Selecting the Optimal System Landscape for Your SAP R/3 Upgrade Project

Arthur Miller

Arthur Miller has been employed at SAP America for over six years. He is the manager of the Upgrade Competence Center, an internal SAP support team that has the mission of supporting SAP consultants and customers through all phases of the release upgrade process.

(complete bio appears on page 62)

As with any major packaged software installation, managing the periodic R/3 release upgrade is a necessary part of life. Upgrading ensures that you have access to the latest core functionality of the R/3 product, that you can support the full mySAP.com product suite as well as the latest third-party tools and packages, and that you will receive current support from SAP in the form of Support Packages, HR legal updates, and bug fixes published in SAPNet. SAP's goal is to make the release upgrade process "manageable" — meaning the upgrade can be completed in a timely manner, at reasonable expense, and with minimal impact.[1]

Anyone who has participated in a previous R/3 release upgrade knows that it's not exactly a trivial exercise. Ideally, each and every business process used in your R/3 system must be re-tested for its functionality at the new release. Additionally, your authorizations scheme and custom Workbench developments (e.g., reports and interfaces) must also be tested to ensure they continue to function as expected. If you're not exactly sure how to evaluate these areas, then an upgrade project requires that you first get your house in order in these areas.

In an *ideal* world, the process of upgrading your production landscape[2] would be relatively simple: you would upgrade your

[1] While this article focuses on SAP R/3 system upgrades, many concepts apply to other mySAP.com application components, such as SAP Business Information Warehouse (SAP BW), SAP Advanced Planner and Optimizer (SAP APO), and SAP Business-to-Business Procurement (SAP BBP). Since these products are based on a similar technical architecture to R/3, upgrades may be managed similarly. For more specific information, contact the respective competency centers for these products.

[2] A "landscape," as used here, refers to the R/3 systems in use by an enterprise, the clients in those systems, and the policies governing the management of changes to those systems.

development system, move on to your quality assurance system, and finish with your production system — one-two-three, and you're all done!

Reality, of course, presents an entirely different picture. The impact of the new release is simply too great in a complex, integrated system like R/3 that has been configured to a customer's exact specifications. You generally need to make changes to the new system that you will be rolling out, as well as support ongoing changes[3] to the production system for the duration of the upgrade project. Because change management is so critical to the upgrade process, you want to make sure that you employ a system landscape that makes it easy to manage:

- **Changes that support the future rollout of the upgrade:** Modification adjustments, changes to Customizing settings to either preserve existing functionality or make use of new functionality, and new ABAP Workbench development activity could all be part of the changes made to the new release *before* it is ready for production activity.[4]
- **Changes that support the existing production system:** Since an upgrade project may take months, it is not realistic to think that no changes will be allowed in the production system prior to the upgrade of that system. Instead, such changes must be made in isolated environments, tested properly, and introduced into the production system in accordance with the quality control procedures already in place. Since these changes also need to be present *after* the upgrade, they must be included in the activities being performed as part of your upgrade project.

The degree of change that must be managed in parallel with the upgrade project varies widely among implementations. At one end of the spectrum, some implementations are able to all but eliminate production changes for the duration of the upgrade project, allowing only qualified emergencies. At the other end, some upgrade projects are forced to conduct their evaluation work alongside a significant degree of change in the existing production system. This change could be the result of ongoing rollouts of new users or functionality, or simply due to a relatively unstable environment that is still "working out the kinks."

There are probably an infinite number of different upgrade landscape strategies that have been proposed

The Repository Switch Upgrade

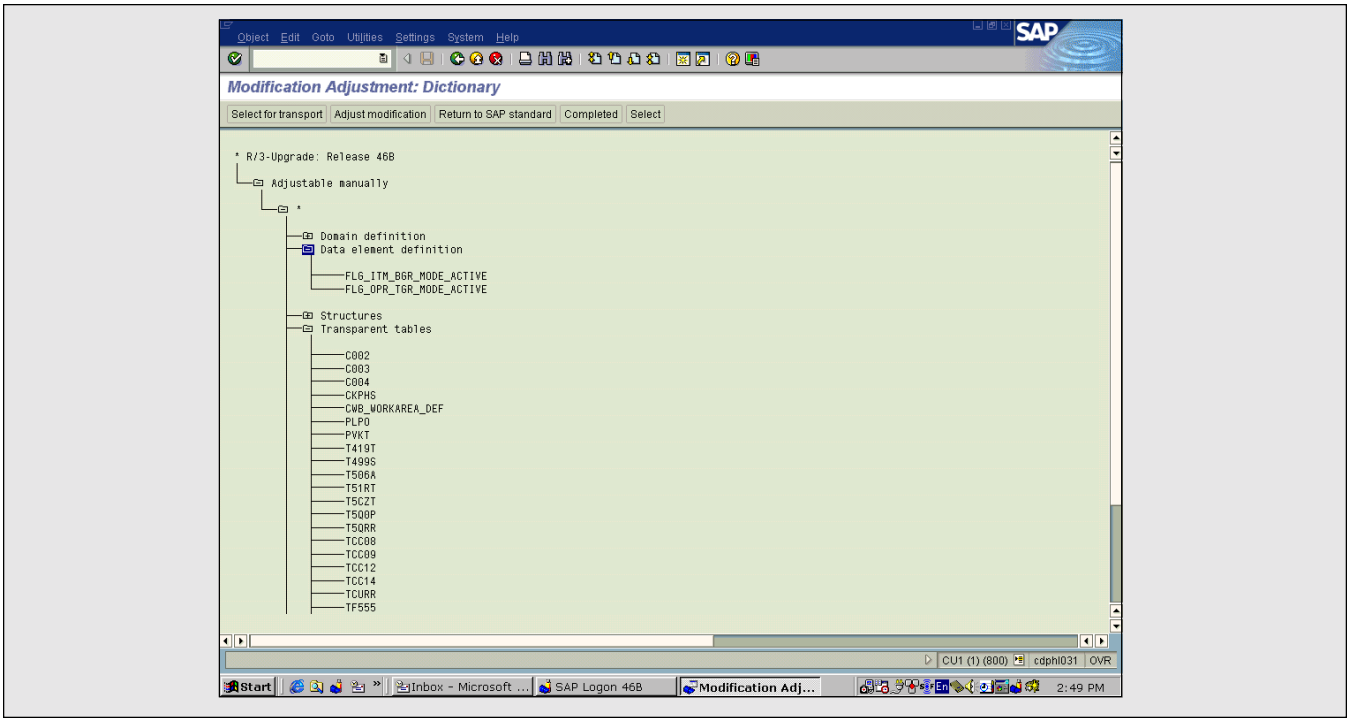
In an upgrade, a new release of software is applied to the R/3 Repository, which includes the entire ABAP Workbench library of ABAP programs, screen and menu layouts, and the ABAP Dictionary and its assorted components. Regardless of the net change between a source release of a system and the release it is being upgraded to, a complete copy of the repository is delivered with each upgrade. We refer to this as the "repository switch upgrade."[5] Since the complete, standard repository of a given R/3 release is imported into the system (regardless of what release was present prior to the upgrade), a series of adjustments are performed to retain customer modifications and new objects. Elements of the old repository that were created or modified by the customer and are needed after the new repository is delivered are copied into the new repository. Finally, a "switch" is effected, and the old repository is deleted from the system.

