UPGRADE

Microsoft Dynamics™ NAV*

Best Practices for Upgrading

White Paper

This white paper describes the best practices for upgrading Microsoft Dynamics NAV from one release to the next. It is intended for Microsoft Dynamics NAV Partners who want to gain an insight into the recommended processes to follow during an upgrade project.

Date: March, 2006

* Microsoft Dynamics NAV, formerly Microsoft® Business Solutions–Navision®

www.microsoft.com/dynamics/nav

Microsoft Dynamics

Table of Contents

Introduction

This white paper describes the best practices for upgrading Microsoft Dynamics NAV from one release to the next. The paper gives practical advice to help you manage the entire upgrade process.

It was written based on the insight of a select group of experienced Partners and other experts who have detailed knowledge and extensive practical experience of the upgrade process specifically for Microsoft Dynamics NAV.

Tectura, Columbus IT Partner, impuls Infromationsmanagement GmbH, and Multum d.o.o are gratefully acknowledged for their valuable contribution to this white paper.

Acknowledgements:

- **Merete Spiegelhauer, Navision Consultant, Tectura, Denmark**
- **Per Vestergaard-Laustsen, System Consultant, Columbus IT Partner A/S, Denmark**
- **Rainer Fischer, Navision Business Unit Manager, impuls Informationsmanagement GmbH, Germany**
- **Borut Jures, Freelance Consultant, Multum d.o.o, Slovenia**

If you have questions or require further information after reading this document, please contact Frank Fugl, Senior Product Manager, Microsoft Dynamics NAV at ffugl@microsoft.com.

The Value of Upgrading

Upgrading can provide your customers with powerful new functionality that can help their business grow. By taking advantage of improved functionality, new features, and enhanced capabilities, customers can reduce their costs and keep their business processes up to date.

Concerns about upgrading are very real for customers and include a fear that upgrading will be a costly and time-consuming process that is not justified in terms of the final results. Customers can also be concerned that their existing solution won't be around long enough to justify an upgrade effort.

However, upgrading is now easier than ever and by following best practices you can help your customers add value to their solution with a minimum disruption to their business. Overall, the longer your customers wait before upgrading, the more time-consuming and costly the upgrade can be. On the other hand, upgrading regularly to the newest functionality gives immediate realization of improvements in productivity for the customer. There are many advantages for you and for your customers to keep on the upgrade path. Highlighting these can help to convince your customers of the value of upgrading.

Advantages for You

If you keep your customers up to date through upgrading, you can focus on fewer versions and concentrate on keeping your developers up to date on the latest versions. You can offer your customers additional services (in addition to new implementations) and with upgrade projects, you have a chance to re-factor your own code and to implement your own add-ons.

Upgrading:

- **Increases customer satisfaction**
- **Improves your competitiveness**
- **Offers a chance to implement hot fixes in one go**
- **Offers a chance for you to implement your own add-on solutions**
- **Decreases your support effort**

Advantages for Your Customers

If you keep your customers on the newest version, they will realize the benefits of the new functionality right away and you save them extra costs later.

Upgrading:

- **Keeps business processes up to date with customers' changing business needs.**
- **Allows expensive customizations to be replaced by standard functionality built into the latest release.**
- **Resolves localization issues that have only been implemented on latest versions.**
- **Allows for a broader integration to other Microsoft products.**

Prerequisites

Before you embark on an upgrade project, there are a number of things that need to be in place to ensure a smooth implementation. It is vital that you know the current version and the new features and changes to the new version in depth. This depth of knowledge is achieved by following training (in particular, the What's New training), reading documentation (for example, the changes.doc), and by upgrading your certification to the next level. You will need to research compatibility issues and the changes to functionality in the new version to ensure a successful upgrade and to take advantage of the latest release's new and enhanced features. You should familiarize yourself with the upgrade tools available in the Upgrade Tool Kit.[1]

It is also very important that you understand the Data Model, and know what changes have been made in the new version because you might need to update your customer's data as well the application objects. If you want to do a full upgrade, you need to understand which data has been changed by the upgrade and why. For more information about this, see the changes.doc. All changes are grouped under specific functionality. Partners should also use the Developers Toolkit to check for changes in data structures. This is covered in section 5.4 "Compare Two Versions" in the DevTool Manual.pdf file, which is part of the Developers Toolkit.

[1] **Note:** The Upgrade Tool requires a Partner License.

The Roles and Phases of an Upgrade Project

An upgrade project is similar to an implementation project and therefore has the same roles and follows the same phases.

To ensure that your upgrade project is successfully managed, carefully consider the skills that are needed in your team to manage the different phases and milestones of the project. With the (MSF) Team Model, there are a number of different roles that a team should consist of. Each role is not necessarily one person; one person can fulfill a number of different roles. However, some combinations are not suitable (for example, a developer shouldn't be a tester).

Best Practice
If possible, it is an advantage to recreate the original team responsible for previous implementations.

Program Manager

The focus of the program manager is to meet the goal of delivering the product within project constraints. This role covers four key functions: project management, solution architecture, process assurance, and administrative services. This role is typically covered by a project leader or business consultant with customer contact, whose main task is to ensure delivery of the product. This role should be represented at the customer site throughout the project.

Product Manager

The key role of product management is to satisfy customers. This role covers four key functional areas: marketing, business value, customer advocate, and product planning. In an upgrade project, this role is often covered by the sales representative whose primary task is to ensure the customer is satisfied.

Developer

The developer or application programmer is in charge of "building to product specification". This role covers: technology consulting, implementation architecture and design, application development, and infrastructure development.

If the upgrade has many customizations that can be replaced by standard Microsoft Dynamics NAV functionality, the developer should have a good understanding of the standard program.

Tester

The goal of the test role is to approve the upgrade for release, only after all product quality issues are identified and addressed. This role covers: test planning, test engineering, and test reporting.

User Experience

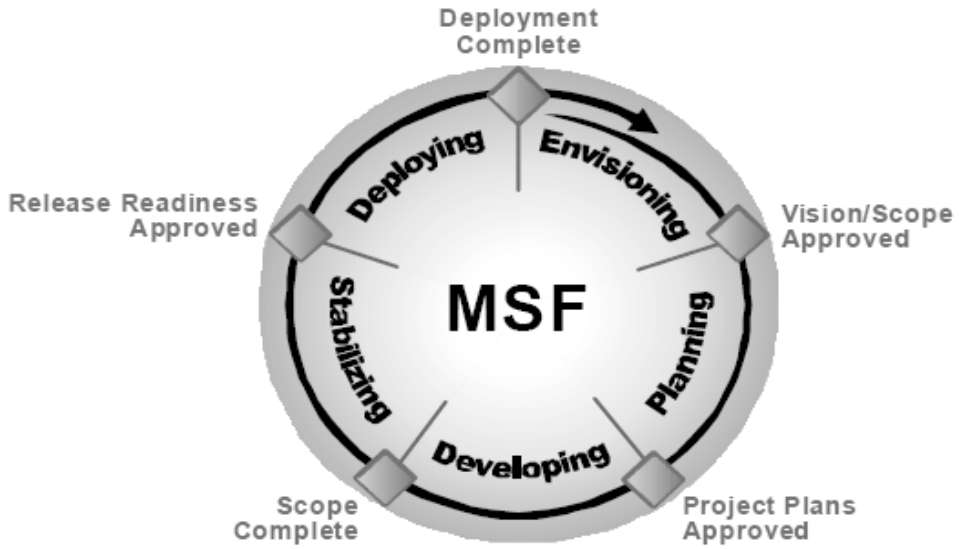
The goal of the user experience role is to enhance user effectiveness. This role covers: accessibility, internationalization, technical communication, training, usability, and graphic design. The user experience role is in charge of training and documentation.

Release Management

The goal of release management is to ensure smooth deployment and on-going operations. This role covers four key areas: infrastructure, support, operations, and product release management. In an upgrade project this role is typically covered by the partner organization that signs off on big projects (alternatively, it would be done by the Program Manager/Project leader and developer).

The Phases in an Upgrade Project

The phases of an upgrade project are illustrated in the Microsoft Solutions Framework (MSF) model (these are described in more detail in the following sections of this document):



Envisioning

The envisioning phase involves a high-level requirements analysis and is one of the most important phases in an upgrade project. With proper planning, identification of business processes, and scoping, the project has a much higher chance of being successful. It is very important at this stage that you define the success criteria for the upgrade and agree this with the customer. The envisioning phase is mainly driven by the product and program manager.

The following sections describe in more detail the things to consider and the tasks that are part of this phase.

