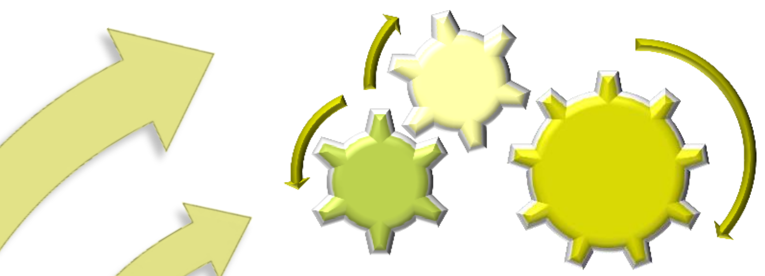
REMAINSOFTWARE

WHITE PAPER

What IT Departments Should Know about Packaged-Software Change Control

Effectively managing changes to customized third-party software



Purchasing off-the-shelf software is frequently the most cost-effective way to fulfill business requirements, but packaged applications must often be customized to meet unique needs. This presents a challenge when the vendor delivers an upgrade. When implementing the new version, you need to ensure that your customizations and the vendor's revisions both remain intact and function properly.

© REMAIN SOFTWARE, 2015

The IT department's dilemma: Retrofitting upgrades into customized packaged software

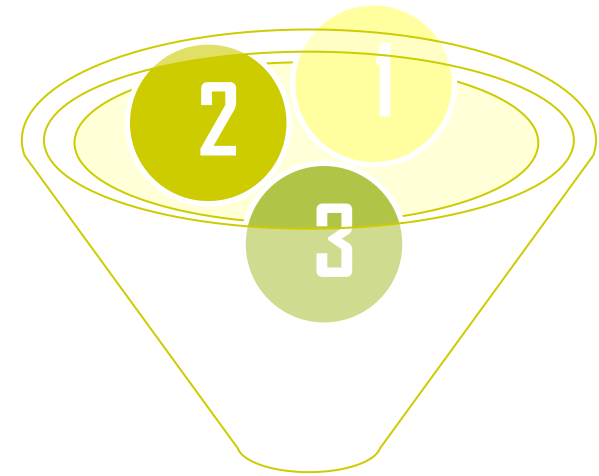
Packaged software offers at least two advantages. For one, by spreading development costs across all customers, vendors can sell the software at a cost that is lower than what a customer would have to spend to develop it from scratch, while still generating a profit for the vendor. And, second, customers can offload the costly and technologically challenging job of fixing bugs, delivering new and enhanced functionality and adapting to modern technologies and security standards.

As a result of these advantages, packaged software plays an important role in many organizations' application portfolios, sometimes comprising the entire portfolio.

Yet, off-the-shelf software also carries at least one serious disadvantage. It is typically delivered in a one-size-fits-all format that delivers identical functionality to all customers. Consequently, organizations may find that they have to adapt their business processes to the software, rather than the other way around. And if the organization has unique, proprietary processes that give it a competitive advantage, the unmodified application likely won't support those processes.

In an attempt to achieve the best of both worlds, many organizations customize packaged software to make it compatible with the organization's processes and to better align the application with its business objectives.

This customization creates a problem. What happens when the vendor upgrades its application? If a customer sees sufficient benefit in the upgrade it has to reconcile three different versions of the software: the original vendor version, the customized production version and the upgraded vendor version. The IT department must somehow implement the delta changes between the original and new vendor versions, without overwriting or breaking the customizations.



With the aim of providing a quick insight into the related challenges, pitfalls and solutions, this white paper describes the problem, its history and the best practices for the mass retrofitting of two systems—the vendor upgrade and the customized production version—with a common ancestor.

History

ERP application suites have become a mainstay in many organizations using IBM i. ERP brands in this market include LX (formerly BPCS), MAPICS, JD Edwards, System 21 and PRISM—and those are just the market leaders; there are others. And, while the ERP is usually the application at the heart of businesses operations, it obviously does not represent the full breadth of the packaged software market—far from it. Because of its central role in the business, for the sake of brevity, the remainder of this white paper will refer primarily to ERP software, but the discussion can be applied equally to all packaged software.

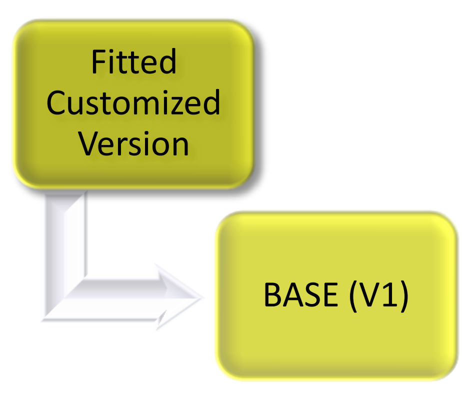
When customers acquire a software license for a comprehensive business application, such as an ERP, they often also obtain, as part of the contract, authorization to change the source code for internal use. This enables them to customize the software to best fit their workflows.