[3] The term *change* in this context refers to changes in functionality effected through updates to Customizing settings or to the ABAP Workbench. Changes to master and transaction data are obviously part of normal production activity and are not considered "changes" in this sense.

[4] The latter two types of changes — changes to Customizing settings and new ABAP Workbench development activity — will be imported into the newly upgraded system after the upgrade. Modification adjustments are imported or manually performed at their respective points during the upgrade itself.

[5] As of R/3 Release 3.0, SAP introduced the *repository switch upgrade* as the means of delivering a new release of R/3 to an existing system. In the "old" days, before Release 3.0, only the changed repository objects were delivered with an upgrade; this method was called a *delta upgrade*. For example, the upgrade process between Releases 2.1D and 2.2C delivered a different set of objects than an upgrade from 2.2B to 2.2C. This strategy, while efficient from a data volume standpoint, led to many inconsistencies and was difficult in general for SAP to support with a high level of quality.

Figure 1 The System Identifies the Necessary SPDD Adjustments



from time to time, but for the most part, and depending where they fall on the aforementioned spectrum, all upgrade projects use a variation on one of the following two landscape strategies:

- **Method A:** Rehearsal of the upgrade process on a separate, standalone R/3 sandbox system, with the intent of rolling out the upgrade to the production landscape at a later point in time.
- **Method B:** Upgrade of the production landscape's development system right away, followed after a period of time by the upgrade of the QAS system, with the maintenance of additional, temporary development and QAS systems to support the production system at the old release.

Understanding the principles, benefits, and limitations of each of these two landscape strategies[6] will help you decide which approach best suits your upgrade project. No single strategy could apply to all upgrade projects. As such, the goal of this article is to present the two basic approaches, Method A and Method B, and allow you to decide which elements of each to incorporate in your own plan.

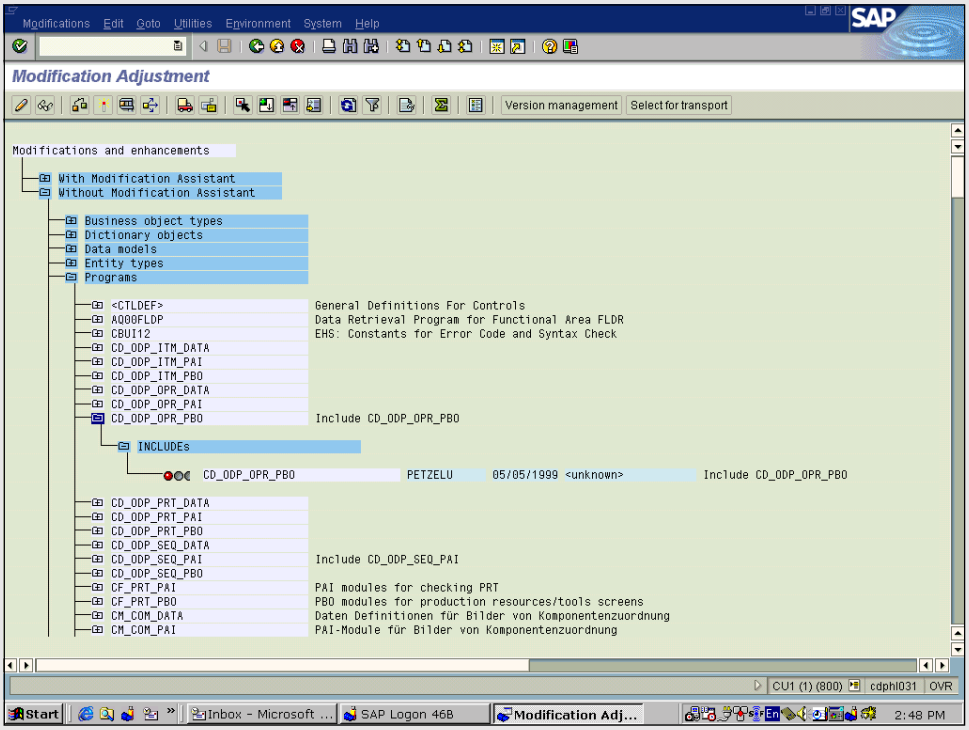
[6] For the sake of simplicity, a basic three-system landscape will be used in the examples. Extending the concepts to larger landscapes should be relatively simple once the basic principles of each strategy are understood.