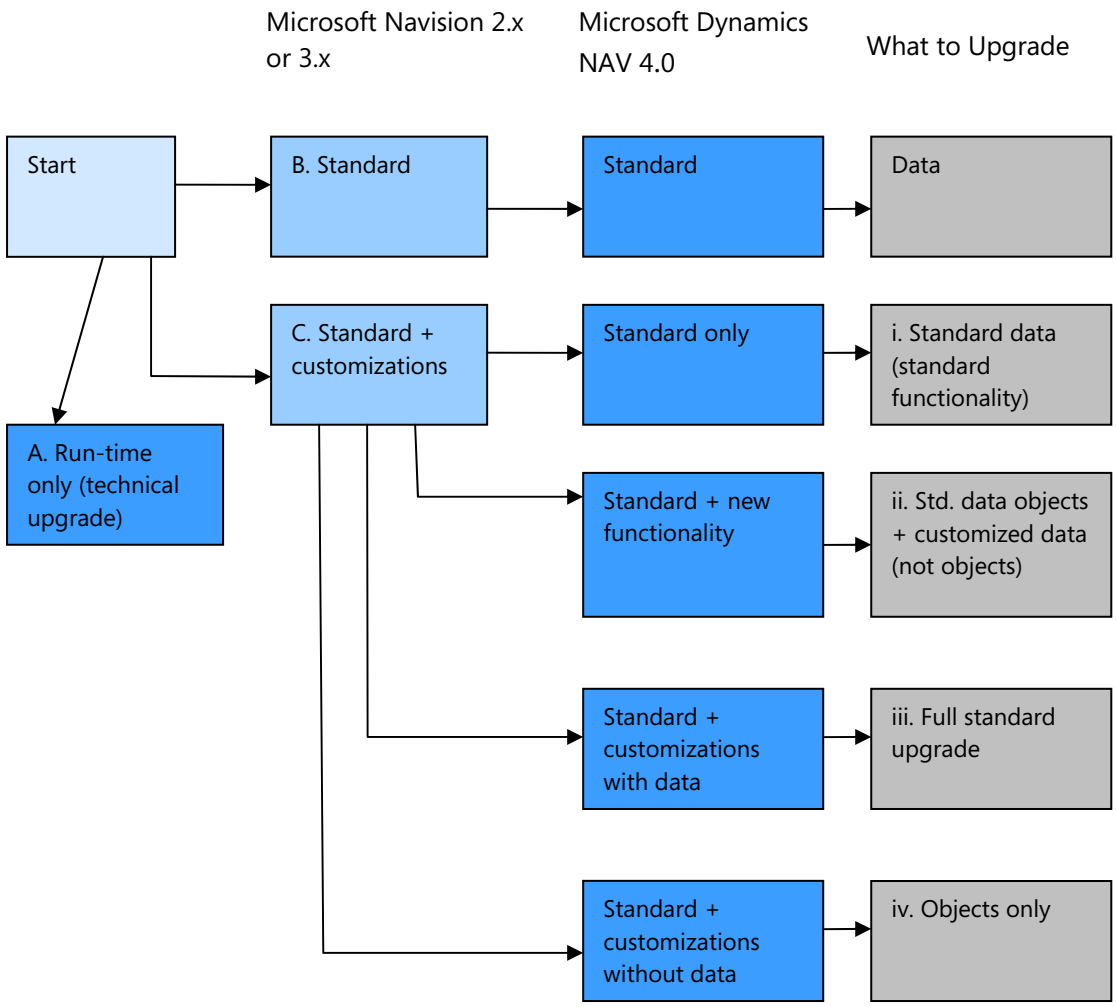
Understanding the Scope of Customizations

Ensure that you are prepared before you go to the customer. One way of ensuring that you meet your customer's needs is to spend time getting to know or preparing the customer's use cases. For larger projects, this is extended to include an Analysis Phase. This will enable you to present new features to the customer. You can interview the customer to find out what they are doing and why so that you gain a thorough understanding of their business and usage of the program.

You can use the decision tree diagram (see the following section) to find out how the level of customizations in the customer's original version will affect the type of upgrade you choose to implement. An important thing to consider in this phase is what the goal of the upgrade is, such as better usability or new features. This is important so that you can measure the success at the end. Typical things that you need to take into account are: your customer's budget, expectations, and whether the upgrade is a purely technical one.

Decision Tree Diagram

The following diagram shows the different decisions involved in upgrading from Microsoft Dynamics NAV 2.x or 3.x to Microsoft Dynamics NAV 4.x.



Run-time only (A): An upgrade can involve just upgrading run-time in response to a bug, or upgrading purely for performance only. Although this is not generally considered best practice, there can be situations where this type of upgrade is relevant. For example, if you migrate to Microsoft Dynamics NAV 4.0 SP1 to take advantage of the improvements in C/SIDE.

Standard upgrade (B): The customer uses only standard Microsoft Dynamics NAV functionality with standard objects in the database, so there are no objects that are modified. In a pure standard to a pure standard upgrade, you can follow the steps in the "Upgrade Quick Guide", which can be found in the Upgrade Toolkit. Also, see Chapter 3 "Upgrading to Multilanguage" and Chapter 4 "Customizing the New Standard Objects" in the "Upgrade Toolkit" manual.

Standard + customizations (C): When there are customizations involved, there are different possible scenarios:

i. Your customer might have a standard system with customizations that are no longer relevant and therefore doesn't want to upgrade the customizations from the old version. In this case, you can do a standard upgrade of standard data. This situation is similar to the standard upgrade. You can follow the upgrade procedure for upgrading data in the Upgrade Toolkit manual and then discard the data from old customizations. However, you need to make sure that there is no data based on the customization in the database when you do the upgrade, and you have to create some conversion tools to delete the data that is not included in the upgrade.

ii. The functionality of the old customizations is covered by the standard functionality in the new version. For example, the Partner has developed support for multiple dimensions in version 2.x, which is now covered in the standard Microsoft Dynamics NAV granule. In this case, you can use the standard upgrade objects and in addition write some customized upgrade objects (you can also write some conversion tools that can move the data). Customized data means writing your own data that is similar to these tools but matching the specific customizations. Customized objects are not transferred to the new version as they are replaced by the standard ones. Other objects that reference old customization must be changed to use the new standard functionality.

iii. Customizations have been made within the base-application objects. In this case, you can create a new set of modified objects based on the new version, incorporating the necessary modifications. Before you do this, analyze the original purpose and intention of the modifications, and then rethink/reengineer them to fit and utilize the new base application and its features and functions. Consider whether it is an option to buy extra (new or enhanced) functionality from Microsoft or a 3rd party vendor that can solve the problems addressed by the modifications. After this analysis (which should be part of the plan and the quote given), you can plan the actual conversion and develop the conversion tools you need, still using the Microsoft conversion paradigm.

iv. The customer wants to upgrade from an old version with customizations to a newer version with customizations, but the customer does not want all the data. The customer may want to upgrade just the good quality data and discard the poor quality data or data containing errors. For example, ledger entries could be discarded but data about customers, vendors, and articles should be upgraded. In this case, you can follow the standard upgrade procedure for objects only.

Once you have gained an insight into the type of upgrade that your customer needs, you can work out how much time is involved. A good indication of the scope of the project can be gained by looking at the number of customizations that have been made to the original version. The more changes to the original version, the more time is needed for the upgrade.

Best Practice
An important thing to keep in mind is whether it is worth doing an upgrade at all, or whether you would serve the customer better by starting over with a fresh implementation. For example, if you are upgrading from a text-based version, the best practice is to start over with a new implementation because the upgrade is simply too big. The same considerations are appropriate when looking at the customizations; you should look at whether these features are now available in the standard version or in an add-on solution of Microsoft Dynamics NAV or in a partner add-on solution requiring a new version.

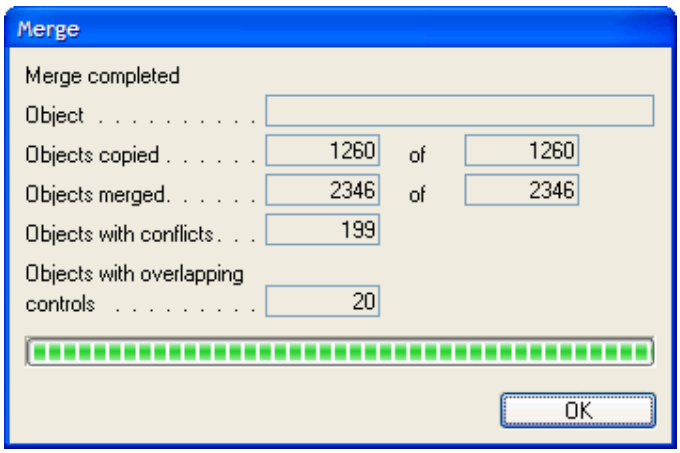
Estimating Time and Costs

When you estimate the time involved in an upgrade project, take into account the deadline that you are aiming for, and the budget. For example, if error correction is the main driver for the upgrade, then the time constraint is a pressing one.

