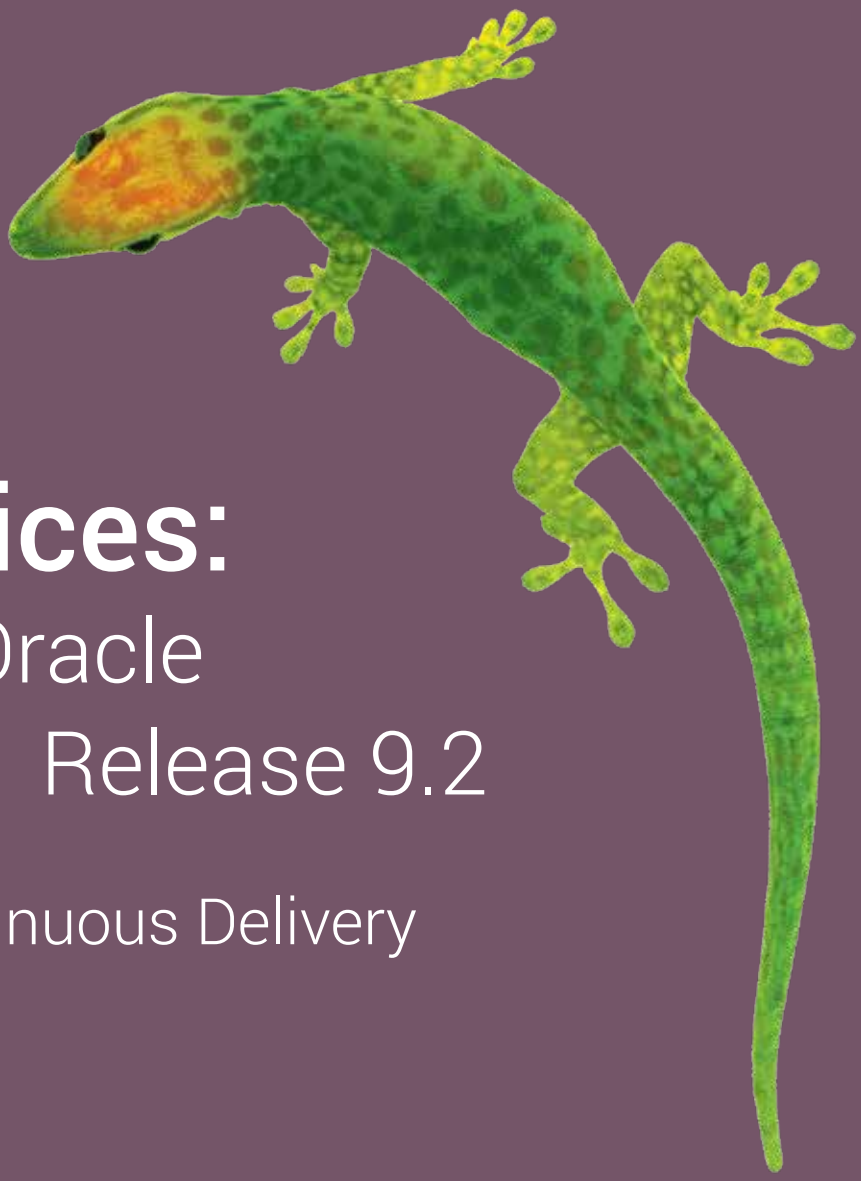
dws
innovation in action



Best Practices:

Upgrading to Oracle JD Edwards E1 Release 9.2

Positioning for Continuous Delivery

Contents

1. Executive Summary

JD Edwards EnterpriseOne (JDE E1) Release 9.2 will be on Extended Support until at least October 2028 and Oracle currently has no plans to release a 9.3 version of the software.

From Release 9.2, with significantly enhanced software packaging and delivery processes, Oracle is adopting a continuous delivery model. Going forward, customers will no longer need to suffer the cost and disruption of a major upgrade to take advantage of enhancements.