Method A: Rehearsal of the Upgrade Process on a Separate Sandbox System

When upgrading a system landscape to a new release, you must perform the following general steps for each system in the landscape:

1. Apply the upgrade to the system and make any necessary SPDD adjustments during the process. This process includes the entire technical upgrade (see sidebar entitled "The Repository Switch Upgrade"). The system will identify for you which objects, if any, must be addressed in the SPDD adjustment, as shown in **Figure 1**. See sidebar for more information on SPDD adjustments.

Figure 2 The System Identifies the Necessary SPAU Adjustments



Why Are Modification Adjustments Necessary During an Upgrade?

Adjustments of ABAP Workbench objects are necessary in one specific case: when both SAP and the customer have made changes to the object since the prior release. (If SAP has made a change to a given object but the customer has not, then there is no conflict; conversely, if SAP has not updated the object, then a customer modification can be retained after the upgrade without question.)

This adjustment cannot be performed automatically by the system. Someone must examine both versions and determine what the final result should look like. Since SAP's changes are desirable in almost all situations, adjustment usually involves modifying the SAP code yet again to include the customer's new functionality.

SPDD is used to identify the ABAP Dictionary objects that affect data storage. This step is done during the upgrade, prior to activation of the new ABAP Dictionary to avoid potential loss of data in the underlying database tables. All other ABAP Workbench object conflicts are identified in SPAU at the very end of the upgrade.

2. At the very end of the upgrade, make the necessary SPAU adjustments (see sidebar on modification adjustments). Again, the system will have determined which objects require attention in this capacity, as shown in **Figure 2**.

3. Apply any new change requests to the system. These changes will represent the customer-specific adjustments to the new release to preserve existing functionality and/or incorporate new functionality, and should be considered as one of the primary deliverables of the upgrade project.

4. Perform a final set of acceptance tests to validate the system. These tests, which may be executed manually or via automated testing tools, should put your business processes through their paces and allow you to see whether or not the upgraded system is performing as expected. The rigor and thoroughness to which you conduct these tests is up to you; I have found that upgrade projects vary greatly in this respect.

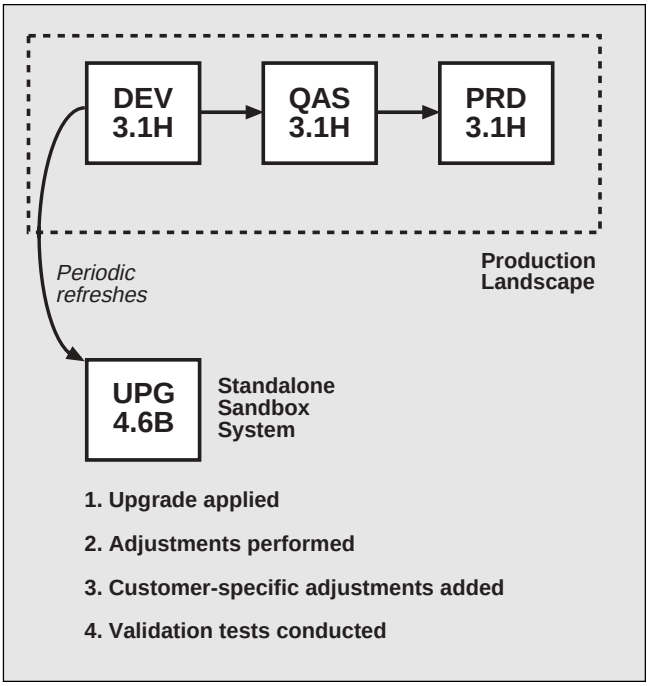
It is important to understand that it will take some time and a series of iterations before the exact tasks involved in these general steps can be refined and proven.

The basic principle of Method A is to rehearse and refine these tasks on a separate upgrade testing system — this separate system is identified as "UPG" in **Figure 3**. You create UPG as a database-level complete copy of any of the existing systems in the landscape. (In this figure, the standalone system is a replica of the development system. The question about which system in *your* three-system landscape should be copied onto the sandbox system — the development, QAS, or production system — will be answered in just a moment.) The four steps listed above are then executed (obviously, the first time such a system is created, there would be no customer-specific adjustments in step 3).

Figure 3 illustrates this approach for an upgrade from Release 3.1H to 4.6B.

As the process is repeated, the UPG system is necessarily destroyed and re-created as another complete database-level copy of one of the existing systems. The trick to this approach is retaining the changes made in the UPG system through each of these refreshes. During the first refresh or two, your project team will most likely be in a "discover-and-learn" mode. Thus, retaining changes may not be

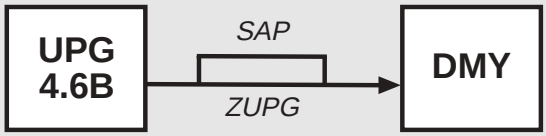
Figure 3 The "Method A" Architecture: One Upgrade Test System



TMS Configuration for Landscape Method A

If you are using Method A (using a separate system for upgrade testing), you do not need to modify the existing TMS configuration to support changes within the production landscape. You do, however, need to provide for the export of changes from the upgrade sandbox UPG. You can accomplish this by defining a virtual system as a consolidation system for UPG, as shown below.

TMS Configuration for Landscape Method A



You can give this system any name (e.g., "DMY" in the diagram). In addition, you must also define a customer-object transport route (shown as "ZUPG") between UPG and your virtual system. Without this route, change requests created in UPG are not transportable and cannot therefore be exported from the system. If your policy is to allow modifications of SAP objects during the upgrade process, then you also need to establish the "SAP" transport route between these two systems.