A customer workshop can be a good way of planning the project and in helping you to assess the scope of the project.

In order to give the customer a realistic quote, you need to estimate the size of the project. Remember to consider the license and the additional granules that are needed, and include this in the price.

As a general rule of thumb, an automatic merge can take between 5 and 8 hours to be processed by a computer. The actual preparation before this merge should not be very time consuming. You can use the merge tool (in the Navision Developer's Toolkit) to estimate how much work is involved. The automatic merge process is described in 'Chapter 5 Compare & Merge Tool' of the 'Microsoft Business Solutions-Navision Developer's Toolkit' manual.



You also need to take into account the different phases of the project when estimating the time involved. Estimates provide information to help calculate the budget and identify the resources needed to perform the work. Break down the estimated time required to complete tasks by area and calculate how much time is required for the completion of each phase of the project. The plan should include time estimates for:

- **Design**
- **Development**
- **Testing**
- **Training and documentation**
- **Implementation**
- **Project management**

For larger, complex upgrade projects, we recommend that you propose an **analysis phase**. This might be needed if significant solution design changes will result from the upgrade or be implemented during the upgrade. For example, this phase might be required if additional functionality is now part of standard functionality in a newer version. If necessary, you should prepare a quote for the analysis. Otherwise, the quote should be for the full upgrade.

An analysis leads to:

- **A detailed scope for the existing object customizations to be upgraded.**
- **Specifications for solution design changes.**
- **A document containing interface requirements.**
- **A decision to bring additional ISV Solution and/or modules.**
- **Functional requirements and test plans.**

Based on these documents, a proposal is accepted by the customer and these documents are part of the contract that is then agreed between the customer and you.

Many customers think that an upgrade will take months to implement, when often the deployment can be done over a couple of weekends. The upgrade is done in two parts: first the Partner merges the objects and tests how well the data has upgraded and second, the actual upgrade is carried out at the customer's site.

It is important that you have an accurate idea of the time that is actually involved for the upgrade at the particular customer site. This can vary from one day to one month depending on who they are, what they need, and the functionality and customizations that they have. You should plan the deployment phase to be carried out during out-of-office hours (for example, weekends are good for most customers).

The customer signs off on the agreed scope of the project when they accept the proposal. This can be for either an analysis or a conversion project.

Upgrade Proposal

Possible contents of an upgrade proposal might include:

Executive Summary:

- **Brief summary of the proposal appropriate for a customer executive audience**
- **Key milestones and dates**

Background:

- **Description of the customer's background situation and expectations of the project (to demonstrate understanding of the customer's business and environment)**
- **Upgrade benefits – describe the upgrade value proposition**
- **Reasons for why the customer chose the particular Partner for the upgrade**

Upgrade Overview:

- **Presentation of how the upgrade team plans to address the customer's expectations, project goals, and objectives**

Proposed Solution:

- **Summary of the Project Scope – list of scope items described in the Scope Document and approved by the customer**

Project Timeline:

- **Project schedule - the schedule provides the basis for the customer to verify timelines**

Project Organization:

- **Roles and responsibilities – including knowledge, skills, abilities, and team structure**

Project Management Approach:

- **Specification of project management responsibilities for the project**
- **Communication plan**
- **Issue management procedure**
- **Change control**

Pricing:

- **Statement regarding a fixed price or an estimate based on time and materials**

Project Assumptions:
Describe key assumptions made with respect to the scoping, approach, estimating, and pricing. The assumptions can include, but are not limited to, statements regarding:

- **Schedule, timeline, and cost**
- **Customer performance**
- **Customer or third-party provided information**
- **Infrastructure or application functionality**
- **Data migration**
- **Timely decision-making by customer**
- **Dependencies**

Planning

In the planning phase, you can avoid future problems by identifying and planning for risks. Getting the project right from the outset means that you increase your chances of a successful upgrade project. This phase is mainly driven by Program Manager.

In this phase, you need to develop the following:

- **Development plan**
  This is the part of the project plan that deals with merging customized objects. The developer is in charge of this.
- **Test plan**
  Set up a generalized testing script or test procedures for data validation, data processing, stress and workload, client/server performance, and application functionality. The customer is heavily involved in this part of the project.
- **Deployment plan**
  The Deployment Plan documents core activities that are necessary to effectively deploy the upgrade. It includes an overview of activities necessary to get the upgrade into a production environment such as installation, configuration, and initial operational activities. The customer should be included in this plan to ensure that they have the right resources available, and that the deployment fits with their schedule.

The planning phase should include a Gap Fit Analysis (including a plan of how to treat customizations), and a detailed design of the upgrade solution.

You are ready to move to the next phase when the plans are signed off by the customer.

Gap Fit Analysis

A high-level Gap Fit analysis is the process of identifying the gaps between the old version of Microsoft Dynamics NAV and the newer version of the objects. These can be identified as a result of the decision made using the decision tree diagram as a way of addressing the processes needed by the customer to perform their business. This is best done in a workshop by actually trying to perform the activity in Microsoft Dynamics NAV. For each activity, the workshop participants must decide to what extent Microsoft Dynamics NAV fulfills their business needs. Use cases should be developed to help you identify the critical business processes.

Every discrepancy between the business support needed and the functionality offered must be identified as a "GAP" and every gap must be described in the GAP FIT worksheet accordingly.

For each gap identified, a decision has to be made. The team must decide, following discussions and agreement with the customer, whether upgrading Microsoft Dynamics NAV will ensure that the program accommodates the activity in question. Alternatively, the Partner may consider third-party solutions or customize the current version of Microsoft Dynamics NAV – or even adjusting the particular business processes to fit the standard system. In some cases, this is an overseen option which can be cheaper and still quite efficient in the long run.

Customizations

In the planning phase, you need to plan how you want to use the merge tool if the customization is utilizing the base functionality in a new way.

To get an overview of the customizations that are needed in the upgrade version, talk to the user and have them explain how they work. The user may already be using the program in an unusual way.

Use the Merge and Compare tool to see what has been changed in the original version. You will not be able to see why the changes were made. For this information, you will need to refer to the documentation if this exists, the users, or the original developer. It is important that you gain as much information as possible about the customizations that have been made to the original version. Identify the changes in the application and find out what they do.

Best Practice
Read up on any documentation that exists – for a new customer, develop documentation yourself and look carefully at the code.

Upgrade Design

As a result of the Gap Fit analysis, you might need to create an upgrade design for the object and the data upgrade based on the gaps that you have identified. This will depend on the outcome of the decision tree. The design should describe in detail the changes that you will make to existing objects and how you intend to do this. This should include what can be merged automatically using the Navision Developer's Toolkit and what needs to be changed manually, the objects that can be taken from the new version without any merge and the ones will remain unchanged. The development design is made be the developer and the test design by the tester. The design has the same design process (development plan/test plan) as for new implementations and the same guidelines apply. For example, you can use temporary tables to store data during the upgrade process; you delete these afterwards. These temporary tables are used to convert the existing data to match the new data design. You should follow the same procedure as the one suggested by Microsoft.

You need to have developed an upgrade deployment plan of your own (you can base it on the Quick Guide and customize the objects mentioned in the Quick Guide to your specific project). Evaluate the standard plan and see how it fits. On the day that you make the actual conversion of data, you must have a detailed plan to follow so that it all goes as smoothly as possible. This is particularly relevant when upgrading third-party add-on-applications (particularly when there is more than one) where the sequence of tasks might be very important for the results of the upgrade, or the third-party product might introduce restrictions of its own upon how the entire conversion can be carried out.

If the upgrade is not a standard upgrade, you need to make your own Quick Guide based on the standard upgrade quick guide from the upgrade toolkit. You should include your own upgrade objects in the fob-files, and modify the clean-up codeunit (this deletes objects, that are only used in the upgrade process) to also include your objects. In the Quick Guide, include any steps that need to be carried out. For example, run the objects that delete data in the existing database; run the objects that move data from one table to another, and clean up. While you test the upgrade, you can make notes

about the time used for each step and the size of the database. This will be useful when doing the "live" upgrade.