As stated in the introduction and as will be expanded in the following section, when organizations customize software packages they undertake a significant challenge when trying to keep up to date with vendor-issued upgrades and patches.

The Challenge

Your ERP diverges from the base code supplied by the vendor from the moment you implement your first customization. With each new customization, the code slowly drifts farther from the version supported by the supplier.

When the vendor issues a patch or an upgrade to the base code, if you want to adopt the new version you must integrate your revisions into the new vendor version or integrate the vendor's revisions into your customized production version. If you fail to do so, you will lose your customizations.



Customization Equals Divergence

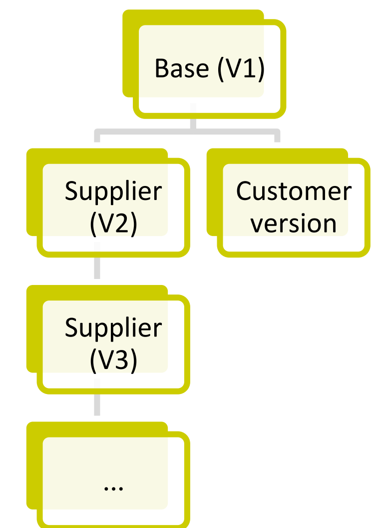
By definition, the benefits of the customizations were sufficient—or even possibly inescapable business requirements—for you to make the investment necessary to develop, implement and maintain them. Thus, forfeiting the customizations is likely not an acceptable option.

Another part of the problem is that the available testing tools are still inadequate, making the integration of vendor revisions with customized code a risky proposition.

Consequently, applying vendor-supplied upgrades to a customized application can be a major challenge, providing a strong incentive to follow the old adage, "If it ain't broke, don't fix it." As a result, upgrades may be put on the shelf. The production code then gets further and further from the vendor's current version as vendor revisions keep coming.

© REMAIN SOFTWARE, 2015

This is not to say that customers who customize packaged applications never install vendor upgrades. Sometimes, the need to do so is too great to ignore. For example, just before the turn of the century there was a large surge of implementations of vendor patches to deal with the widespread Y2K problem. Judging from the fact that the world as we know it didn't implode—as some of the gloomiest of prognosticators predicted it would—that mass-patching effort was successful.



Divergence increases with each vendor-supplied revision

Today, more than one and a half decades later, pressure to upgrade to the vendor's current version level has built up to the point where many organizations can no longer ignore it. This time, the arguments in favor of becoming current are more varied. They include the benefits of modernization, the value that can be derived from new functionality, the need to increase security or meet new regulations, and the desire to keep the vendor's warranty in force, to name but a few.

When the decision is made to implement the vendor's current version, the effort required to reconcile it with any customizations applied in-house, along with the risks incurred when doing so, may be colossal. A solid strategy and effective tools are required to perform the work efficiently and successfully.

Strategy for Retrofitting Upgrades into a Customized Version

When the pressure to implement a vendor-supplied upgrade increases to the point where the benefits that the upgrade offers outweigh the challenges and costs that the implementation imposes, an effective strategy for retrofitting the upgrade into your customized source code includes the following steps:

1. *Find the affected source code*
2. *Compare the sources and merge the version differences*
3. *Compile the merged source code*
4. *Retrofit dependent systems*
5. *Test*
6. *Deploy*

Technology Options

Source compare-and-merge programs are the most commonly used solutions to support the management of multiple application versions. In this section, we present the output of the following three source compare-and-merge programs as possible alternatives in your strategy to retrofit vendor-supplied changes into a customized version of an application:

MRGSRC - This program is part of the ADT (Adapter Development Tool) that runs on the IBM i.

KDiff3 - This open source program runs on your PC and is able to do a three-way merge.

TD/OMS Fusion Pro (Source Change Control) - This Remain Software solution was developed primarily to overcome the challenge of mass-retrofitting vendor changes into customized sources.