The DMY transport buffer contains a sequential list of the changes exported from UPG. If you need to import these changes into another system or back into UPG after a refresh, you can copy the transport buffer at the operating system level and use it to perform the import. You can rename this transport buffer to the proper SID and use it to execute a **tp import all** command, or simply use it as a reference.

Figure 4 Considerations When Choosing a Source System for the Upgrade Sandbox

Source System	Characteristics	Suitability for Upgrade Testing
Development	Source of all configuration and development activities. Probably many in-process or miscellaneous changes that have never been transported elsewhere.	Probably suitable for functionality testing, depending on how congruent the configuration and repository are with the QAS and production systems.
Quality Assurance	Same functionality as production, but may not have much, if any, "real" production master/ transaction data.	Suitable for functionality testing, depending on the requirement for "real" application data.
Production	Valid configuration with real master/transaction data.	Ideal if technically feasible, due to the presence of real application data. May not be feasible depending on size of the database.

necessary, and you can treat the system as a "sandbox" to help you evaluate the new release. Eventually, however, you will need to preserve these changes by releasing the change requests prior to the refresh and rebuild of the UPG system. You can then import the same change requests back into the UPG system as part of step 3 of the next refresh. (See the sidebar on the previous page for more suggestions as to how the Transport Management System, or TMS, configuration of such a landscape should be configured.)

Which System Should Be Used As a Source for Your Sandbox System?

Before you create the UPG system, you have to decide which of the existing R/3 systems should be used as the source for creating the upgrade sandbox. In a standard three-system landscape, the development system, quality assurance system, and production system each has characteristics that make it more or less ideal than the others for this purpose.

> ✔ ***Tip***
> *As you decide whether or not it is feasible to use the production system as the source system for the upgrade sandbox, remember that there are currently no tools or methods for extracting a logically consistent subset of master or transaction data. Thus, you must copy the entire production system if real production master/transaction data is needed for testing.*

Figure 4 compares the merits of using each system.

The most often-used system is the production system, because it contains real-world application data. A key consideration in choosing to use the production system as the source for your upgrade sandbox is obviously going to be the volume of production application data in that system. If your database is particularly large, the costs involved with duplicating it could be prohibitive — e.g., the size may exceed your available disk storage space, in which case you would need to purchase additional space.

If your production system is, in fact, prohibitively large, the QAS system is a good second choice. Here, the benefits of proper change management processes become evident. If managed correctly, the QAS and production systems should be functionally identical, in the sense that they contain near-identical repository and configuration settings in the primary QAS client. They differ only in the volume of application data present in the system. In such a setup, QAS would be a perfect choice as an upgrade sandbox, since it will allow for valid functional tests without requiring duplication of enormous amounts of application data.

A development system would be useful for some early versions of the upgrade sandbox, but is probably

not suitable for more detailed upgrade testing. A development system is probably more "out of synch" with production than any other system, and is also unlikely to contain master/transaction data that even remotely resembles production system data.

Managing Changes to the Production Landscape

When considering how to manage the production landscape changes during an upgrade project, you have three options:

- **Manually re-enter all DEV changes in UPG at an appropriate time** (e.g., once the change has passed QAS testing and is approved for import into PRD). These changes are included in the upgrade impact analysis, but excluded from the adjustments made in step 3 since they would already be part of the base configuration.

  Effectively maintaining and enforcing this policy might be problematic — for example, a Customizing transaction may change entirely between two releases, obscuring the re-entry process. Since the upgrade team will rely heavily on *proper* replication of changes, you need to insert some sort of QAS into the transport process to ensure that the changes made within the production landscape are indeed replicated properly into the upgrade testing system.

- **Import all DEV changes into UPG via change requests at an appropriate time.** This method may be faster than manual re-entry. However, there is a risk that the import will overwrite changes made as part of the upgrade project, due to the way changes are transported (e.g., the entire Workbench object or Customizing table entry is included at one time). More significant is the fact that this option involves transports between differing releases of the R/3 system, which makes it inadvisable in most cases.

  (See the sidebar on page 54 for a discussion of cross-release transports.)

- **Introduce changes into UPG at the next refresh of that system**, instead of bringing production support changes directly into UPG. You can rely upon the testing process in step 4 to identify any new problems that arise as a result of changes to the base configuration.

Your choice of policy will depend on many criteria. Primarily, you need to estimate and consider the volume of production changes that need to be supported. A high volume of production changes will mean propagating those changes more frequently to the upgrade testing system. The frequency of UPG system rebuilds is also a key factor.

If your plans call for rebuilding UPG only three times during an eight-month project, you will have relatively few chances to understand the impact of those new production changes at the new release. Refreshing more frequently diminishes the impact of manual re-entry, since the changes will appear in the UPG system in a relatively short amount of time.

In general, I tend to recommend a more frequent refresh of an upgrade sandbox system, with no real attempt to continually update those systems with changes in the production landscape. The actual tolerable refresh frequency will depend on the volume of transports going through your production landscape, but one refresh per month should be sufficient for a majority of upgrade projects.

> ✔ ***Tip***
> *Managing an upgrade project alongside significant change to the production landscape (e.g., new rollouts of users or functionality) will certainly increase the length of time and total cost of your upgrade project.*

Cross-Release Transports

Generally, SAP neither recommends nor supports cross-release transports. However, since there are cases where it may save time and manual effort, let's examine whether you should even consider these types of transports. Primarily, the contents of a change request determine whether a cross-release transport is technically feasible, as summarized below.