Best Practice

In this scenario it is good practice to use the standard Microsoft methodology:

1. Remove data from fields that you are deleting (you might have to save some data in temporary tables).
2. Create a batch job to run before you delete all standard objects and relevant tables.
3. Move the data that you want to preserve to the tables specifically created for this purpose.
4. Create another batch job to run after you have deleted all the old objects (using the MBS upgrade tool).
5. Save your own batch job and table objects in two separate object files (one for importing and running before the upgrade of objects and the other for importing after you have deleted all objects).
6. Delete all objects except the tables that contain the data that you need (using the Upgrade Tool). This ensures that the objects that haven't changed will also be included eliminating the risk of forgetting to import/update some objects. It also relieves you of having to decide which objects to import, and at the same time ensuring that any obsolete objects are deleted leaving a clean database.
7. Import all the new objects.
8. Move the temporary data into the new fields.

Development

In the development phase, the customized objects from the old version of the database are merged with standard objects from the new version. The developer is the key driver of this phase. Based on the findings of the analysis in the previous phase, you can start to develop the customizations that are needed in the new version that you are upgrading to.

If the upgrade project is a large one, for example, with complex customizations and performance issues, you will need several developers with different areas of expertise to cover all the features. Different types of upgrades need different levels of skills in developers, so you should either assign junior developers or experienced consultants as appropriate to the project.

When you have finished making these customizations, you can do the following:

1. Save the modified objects in a separate file.
2. Store the data in temporary tables that you will later need to delete or overwrite.
3. Create the batch jobs that will save the old data in temporary locations.
4. Place the saved data from the temporary tables correctly in the new application tables.

The Upgrade Toolkit consists of a set of tools and procedures that are designed to help you upgrade. The tools and procedures that you are encouraged to use vary depending on the version you are upgrading from and to. We suggest you look at the following:

- **C/AL Programming Guide**
- **Application Designer's Guide**
- **Terminology Handbook**
- **GUI Guidelines**

For more information on these, see the section on Resources.

To evaluate your code, you should use the GUI Check Tool to ensure all your code is created according to Microsoft Dynamics NAV standards. This tool checks whether your development is according to the application style guide.

We recommend that you split the development into several short iterations followed by a test of each to show whether the data has transferred correctly. An iteration can last anything from a couple of days to a week.

The upgrade has two parts: the data upgrade (consisting of data preparation and data conversion) and the merge of the customized objects into a new Microsoft Dynamics NAV application.

Note: When you move to SQL server you must verify that all your customizations are SQL compliant. Not only customizations need to be checked but also data formats that are different from Navision Server to SQL Server. These need to be checked using the provided tool (objects). See the Upgrade Toolkit, chapter 7 *"Migrating to the Microsoft SQL Server Option for Navision 4.0"* and section 4.2 *"Locating Illegal Locktable Calls."* You can also refer to the section about how to upgrade in the Microsoft Navision SQL Server Resource Kit. This outlines how to develop for SQL server. Additionally, you can refer to the standard code in Microsoft Dynamics NAV.

You can also refer to the decision tree diagram and the following examples.

Using the Compare and Merge Tool

In the planning stage, you decided how you wanted to use the merge tool if customizations are utilizing the base functionality in a new way. This involves manual work, where you go into the new version and manually find out how the customizations are being used. If you make a note of these customizations, you can use these as a starting point for the actual merge.

For the standard forms that have been changed, the recommendation in the Navision Developer's Toolkit is to merge them. However, if there is modified code in the tables, this does not always merge satisfactorily. In this situation, it is therefore better to manually copy the changes in the Microsoft Navision client.

In the Developer's Toolkit you only see properties, not the actual form. Because modifications are not always documented and customers can be sensitive about their interfaces, it is not always a good idea to upgrade the forms and reports. Instead you can migrate the Microsoft Dynamics NAV changes (if any) since these are comparatively well documented and therefore easier to identify and migrate.

You must decide whether to customize standard forms or create new ones. Both of these options are worth considering as there are advantages and disadvantages to both. If a form hasn't been changed too much and the fields are on the original tabs, this makes it easier to customize. In addition, take into consideration how this decision will affect future upgrades.

The Microsoft Dynamics NAV Form Designer is an excellent tool for rapidly designing new forms, so consider building smaller componentized forms instead of changing existing forms, in order to minimize the upgrade work. Having built smaller componentized forms, it is possible to upgrade the parts independently and it is not necessary to consider the interaction between information.

Best Practice
Consider the following best practice concerning reports to minimize upgrade and migration work:

If it is necessary to add business logic (C/AL code) to a report, consider adding the business logic in a codeunit and use the report as the "presentation" layer. By separating the business logic from the report, it is possible for parts of the code to access the code in the codeunit and even make a call through the Application Server.

How to Handle Standard Objects (Standard Upgrade)
When you export from an existing customer database, you can either export:

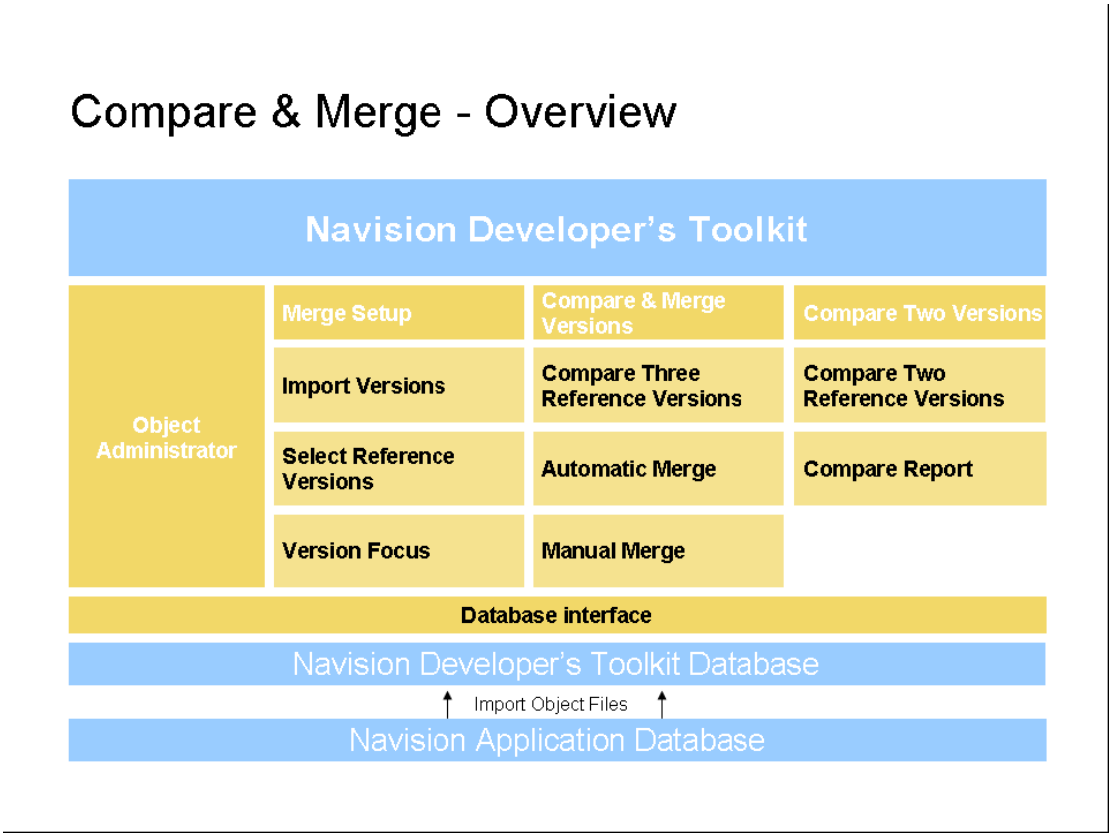
a) only changed objects, or
b) all objects if you don't know what has been changed and there is no documentation.

Warning: Although option a) is the best practice, you might not know what the changed objects are. If you do not export all the objects that have been changed (these are indicated by the modified flag or because the version list is different to the standard objects), the merge will not be successful because the resulting code will contain errors. A bug will be found and this could even damage your data. Usually this is detected during the compile of all objects. If this inconsistency is not detected through the compile, it can damage the customer's data during run-time. If you cannot be sure that by filtering on the version list you get all the changed objects, then you should choose option b).