Once they have made the move to 9.2, JDE E1 users will be able to run code-current change event projects much easier.

The ability to stay code-current will heavily influence an organisation's ability to make the most of the latest software developments and help them compete effectively in today's digital economy.

A prerequisite to getting the most out of any continued investment in JDE E1 in the years to come is to upgrade to Release 9.2. This paper focuses on best practice advice for upgrading your JDE E1 system and laying the foundations that will help you stay code-current once you are on 9.2.

When planning the software development required during your upgrade (i.e. the retrofitting of your modifications) it is imperative that you reduce the number of modifications you are bringing forward to 9.2. Fewer modifications mean less work.

It is also important to think about how the retrofitting work will be executed as part of a comprehensive and well-managed upgrade project.

Upgrade projects can be complex, but the technical aspects of an upgrade, and any future code-current change event project, can be highly structured, outsourced, systemized and delivered for a fixed price on fixed timelines with near zero defects.

Another important aspect of your upgrade is the functional testing that is required. Testing time and effort is significant in every project, so there is no better time to think about your strategy for testing than during the planning stages of your upgrade project.

Advances in test planning and automation software allow organizations to test smarter; speeding up the process and easing the burden of testing on both your upgrade and any future code-current change event.

By thinking about the subjects raised in this paper, and adopting appropriate strategies and plans, you can run a better Release 9.2 upgrade project and position your organisation to make the most of Oracle's continuous delivery model.

Continuous delivery demands different thinking, coupled with products and services that support a culture of uninterrupted service. Business leaders that rely on JD Edwards are starting to demand this of their IT organizations.

2. Planning Your JDE E1 Upgrade

When planning your upgrade to Release 9.2, there are some key factors that warrant attention. These fall into the following categories:

- Reducing the modified footprint
- Identification of the types of modifications and how they will impact the upgrade
- Determining the best upgrade approach
- Calculating how much effort will be required

Some aspects of your upgrade are easily quantifiable, such as installing new hardware and setting up new databases. But when it comes to uplifting your custom code, (AKA retrofitting your modifications), this is notoriously difficult to estimate. How long will it take? What resources will you need, etc.?

2.1 Reduce your modified footprint

It is common for users of JDE E1 to customize their implementation of the software; making modifications to the standard code and tailoring functionality to better suit their specific business processes.

Over time, these custom objects can become a support issue; either because they become further adapted as business needs evolve, or because of changes made by Oracle to the base code.

In addition, many custom objects become obsolete; as enhancements made by Oracle remove the need for the customization or because the custom objects have been superseded.

Each JDE E1 upgrade, whether it is a major version upgrade or a minor code-current change event project, represents an opportunity to review which customizations or modifications are still required.

The ability to identify unused modifications will reduce the effort required for the upgrade. Time will not be wasted uplifting unwanted code and future support and maintenance efforts will be reduced.

Where do my modifications exist and how extensive are they?

A clear understanding of the existence, or extent, of any modification is a core component of any upgrade project. If you do not have an accurate view of your modified footprint, it could have wide ranging implications for the project. It may be useful to think about your modified footprint in terms of program objects.

Existing Oracle tools may under or overstate the number of objects in the 'affected' objects list. They may simply list modified objects without any concern given to your custom objects, or to your copies-of standard objects.

It is also likely that they will not consider the effect of any "net-change" coming from Oracle's new-release code and how those code changes will impact your existing modifications.

We cannot overstate the importance of clarity when it comes to understanding the extent of your modifications. The more extensive they are, the longer it will take to uplift the code.

As an alternative, you could rely upon your technical documentation. However, the effectiveness of this is going to be determined by how accurate, up-to-date and comprehensive your documentation is. Even the most diligent of IT operations can struggle to maintain a 100% accurate record of all their system modifications.

Failing to identify modified code could cause system failure or render users inoperative, as code they rely on has not been taken up to the new EnterpriseOne release.

Are the modifications still required?

Assuming you can identify accurately all the modifications you have made to your implementation of JDE E1, how can you identify which are still required by the users?