Feasibility of Cross-Release Transports

Type of Object in Change Request	Cross-Release Transports Allowed?	Explanation
SAP-owned ABAP Workbench objects	Definitely not	Since SAP may have changed the object, transporting from a different release could introduce changes that are incompatible with other objects in the repository.
Customer-owned ABAP Workbench objects	Yes, in most cases	Unless the object has also been changed by someone at the new release, a cross-transport object usually poses no problems. Whether the program functions as before depends on separate factors that must be evaluated as part of the upgrade impact analysis.
Customizing table/ view data	Possible	If the underlying ABAP Dictionary structures have not changed significantly, transports usually can be imported without a problem. Dictionary inconsistencies are usually identified as warnings or errors in the import logs. It is always prudent to examine the relevant transactions in the IMG to review the results of the import.
Transport-defined objects other than above — e.g., R3TR AM17, R3TR COC1, R3TR SRTR	Probably not	These objects represent special transport rules embedded in the source code of the OS-level utilities that perform the export/import. Since a cross-release transport usually also involves different versions of the utilities (tp and R3trans), problems may result. Potential problems in this case may not be identified in the import logs, so you should only attempt such transports with caution.

Note: This information is not an endorsement of cross-release transports and is provided for informational purposes only.

Variation: Using Two Separate Rehearsal Systems

Alternatively, you could implement a variation of the Method A strategy that incorporates two upgrade test systems at the new release. **Figure 5** shows the architecture for this Method A variation with two separate systems for testing the upgrade (called UPG and UP2 in the diagram). Using a second system allows you to release and transport changes to a separate environment. Since testing the transport itself is part of the change management process in general, using a second system should result in higher-quality transport requests for step 3 — although at the expense of maintaining an additional system. Finally, applying the upgrade process to more systems gives you more

Figure 5 Method A Architecture — Two Upgrade Test Systems

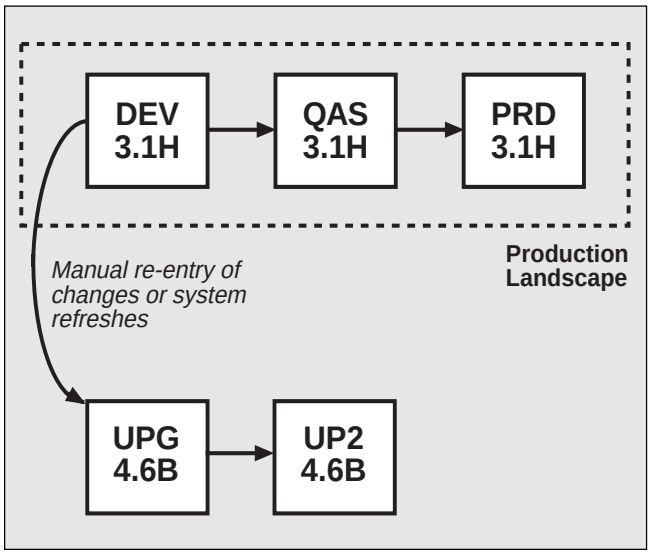
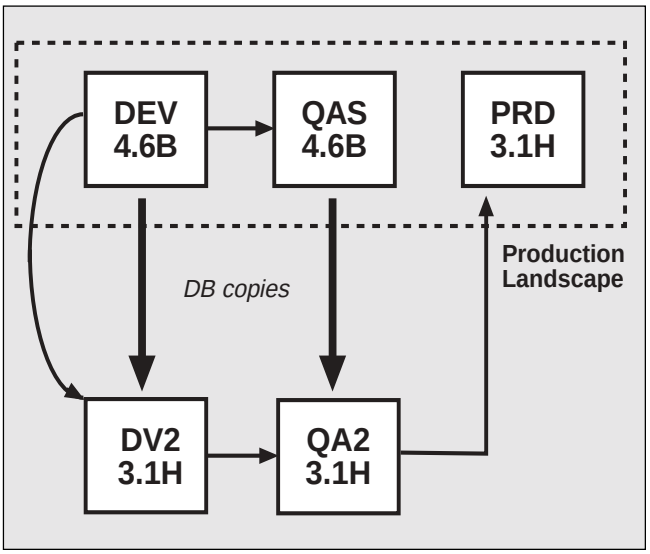


Figure 6 Method B Architecture — Two Production Support Systems



practice (a key element of upgrade success), perhaps reducing the frequency of refreshing UPG and UP2.

When executing the actual upgrade of the production landscape, you perform steps 1 through 4 for each of the systems in the production landscape: DEV, QAS, and finally PRD. You can then discard the UPG and UP2 systems, since they are no longer needed.

Clearly, the phased-rollout approaches for both initial implementation and upgrades of R/3 are highly similar, in the sense that you must manage changes for both the new and existing landscapes at the same time. Thus, the architecture shown in Figure 5 could also be part of an overall strategy to support an entire implementation life cycle. In this manner, you could rebuild the UPG and UP2 systems to support the next phase in the rollout plan.

On the surface, this alternative Method A architecture looks similar to Method B (the second upgrade landscape method). But looks can be deceiving. As we delve beneath the surface, you will see that Method B supports a very different approach to a release upgrade.

Method B: Maintenance of Temporary DEV and QAS Systems to Support the PRD System at the Old Release

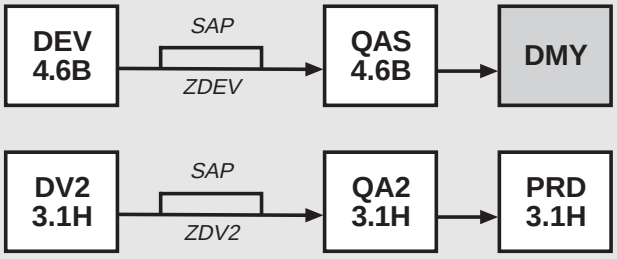
The second upgrade strategy, Method B, consists of directly upgrading the existing development system, without prior testing on a separate upgrade system. After a period of time, the QAS system is upgraded, followed by the upgrade of the production system. During this time, you implement secondary development and QAS systems to support the production system that is still at the old release.