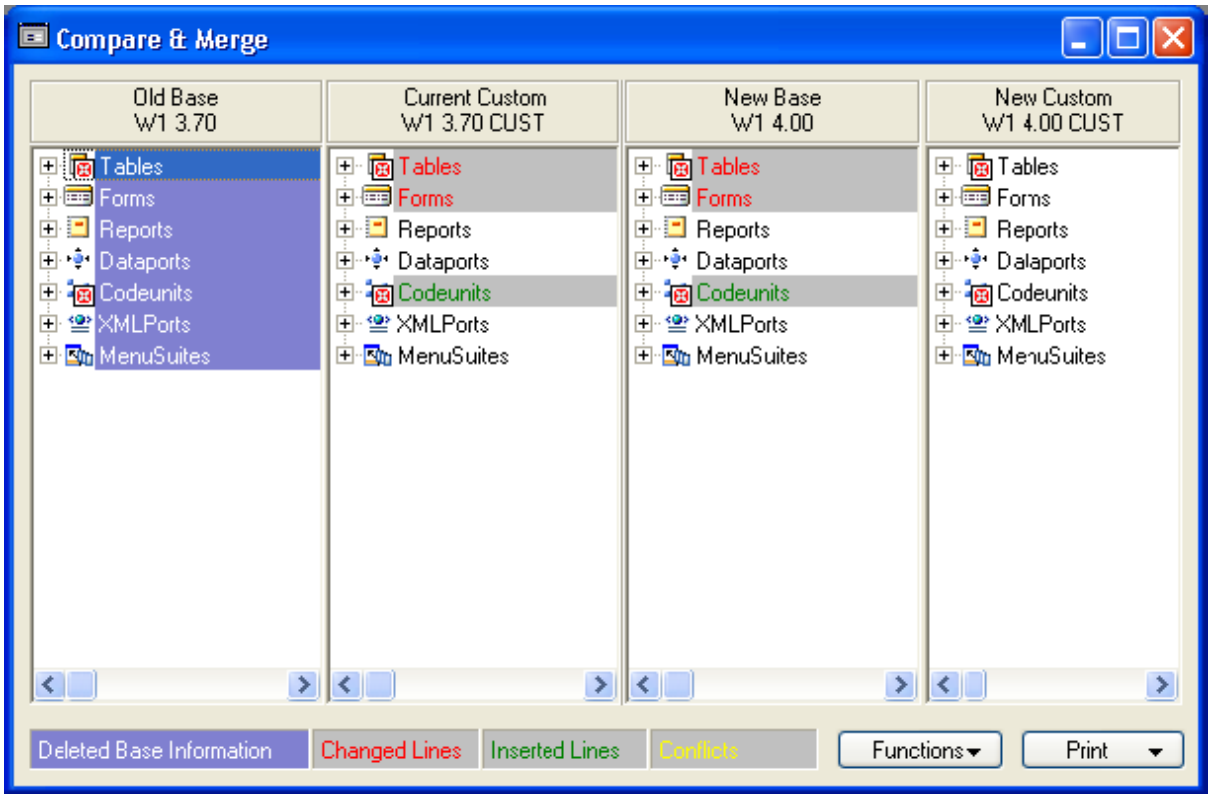
To avoid problems, before exporting only the changed objects, you should use the merge tool to compare the customer's version with the standard version. This will show you the objects that have been changed and identify the differences – and then only use the merge tool to merge the changed objects. To quickly find out which objects have been changed, you can use the Compare Two Versions option of the Navision Developers Toolkit to compare the old standard version of the database with the current customer version. This way, you will identify the changed objects and you can mark them as such using the Version List field in the Object Designer.

Easy Merge of Code/Objects

The Compare & Merge tool is designed to compare three different versions of Microsoft Dynamics NAV objects and to merge these versions into a fourth version. The Compare & Merge tool is based on the structure of a Microsoft Dynamics NAV object. This structure is used to apply compare and merge rules that will lead to an improved and structured merge result.



The following **Compare & Merge** window illustrates an example merge:



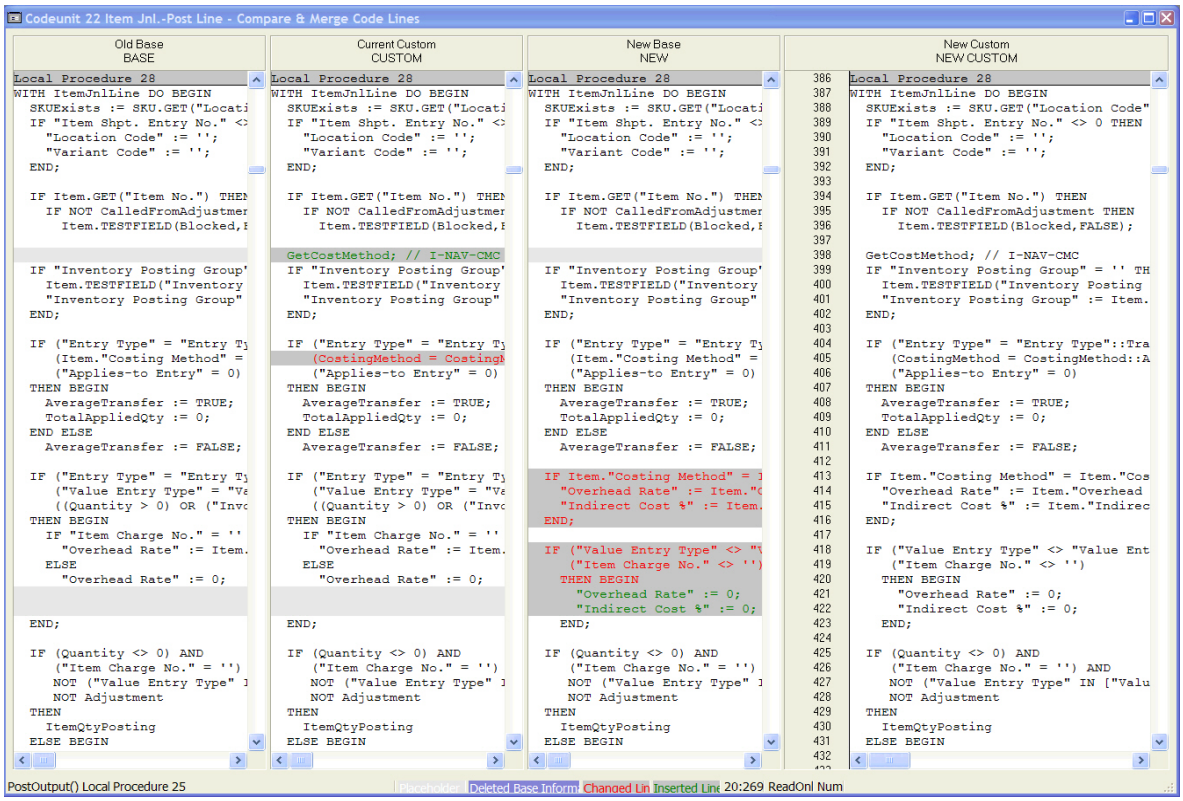
All columns are synchronized vertically and horizontally. You can expand objects to see the details and collapse objects to reduce the object details shown in the **Compare & Merge** window.

The colors reflect differences between the reference versions. A conflict bitmap indicates that the program could not merge automatically. You should always check the suggestion in the New Custom Version and either mark this suggestion as accepted or change it.

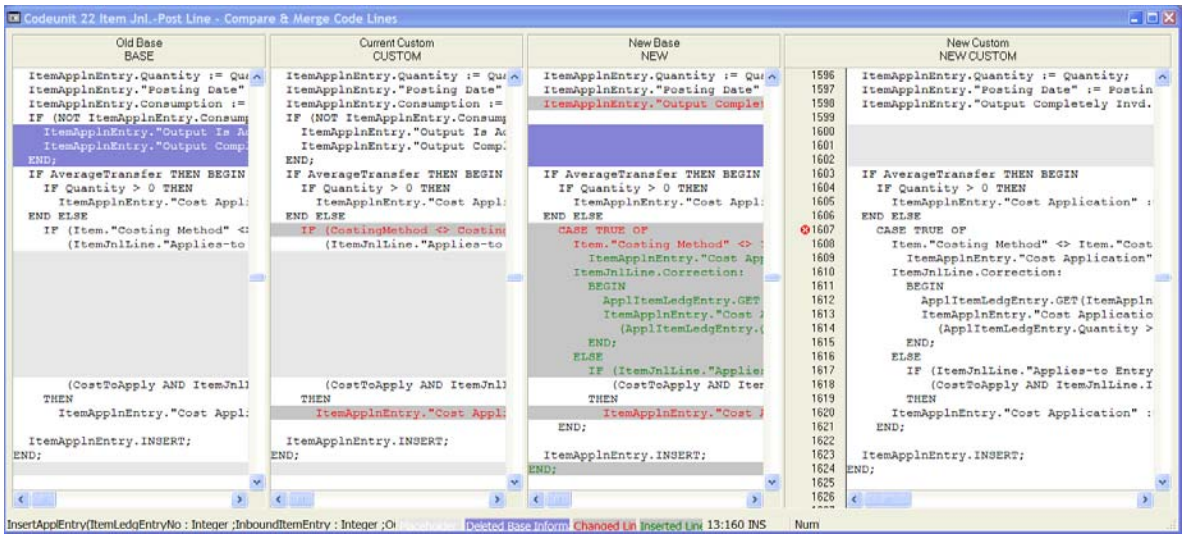
You can change all details of an object in the New Custom Version only. If you want to undo your changes in an object, you can click **Remerge** in the shortcut menu on the object level. This remerge will overwrite all your changes in this object.