Find the Affected Source Code

Ideally, you will have the following three versions of the application source code at your disposal:

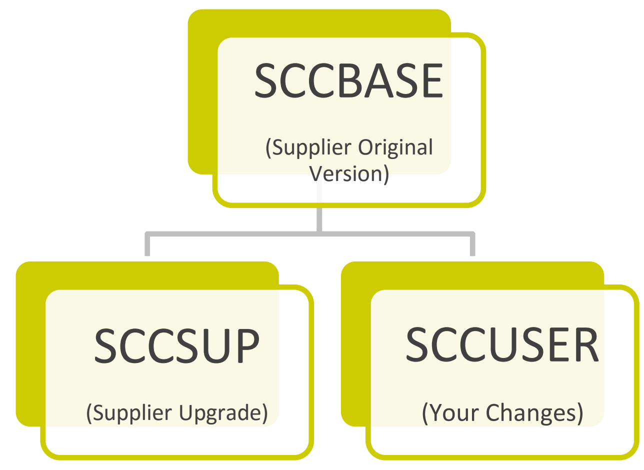
- **Your production version.** You will almost always have this version. It is exceptionally rare that an organization will lose the customized production source and have only the compiled code. If you don't have the production source code, it will be virtually impossible to isolate the changes and retrofit the new version into it.

- **The new version.** You will almost always have this version as well. The only instance where that would not be the case is if the vendor rescinded a previous policy of providing source code. Again, without this version of the source code it will be impossible to perform the necessary retrofitting.

- **The original vendor version.** This is a nice-to-have, but it's not necessary for performing the retrofit. Having this version of the source enables you to perform a three-way compare. Without the original base version you will have to spend considerably more time finding changes and merging your customized version with the vendor-supplied upgrade.

  Thus, if the original source is no longer available on your system, it is worth the effort required to track it down. Even a version that is older than the one you customized is better than nothing. If there are no earlier versions to be found within your organization, try asking the vendor if it can supply a copy.

After you have found all of the versions of the source code, copy all of the objects and sources in each version into one library per version. For the purpose of this white paper, we've labeled these three libraries as SCCBASE (the original vendor-supplied base code), SCCSUP (the vendor-supplied upgrade) and SCCUSER (your current customized source code).



It is important to be able to test the final merged application against the original customized version and the vendor-supplied base version. Thus, you should keep the SCCBASE, SCCSUP and SCCUSER libraries isolated from your normal workflow libraries, including your development, test and production libraries. If you merge them all into a single library and that prevents you from testing the retrofitted upgrade against the original versions then you will have to find an alternative strategy for performing the retrofit and testing it.

Compare the Sources and Merge the Differences

To visualize the compare and merge process, consider the following samples of original vendor-supplied code, user-customized code and vendor-supplied upgrade code. To avoid overburdening this white paper, we've kept the samples very small, but they include all of the major issues that your source-compare tool will have to take into account.

```
10 FOMAPPL1 IF   E           K        DISK
20 C                      READ OMAPPL1                  99
30 C                      SETON                     LR
```

Vendor-Supplied Base Version (SCCBASE)

```
10 FOUAPPL1 IF   E           K        DISK
20 C                      READ OUAPPL1                  99
30 C                      EXSR USER
40 C                      SETON                     LR
50  *
60
70 C            USER      BEGSR
80 C            'logoff'  DSPLY
90 C                      ENDSR
```

Vendor-Supplied Upgrade Version (SCCSUP)

```
10 FOUAPPL1 IF   E            K        DISK
20 C                       READ OUAPPL1                  99
30 C                       EXSR USER
40 C                       SETON                     LR
50  *
60
70 C             USER      BEGSR
80 C             'logoff'  DSPLY
90 C                       ENDSR
```

User-Customized Version (SCCUSER)

In the above example, the user replaced the file that the code reads with her version of the file (line 10 and 20). This is a major change. The vendor-supplied upgrade still uses the same file as the original vendor version. How should the merge tool handle this? Should it employ the version of the file specified by the user or does the upgrade make this unnecessary or, possibly, unworkable?

The user also changed the base file to add a call to a subroutine and added that subroutine to the end of the source code (lines 30 and 50-80). The vendor did the same thing in the upgraded version, although the two subroutines are different. In this case, the merge tool must recognize that there are two independent additions to the files—one in the user-customized version and one in the vendor-supplied upgrade. It might also flag a conflict on lines 30 and 50-80 and leave it up to the user to decide what to do with it.

Below, we consider how MRGSRC, KDiff3 and TD/OMS Fusion Pro each handle the compare and merge process for these samples.

MRGSRC

When using MRGSRC to compare and merge the differing versions, the tool reported one conflict, which we will examine later. As you can see from the resulting source below, there is something not quite right with the MRGSRC merged code.

From a technical perspective, one can understand why MRGSRC made the choices it did, but the result is incorrect. In addition, we see that it kept the call to the USER subroutine, but the routine itself disappeared. Could this be the conflict that MRGSRC reported? As can be seen in the resulting output, there is a report of a conflict, but the tool does not say what the conflict is. You have to track that down yourself.