2.2 Identify the types of modifications and how they will impact the upgrade

Modifications can take several forms. You may have created new custom code, modified standard code or taken copies of the standard code and modified those.

Each type of modification brings its own challenges, with different levels of complexity. Copies, for example, tend to be the most complex and pure custom objects the least (Note: pure custom objects are those developed from scratch with no direct impact on other objects in the system).

It is important to know which of your enhancements work independently versus those that interact with, and make calls to, standard JDE objects.

As an example, you may have two Universal Batch Engines (UBEs) the same size. One UBE may work in isolation and is therefore standalone, whereas the other makes several calls to JDE objects.

In this second case, you need to understand by how much this UBE will be affected by the changes made to the standard JDE objects called by this UBE.

Another question you need to ask is, are you able to determine if the standard JDE objects you call have changed or not?

You then need to consider if any mods you have made to standard objects will be affected. In a worst-case scenario, Oracle may have removed this object completely. So how will that affect you?

Version Overrides give you the ability to modify a version's Event Rule (ER) logic and layout at the section level. The JD Edwards Object Analyzer can help you identify version overrides, but they are sometimes overlooked.

Whilst some organizations do not allow version overrides as a policy, some get shipped out of the box by Oracle and power users may make copies of these; which in effect spawn new modifications (version overrides).

These additional spawned modified versions may exist discreetly in production.

"It is important to know which of your enhancements work independently versus those that interact with, and make calls to, standard JDE objects."

2.3 Determine the best upgrade approach

When you uplift a modification, you need to determine whether it is better to:

1. Keep your modified copies-of-standard objects and test them thoroughly to ensure they continue to function correctly.

2. Determine the Oracle-sourced enhancements to the based-on object in the to-release and port these to your existing copy.

3. Re-copy and re-apply your modifications to a 'fresh' copy of the to-release's based-on object.

Although there is no absolute right or wrong method, each copy-of-standard should be reviewed on its merits; considering aspects such as the extent and complexity of your customizations versus Oracle's own changes in the to-release.

For example, a std-JDE object may only have 2 lines of code added to it in the new release. However, the custom modifications applied may span a hundred lines or more. The ability to see this enables the best approach in retrofitting an object; saving time and effort.

When considering how to uplift those modifications that are still required, do you have access to all the original developers? This may be relevant if you have particularly complex modifications, as a lack of accurate documentation may cause problems.

2.4 How long will it take?

If you are unable to accurately map your modified footprint and determine a best upgrade approach, then establishing a timescale for your project is going to prove difficult.

Assuming you have identified both, how do you estimate the likely duration of your project?

The one-third rule

An early theory suggested that whatever the effort required to develop the modified footprint originally, about one third of this effort will be needed to upgrade it.

But how many people understand the true effort and time involved; especially after they will have undergone subsequent changes over extended periods of time?

Analysis by DWS (based on hundreds of upgrade projects) reveals the one-third rule to be grossly inaccurate.

The sample and extrapolate method

The method most people use is the sample and extrapolate method.

This is where the vendor (or customer) interrogates an example of a small, medium and large modified object; before extrapolating the results across the whole system. This provides an estimate of how many modifications fall into each category.

This has also proven to be inaccurate, as it relies on a lot of assumptions about the consistency of the complexity of the modified code.

Sample and extrapolate, though flawed, has been widely used as there was no specialist tool available to forensically analyze the modified footprint. Until now.

DWS Dimension Analyze™

Dimension Analyze™ is a software-led service that has been successfully deployed by hundreds of JD Edwards EnterpriseOne customers; enabling them to audit and scope the development effort required during an upgrade to an unprecedented level of accuracy.

The service provides a forensic level of analysis of every object, every line of code and every property setting; down to pixel movement level of detail.

Dimension Analyze™ identifies the from/to base net change for every modified object and identifies the net change type and severity of impact against every modified object.

It also identifies all modified/custom UBE (Batch) objects that are no longer in use, so do not need to be upgraded.

