damage only gets compounded if there are downstream applications or systems that rely on this data for their operations or reporting.

Upgrade Gap Analysis

The success of an upgrade depends on the additional efficiency it brings with it. Naturally, while implementing a solution, based on business requirements, the IT team would validate the out of the box functionalities and create custom solution. And with each upgrade, these custom components need to be validated to make sure it works with the new version too, if not make necessary changes to make it work. Depending on the size of Oracle Applications in the company, the effort involved in identifying the gap can be a very tedious and error prone. There are several tools available in the market and even some provided by Oracle to analyze the instances for the impact, but there are limitations with these tools and not to mention the cost factor associated with it. Some of limitations as follows -

- Binary objects like forms and reports are ignored from the scan.
- Compiled code such as java classes are not analyzed
- Non-database objects that reside in the server such as shell scripts, pearl scripts and in certain cases the SQL files that are stored in the server are ignored.
- Objects in the path outside of Oracle recognized path are not scanned,

While some tools are better than others, there still will be a need for further analysis to get the complete impact analysis to plan the remediation activity. There are several ways that the upgrade team can identify the gaps, and in particular the application configurations that need to be carried over to the upgraded system and these are the focus areas that most tools in the market do not address.

Separation of Duties (SOD) is an extremely important control that needs to be retained by the application irrespective of the version. This is a strategy, but which the companies force limitations, thereby needing multiple resources / teams to complete a task. This is a deliberate attempt to ensure that no one team or person has a power to perform a complete flow, even within the same practice. For instance, a company can have a SOD defined such that the same person or team cannot place a Purchase Order and receive the same. It controls can be such that there is a centralized buyer, who can place orders and the receiving split across multiple warehouses, which act as receiving locations, who can enter the details in the system. In Oracle, as a product, this entire activity is provided under a single responsibility and the implementing companies, enforce the separation of duties by creating custom responsibilities and filtering some functionalities using menu exclusions, which might get erased as part of the upgrade, especially, if the same responsibility is retained.

Forms Personalization is yet another customization the need to be factored in while upgrading Oracle. There are special add-on functionalities that are defined at the form or function level, without disturbing the core functionality. These are designed to trigger based on specific events on the form to facilitate the custom needs of the department. These can be a simple personalization defined using the personalization function in the front end or a bit more invasive by coding at CUSTOM.pll. Example, there can be a forms personalization defined at WHEN-VALIDATE-RECORD trigger, which is used to trigger a message to the user with the validation results based on the data entered by the user. While in case of minor upgrades, this personalization are preserved, in other cases, it is usually reset, erasing the custom functionalities and making the form behave as per its factory settings, which could be an inconvenience or even a problem in the day-to-day operations and hence can not be ignored in a upgrade process.

The next case is a practice that is generally frowned upon yet can be seen quite frequently. It is the condition, where the Oracle standard object is customized by retaining the same object name. This can be seen in situations, where the customization expected in minimal, but the piece of code where this change needs to be introduced in deep in the flow that the implementing team does not want to go through the hassle of recreating the entire flow. This is a very risky change and should have detailed documentation of the changes made and should be part of a checklist in every post-patch validation, since this could easily be overwritten and the changes lost. Its also important to note that these changes will most likely not be captured by any tool in the market since they would be functioning under the assumption that objects with standard names are preserved and look for gaps only in custom objects. One such possible customization is that if a payable invoice is paid in a third-party system and the payment interfaced to Oracle, to keep the payment number in sync with what was generated, the check generation piece, which is several layers into the payment process need to be modified not to generate the check number based on payment profiles, but to use the check number pushed as part of the interface. In this case the implementation team can opt for modifying just the code snippet and retaining the standard name, instead of creating a whole custom process.

There are several ways to identify these set-ups and other impacts with the new version. However, there are a few steps that the IT team can do to capture these details and port them to the new system.

One of the most straight forward method is the brute force method. In this method, a database link is created between the existing instance and the new version's sandboxinstance and each set-up are scanned between the two instances and the difference is captured. This can be used in any type of objects; may it be a PL/SQL code or a form or a report. Deep down, almost all tools do the same in some form or the other, though each object type might need different type of comparison. Such as,for PL/SQL code, it would be a cleaner and more precise approach, to compare the codes using the DBA_SOURCE tables for each object, which would provide every last detail. Also, for the binary objects such as forms and reports, they can be converted into a text format, such as fmx or rex, which can be scanned easily to identify the objects involved and impact identified. While this can be quite accurate, the data can be huge, and a lot can be quite irrelevant that might overwhelm the implementation team.

The other approach is quite similar to the above method, but with minor differences. In this approach, using the same DB link, the objects can be remotely compiled and the error message captured related to the invalid objects, which can point to the remediation plan. However, this has a limitation too. While this can be concise and can be used to identify the technical issue, the functional gaps cannot be identified. Since the object compilation largely depends on the underlying object being valid , but does not care if the functionality or the data it stores changed.

Other way is the use of AI and BOTS to do a regression analysis on the new version using predefined inputs and expected outputs to find if the new version meets the business requirements. This method is gaining popularity and is explored by various organizations. A properly defined BOTS can eliminate much of the errors that might happen with manual testing. This can be precisely tuned as per the need to deliver accurate and targeted analysis. But it comes with a cost. It takes a long time to perfect the BOTS with various flows and combinations that the application can take. Also, it can be expensive, since the company needs to invest in such automated tools and resources who can design and configure these tools.

Conclusion

One needs to understand that not all gaps need to be remediated. There are several cases where the customizations that were built to satisfy a customer's need has been incorporated as a standard feature in the new version. In other cases, there might be a chance for the business to adopt a standard process instead of custom one owing to challenges faced with the customizations in terms of reliability, performance, or even finding the right resources to support it. Hence it is an important, the gap analysis is not seen as a static piece, which is done at one point and move on, but done in every stage and the transformation is captured as the project progresses and necessary action is taken. It is also in the best interest of the IT department to keep the customizations to a minimum, since it is an additional over head in terms of time, effort and money and should aim at bringing down the customization or at least stick to the old count, otherwise, there needs to be a serious deliberation to see what other value-add does the new version bring to the organization and progress accordingly..

References

1. *ORACLE UPGRADE ASSESSMENT AND ROADMAP SERVICE FOR ORACLE EBUSINESS SUITE,https://www.oracle.com/assets/upgrade-assessment-ebs-1989411.pdf,*
2. *Best Practices for Upgrading Oracle E-E-EE-Business Suite https://www.oracle.com/us/products/consulting/resource-library/best-practices-e-business-069831.pdf, July-2011*
3. *Harness AI and Application Innovation for Successhttps://www.oracle.com/a/ocom/docs/artificial-intelligence/cio-harness-ai-and-app-innovation-solution.pdf*