One of the advantages of MRGSSRC is that you can run it in interactive mode. Doing so for these sources shows the following screen.

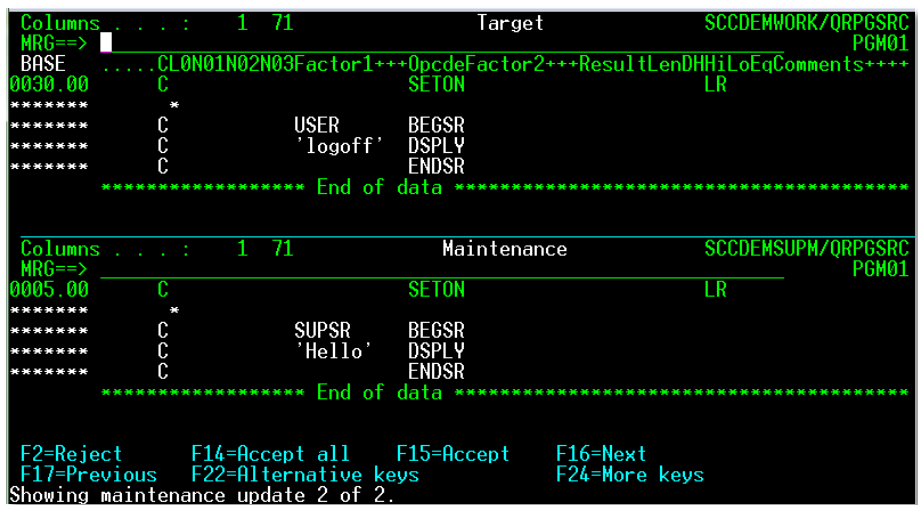
© REMAIN SOFTWARE, 2015

```
 Columns . . . :    1  71                Target              SCCDEMWORK/QRPGSRC
 MRG==>                                                                 PGM01
 BASE    ...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7
         *************** Beginning of data ************************************
>>>>>>> ----------------------------------------------------------------------
0001.00      FOUAPPL1 IF  E            K        DISK
0009.00      C                      READ OUAPPL1                    99
0009.01      C                      EXSR USER
0030.00      C                      SETON                       LR
*******       *

 Columns . . . :    1  71             Maintenance            SCCDEMSUPM/QRPGSRC
 MRG==>                                                                 PGM01
0002.00      C                      READ OMAPPL1                    99
>>>>>>>      C                      EXSR SUPSR
>>>>>>>      C                ', World!'DSPLY
0005.00      C                      SETON                       LR
*******       *
*******      C                SUPSR     BEGSR
*******      C                'Hello'   DSPLY

 F2=Reject      F14=Accept all    F15=Accept     F16=Next
 F17=Previous   F22=Alternative keys             F24=More keys
Showing maintenance update 1 of 2.
```

To resolve the conflict, you can ignore the merge proposal from MRGSRC and copy the two lines below to the correct location manually.



```
 Columns . . . :    1  71                Target              SCCDEMWORK/QRPGSRC
 MRG==>                                                                 PGM01
 BASE    .....CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++
0030.00      C                      SETON                       LR
*******       *
*******      C                USER      BEGSR
*******      C                'logoff'  DSPLY
*******      C                          ENDSR
         ***************** End of data ****************************************

 Columns . . . :    1  71             Maintenance            SCCDEMSUPM/QRPGSRC
 MRG==>                                                                 PGM01
0005.00      C                      SETON                       LR
*******       *
*******      C                SUPSR     BEGSR
*******      C                'Hello'   DSPLY
*******      C                          ENDSR
         ***************** End of data ****************************************

 F2=Reject      F14=Accept all    F15=Accept     F16=Next
 F17=Previous   F22=Alternative keys             F24=More keys
Showing maintenance update 2 of 2.
```

This is where the conflict appears. Again, we are able to manually copy the changed code to the correct location. The result after the manual intervention is something that looks correct. However, it is important to note that conflicts must ***always*** be revisited. No merge program, including Remain Software's, can understand context.