With this detailed information available, it is possible to not only provide answers to the key questions raised in this paper, but also to estimate the upgrade effort required for each modified object down to an hours/minutes level of detail.

The net result can lead to some significant savings in both time and resource:

For one client upgrading from Xe to 9.1, Dimension Analyze™ enabled a reduction in the modified footprint of 63%; saving 318 developer days.

With another customer planning their upgrade to Release 9.2 from 9.0 we identified more than 1,500 objects that did not need to be upgraded, reducing their modified footprint by 39% and saving them close to 250 man-days of development effort.

Our best result with a customer upgrading to Release 9.2 is an 80.9% reduction in their modified footprint.

“For one client upgrading from Xe to 9.1, Dimension Analyze™ enabled a reduction in the modified footprint of 63%; saving 318 developer days.”

3. Executing Your JDE E1 Upgrade

With a definitive list of modified objects to be uplifted, you need to decide what is the best upgrade approach for your modified code. Do you take the new object and reapply the changes you made? Or do you take your revised objects and apply Oracle changes to your version? How do you decide which will be quicker?

Once you have determined the effort required and the approach to be taken with each object, you need to decide whether you want to carry out the uplift yourself or contract out some, or all, of the development work.

After all, your own development resource may be best employed supporting the ongoing day-to-day operation of JDE E1.

As we consider the challenges that face your developers, project managers and quality assurance teams, we will also look at what you might expect should you outsource elements of the development work.

3.1. Development

There are many issues developers need to consider when executing an E1 upgrade, including:

- Identifying modifications to standard objects
- Identifying modifications to copies of standard objects
- Obsolete objects
- Changes to standard hooks
- Custom indexes on standard tables
- Unicode and best practice changes in C business functions
- Obsolete APIs and custom client code
- Client only functions
- Table I/O mismatches
- Version overrides

Previously, we discussed how you might identify all modifications to standard JDE objects and to copies of standard objects. This information needs to be available to your developers when carrying out the upgrade.

Obsolete objects – If you miss a custom object that references a standard JDE object that has become obsolete (or no longer exists in a future release) it will cause issues downstream, during user acceptance testing.

Changes to the standard-hooks – There may be a change in the parameters passed into a function call, which ultimately affects the results of using this function in custom code.

Knowing about these changes, and the potential impact during the retrofitting of an object, reduces the likelihood of issues during user acceptance testing.

Custom indexes on standard tables – As an example, a custom index may have been added to the standard table. For the sake of this example we will call it index 5. In the future release, Oracle has added a new index to the table, also called index 5, which causes a conflict.

How do you check for these pitfalls so you can verify and retrofit the use of the custom index correctly and, in turn, ensure the consistent behaviour of the custom code and index?

Unicode and best practice changes in C business functions – If upgrading from a non-Unicode release, ensuring your C Business Functions meet Unicode standards is only partially covered by Oracle's tool set. Some of the changes will need to be made manually by a programmer.

Obsolete APIs and custom client code – Like the calling of obsolete objects and business functions, APIs called from custom or modified business functions may have been changed by Oracle or replaced by a new API.

Understanding what has changed, how it should be replaced (and by what) and where they are used in custom modifications helps for a successful upgrade.

Client only functions – If upgrading from a Windows-based release to a web-based release, any client-only business functions you may have developed may not automatically function correctly and often require re-engineering / re-design. Knowing where these function types exist before the upgrade starts is critical to prevent overrunning.

Table I/O mismatches – An upgrade presents a good opportunity to review the existing code/logic that may be causing issues in the current release. Issues such as mismatches between the indexes used by a Select and its following Fetch, for example, are issues that can cause testing problems if upgraded "as is".

"Understanding what has changed, how it should be replaced (and by what) and where they are used in custom modifications helps for a successful upgrade."

3.2 Project Management

A lot goes into managing an upgrade and organizations upgrade for many different reasons. With Release 9.2, a major justification for any upgrade will be to get current, so you can stay code-current.