The advantage of this method is that it reduces the overall time to upgrade by minimizing the actual number of upgrades necessary to move the entire landscape to the new release. The reason for this is that only three upgrades are ultimately required to get the entire landscape to the new release. Also, depending on the volume of changes within the production environment during the upgrade project, it may not be necessary to maintain a secondary development system (this is discussed later in the article). **Figure 6** shows the Method B architecture with two

TMS Configuration for Landscape Method B

If you are using Method B (upgrading your production landscape and using secondary development and QAS systems for production support), you will need to modify the existing TMS configuration. Since you no longer want automatic transports from DEV or QAS to PRD, you need to delete the delivery route between these two systems. As shown below, set up a standard TMS configuration between DV2, QA2, and PRD that is basically identical to the one that previously existed for DEV → QAS → PRD.

TMS Configuration for Landscape Method B



Create a virtual system (shown as "DMY") to use as a delivery system from the 4.5 QAS system. The transport buffer for this non-existent system represents the transport requests that should be applied to the production system after the upgrade, in their proper import sequence. Keep in mind that these changes should include all the changes made to support the production environment made in DV2/QA2, since those changes should have been manually entered in DEV and transported to QAS along with upgrade-related changes. With proper testing and quality control, applying these changes after PRD has been upgraded should not be harmful.

Again, you need to decide whether modifications to SAP objects are allowed in DEV, DV2, or both, and add the "SAP" transport routes to the configuration accordingly. In addition, you need to determine whether to permit changes to customer-owned objects in DV2. These objects would be treated as *repairs* in DV2, since they would still reflect that the originating system was DEV. If your policy is to allow these changes to be made and transported to QA2, set up the "ZDV2" transport route accordingly.

temporary systems for supporting production (called "DV2" and "QA2").

Method B consists of directly upgrading the existing development system, without prior testing on a separate upgrade system. After a period of time, the QAS system is upgraded, followed by the upgrade of the production system. During this time, you implement secondary development and QAS systems to support the production system that is still at the old release.

Based on this method, you can upgrade an entire landscape by following these steps:

1. Make complete database-level copies of your DEV and QAS systems. Give the copies unique names, such as DV2 and QA2, as shown in Figure 6. Continue running the old release on these systems for the purpose of providing production support.[7]

[7] For more information on the specific tasks involved in performing a database-level copy of an R/3 system, refer to the document *R/3 Homogeneous System Copy*, which is available as part of the R/3 system installation kits.

2. Apply the upgrade, including modification adjustments, to the DEV system.

3. Perform a series of impact analysis activities on the DEV system, capturing any necessary changes in change requests. These activities will involve research and education on the new release, as well as changes necessary to preserve existing business functionality.

4. Apply the upgrade, including modification adjustments, to the QAS system.

5. Maintain DEV and QAS at the new release, applying transports as necessary from DEV to QAS. Meanwhile, enter any production-support changes in the DV2 system, then transport them through QA2 and finally to PRD.

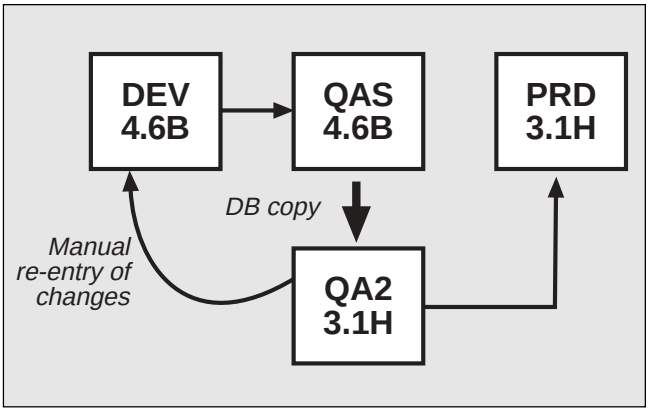
6. Upgrade the PRD system, make any modification adjustments, and import any change requests generated on the DEV/QAS systems.

7. Discard the DV2 and QA2 systems, as they are no longer needed.

The advantage of Method B is that it reduces the overall time to upgrade by minimizing the actual number of upgrades necessary to move the entire landscape to the new release. The reason for this is that only three upgrades are ultimately required to get the entire landscape to the new release.

Incorporating production support changes remains an issue with this method. However, the upgraded systems will never be rebuilt or refreshed. Therefore, you *must* re-enter all production support changes in DEV, either manually or via change requests. In general, this method is suitable for more stable production systems (e.g., those with relatively few changes being introduced) or those upgrade projects that are relatively simple in their scope and complexity. (See the sidebar on page 56 for suggestions on the TMS configuration for this landscape.)

Figure 7 Method B Architecture — One Secondary Production Support System



Variation: Using a Single System

If the volume of production support changes is very low, you probably do not need to maintain both DEV and QAS systems at the old release. **Figure 7** shows the architecture for this Method B variation with a single system for supporting production (called "QA2" in the diagram). Since changes would be rare and made only in an emergency, you would make them directly in the QA2 system and then transport them to the PRD system. Obviously, this approach is not ideal if the volume of changes is high (e.g., numerous changes introduced to PRD every week).

Choosing Between Method A and Method B

Obviously, every project is different. No single upgrade strategy works with every project for every customer. In one sense, the difference between the two methods is primarily philosophical. Should you develop, test, and retest the upgrade process in an isolated environment? Or, should you start by directly upgrading the landscape, taking care to

Figure 8 Selecting the Right Upgrade Strategy

Decision Factor	Method A	Method B
Number of additional R/3 systems required?	One system (potentially two, based on your requirements).	Two systems (potentially one, based on requirements).
Experience obtained with the upgrade process?	Many practice upgrades are executed.	Practice is limited to the DEV and QAS systems.
Best strategy given a high volume of changes within the production environment?	Yes, because production support changes may not need to be entered manually in upgrade system.	No, because all production changes must be entered manually.
Best strategy for a shorter, less complex upgrade project?	Maybe not, because the value added by the extra systems decreases.	Perhaps, because the time involved is greatly minimized.

provide for changes within the production environment as needed? SAP's recent extension of support for Release 3.1I and 4.0B make this decision potentially more important. For example, an upgrade from 3.1H to 3.1I is going to be quite low in its complexity due to the small functional and technological deltas between the two releases, while an upgrade from 3.0F to 4.6C is going to be at the other end of the spectrum entirely.