Example 1: Code Merge: Changing the Codeunit (Easy Situation)

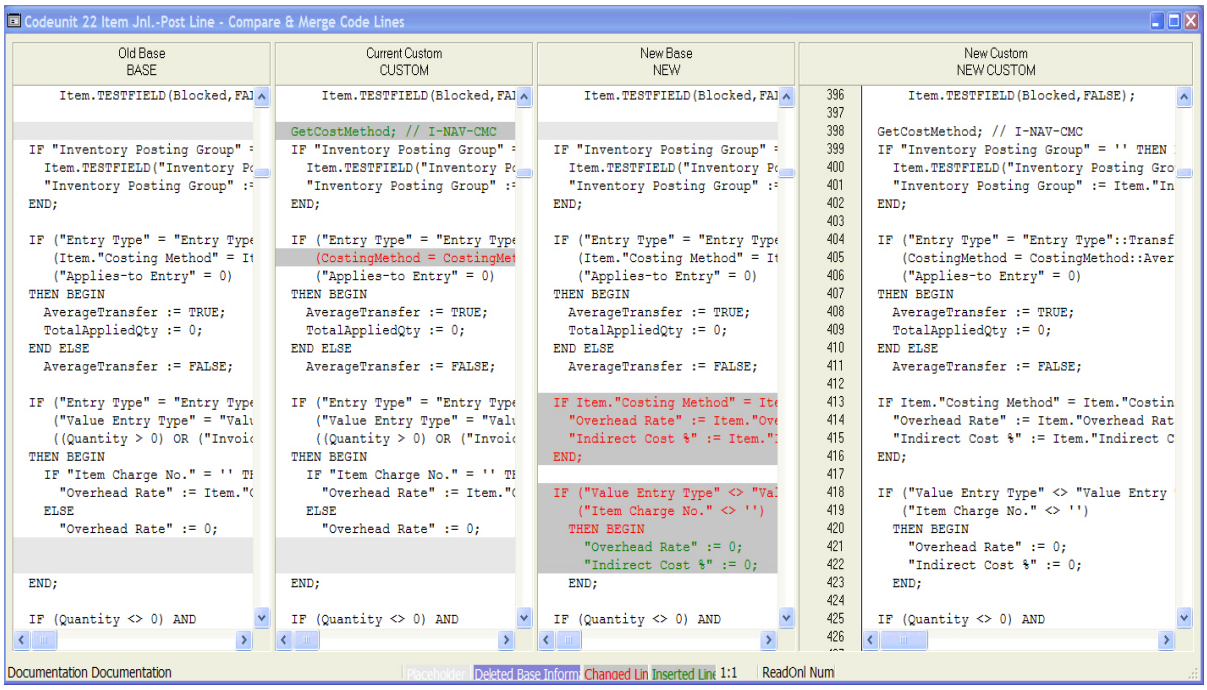
The following examples are for three possible code merge situations. The customization that is merged here is in codeunit 22 and replaces the standard Microsoft Dynamics NAV item costing method with a new, customer-specific method. The first 3 columns show the old standard version, the customer's version, and the new standard version. The last column shows the new version, based on the new standard version and the customer's changes – done by the automatic merge.



This is a simple code merge. The customized code does not have a conflict with new code and therefore the result after the automatic merge in the new custom version is fine.



This is a typical example in an upgrade where you have to make a decision about the new code. The new version has been changed in the same place as where you have made a customization. Therefore, the Developers Toolkit does not merge the customization and you have to decide where to place the customization by editing the "New Custom Version".



This is the most challenging scenario for a code merge. There is no obvious conflict between the three versions. However, in the new version, the new code affects the business logic of the customization because it introduces the item costing method in a new place in the codeunit. It is not difficult to resolve this issue by editing the new custom version, but there is no warning that there is an issue. You will only become aware of it when you manually go through all the code changes or when a test case discovers it as an issue. This highlights the need for building test cases and trying them out with the customer. In some cases, this can be the only way to discover problems or malfunctions, and the quality of the rest of the upgrade is also greatly improved.

Best Practice
In this situation, you can do a search for the "Costing Method" field in the old base version and the new base version using the "Where Used" function in the Navision Developer's Toolkit to find all the places where the field is being used. Then, you can take the delta of these two where-used results to look through all the new places in the code where the field is being used in a new way.

Example 2: Forms and Reports

After finishing the merge of forms, the Compare & Merge Tool automatically checks forms for overlapping controls. The result is shown in the status window after the automatic merge process. In the Compare & Merge window, click Print and click Overlapping Controls to see a list of overlapping controls in forms and request forms in reports. These overlapping controls can be rearranged either directly in Microsoft Dynamics NAV or in the Navision Developer's Toolkit by changing the properties.

Complex Merge Situation

There are, however, other situations where you cannot just use the Compare and Merge tool. For example, if a complete function has been moved to another part of the system and this has been customized. In this case, you need to do manual work. However, this scenario is the exception. The

older the version you are upgrading from, the more likely this situation is. If you have upgraded regularly then this is less likely.

An example where manual work is needed is with dimensions. In Navision Financials there were no multi-dimensions; in Navision Attain 3.0 this was changed. Therefore, the two dimensions: Department and Project had to be changed everywhere in the application.

Best Practice
It is very important to know what has been changed from the standard and why. If, for example, you developed your own customizations because these were not available in the standard version, then, when upgrading such a database to a version which already includes similar functionality as part of the standard application, then you need to decide what to do with your customizations. The best practice is for you to replace your customizations with the standard version. This involves changing code in many places. It could also mean changes in the design of your customized solution.

How to Deal with Customizations

For complex merge situations, such as replacing your version of dimensions with the standard functionality, we recommend that you use the 'Where Used' functionality in the Navision Developers Toolkit (NDT) to prepare a list of code that has been changed. This means using the Where Used function for:

- **Tables that contain the data for the your version of dimensions**
- **Fields that have been added to standard tables needed for this customization**
- **Your codeunits that are used**
- **Standard functions that were modified to support your version of dimensions**

After you have the list of changed code, you can prepare a migration plan from the existing version of dimensions to the standard version. This should include the redesign of code and transfer of existing data. When required changes for moving to the standard functionality are implemented, this code can be used as objects of the customer's current database as described in Upgrade Toolkit.

Test Deployment

Testing helps identify any compatibility problems and will help you validate data and organize the upgrade process. This phase entails establishing a test environment and composing validation scripts and application functions to confirm a successful upgrade.

You should ensure that you have upgraded the client license to include the upgrade and other versions/modules. Also check that your customer has rights for any new tables and forms that you have created (all new objects). When you are moving data around:

- **Create your own mini Quick Guide outlining the steps you are going to take. Base it on the one in Upgrade Toolkit including the specific changes you've made in the object file.**
- **Create your own object file (clean up: move data and remove temporary tables).**
- **Go through use cases (see the test plan). Run through the same infrastructure as the customer.**

If the upgrade is not a standard upgrade, you need to make your own quick guide based on the standard upgrade quick guide from the upgrade toolkit. Include your own upgrade objects in the fob-files, and modify the clean-up codeunit (which deletes the objects that are only used in the upgrade process) to also include your objects. In the Quick Guide, include any steps that need to be carried out, for example, run the objects that delete data in the existing database; run the objects that move data from one table to another, and clean up.

While testing the upgrade, make notes about the time used for each step and the size of database. This will be useful when doing the "live" upgrade.

Iteration for test deployment:

1. Run through the upgrade document that you prepared during the development phase, which is based on the Quick guide.
2. Proceed with the deployment.
3. Verify results.
4. Run test cases.
5. Verify results.
6. Fix errors either in the data or code.
7. Repeat the iteration until there are no errors.
8. The customer signs off on the test deployment (the customer agrees that the functionality is ok).