Whether this is your justification or not, every 9.2 upgrade, and all the change event projects that will follow your upgrade, will involve some degree of business impact.

It is the change to the business that should be the focus of the project.

Behind the business change, there may be a lot of technical change (e.g. cloud and/or infrastructure changes, retrofit development effort, etc.). The more technical aspects of a change event project are excellent candidates for outsourcing.

Outsourcing is a great way of mitigating risk, as you employ an organization that is expert in a specific area, allowing you to concentrate on the business change and value-adding aspects of your project.

Whether you choose to contract a specialist service provider or not, some of the challenges that relate to the project management of a technical sub-project that is focussed on retrofitting during an upgrade include:

- Bundling
- Managing Resources
- Status Tracking
- Monitoring Progress

The Retrofit Project Manager needs to consider the overall project plan and timeline, requirements of the critical business processes, object dependencies and how to bundle objects together into smaller projects to meet defined deliverables and milestones.

Being able to do this accurately will enable you to plan resources for testing and deliver the project in manageable bundles (an agile approach) rather than waiting till the end of the development activity before performing all the testing (known as the waterfall approach).

It would clearly be helpful if the Retrofit Project Manager can produce a detailed project plan; one that provides visibility and accountability at an object level and enables the effective management of resources and tracking of the progress of the sub-project.

3.3 Quality Assurance

An important question to ask at this stage is: how do your Quality Assurance (QA) staff ensure that the code delivered by the developer has addressed all the challenges we've identified?

What tools exist that can help your QA staff provide your end users confidence that the upgrade has gone smoothly and that they will experience few problems?

The key issues for the Quality Assurance Officer are to minimize defects and maximize quality. Part of this is ensuring that as much as possible is done correctly first time.

Where any reworking is necessary, the QA function needs to track this activity and ensure the correct version is included in the uplifted system.

You need to be sure you have an accurate list of all modified code still in use to make sure every required object is uplifted to the new release, with nothing overlooked.

It would be helpful to have access to a knowledgebase of issues; such as unknown or missing columns and parameters, known obsolete and retired APIs etc.

Unfortunately, many employees involved in upgrades have little or no previous experience to call upon.

4. Planning Your Upgrade Testing

As you may be aware, Oracle has been promoting the concept of code-current to its JD Edwards customers. As supporters of this strategy, we conducted research among 100 JD Edwards customers, from around the world, to understand the barriers to adopting a code-current approach.

The most common reason given (by 76% of responders) was that testing is too complex and time-consuming. Testing, therefore, is a key challenge that needs to be (or should we say "can opportunistically be") addressed when carrying out your upgrade.

Whilst there are testing solutions available in the market, most users resort to manual testing rather than use the existing solutions. Why is that?

Our research revealed that current testing options available have their own problems:

- the need for specialist staff
- testing is still too time-consuming and too complex
- the fact that changes to the code will require complex changes to the testing solution

4.1 Key Considerations

Changes to your system during an upgrade may be considerable, but where exactly are those changes and when can your team start testing different aspects of your upgraded system?

Whether we are talking about a Release 9.2 upgrade or a code-current change event project post-upgrade, it is important to be able to understand the full impact of the changes. You need to know exactly what has changed, which objects are affected by the changes, what dependencies exist and where you need to go to test the changes.

During an upgrade project that spans many months, retrofitted objects will be completed and delivered at different times. Best practice would be to consider different cycles of testing and testing early.

If you test thoroughly, as you get toward the end of the project you can more seamlessly transition from project team testing activities (e.g. unit testing, system testing, integration testing) to end user acceptance testing.

With the completion and delivery of each retrofitted object, or bundle of retrofitted objects, you should be able to plan and execute testing.

Equally importantly, you need to be clear regarding what does not need testing, so you don't waste time, effort and money testing objects and processes that have not been impacted.

"76% of JD Edwards customers believe testing is too complex and time consuming."