Figure 8 contrasts key characteristics of the two methods.

Further Considerations

The following sections include some important considerations to keep in mind when undertaking a system landscape upgrade — i.e., documenting the landscape plan, factors that will affect your investment in the upgrade, and managing downtime when executing the upgrade.

Documenting the Landscape Plan

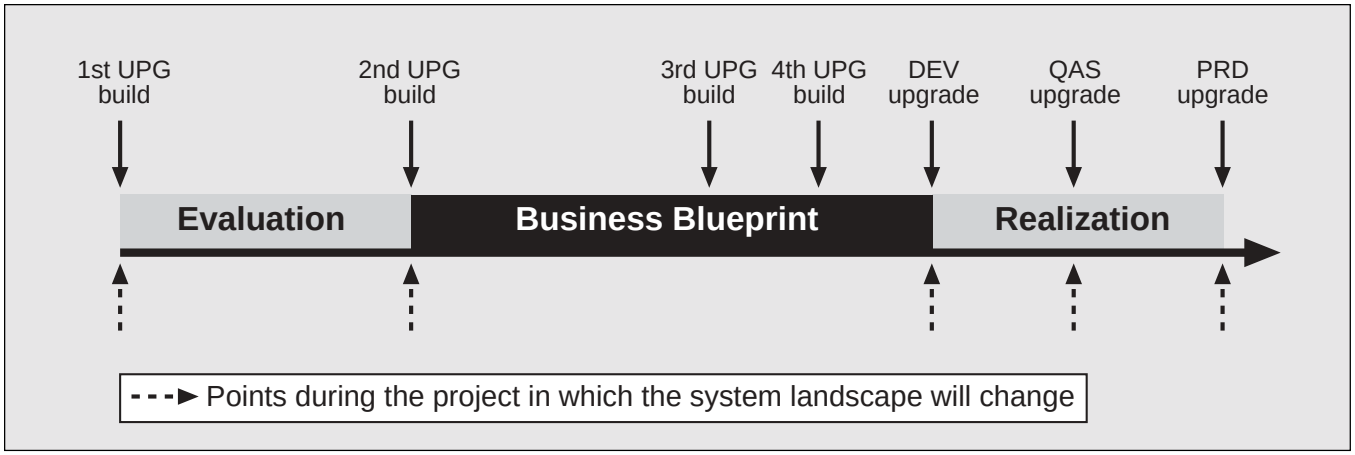
A landscape strategy is not static; it changes and evolves along with the project as a whole. Thus, it is usually helpful to correlate the changes in the landscape to the high-level phases of the upgrade project plan, and define the landscape separately for each unique situation. **Figure 9** shows an example.

During the Evaluation phase, no changes in UPG are retained. UPG is rebuilt at the end of Evaluation, and all changes will be preserved from then on. Two more refreshes of UPG will again occur during Buiness Blueprint, although these don't really change the landscape per se. Once the Realization phase begins, and the production landscape is upgraded, the landscape will change with each step of the way to reflect new procedures and potentially the loss or addition of systems in the landscape.

Then, with each step in the evolution of the upgrade project, a more detailed landscape strategy can be documented. The landscape itself, change management policies and procedures, and other key assumptions should be documented clearly for each step.

A landscape strategy is not static; it changes and evolves along with the project. Thus, it is usually helpful to correlate the changes in the landscape to the high-level phases of the upgrade project plan, and define the landscape separately for each unique situation.

Figure 9 Upgrade Project Timeline



Contributing Factors

Regardless of the method you choose to use, certain factors may well influence the investment of time and money required to upgrade your R/3 systems to a new release. If you are just starting to consider whether to upgrade, or if you are still assessing the potential costs of upgrading, answering these questions may help you identify potential areas of concern:

- How well are your system modifications documented? Do you know which modifications you will still need at the new release? Keep in mind that the upgrade considers manually applied OSS notes to be "modifications," so having a list of such notes will be very helpful in resolving SPAU conflicts.

- How easily can you test the necessary business processes on the system? Does the project team have a good understanding of exactly what must be tested? While ideally every implementation has a defined testing process that is used to test all changes (for example, the production-support changes that often occur regularly), the upgrade project often reveals that no documented, defined testing practice is in place already. The release upgrade thus forces the issue, requiring the project team to address this deficiency as part of the upgrade.

- Based on your decision with regard to your upgrade landscape strategy, how many extra R/3 systems will you need? Do sufficient hardware resources exist to support the extra system(s)?

- If you decide to use an upgrade test system, which production landscape system will you use for creating it? Are there any issues with database size or application data consistency across the various systems?

Managing Production Downtime When Executing the Production Upgrade

With any production upgrade, minimizing productive downtime while the system is being upgraded is a primary concern. To understand the concept of *downtime* as it relates to upgrade runtime, consider the high-level phases of an upgrade as shown in **Figure 10**. Based on your database log archiving strategy — which controls how and when the database redo logs are activated, and affects overall runtime as well as whether a full offline backup is required post-upgrade — you may be able increase productive uptime by running parts of the upgrade in parallel to productive activities. While there will likely be some impact on system performance, at least you can keep the production system available for as long as possible.