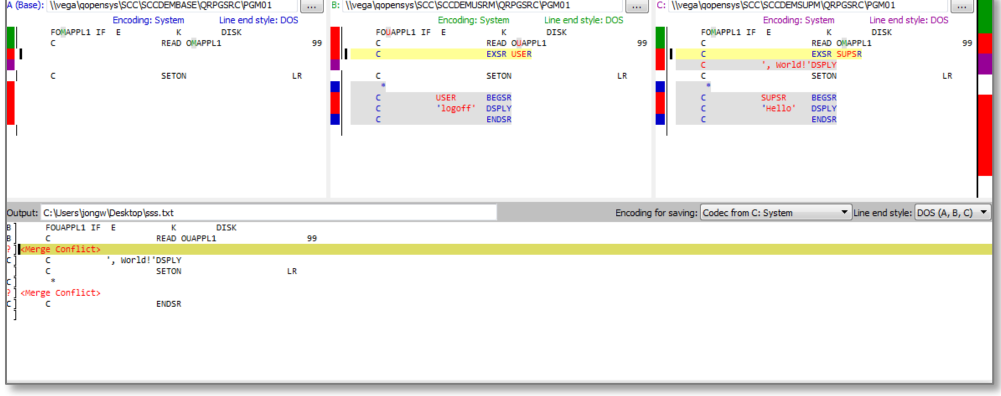
```
00      FOUAPPL1 IF  E          K        DISK
01      C                    READ OUAPPL1                  99
02      C                    EXSR SUPSR
03      C            ', World!'DSPLY
04      C                    EXSR USER
05      C                    SETON                     LR
06       *
07      C            SUPSR     BEGSR
08      C            'Hello'   DSPLY
09      C                    ENDSR
10       *
11      C            SUPSR     BEGSR
12      C            'Hello'   DSPLY
13      C                    ENDSR
```

KDiff3

When using KDiff3, you must first move the sources to the IFS. After completing the merge, the source must be copied back to the appropriate library member. This normally results in the loss of the source line change date and sequence number. However, when using a specialized copy function, these changes can be included in the compare.

After starting KDiff3, the tool pops up a color-coded three-way compare, rather than a two-way compare. It too then detects conflicts that must be resolved manually.

KDiff3 makes the resolution process easy. You simply select the conflict and then choose which source to use in the final product.

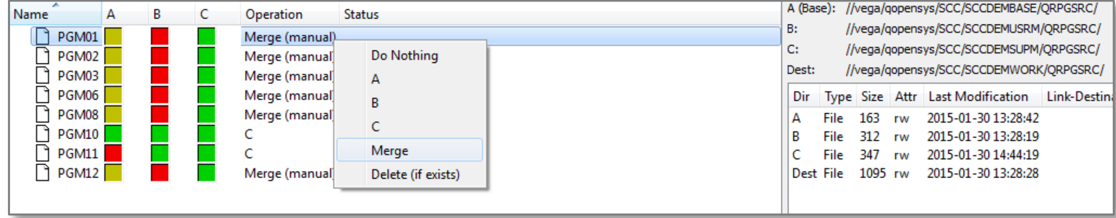


Doing so with our sample code versions results in the following source member:

```
01  FOUAPPL1 IF  E           K        DISK
02  C                     READ OUAPPL1                  99
03  C                     EXSR USER
04  C                     EXSR SUPSR
05  C            ', World!'DSPLY
06  C                     SETON                     LR
07   *
08  C            USER      BEGSR
09  C            'logoff'  DSPLY
10  C            SUPSR     BEGSR
11  C            'Hello'   DSPLY
12  C                     ENDSR
```

However, this is still not quite right. As you can see, KDiff3 misses the ENDSR line that should come after line 9.

As depicted below, KDiff3 can also merge a complete directory, but any conflicts must be merged manually.



TD/OMS Fusion Pro

Remain Software’s Source Change Control tool, TD/OMS Fusion Pro, which is part of the TD/OMS Application Lifecycle Management framework, takes a different approach to the compare and merge process.

The first difference from other tools comes in the speed of the conversion process. One reason for this greater speed is that TD/OMS Fusion Pro allows you to compare complete libraries in one go.

Another difference is that TD/OMS Fusion Pro divides the process into two steps. In the first step, user changes are isolated from the base. You can also use this step to isolate the supplier changes from the base. These comparisons give some idea of the processes that will be required to successfully retrofit the code.

Consider the following two figures, in which TD/OMS Fusion Pro isolated the changes in the user-customized and vendor-supplied versions. Note the modified line numbers signaling the source differences.

```
02DD     FOMAPPL1 IF   E            K        DISK
03DD     C                      READ OMAPPL1                 99
06II     FOUAPPL1 IF   E            K        DISK
07II     C                      READ OUAPPL1                 99
08II     C                      EXSR USER
10       C                      SETON                    LR
12II      *
13II     C            USER      BEGSR
14II     C            'logoff'  DSPLY
15II     C                      ENDSR
```

Isolated User Changes

```
01       FOMAPPL1 IF   E            K        DISK
02       C                      READ OMAPPL1                 99
04II     C                      EXSR SUPSR
06       C                      SETON                    LR
08II      *
09II     C            SUPSR     BEGSR
10II     C            'Hello'   DSPLY
11II     C                      ENDSR
```