DWS Dimension Focus™

Dimension Focus™ is the first product available to JD Edwards customers that will analyze your entire EnterpriseOne system to identify exactly which objects are impacted by any changes you are introducing into your system and how significantly they are affected.

This intelligence is then used to inform your test planning and management. It can help you test early and can ensure that you only test what needs to be tested.

It does so by investigating the programs that you use and how they will be affected by the changes you are introducing. It can also analyze the lower level objects that are used by these programs; such as tables, business views and business functions.

Dimension Focus™ also knows about the interdependencies (at the Event level) between these objects and can build up a multi-level hierarchy of how they are interlinked.

This cross-reference tree extends multiple levels above the ESU-affected objects, which may include not only APPLs & UBEs but also lower-level objects such as TBLEs, POs, BSFNs & DSTRs, each of which may be called by multiple 'parent' objects and events.

If you are unfamiliar with some of these JD Edwards terms, there is a glossary at the back of this paper.

Using advanced analysis techniques unique to DWS, Dimension Focus™ identifies those objects that would be directly changed as a result of making a change to your system. For some changes (e.g. a modification, or the application of an ESU) this could be as small as 5% of the total number of objects in your system.

5. Executing Your Upgrade Testing

Feedback from JDE customers and partners indicates that testing can account for up to 65% of the overall effort of an EnterpriseOne upgrade, so it is an area where any improvements could make a big difference.

5.1 Key Considerations

What are the key challenges and considerations when it comes to the functional testing of your EnterpriseOne upgrade?

- Test planning
- Build up-to-date test scenarios
- Resources
- Test execution
- Results tracking
- Regression testing
- Agile v Waterfall

Test Planning

Arguably, when performing an upgrade everything needs to be tested. The challenge therefore is in planning your testing. What are you testing and when? How does all your testing come together to ensure that when you hand over to users for acceptance testing your system has already been thoroughly tested?

With any change event project the challenge is to identify just what needs testing and what does not. As we mentioned earlier, some ESU's or customer development projects may only impact 5% of your total custom programs and standard objects.

Build Up-to-Date Test Scenarios

For any programs that have been impacted by changes Oracle has made, you will need to test. You may be able to run previously created test scenarios, or modify any existing test scenarios; or you may need to create new test scenarios.

If you do have an existing script, how easy is it to modify those tests? How easy is it to test the changes you have made to those scripts? After all, a test script often requires highly technical test programmers to develop the test scripts, so they need testing too.

It is quite common to find that existing tests can be difficult and time-consuming to update, even to cope with quite simple changes to the user interface. You should strive, wherever possible, to keep to a minimum any brittle user interface (UI) tests that are tough to maintain.

Resources

What resources are available to help create test scenarios? Are you using a testing tool that requires expensive and scarce specialist test engineers to develop the scripts required? Or are you relying on manual testing that requires multiple end users to manually run through their business processes on the new version of E1?

Either way, testing can become a resource bottleneck and requires careful planning. When it comes to executing the tests, you need to focus your testing effort on the code that has been affected by the changes in the upgrade, or in code you have changed yourselves.

If you are relying on manual testing and bugs are identified, these will need to go back to the development team to be fixed. Your developers will need to know exactly what the user did and what happened or did not happen as expected. They will require screen shots to help them.

Results Tracking

Once a problem has been fixed it will need to be retested. This retesting cycle may need to be repeated several times until the code is performing correctly. The problem with this approach is that each time it goes back to the users for retesting, it is taking up more of the users' time, so they will probably be less diligent with the testing on each subsequent cycle.

This could result in bugs creeping through to the live production environment, with potentially serious consequences. According to Carnegie Mellon University it costs fifty times more to fix a problem if it has gone into production than to find it during development.

Regression Testing

Regression testing is the process of testing changes to computer programs to make sure that the older programming still works with the new changes. This has been a basic requirement of software testing, as up until now, it has not been possible to identify if any EnterpriseOne code is unaffected by changes or not.

With the sheer volume of testing that may be required, it is important to keep an accurate record of what has been tested satisfactorily and what still requires final testing.