The next step is to train the customer in the new functionality. You could offer this in the form of a workshop.

Use the information from the test deployment to fine-tune the timing of your upgrade plan and revise your planning if necessary. Make sure that you test permissions and test using the client license. After the customer has signed off on the key functionality/data, you can "go live" with the new system with the upgrade.

Updates to Documentation

It is very important that you document all the changes you make to ease future development and upgrades. It is worth keeping a log of changes of the upgrade, describing what you did and why. You should also update your design document, quick guide, version list, and online Help during the upgrade.

Any changes made in the objects needs to be documented in the Documentation Trigger including a reference to the code. The version list should also be updated. If this is done every time an object is modified, the next upgrade process will be much easier, because the modifications will be easily identified.

Best Practice
The best practice is to assign a number to each modification made and mark this in the code. This documentation also reduces the number of bugs, because writing the changes down forces you to think about why and how a change was made. See the following example:

Somewhere in codeunit 80 :

```
IF SalesLine.Type = SalesLine.Type::"Fixed Asset" THEN BEGIN
  SalesLine.TESTFIELD("Job No.",'');
  SalesLine.TESTFIELD("Depreciation Book Code");
  DeprBook.GET(SalesLine."Depreciation Book Code");
  DeprBook.TESTFIELD("G/L Integration - Disposal",TRUE);
  FA.GET(SalesLine."No.");
  FA.TESTFIELD("Budgeted Asset",FALSE);
  //CITPPVL20060309001 >>
  UpdateResource(FA."No.");
  //CITPPVL20060309001 <<
END ELSE BEGIN
  SalesLine.TESTFIELD("Depreciation Book Code",'');
  SalesLine.TESTFIELD("Depr. until FA Posting Date",FALSE);
  SalesLine.TESTFIELD("FA Posting Date",0D);
  SalesLine.TESTFIELD("Duplicate in Depreciation Book",'');
  SalesLine.TESTFIELD("Use Duplication List",FALSE);
END;
IF ("Document Type" = "Document Type"::Invoice) AND (SalesLine."Shipment No." <> '') THEN BEGIN
  SalesLine."Quantity Shipped" := SalesLine.Quantity;
  SalesLine."Qty. Shipped (Base)" := SalesLine."Quantity (Base)";
  SalesLine."Qty. to Ship" := 0;
  SalesLine."Qty. to Ship (Base)" := 0;
END;
```

In the Documentation section of codeunit 80

```
Documentation()
//CITPPVL20060309001 >>
Change made to better integrate Project Management resources with FA
Symptoms : N/A
Problem : When purchasing a new asset we cannot automatically just use it as a resource in our projects
Solution : Whenever posting to a fixed asset, the resources are automatically updated
//CITPPVL20060309001 <<
```

Deployment

The Deployment phase is often described as the "turnover to the new system" or the "Go Live" phase. A successful deployment phase depends largely on the work done in the previous phases.

When you are ready for the implementation of the upgrade, it is important to use the structure provided in the Upgrade Tool Kit and Developer Tool Kit to ensure a smooth implementation.

In the deployment phase, you carry out the actual implementation of the upgrade. This includes ensuring that the customer is aware of the next steps and has a full overview of the process and their own involvement in it.

Upgrading the Database

Upgrading the database occurs at the customer site during out-of-office hours, (typically weekends, holidays, or evenings).

Before starting, you must switch off all clients and back up the existing system and store it in a separate location. Make a file-level copy of the full database by running the HotCopy program (this is installed when you install the Navision server – for more information, see "Making Database Backups in Microsoft Business Solutions–Navision" on the product CD), so that if you need to restore the backup, all the indexes are in place (in case you need to roll back). On the SQL server side, do a full SQL backup. Test your backups by running a "test restore" before starting to ensure that you have a working backup. You should run in maintenance mode when backing up.

Follow your Quick Guide to help you with upgrading the data. Update client, servers, and licenses.

Mitigation

Be prepared for what can go wrong and have pre-prepared strategies to resolve issues. For example:

- **Have extra hard drives available if you need them.**
- **If possible, use a test server as a backup production server running the old version of the system so you can switch to this system, while doing the actual restore on the production server.**
- **For large native databases, have a copy of database files on extra disks. If the upgrade is not successful you can use this copy of the database where all indexes are already built.**
- **For large SQL databases, prepare a SQL Server backup with all indexes already built.**

If you run into problems with the upgrade, you need to ensure that you have enough time to roll back to the old production environment. Remember that a restore can take several days (with the Microsoft Navision client).

Best Practices for Restoring

- **If using a test server as a backup production server, switch to this system.**
- **If using a native database, copy the backup copy of database files to the production server.**
- **If using SQL Server backup, do low-level SQL restore.**

Migrating to SQL Server

Use the following documents to refer to how to install SQL server and how to install Microsoft Dynamics NAV on the SQL Server:

- **"Installation & System Management: Microsoft Business Solutions–Navision® SQL Server Option" on the Microsoft Dynamics NAV CD**
- **The white paper "Microsoft Business Solutions–Navision SQL Server Option Resource Kit," which is part of the "Microsoft Navision SQL Resource Kit"**
- **"Performance Troubleshooting Guide for Microsoft Business Solutions–Navision"**

These resources cover the additional things you should do when upgrading with SQL server.

Best Practice
The best practice is to upgrade to the latest version before migrating. It is also important to know the differences between the different databases.

One approach is to migrate to the SQL server in two steps: Perform an upgrade to a native database. And after a testing period you can migrate to SQL server.

You can plan to do these steps on two consecutive weekends, for example.

Testing

You must test your deployment. Set up a local upgraded database and give your key end-users a short introduction to what has been upgraded, and then turn them loose on it for testing. (The users participating in the test must have been through user training before performing the test.) You can run the same use cases as discussed in the planning stage.

Do not install the test version on the customer's server; also note that the testing must be done with the customer's license, so that problems with the license will be discovered before "Go-Live."

Issues may be found in the test. If something is not working you must have resources at the customer site available to resolve problems. You should review issues and make sure they really are problems with the upgrade rather than a change in requirements. Do not accept a change in requirements as a bug. Make sure that all valid bugs are logged.

Once the issues have been resolved, let the users perform further testing. Finally, get the customer to sign off on completion of the work and proceed to the "going live" stage.

Going Live

The program manager (or project manager) at the customer site signs off on whether the going live is a success or not. When going live, you should provide a business consultant to be at the customer site to help the users get started. If there are big changes from the old version, the business consultant should be there for more than one day. If the deployment was not a success you have to roll back. Allow time to roll back if there are problems and ensure that you have consultants available at the customer site when going live.

The project is not considered complete until the customer has given a final approval. This is due once all processes are signed off and approved, and all bugs are resolved.

Follow Up

After going live, you must follow up with the customer on any pending items that should be closed and, most importantly, the customer's satisfaction with the project – this must be measured and documented.

The activities in this phase are on-going activities performed after the project is closed and throughout any future involvement with the customer.

When following up with the customer, you can gather information about issues that might exist and decide what to do next. Your sales representative could call the customer and compile a list of outstanding things to do. You can also plan in advance that an upgrade consultant visits the customer a couple of weeks after the upgrade to see how things are working and help with any issues.

Evaluating Upgrade Success

Evaluation of the upgrade is both for you and your customer's benefit. We recommend that you carry out two evaluation meetings: one internally and another at the customer. At these meetings discuss the goal of the upgrade and verify that it was reached. Consider the following questions:

- **Were the problems solved by the upgrade?**
- **Could it have been done better?**
- **What did we do right?**
- **What did we do wrong?**
- **Did we meet the customer's expectations?**
- **Are we learning lessons again that we thought we'd learned years ago?**

In addition, look at your internal processes and refer to your success criteria – were they reached? Validate the different phases of the project – for example, were there problems in the planning phase?

Key Learning Points

After reading this white paper, you will have learned the following:

- **There are many advantages to upgrading, both for you and your customer, and especially to upgrading regularly.**
- **Understanding the data model and having an in depth knowledge of the current and newer version of the product before embarking on an upgrade project are essential.**
- **The importance of ensuring that you have identified and assigned all the roles to address the different phases of the project.**
- **The success of the upgrade can be ensured by using the envisioning phase to plan and define the success criteria, identify the business processes, and understand the scope of the customizations.**
- **For a larger upgrade project, spending time on an analysis phase including the scope and specifications for the upgrade, as well as functional requirements and test plans, is essential.**
- **Ensure that you have an accurate idea of the time that is actually involved in the upgrade and formalize this by preparing an upgrade proposal.**
- **Spending time in the planning phase allows you to avoid future problems by identifying and planning for risks (by conducting a gap fit analysis) and preparing a development, test, and deployment plan.**
- **In the development phase, use the best practices in the form of the tools and procedures that are available as part of the Upgrade Toolkit to help you.**
- **Use the GUI Check Tool to ensure all your code is created according to Microsoft Dynamics NAV standards.**
- **Before exporting only changed objects, you should use the merge tool to compare the customer's version with the standard version. This will show you the objects that have been changed and identify the differences – and then you only need to use the merge tool to merge the changed objects.**
- **In complex merge situations, the best practice is for you to replace your version of dimensions with the standard version. This involves changing code in many places. It could also mean changes in design of your customized solution.**
- **During test deployment, you need to check that you have upgraded the client license to include the upgrade and other versions/modules. Also you need to check that your customer has rights for any new tables and forms that you have created (all new objects).**
- **To prevent overwriting and overlapping, it is very important that you document all the changes you make during the upgrade.**

Appendix

Upgrade Checklist

Prerequisites	Know the functionality of all the relevant versions	
	Research compatibility and functionality changes	
	Understand data model	
	Ensure your organization has the necessary resources to cover the required roles	
Diagnostic	Define success criteria for upgrade and agree this with customer	
	Develop use cases to reflect the customer's business processes	
	Define scope of project	
	Include a customer workshop (if necessary)	
	Estimate time and costs	
	Prepare an upgrade proposal (use decision tree diagram to help you do this)	
Planning	Create a development plan	
	Develop a test plan	
	Develop a deployment plan	
	Perform a Gap Fit analysis	
	Plan how you use Merge tool	
	Read existing and develop new documentation for the new functionality	
	Develop a detailed upgrade design	
Implementation	Use the structure provided in the upgrade and developer toolkits	
	Use GUI check tool to ensure code meets Microsoft Dynamics NAV standards	
	Split development into short iterations followed by tests for each iteration	

Test Deployment	Establish test environment	
	Compose validation scripts and application functions to confirm successful upgrade	
	Upgrade client license to include the upgrade and other versions/modules	
	Check that your customer's license file has the rights for all new objects	
	Create your own Quick guide	
	Create your own object file	
	Follow iteration for test deployment	
	Train customer in new functionality	
	Test user rights	
	Fine-tune Deployment plan based on test	
	Document all changes on reports and forms and update design document	
Deployment	Switch off all clients and back up the existing system and store separately (run in maintenance mode during backup)	
	Make a copy of full database (and if relevant do a full SQL backup)	
	Test your backups by running a test-restore	
	Follow Quick guide and restore everything	
	Update the Microsoft Dynamics NAV client and licenses, as well as servers	
	Test your deployment (do not install test version on customer server)	
	Resolve any issues	
	Get customer to sign off on completion (Roll back if necessary)	
Follow up	Follow up with customer and evaluate upgrade success	
	Internally evaluate project success	

Resources

Partner Source

On Partner Source you can find all the resources you need.

Application Designer's Guide

You'll find the Application Designer's Guide on the Microsoft Dynamics NAV Product CD. This is a very comprehensive document that all of your developers should read.

Development Training Guides

Development I - C/SIDE Introduction Training (Course 8359B)
Development II - C/SIDE Solution Development (Course 8401A

Changes Doc

C/AL Programming Guide

You can find this on the Microsoft Dynamics NAV Tools CD. Particularly relevant are the guidelines outlined in Chapter 2: Naming Conventions and Chapter 3: Number Conventions.

Developer's Tool Kit

This manual describes various aspects of using the Navision Developer's Toolkit in your daily work. It provides an overview of what the Navision Developer's Toolkit contains what it can do, and how it works. All the examples in this manual are based on Microsoft Dynamics NAV version W1 4.00.

This manual does not cover every detail of the Navision Developer's Toolkit. If you need to find out more about certain fields or windows that are not described in this book, you can use the online Help, which provides guidance for all the windows in the program.

GUI Check Tool

This tool helps you check the GUI in your application. The guide is a step-by-step guide to check your GUI.

GUI Guidelines

Prior to version 4.0, Microsoft Dynamics NAV maintained strict GUI Guidelines to support developers. Although these guidelines were not updated for 4.0, the content remains valid. You can locate the GUI Guidelines on the Microsoft Dynamics NAV Tools CD. You should make sure that your developers review this guide. It is an excellent tool for understanding the strict standards inside the Microsoft Dynamics NAV application. The benefit of using the GUI Guidelines is that you can check them with the GUI Tool provided on the Microsoft Dynamics NAV Tools CD. Be aware that this tool is consistent with the GUI Guidelines prior to Microsoft Dynamics NAV 4.0, so the standard Microsoft Dynamics NAV code that has new user interface features will show up in the tool.

How to Structure Your Indexes

This information is part of SQL Server resource kit.

How to Design Code for Easy Upgrade

See the Application Designer's guide and Solution Developer training.

Installation Guides

You can find these on the Product CD.

Microsoft Dynamics NAV Rapid Implementation Methodology (RIM)

Only for scenarios when migrating data.

Performance Troubleshooting Guide

The Performance Troubleshooting Guide is part of the Microsoft Navision SQL Resource Kit This material is designed to help you identify performance problems in a Microsoft Dynamics NAV application. It describes how to troubleshoot on both Server options and describes and explains how to use the debugging tools that exist in Microsoft Dynamics NAV to identify performance problems. It also describes how to use the troubleshooting tools that come with this guide. Furthermore, it contains a brief description of some of the Microsoft SQL Server tools that you can use.

This document describes some of the most common performance problems and the reasons that can cause them. The topics covered include:

- **The Client Monitor and the Code Coverage tool**
- **The new performance trouble shooting tools**
- **Hardware setup and performance**
- **Setting up the test environment**
- **Identifying the clients that cause performance problems**
- **Profiling a task with the Client Monitor**
- **Identifying the worst server calls and the keys and filters that cause them**
- **Identifying the tasks that cause deadlocks on Microsoft Business Solutions–Navision Database Server**
- **Using the SQL Error Log to identify the clients involved in deadlocks on SQL Server**
- **How to identify locking problems**
- **How to set up locking order rules and check whether or not your application follows these rules**
- **Identifying index problems on SQL Server**
- **How to identify bad C/AL NEXT statements on SQL Server**
- **How to use Excel pivot tables to get an overview of the data in the Client Monitor.**

SQL Server Resource Kit

Use this tool for planning your SQL implementations. This toolkit contains the following;

- **Application Benchmark Tool**
- **DB Sizing Worksheet**
- **SQL Server Resource Kit**
- **Performance Troubleshooting Guide**
- **Objects to use the tools discussed in the above material.**

SQL Installation Guide (Installation and System Management: Microsoft Dynamics NAV SQL Server Option)

This manual is part of the Product CD and describes how to install and maintain the SQL Server Option for Microsoft Dynamics NAV. However, we recommend that the installation and customization process is carried out with the assistance of a Microsoft Certified Partner or by someone who has Microsoft SQL Server training.

Terminology Handbook

This can be found on the Microsoft Dynamics NAV Product CD. It is an excellent guide for terminology in the C/SIDE development environment. In particular, see the section on the naming of objects in Chapter 4 and the Terminology Review Checklist in section 7.4.

Upgrade Tool Kit

This tool kit contains the information necessary to successfully upgrade from Navision Financials®, Navision Manufacturing, Commerce Portal, Commerce Gateway, Navision Attain® or Microsoft Dynamics NAV (formerly Microsoft Business Solutions–Navision) to Microsoft Dynamics NAV 4.0 and to migrate to the SQL Server Option for Microsoft Dynamics NAV 4.0.

Upgrading to Microsoft Dynamics NAV 4.0 is the first step in the process of migrating to the Microsoft SQL Server Option for Microsoft Dynamics NAV 4.0. The tools described in this tool kit are located in the Upgtk folder on the product CD. You will find a detailed log of the differences between the specific products and Microsoft Dynamics NAV in the changes.doc file that is located in the same folder.

User Rights Setup

This tool is available on the tools CD, in the User Rights Setup Folder. The document provides short descriptions of the central forms and functions of the User Rights Setup feature and some procedures for effective use. It includes the following sections:

- **Introduction**
- **User/Role Combinations form**
- **User Rights Setup form**
- **Give Permissions wizard**

Microsoft