Figure 10 Phases of an Upgrade and Their Effect on Productive Operations

Upgrade Phase	Key Activities	Status of Productive Operations
Initialization	Version and environment checks, upgrade tools imported, and upgrade processing parameters are entered.	Uptime
Data Transfer	New Repository is imported into the database; analysis is performed on customer modifications.	*Optional*
Pool Transfer	Some pool tables are converted into transparent tables.	Downtime
Basis Adjustment	Old release is shut down; repository of new release begins activation.	Downtime
Application Adjustment	Adjust modifications (SPDD early, SPAU late), convert application pool tables, and perform all other adjustments to the new release.	Downtime

You need to evaluate several factors when estimating the upgrade runtime. Carefully consider, and, if necessary, address each of these factors before beginning a production upgrade, especially if minimizing downtime is a major priority:

- **Database Server Hardware**
  Note that the upgrade process *does not use application servers in a system*. Generally, these servers are shut down at the beginning of downtime and are not started again for the duration of the upgrade. Therefore, the speed and processing power of the central database server is a primary factor in determining upgrade runtime. In addition, investing in intelligent disk subsystems can pay significant dividends during upgrades, as they increase the overall I/O throughput of the system.

- **General Database Performance**
  If a database is heavily fragmented, performance degradation may cause the upgrade to run longer than is necessary. In general, consider cleaning up large databases (e.g., perform reorganizations) before the upgrade to minimize this factor.

- **Database Size**
  Although generally not the major factor in upgrade runtime, database size can be critical depending on the specific tables undergoing conversion and how big those tables happen to be in a given system. You can determine whether database size presents a consideration for you once the initial upgrades have occurred. Also, additional clients tend to slow the upgrade; delete any unnecessary clients before upgrading.

- **Unexpected Problems**
  Remember that encountering a problem that you have not seen on previous upgrades can result in unexpected delays while you try to resolve the problem. A single never-before-encountered error that arises during a production upgrade could prevent completion of the upgrade, if you cannot summon the necessary support resources in a timely manner. For this reason, I strongly

recommend that you make every attempt to practice the upgrade multiple times and fully document each and every error or warning message that occurs — as well as its eventual resolution.

One common R/3 upgrade challenge is not having any available servers that are the size and class of the production database server. This scenario makes it impossible to conclusively determine how fast the upgrade might run on that server. By running the upgrade on smaller servers, you can at least determine an *upper limit* on upgrade runtime and use it to establish your schedule. Obviously, if the test server is considerably smaller than the production server, your estimates will be extremely conservative.

As a protective measure, standard upgrade procedures call for a "drop-dead" time. At this prespecified time, if the upgrade has either not finished or not passed the testing/validation process, you abort the upgrade and restore the system to a point prior to the upgrade. You can enlarge this time window by implementing a standby database. Some data storage technologies, such as those offered by EMC Corp., allow for breaking the disk mirrors at a specific point in time; in case of emergency, the restore can be performed from that mirror. Investigate your options, given the technology in place at your implementation.

With any production upgrade, minimizing productive downtime while the system is being upgraded is a primary concern. Based on your database log archiving strategy — which controls how and when the database redo logs are activated and affects overall runtime, as well as whether a full offline backup is required post-upgrade — you may be able increase productive uptime by running parts of the upgrade in parallel to productive activities.

Conclusion

In my six years with SAP, I have been directly involved with 15 different customer upgrade projects. I have found the following practices to be generally accepted throughout the extended SAP community as conducive to successful upgrade projects:

✔ **Practice, Practice, Practice**
Without doubt, having significant experience with the upgrade process (the technical upgrade, modification adjustment, customer-specific adjustments to the new release, and testing/validation procedures) is key to the success of the overall upgrade effort.

✔ **Control the Scope**
To minimize project complexity and reduce the overall duration of the project, consider restricting the implementation of new functionality during the upgrade process. Focus first on preserving existing business processes at the new release, then take advantage of new capabilities after the upgrade.

✔ **Utilize Scripting/Testing Tools**
Since the testing and validation procedure should be performed many times, investing in supporting tools can pay significant dividends. For example, R/3 comes with the Computer-Aided Testing Tool (CATT), and Mercury Interactive offers the WinRunner product. As with any tools, they will require an investment of time to learn properly. In addition, the process of developing test scripts will help clarify the business processes in use and the data necessary to validate them, and this will pay dividends long after the upgrade project has concluded.

✔ **Track Modifications**
Avoid wasting time during the upgrade to investigate modifications that occurred years ago and for which you have no documentation whatsoever. Instead, set up a process now to track all modifications to the R/3 system, and establish policies regarding long-term ownership of those modifications. With no one available to "vouch for" or otherwise defend the presence

of a modification, you face the risk of not properly including it in the new release.

✔ Clean House

A release upgrade can be a "clarifying" endeavor. Some projects accumulate a formidable number of extraneous clients throughout the landscape, many with poorly defined roles. The upgrade project is a great time to re-evaluate the presence of these clients, delete the unnecessary ones, and simplify the landscape and transport process in general. As mentioned before, database size can have a direct effect on upgrade runtime, so consider the use of SAP's Archiving functionality to remove unnecessary master and transaction data from the production database prior to the release upgrade.

Arthur Miller has been employed at SAP America for over six years. He is the manager of the Upgrade Competence Center, an internal SAP support team that has the mission of supporting SAP consultants and customers by providing direct support and information about available services, tools, and accelerators through all phases of the release upgrade process. Formerly, he was a technical consultant and member of SAP's Platinum Consulting Organization specializing in R/3 release upgrades, the Transport Management System, and system landscape management in general. Prior to joining SAP, he was a consultant with Andersen Consulting in Chicago, Illinois.

Mr. Miller, his wife, and two daughters live in suburban Chicago. He has a Bachelor's degree in computer science from Northwestern University. He can be reached at arthur.miller@sap.com.