Isolated Vendor Changes

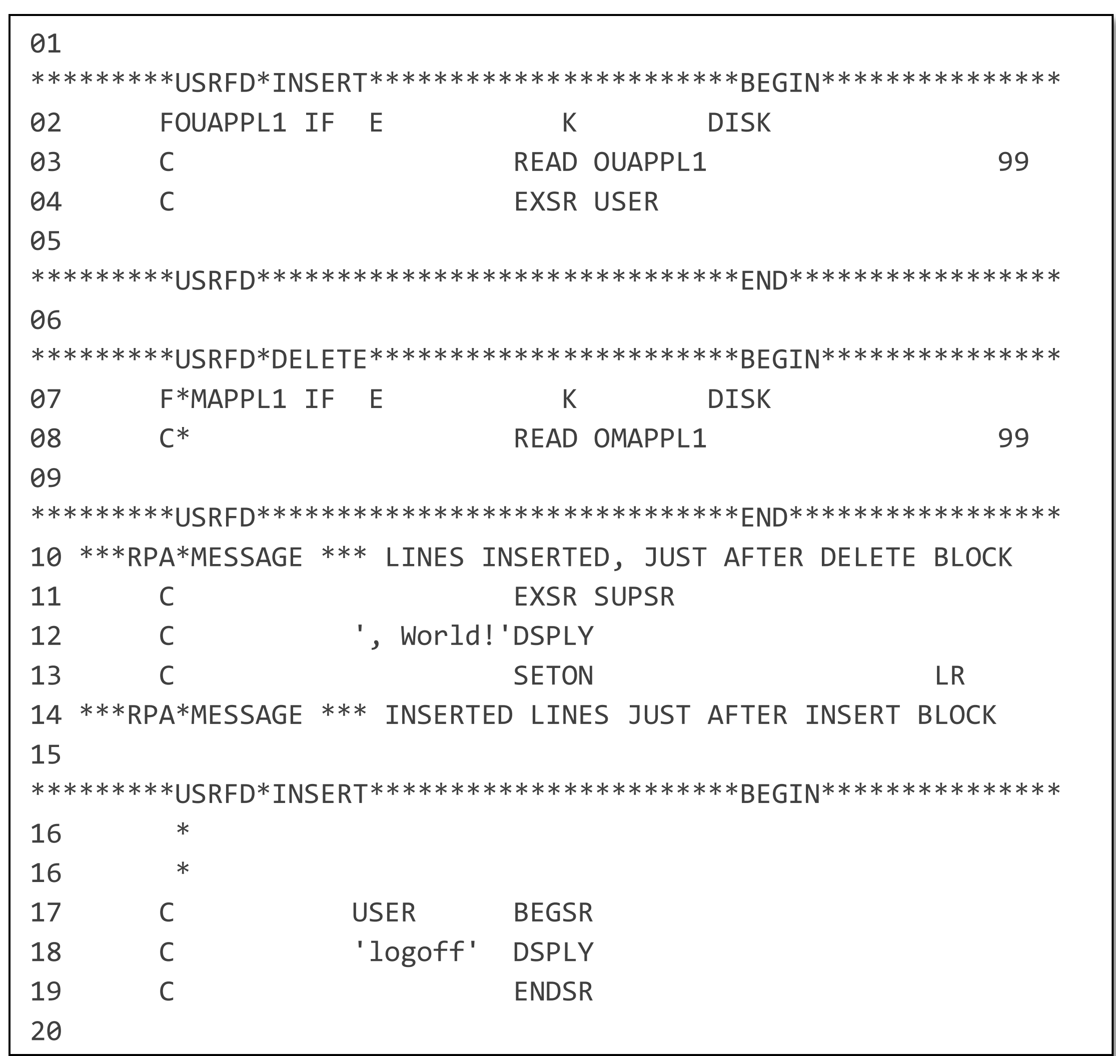
Now that the differences have been made apparent, a proper merge can be performed and the developer can be advised as to how to proceed.

Annotated Merged Version

First, the tool delivers an annotated merged version. This is source code that can be compiled, but it notes all of the details of the merge so the developer can see exactly what happened. The developer can then revert or replace older lines as appropriate.

There is some value to leaving the source annotated beyond this step. If a problem is discovered in test or, worse, in production, the annotations can help the developer to find the cause of the error more quickly.