Agile v Waterfall

In the Waterfall testing model, no testing is started until all development activity is completed, so no usable software is produced until very late in the whole upgrade process.

With an agile approach, the software is tested as it becomes available and in a usable form, so testing of one bundle of objects can be tested whilst development is carried out on the next bundle of objects.

5.2 Testing Options

Manual testing seems to have been the only real option for most companies running JD Edwards until quite recently.

Some larger organizations, with complex environments and considerable resources, use solutions like Oracle Application Testing Suite (OATS) or Hewlett Packard Quality Centre (HP QC). These larger companies, however, are the exception.

Smaller more specialist testing solutions are on the market but none of these have been tailored to work well with JD Edwards EnterpriseOne and consequently also require quite specialist and technical skills. What the JD Edwards EnterpriseOne community wants are testing products that can be adopted and used during the course of every project, and that can be used by business analysts and super-users.

"What the JD Edwards EnterpriseOne community wants are testing products that can be adopted and used during the course of every project."

DWS Dimension SwifTest™

Dimension SwifTest™ is the only testing solution designed specifically for, and integrated with, JD Edwards EnterpriseOne.

It has been designed with the non-technical end user in mind to make the creation, editing, scheduling and execution of testing as simple and efficient as possible.

Dimension SwifTest uses unique scanning technology to automate the generation of test scripts for all test scenarios. It is natively integrated with JDE E1 to provide the easiest and quickest way to generate and edit test scripts for JD Edwards applications.

Test scenarios are easily updated and edited, unlike in traditional record and play products that require a whole sequence to be rerecorded each time a change is required.

In addition, it tracks the status of tests and their results, provides clear reporting on test results and pinpoints all test failures with user-friendly information regarding the reasons for the failures.

Dimension SwifTest™ can significantly reduce the time and effort associated with functional testing. Savings of up to 60% can be realized during test execution, with further savings of 70% achievable during the creation and maintenance of test scripts.

Used in conjunction with Dimension Focus™ the savings can reach as much as 85%.

Glossary of Terms

Term/Acronym	What is stands for	Note
APPL	Interactive application object in E1	
BSFN	Business Function	Written in "C" or NER, these objects can be called by other objects, including other BSFNs, to perform an action or set of tasks within a functional process.
CNC	Configurable Network Computing	This is used for JD Edwards systems and security set-up, configuration and management.
DSTR	Data Structure	A vehicle made up of an element or group of elements used to pass data between a calling object and the BSFN it calls. Data can pass in either direction.
ER Logic	Event Rule Logic	This is the coding language employed by Oracle E1 applications, batch-jobs and other object types to control functionality within each object/ process.
ESU	Electronic Software Update	The mechanism by which new software is released by Oracle to perform partial or complete upgrades to JD Edwards.
PO	Processing Option Template	A set of options attached to an APPL or UBE that can be set/changed by the user to control the functionality or "logic flow" of the APPL or UBE.
TBLE	Table	Table in the DB for storing data
UBE	Universal Batch Engine	This is used for data extraction, transformation, publication, and distribution.

dws
innovation in action

"I would not start planning a future upgrade or migration without first consulting DWS. In a previous project DWS confirmed assumptions we had long held while providing additional detail to enhance those assumptions. In addition, we were enlightened to several surprises that the DWS process uncovered that has helped us reduce risk and cost in our migration planning."

Applications Director, Lafarge

About DWS

DWS Consultants is a leading provider of Oracle JD Edwards EnterpriseOne software services.

Since 1998, we have been providing specialist analytics, development and testing services to organizations looking to customize, upgrade or support their JD Edwards deployment.

DWS supports a global client base with bespoke module development, CNC, testing and software upgrade services; using our proprietary Dimension™ Suite of tools.

For further information please visit our website, or contact us:

UK: +44 (0) 1494 896 600 US: +1 888 769 3248 ANZ: +64 9427 9956
sales@dws-global.com **www.dws-global.com**

ORACLE Gold Partner Cloud Standard