The annotated version of the sample merge is provided below:



```
01
*********USRFD*INSERT***********************BEGIN***************
02       FOUAPPL1 IF   E            K         DISK
03       C                     READ OUAPPL1                  99
04       C                     EXSR USER
05
*********USRFD******************************END*****************
06
*********USRFD*DELETE***********************BEGIN***************
07       F*MAPPL1 IF   E            K         DISK
08       C*                    READ OMAPPL1                  99
09
*********USRFD******************************END*****************
10 ***RPA*MESSAGE *** LINES INSERTED, JUST AFTER DELETE BLOCK
11       C                     EXSR SUPSR
12       C           ', World!'DSPLY
13       C                     SETON                     LR
14 ***RPA*MESSAGE *** INSERTED LINES JUST AFTER INSERT BLOCK
15
*********USRFD*INSERT***********************BEGIN***************
16        *
16        *
17       C           USER      BEGSR
18       C           'logoff'  DSPLY
19       C                     ENDSR
20
```

Cleaned Merged Version

Lastly, you can run a command that cleans-up the annotations, producing the final source, as follows:

```
00      FOUAPPL1 IF  E            K        DISK
01      C                    READ OUAPPL1                  99
02      C                    EXSR USER
03      C                    EXSR SUPSR
04      C           ', World!'DSPLY
05      C                    SETON                     LR
06       *
07      C           USER      BEGSR
08      C           'logoff'  DSPLY
09      C                    ENDSR
10       *
11      C           SUPSR     BEGSR
12      C           'Hello'   DSPLY
13      C                    ENDSR
```

Compile

The first chance to determine whether the source was merged correctly is after you compile it. This will tell you if the merge was successful from a technical point of view. In 90 percent of the cases, this technical validation and a high-level functional validation are performed simultaneously. However, further testing is required to fully validate the retrofitted application upgrade.

The TD/OMS base product can help to streamline this phase. Its 'Work with Solution Compile' function provides access to compiling options that allow you to create objects in the system without leaving the TD/OMS Solution Maintenance function. Before execution, default 'create or compile' command-sets can be defined for the creation of different types of objects. These pre-defined commands can be started from the solution maintenance function or during the fix-transfer process. Once defined, compile commands will be used during the transfer process as well.

Retrofit Dependent Systems

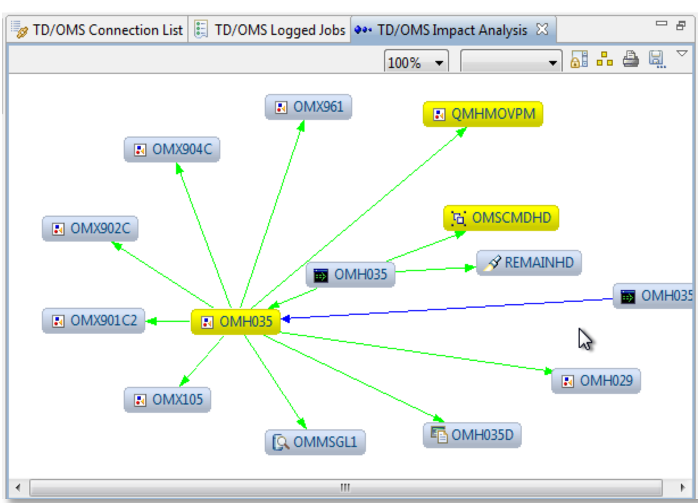
Programs rarely stand alone. They often interact with other programs to create a complete application. What's more, there may also be interfaces between applications. For example, you may have:

- Created satellite applications that replaced a complete block of the supplier code or provided a separate enhancement unique to your company.
- Added code to your financial application to retrieve data from the ERP system.
- Created a replacement warehouse management application that hooks into ERP files that have now changed.
- Placed calls in your warehouse management application to ERP programs that now have different parameter lists.
- And so on.

Changing the base software may require changes in these other programs as well. Thus, after retrofitting the vendor's changes into your custom-modified source, you then must retrofit dependent systems where appropriate.

The key to solving this problem lies in knowing how the different components in the system interact, which files are read by which programs, which "foreign" logical files are created over supplier's physical files, and so on.

The TD/OMS framework includes an Impact Analysis module that presents all of this information to the developer.



TD/OMS impact analysis makes dependencies clear

Test

Vendors generally deliver thoroughly tested code that is relatively bug-free (or as close as any large, complex application can come to being bug-free). In addition, your customized production version might not only have been thoroughly tested, but it may also have had years of production experience behind it. Nevertheless, that doesn't negate the need for thorough testing of the retrofitted upgrade.

Because they can't recognize context or determine business requirements, even the best of compare-and-merge tools cannot guarantee that the merge was done correctly. What's more, the interaction of your prior customizations with the new vendor code might introduce new problems.

As a result, you should test the merged code as thoroughly as you would any application modifications, including sufficiently testing your full application portfolio to ensure that all dependent programs will still function properly after the merged code is deployed into production.

The testing phase should encompass the standard variety of tests, including unit, integration and load testing.

An automated testing tool can significantly improve the thoroughness, accuracy and efficiency of the testing phase, particularly if you've already built a test suite designed to put your applications through their paces, including testing all situations that will occur in production. However, if the vendor's upgrade adds new functions, you will have to augment your existing test suite to validate the augmented functionality.

Deploy

Thorough testing is not sufficient to ensure successful deployment to production. In some cases, the order in which objects and files are transferred to production matters. And, of course, they must *all* be transferred accurately. In addition, if the vendor-supplied upgrade modifies any system settings, those new settings must be accurately reflected in the production environment at the time of deployment.

© REMAIN SOFTWARE, 2015

A Software Change Management (SCM) tool, such as TD/OMS, can automate application deployment to ensure that all objects are moved to production in the correct order and all configurations duplicate exactly their conditions in the development and test environments, i.e. the environments where the retrofitted application was validated.

Deployment automation can significantly reduce the time and expense required to implement the retrofitted vendor upgrade. What’s more, it virtually eliminates the possibility of human error during deployment, thereby helping to prevent system failures that might result from an inaccurate deployment.

> Conclusion

Vendors don’t issue upgrades just for the heck of it. Upgrades may improve performance, add new functionality, and/or plug security holes.

Depending on the vendor’s implementation tools and processes, even implementing an upgrade as-is can be time-consuming and cumbersome. However, the complexity, human resource burden and potential for error increase significantly if, rather than implemented as-is, the upgrade must be retrofitted with customizations that you applied to an earlier version.

Before organizations will consider implementing an upgrade, the value it delivers must be greater than the cost of the implementation, including all of the risks it introduces. A Source Change Control tool, such as TD/OMS Fusion Pro, can automate the process of finding differences between software versions and merging your customizations with the vendor’s upgraded code. What’s more, automation can reduce the probability of human error. As a result, the tool can lower the cost and risk of implementation, thereby making it easier to justify implementing the upgrade and gaining its benefits.

© REMAIN SOFTWARE, 2015

About TD/OMS

TD/OMS is an easy to use, flexible and cost-effective Software Change Management solution that supports IBM i (AS/400, Power Systems), Windows and Unix/Linux.

The basics of TD/OMS incorporate fundamental IT business processes such as Incident Management, Configuration Management, Version Management, Release Management, Life Cycle Management and Software Distribution & Deployment.

TD/OMS helps the IT organization to streamline the change process of any type of application, no matter the complexity of the environment. It provides complete control over the software life cycle process and delivers a real-time overview of software components and configuration. Compliance and auditing requirements can be easily met due to the registration of all component movements.

About TD/OMS Fusion Pro

TD/OMS Fusion Pro (Source Change Control), a part of the TD/OMS Application Lifecycle Management framework, is designed to simplify the process and minimize the risks of retrofitting a vendor-supplied application upgrade into existing application customizations. This significantly reduces the challenges of implementing upgrades in a customized environment.

TD/OMS Fusion Pro facilitates comparisons of complete libraries of sources, highlighting differences between the versions. It then provides an annotated merged version of the code that can be compiled as is or modified. TD/OMS Fusion Pro can also automatically clean up the code by removing the annotations.

The result is a more accurate fusion of the vendor's latest version of its software and your organization's pre-existing customizations.

About Remain Software

For more than 20 years Remain Software has been an expert and a market leader in Application Lifecycle Management solutions for the IBM i platform. The innovative and flexible software change and workflow management solutions from Remain Software help organizations to manage their IT assets by simplifying and automating application change and modernization processes, improve workflows and teamwork, and streamline IBM i, Windows, Unix und Linux software development - from defining requirements through design, development and up to deployment and testing.

Simplified and standardized Application Lifecycle Management, time and cost savings, and improved productivity and communication within teams are just some of the benefits that help to deliver high quality applications and customer satisfaction. Taken together, these features and benefits serve to increase organizations' profitability.

Remain Software is supported by an extensive Partner Network. Together with their Value Added Resellers, Remain Software offers a broad range of services and training that maximize the benefits of our solutions.

Remain B.V.
Dukatenburg 82b
3437 AE Nieuwegein
Tel: (+31)30-6005010
Fax: (+31)30-6005019
info@remainsoftware.com
www.remainsoftware.com

Application life cycle management

Software Change Management for small teams

Workflow management

Application modernization support

Get the latest news about application life cycle management. Follow us: