ALM Best Practices Series

For ALM Practitioners

Workflow Best Practices

Document release date: July 2020

Legal Notices

Disclaimer

Certain versions of software and/or documents ("Material") accessible here may contain branding from Hewlett-Packard Company (now HP Inc.) and Hewlett Packard Enterprise Company. As of September 1, 2017, the Material is now offered by Micro Focus, a separately owned and operated company. Any reference to the HP and Hewlett Packard Enterprise/HPE marks is historical in nature, and the HP and Hewlett Packard Enterprise/HPE marks are the property of their respective owners.

Warranty

The only warranties for products and services of Micro Focus and its affiliates and licensors ("Micro Focus") are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Micro Focus shall not be liable for technical or editorial errors or omissions contained herein. The information contained herein is subject to change without notice.

Restricted Rights Legend

Contains Confidential Information. Except as specifically indicated otherwise, a valid license is required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

Contents

Welcome To This Guide

Welcome to the Workflow Best Practices guide.

This guide provides concepts, guidelines, and practical examples for the best implementation of workflows in various organizations.

About Workflows

New technologies, architectures, business trends, and end-user expectations are changing the very nature of applications. As a result, applications themselves are changing. New tools and architectures have emerged that make it faster and simpler to develop and deliver composite applications, rich Internet applications, and interactive Web 2.0 services. New processes such as agile development are being implemented with the hope of making it easier to create adaptable applications quickly.

Application Lifecycle Management (ALM) is one complete solution that covers all the phases that software travels through during its existence. Embracing ALM practices and solutions enables software teams to meet the high expectations and demands of the business. ALM suite can serve various companies to achieve their specific needs based on industry segment, company focus and processes, amount of applications and their types, and so forth. No company works in the same way as the other, even in the same industry and under similar circumstances. The ability to customize ALM projects in many ways to meet your organization's business process needs is therefore an important aspect of each implementation.

One of the strongest tools provided by ALM is the built-in scripting capability used to define, control, and manage the business flows performed within the project. The ALM project administrator can write workflow scripts to customize the ALM user interface, and to control the actions that users can perform.

The purpose of this document is to assist ALM customers to assess their current customization practices and successfully build and maintain efficient workflow scripts using advanced features provided by ALM. All aspects of this process have been researched using best practice data and expertise from various sources including Micro Focus' operating system administrators, Micro Focus' professional services organization, technical

documentation, books from industry experts and personal experience of many customer testing organizations. These guidelines will help reduce the initial creation time and achieve maximum value in operating ALM.

Audience

This guide is intended for:

- Project Administrators
- Template Administrators
- Customization Specialists

Prerequisites

To use this book, you should have a good acquaintance with major phases of Software Development Life Cycle (SDLC). You should also be familiar with the business processes in actual IT organizations.

Operational knowledge and administrative privileges of ALM are essential in implementing these best practices.

Structure

This guide is organized as follows:

- Introduction to Workflow
- Workflow Customization Guidelines
- Workflow Events
- Conclusions

Feedback

If you have questions, comments, or valuable best practice information you want to share, send a message to the following email address:

docteam@microfocus.com

1 Introduction to Workflow

Importance of Workflow

No company is like another – different business processes, industry affiliation, development methodologies, legacy and modern technologies in use dictate the need for unique implementation of ALM. A "one size fits all" philosophy can hardly be found in real IT organizations. Therefore every ALM customer eventually takes advantage of the flexibility provided by workflow scripting.

However, workflow scripts can have a significant impact on a project's and the overall site's performance. Therefore, it is extremely important to develop workflow code that is logical and organized. It is also critical to implement a sound process for developing and maintaining workflow code.

The following steps describe the proper workflow steps in detail.

Common Steps

Understand Project Needs

Before creating or modifying workflow code, it is important to understand the project structure, methodology for working with the project, organizational processes, and the various personas involved.

To succeed with your workflow code, first gain an understanding of each group's or persona's requirements, and determine workflow that takes all groups into account. Identify the common denominators to create a combined process that meets the overall needs.

Create Workflow Requirements Document

Before rushing to write the code, start from the requirements document, which should contain planned workflow customizations. The purpose of this document is to define initial customization. Once the workflow is in production, this document must be updated regularly to include all changes that are implemented.

This document should include the following:

- The complete workflow process
- The required functionality of the workflow

  Here are some examples:

  - Requirement review process - your organization may demand that each requirement must be reviewed and approved before it can be linked to a test.
  - Actions that users or groups can execute according to their permissions.
  - Send email notification when a specific field change is made.
- Layout and format

  For example, you may want to determine which fields are available when creating a new defect, defining different sets of fields per user group, field locations in different tabs, and so forth.

The document should then be approved by all relevant stakeholders.

Write Workflow Code

After the workflow customization document is approved, begin writing the workflow code in a testing environment. For more details on the testing environment, see the *Test Environment* section.

Test Workflow Code

Invite end users to the testing environment to validate your changes. Ensure that the workflow implementation meets their needs.

Move Workflow to Production

Workflow is enforced on the client side. During login, customization and workflow files are downloaded to your local client machine under the following directory: `%temp%\TD_80`.

After the workflow code is moved to the production environment, it is necessary to log out and log in again to access the latest customization and workflow modifications.

Manage Workflow Requests

To be in control of your workflow code, especially when multiple people are involved, define a system to manage new workflow requests.

This system can be used to trace requests, understand the business needs behind them, the impact of the change, its importance, the request scope (how many people need the feature), and so forth. Such a system should also provide the ability to send notifications and status about the requests progress.

One possible solution can be to define an ALM project for the specific purpose of managing new workflow requests.

2 Workflow Customization Guidelines

Project Customization Options

ALM Workflow scripting capabilities are based on different customization sections as explained below. Before writing the code, identify all other customization needs in Project Customization. Those will be used to implement the workflow.

Permission Groups

To protect a project from unauthorized access, ALM enables you to assign each user to one or more user groups. ALM includes predefined groups with default privileges. Each group has access to certain ALM capabilities.

You can create a new group, based on the privileges of an existing group. Choosing an existing group that has similar access privileges to the new user group you want to create minimizes the level of customization you need to do.

Note that setting permissions according to user group can be used not only to enforce accessibility, but also for mail actions, notifications, and so forth.

We does **not recommend** assigning a user to more than one user group.

Project Lists

An ALM project contains a set of predefined lists that are used for default project customization, such as defect statuses and Yes-No lists. Some of those lists can be customized to support individual processes used in your organization. Other lists cannot be customized, as ALM relies on the list's values in its internal system logic. You can also create user-defined lists containing values that you can enter in a lookup list field.

Project Entities

Entities are the building blocks of any ALM project. Entities contain data entered by users for a specific application management process and the data is stored in tables. An entity can be any work object, such as requirements, tests, design steps, attachments, or defects.

Project Customization allows you to set attributes and properties for the ALM entities, such as required fields, read only, and verify value. Each entity contains ALM default fields, called *system fields*. Entities can also include *user fields* that you can create. A user field can be of the following types:

- User list (list of all users in the project)
- List
- Number
- Date
- String

In Project Customization, you can define properties for each project entity, such as defining which fields are required to be filled in by users and for which fields data history is logged. Some of these properties can be set using workflow as well. It is recommended to set the default behavior using Project Customization and change it only in special cases using workflow scripts.

Each entity has a limit of up to 99 user-defined fields. Therefore, we **recommend** working together with all stakeholders to include fields that match most stakeholders' needs and will not become redundant after a short time.

Requirement Types

You can define requirement types for your project. A requirement type defines which fields are optional and which user-defined fields are available. This enables you to create user-defined fields that are only available for requirements of a specific type.

Generic Workflow Rules

Using workflow code, you can further customize your project. You can define settings such as:

- visible and required fields
- the order in which fields are displayed in a dialog box
- which fields display in each dialog box tab
- lists to be assigned to specific fields
- default values for specific fields
- dependencies between field values

You can define these settings according to user group.

Important notes:

- Workflow code overrides any settings defined in the specific customization category in Project Customization.
- You can perform certain customization, such as defining transition rules for user groups or set field properties for requirement types, using either the specific Project Customization page or through workflow code. It is **recommended** to decide on one method for this customization, and not combine both methods.
- Using Automail, ALM enables you to automatically notify users via email each time changes are made to specified defect fields. Using the `SendMail_AfterPost` function, the workflow enables you to define automatic notification for all project entities, add complex conditions, or use it for specific users or user groups. It is **recommended** to ensure that you do not create any overlap between Automail and the workflow function.
- Workflow scripts enable you to control actions of entering and exiting modules, as well as limiting module access. To prevent access to a specific module for a user group, we **recommend** using the Module Access page in Project Customization. Do not block access to modules using workflow scripts as it conflicts with the Module Access functionality.

Test Environment

Before implementing workflow customization on your production environment, we **recommend** validating custom functionality in a test environment that reflects your specific configuration.

What Is Test Environment?

The testing environment is separate from and precisely reflects the production environment. It simulates the configurations and applications installed on the production system, including the database server, software, and production projects. By testing the workflow in your test environment, you can get a better picture of the results you can achieve, while identifying and preventing any potential negative impact to your production environment.

Why Set Test Environment?

Workflow has a great impact on the way your project functions. We **recommend** setting up a test environment for the following reasons:

- It is advisable to test the workflow before going live.
- If the workflow fails, no real harm is done since the test environment is independent of the production environment. Possible harm in production would be data loss, functionality blocked by workflow errors, and so forth.
- Early identification and detection of problems.
- Verification by stakeholders of the planned process.

Debugging Workflow Code

You can debug workflow code in a number of different ways:

Option	Tool Functionality	Used for
Adding Msgbox via ALM workflow	This is a built-in capability of ALM Workflow code. You can add `Msgbox` (message box) to any place in the workflow, in order to view the field value, action name, location in the code, and so forth. Using this method, it is recommended to comment the error handling line in your code: `On Error Resume Next`	Debugging a specific location in the code
Dbgview	This freeware monitor debugs output on your local system and prints out all application events. In order to view only workflow events, you can filter by the word 'workflow'. Please note – this option allows you to see only built-in events and not additional functions that you added. Vendor: Microsoft (SysInternals) URL: *http://technet.microsoft.com/en-us/sysinternals/bb896647*	Viewing the workflow procedures invoked by ALM
Microsoft Visual Studio	This is a commercial product that allows you to validate your code by using breakpoints, inspect variables, and so forth. To attach the workflow code to Visual Studio, produce an error in the workflow code at the point you want Visual Studio to attach. One way to do so is to call a subroutine that does not exist. When Visual	• Debugging a specific location in the code • Viewing the flow of the code that you wrote • Observing the code in runtime

	Studio is installed, such an error in the script will pop up a dialog allowing you to attach to the script. Please note that you should comment `On Error Resume Next` in your workflow code to use this option.	

In addition to the above tools, you can implement your own logger. We **recommend** that if you use your own logger, please ensure you implement the option to enable or disable it according to your needs, to prevent a negative impact on performance.

Dos and Don'ts of Workflow

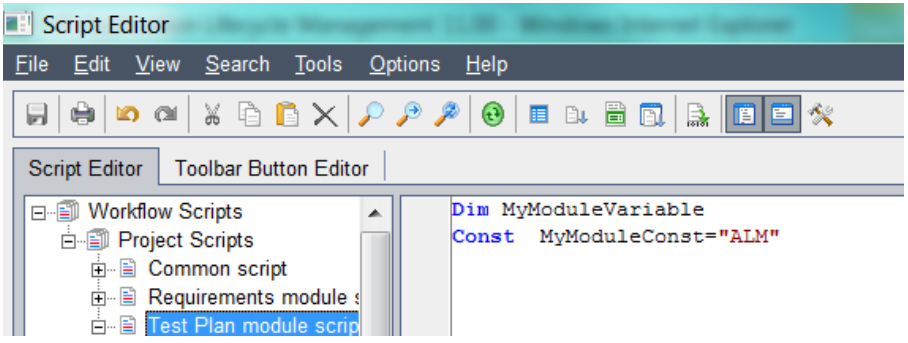
Here are the **recommended** common practices that should help you with mastering workflow.

Do

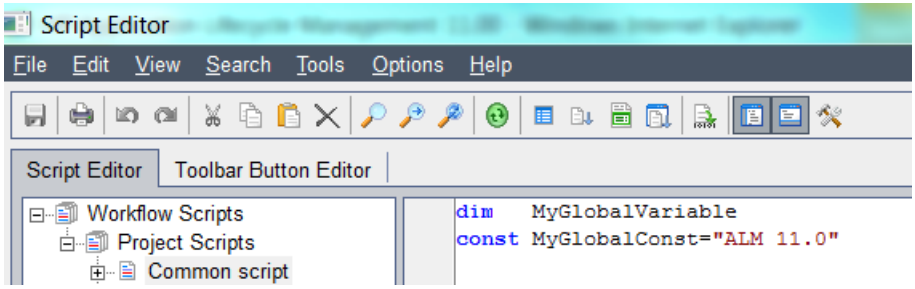
Do Use Globals

In order to pass values between different events, it is necessary to use Global Variables or Global Constant.

The variable can exist at the Module level. The variable can be used in the Module events.



In order to pass values between different module events, the global variable should be defined in the common module.



Do Optimize Code

When programming in VBScript, you may find that you have to repeatedly program the same code. This is usually an indication that you should be using a function to reduce code repetition:

- Use procedures and functions instead of the redundant code

  The following common tasks are examples of good subjects for separate functions or procedures:

  - Setting field properties
  - Setting up fields on the form
  - Setting list dependencies
  - Any procedure that works with OTA API
- Use a Switch statement instead of repetitive ElseIf statements

  The rule of thumb here is that if there are two or more ElseIf conditions, use a Switch statement.

Do Improve Code Readability

In addition to reuse of repetitive code, functions are also useful to increase code readability. The following are the **recommended** good practices to help achieve better code readability:

- Use blank lines to logically separate related blocks of code.
- Make use of introductory (header) comments from the first variable declaration and the last declared variable from the code itself.
- Precede all comments with a blank line.

- Indent code and comments within a procedure by using a two- to four-space tab stop. (The Visual Basic Editor uses a four-space tab stop by default.) As with white space, indents are used to organize code logically and make it visually appealing. The following list contains some general guidelines on using indentation correctly to make your code more readable and maintainable:
  - Indent all code and comments within a procedure at least one tab stop. The only code lines that are not indented are the beginning and end of the procedure and line labels used in connection with your error handler.
  - If you use line breaks to format a procedure's argument list, use tabs to indent the arguments and their data-type declarations, so they are aligned with the first argument in the list.
  - Indent declared variables one tab stop. Declare only one variable on a line.
  - Indent control structures at least one tab stop. If one control structure is embedded within another, indent the embedded structure one tab stop. Indent code within a control structure one additional tab stop.
  - If you use a line-continuation character to break a line of code, indent the new line one extra tab stop. This creates a visual cue that the two (or more) lines belong together. If the line following the continued line is indented as much as the continued line would be, add one more tab to the continued line to distinguish it from the next line.
  - Indent comments to the same level as the code to which the comment refers.

Do Comment Code

Use comment templates like the one below to document your code:

```
'#############################################################
'#Date:
'#Designer:
'#Purpose:
'#############################################################
```

Do Access By Name

It is a common task to access a field or fields in the certain entity:

- Use statement `<Entity>_Fields.Field("<Field_Name>")` to access a field by name.
- Use loop on `<Entity>_Fields.FieldById(i)` to access all fields in the collection. This can be used, for example, to reset the fields' order.

Both methods allow working with the fields of the *current entity*.

The current entity can be defined in the following way:

- Current entity

  The entity for which the current event is triggered. Almost each event points to the entity type for which the fields can be retrieved. For example, in `Defects_Bug_` events, only `Bug_Fields` can be manipulated; in `TestPlan_DesignStep_` events, only `DesignStep_Fields` can be accessed.

- Focused item

  From all entities of the collection defined by the event, only the fields of the currently focused entity can be retrieved. For example, the test on which the cursor is placed, or the current run in manual runner.

To retrieve the fields of other objects of the same or other object type, use OTA API.

Do Reset Prior to Setting New Layout

Make sure you reset the layout for *all fields* before setting the *specific fields*' layout such as `PageNo` and `ViewOrder`.

Since the fields have some default predefined order, it is important to reset this order before defining the new, custom one. Otherwise, you can have a field other than the desired field with the same order, resulting in an unknown order for all remaining fields.

See *Workflow Sample* later in the book.

Do Backup Code

We **recommend** performing regular backup of your workflow code. You can copy all or part of your workflow script and paste into an external text file to save in the file system.

Do Use Global Constant vs. Field Names

In order to facilitate code readability, use Global Constants instead of the field names.

You would need to declare Global Constant for each field name - see section *Do Use Globals* above.

Example:

```
If Bug_Fields.Field(Bug_Status).Value="Closed" then
       Bug_Fields.Field(Bug_Closed_In_Version).isRequired = true
End if
```

Do Cleanup of Objects

Make sure you clean objects at the end of their scope. For each object instance, it is imperative to clean the unused objects. It improves ALM performance and helps prevent errors.

```
set myTDConnection = TDConnection
set myTDConnection = nothing
```

Do Standardize

We **recommend** applying standardization across all projects. The ALM project administrator, who is responsible for more than one project, should use common conventions in all projects. This contributes to code readability and maintainability and enables cross-project functionality.

Do Error Handling

One of the most important factors that affect quality of the workflow scripting is the proper implementation of error handling.

In general, here are some **recommended** simple practices of error handling that would help better control application behavior:

- Use an `On Error Resume Next` statement at the beginning of each procedure and function
- Use `On Error GoTo 0` at the end of each procedure or function
- Show errors to the user in some standard message box

The code should be added once to each workflow script (Defects, Test Plan, and so forth) like in the `PrintError` function below.

Use the Visual Basic `Err` object that contains information about run-time errors for that purpose.

```
Sub GetBug1
  On Error Resume Next
  Set Bug1 = TDConnection.BugFactory.item(1)
  PrintError("GetBug1")
End Sub
Sub PrintError(strFunctionName)
  If Err.Number <> 0 Then
    MsgBox "Error #" & Err.Number & ": " & Err.Description,
    _ vbOKOnly, "Workflow Error in Function " &
    strFunctionName
  End If
End Sub
```

Do Not

Do Not Duplicate

Do not duplicate your code in the `Entity_CanChange` event and in the `EntityChange` event.

The code based on a field change could be written in the `Entity_canChange` event or in the `Entity_Change` event. Please make sure you understand the difference between these events and follow the simple rules below in your workflow code:

- The code that deals with permissions (allow change status) should be written in the `Entity_CanChange` event
- The code that processes dependency values or dependency lists should be written in the `EntityChange` event

Do Not Put Too Much Code

Too much code in the `CanLogin` event or in the `EnterModule` event impacts performance. A common error is to update entities during the `CanLogin`

event.

Do Not Set Other Properties Prior to Visible

Set the `IsVisible` property before setting the `IsRequired` or `IsReadOnly` property of the field.

Setting the mandatory or read-only property for the field that is not visible on the screen is meaningless and is ignored by ALM. So it is important to ensure that the field is visible before setting any of these properties.

See *Workflow Sample* later in the book.

Do Not Mess Workflow Code

Even though an ALM project may be in use for several years, the workflow code should still be clear and extendable.

- Prefer Select over If
- Use functions

Do Not Update Parameters

Do not update parameter values from a workflow function. Specifically, do not update the `NewValue` parameter in the `Entity_CanChange` event.

Do Not Modify in New

In the new entity event, it is **not recommended** to modify actions, because the new entity event is called when the entity is *created* and not when the new entity dialog box is opened.

A common use case is when a user opens the new entity dialog box for the second time. The new entity event will not be called since the entity was already created the first time the dialog box was opened.

Do Not Assign in MoveTo

Do not assign values to a field in the `MoveTo` event. It is a bad practice because the `MoveTo` event would lock the entity.

Do Not Modify After Post

Do not perform any object modification on the `After_Post` event.

Do Not Use Many API Calls

Do not use many API calls, because calling the APIs increases the level of communication between the server and the database. Each call to an API results in a network communication, causing the script to take longer to run.

For example, in order to process 100 entities, try to get all of them in one filter instead of retrieving each one separately.

Using API in Workflow

Use ALM API for Modifications

ALM API provides a separation layer between the user interface (or any application that uses it) and the server logic. We **recommend** following these rules when using API calls:

- Use a predefined `TDConnection` object to get the current session

  When using OTA API from external applications like Visual Basic or Excel, the first step for any application that uses OTA is to create the instance of the `TDConnection` object, initialize the connection to the server, and connect to the database. However, in the workflow there is the predefined `TDConnection` object (in this case `TDConnection` is not only a class name, but also the name of the global variable that contains the instance of `TDConnection`), which points to the same session in which the current user works. This means that access to all ALM collections and objects is always available from any place in the workflow.

- Avoid direct update of the database using the `Command` object because of the following potential problems:
  - Bypassing server mechanism leads to:
    - Entity locking
    - Loss of history
    - Unwanted other functionality (setup mail)
  - High maintenance of the queries
  - Can result in data corruption or inconsistency

- Use mailing methods available in OTA to send the custom mails to the users

  OTA allows access to ALM mailing, which allows you to:

  — Create custom conditions that cannot be implemented using the automatic notification system of ALM

  — Change the subject or the text of the e-mail

  — Send an e-mail to the specific ALM groups or users

  — Send the e-mail from the specific user, rather than "admin" as automatic mail notification does

  The mailing methods are available from any ALM object such as Defect, Test, and so forth, or directly from the `TDConnection` object. Using the `Mail` method from the `TestDirector` object you can send the e-mail that contains that object and your custom subject and text.

Example – Use workflow objects instead of SQL statement

Do not use these commands:

```
Com.CommandText = "UPDATE TESTCYCL SET TC_TESTER_NAME =
'" & Cstr(ASSIGNED_TESTER) & "' " &_
"Where TC_CYCLE_ID = " & iTestSetId & " and TC_TESTER_NAME
is NULL"
Set UpdateRecSet = Com.Execute
```

Instead use this code snippet:

```
Set tstestF = currentTestSet.tstestFactory
Set tsFilter = tstestF.Filter
tsFilter("TC_TESTER_NAME")= ""
Set tsTestList = filter.newList
For each tsTest in tsTestList
    tsTest.Field("TC_TESTER_NAME") = "admin"
Next
```

Minimize Activity on Client

When fetching data from the server, it is **recommended** to filter the information on the server side instead of on the client. The performance overhead of filtering on the client is very high. Loading too many records can also impact the server's performance.

Example – Use filter when working with history

When using the `Command` object to go over the `HISTORY` table, you should create a filter in the SQL by implementing the `WHERE` condition, so it does not bring all recordsets to the client.

Example – Calculating design steps

Design step has a user defined field holding the duration of the step. Our goal is to get the number of design steps with duration bigger than 30 minutes.

The code below represents bad practice:

```
For Each Test In TestLists
    Set DesStepF = Test.DesignStepFactory
    Set DSList = DesStepF.NewList("")
    For Each DStep In DSList
        If DStep.Field("DS_USER_01")>30 Then
          HowManyFound = HowManyFound + 1
        End If
    Next
Next
```

Instead, try using the following code block:

```
Set TestF = TDConnection.TestFactory
Set TestList = TestF.NewList("")
For Each Test In TestList
  Set DesStepF = Test.DesignStepFactory
  Set DSList = DesStepF.NewList("select * from DESSTEPS
WHERE DS_USER_01>30 ")
  HowManyFound = HowManyFound + DSList.count
Next
```

3 Workflow Events

During an ALM user session, as the user initiates various actions, ALM triggers event procedures. You can place code in these procedures to customize the execution of the associated user actions. Event procedures can be functions or subroutines.

General

The following gives some general background on event functions and subroutines as well as naming conventions used in ALM.

Event Functions

These procedures are triggered by ALM to check whether the user's action should be performed. You can place code in these functions to determine whether ALM can execute the user's request. If your code returns a value of false, ALM does not proceed with the action.

For example, when a user clicks the Submit button on the *Add Defect* dialog box, ALM invokes the function `Bug_CanPost` before posting the defect to the database on the server. You can add code to the `Bug_CanPost` function to control whether ALM posts the defect. For example, you can ensure that a user cannot reject a defect without adding a comment.

Event Subroutines

These procedures are triggered to perform actions when an event takes place.

For example, when a user opens the *Add Defect* dialog box, ALM invokes the subroutine `Bug_New`. You can add code to the `Bug_New` subroutine to perform actions that should be performed when a user opens the dialog box. For example, you can change the value of the *Detection Mode* field to `BTW` if the user is not in the `QA Tester` user group.

Naming Conventions

The naming convention in ALM for an event procedure is as follows:

`<entity>_<event>`

Note: Some event procedure names, such as `GetDetailsPageName`, do not include an entity name.

Entities

Entity	Description
AnalysisItem	Reports and graphs data
AnalysisItemFolder	Reports and graphs folder data
Bug	Defect data
BusinessModel	Business model data
BusinessModelActivity	Business model activity data
BusinessModelFolder	Business model folder data
BusinessModelPath	Business model path data
Component	Business component data
ComponentFolder	Business component folder data
ComponentStep	Business component step data
DashboardFolder	Dashboard folder data
DashboardPage	Dashboard page data
DesignStep	Design step data
Resource	Test resource data
Resource Folder	Test resource folder data
Run	Test run data
Step	Test run step data
TestSet	Test set data
TestSetTests	Test instance data

Some extensions may also be supported by workflow.

Common Modules

CanLogin

This event is triggered to check whether the specified user can log in to the specified project. It is to allow or forbid the login to a project. This event can be used to update the users.

Topic	Description
Syntax	`CanLogin(DomainName, ProjectName, UserName)` where `DomainName` is the domain name, `ProjectName` is the project name, and `UserName` is the user name
Type	Function
Returns	True or False
Availability	`CanLogin` (all modules)

Example – Notify user upon login

```
Function CanLogin(DomainName, ProjectName, UserName)
CanLogin = false
Call MsgBox("Hi " & User.UserName & “,” _
            & vbCrLf & "" _
            & vbCrLf & "Your project " & TDConnection.ProjectName
& " was upgraded to ALM 11.0" _
            & vbCrLf & "" _
            & vbCrLf & "The Project was moved to the server :
http://ALM:port/qcbin" _
            & vbCrLf & "" _
            & vbCrLf & "QC Admin" _
            , vbExclamation, "Important Message")
Exit function
End function
```

CanLogout

This event is triggered to check whether the current user can log out of the current project.

Topic	Description
Syntax	`CanLogout`
Type	Function
Returns	True or False
Availability	`CanLogout` (all modules)

Example – Notify user before logout

```
Function CanLogout
Call MsgBox("Hi " & User.UserName & “,” _
            & vbCrLf & "" _
            & vbCrLf & "Your project " & TDConnection.ProjectName
& " will be upgraded to ALM 11.0  on 01/01" _
            & vbCrLf & "" _
            & vbCrLf & "The Project will be moved to the server:
http://ALM:port/qcbin" _
            & vbCrLf & "" _
            & vbCrLf & "QC Admin" _
            , vbExclamation, "Important Message")
End Function
```

ActionCanExecute

This event is triggered before ALM performs an action that has been initiated by the user, to check whether the action can be executed. You can add code to this event procedure to perform actions when the user has initiated a particular action, or to prevent the action from being executed in specific cases.

Topic	Description
Syntax	`ActionCanExecute(ActionName)` where `ActionName` is the action that the user has initiated Actions are in the format `context.action` User-defined actions start with the prefix `UserDefinedActions`
Type	Function
Returns	True or False
Availability	`ActionCanExecute` (all modules)

Example - Prevent defect deletion

```
Function ActionCanExecute(ActionName)
On Error Resume Next
if ActionName= "Defects.DeleteDefect" then
       if Bug_Fields.Field("BG_STATUS").value  =”Closed” then
              ActionCanExecute = true
       Else
Msgbox “You don’t have enough credentials to perform
              Delete” ActionCanExecute = false
              Exit function
       End if
End if
‘……

End function
```

Example - Find action names

```
Function ActionCanExecute(ActionName)
On Error Resume Next
if user.Username=”Project_admin” then
       MsgBox actionname
```

```
End if
End function
```

EnterModule

This event is triggered when the user switches to an ALM module.

You can add code to this event procedure to perform an action whenever the user switches to the specified module.

Topic	Description
Syntax	`EnterModule`
Type	Sub
Returns	
Availability	`EnterModule` (all modules)

Example – Hide a button

```
Sub EnterModule
'hides the button Send Mail in the Defects grid
  On Error Resume Next
    Actions.action("Defects.SendByEmail").Visible= false
  On Error GoTo 0
End Sub


Sub DialogBox(DialogBoxName, IsOpen)
'hides the button Send Mail in the Defect details Dialog
'Use ActiveModule and ActiveDialogName to get the current
context On Error Resume Next
if (DialogBoxName="actBugDetails" or DialogBoxName="Details"
or DialogBoxName="Bug Details") and IsOpen=true    then
       Actions.action("BugDetails.SendByEmail").Visible= false
End if
  On Error GoTo 0
End Sub
```

ExitModule

This event is triggered when the user exits the specified module.

Topic	Description
Syntax	`ExitModule`
Type	Sub
Returns	
Availability	`ExitModule` (all modules)

DialogBox

This event is triggered when a dialog box is opened or closed.

Topic	Description
Syntax	`DialogBox(DialogBoxName, IsOpen)` where `DialogBoxName` is the name of the dialog box, and `IsOpen` indicates whether the dialog box is open
Type	Sub
Returns	
Availability	`DialogBox` (all modules)

Example - Identify view type

This example helps identify the current view type - Grid, Details, New Entity. The type is maintained in a global variable in the common module called `DialogIsOpen`.

```
Sub DialogBox(DialogBoxName, IsOpen)
On error resume next
If DialogBoxName="New Bug" and IsOpen=true then
         DialogIsOpen = "NEW"
             Else
```

```
                    DialogIsOpen =”OTHER” ‘Details Or Grid
End if
 On Error GoTo 0
End sub
```

CanCustomize

This event is triggered when a user attempts to open the Customization window, to check whether the user can customize the specified project.

Topic	Description
Syntax	`CanCustomize(DomainName, ProjectName, UserName)` where `DomainName` is the domain name, `ProjectName` is the project name, and `UserName` is the user name
Type	Function
Returns	True or False
Availability	`CanCustomize` (all modules)

Example – No entry to customizations

This example prevents entrance into customizations for unauthorized users.

```
Function CanCustomize(DomainName, ProjectName, UserName)
on error resume next
if  User.IsInGroup("TDAdmin”)
  then CanCustomize = true
else
    MsgBox User.FullName & vbcrlf & vbcrlf & "You don't have
enough privileges" &vbcrlf & vbcrlf &"Please Open a SR in
Project Center Admin", vbExclamation, "Not Allowed"
   CanCustomize = false
end if
On Error GoTo 0
End Function
```

Attachment_New

This event is triggered when an attachment is added to ALM.

Topic	Description
Syntax	`Attachment_New(Attachment)` where `Attachment` is the `lAttachment` interface
Type	Sub
Returns	
Availability	Attachment_New (all modules)

Attachment_CanOpen

This event is triggered before ALM opens an attachment from the server, to check whether the attachment can be opened.

Topic	Description
Syntax	`Attachment_CanOpen(Attachment)` where `Attachment` is the `IAttachment` interface
Type	Function
Returns	True or False
Availability	`Attachment_CanOpen` (all modules)

Attachment_CanPost

This event is triggered before ALM updates an existing attachment on the server, to check whether the attachment can be updated.

Topic	Description
Syntax	`Attachment_CanPost(Attachment)` where `Attachment` is the `IAttachment` interface

Type	Function
Returns	True or False
Availability	`Attachment_CanPost` (all modules)

Attachment_CanDelete

This event is triggered before ALM deletes an attachment from the server, to check whether that attachment can be deleted.

Topic	Description
Syntax	`Attachment_CanDelete(Attachment)` where `Attachment` is the `IAttachment` interface
Type	Function
Returns	True or False
Availability	`Attachment_CanDelete` (all modules)

GetDetailsPageName

This event is triggered by ALM to retrieve the name of the page (tab) that has the index number specified in `PageNum` in the following dialog boxes:

- An entity's Details dialog box
- An entity's New <entity> dialog box

Topic	Description
Syntax	`GetDetailsPageName(PageName, PageNum)` where `PageName` is the default page name (for example, Page 1) and `PageNum` is the page number. Note: The page number is the absolute page number, regardless of the page relative position in relation to the other displayed pages in the dialog box

Type	Function
Returns	String containing the page name
Availability	`GetDetailsPageName` (all modules)

Example - Update Tab

```
Function GetDetailsPageName(PageName,PageNum)
  On Error Resume Next
     Select Case activemodule
        Case "Requirements"
           Select Case PageNum
                  Case 1
                    GetDetailsPageName="Req_Details-First Tab"
                  Case 2
                    GetDetailsPageName="Req_Details-Second Tab"
                  Case 3
                    GetDetailsPageName="Req_Details-Third Tab"
           End select
        Case "Defects"
           Select Case PageNum
                  Case 1
                    GetDetailsPageName="Def_Details-First Tab"
                  Case 2
                    GetDetailsPageName="Def_Details-Second Tab"
                  Case 3
                    GetDetailsPageName="Def_Details-Third Tab"
           End select
     End select
  On Error GoTo 0
End Function
```

```
Function GetNewBugPageName(PageName,PageNum)
On Error Resume Next
  Select Case PageNum
         Case 1
           GetNewBugPageName="Def_Details-First Tab"
         Case 2
           GetNewBugPageName="Def_Details-Second Tab"
         Case 3
           GetNewBugPageName="Def_Details-Third Tab"
  End select
  On Error GoTo 0
End Function
```

Entity Modules

Entity_New

This event is triggered when an object is added to ALM. You can add code to this event procedure to perform an action when a new object is added.

Topic	Description
Syntax	<entity>_New
Type	Sub
Returns	
Availability	AnalysisItem_New AnalysisItemFolder_New Baseline_New Bug_New BusinessModelFolder_New BusinessModelPath_New Component_New ComponentFolder_New ComponentStep_New

	Cycle_New DashboardFolder_New DashboardPage_New DesignStep_New Library_New LibraryFolder_New Release_New ReleaseFolder_New Req_New Resource_New ResourceFolder_New Step_New Test_New TestConfiguration_New TestFolder_New TestSet_New TestSetFolder_New

Entity_MoveTo

This event is triggered when the user changes focus from one object to another.

You can add code to this event procedure to perform actions when the user changes the focus.

Topic	Description
Syntax	<entity>_MoveTo
Type	Sub
Returns	
Availability	AnalysisItem_MoveTo AnalysisItemFolder_MoveTo Baseline_MoveTo Bug_MoveTo BusinessModel_MoveTo BusinessModelActivity_MoveTo BusinessModelFolder_MoveTo BusinessModelPath_MoveTo

	Component_MoveTo ComponentFolder_MoveTo (formerly MoveToComponentFolder) ComponentStep_MoveTo Cycle_MoveTo DashboardFolder_MoveTo DashboardPage_MoveTo DesignStep_MoveTo Library_MoveTo LibraryFolder_MoveTo Release_MoveTo ReleaseFolder_MoveTo Req_MoveTo Resource_MoveTo ResourceFolder_MoveTo Run_MoveTo Step_MoveTo Test_MoveTo TestConfiguration_MoveTo TestFolder_MoveTo TestSet_MoveTo TestSetFolder_MoveTo TestSetTests_MoveTo

Example - Update setup when moving

Update setup when moving to another entity.

```
Sub Bug_MoveTo
        Select Case Bug_Fields.Field("BG_STATUS").value
           Case “New”
               Setup_Status_New
           Case “Open”
               Setup_Status_Open
           Case “Fixed”
               Setup_Status_Fixed
           Case “Closed”
```

```
                Setup_Status_Closed
        End select
End sub
```

Example - Dependency list

The following code shows how to change a list associated with a field according to the value of a different field.

Let's assume there are user-defined fields added to the Requirement entity named *SUB_AREA* (RQ_USER_01) and *TESTING_AREA* (RQ_USER_02) and there is a user-defined list added per each testing area with the name *SUB_LIST_<testing area>*.

This code should be called in the `<entity>_MoveTo` and in the `<entity>_FieldChange` event.

```
Req_Fields.field("RQ_USER_02").List = Lists("SUB_LIST_" &
Req_Fields.field("RQ_USER_01").value)
```

Entity_ FieldCanChange

This event is triggered before ALM changes a field value, to determine whether the field can be changed.

You can add code to this event procedure to prevent a field from being changed in specific cases.

Topic	Description
Syntax	`<entity>_FieldCanChange(FieldName, NewValue)` where `FieldName` is the name of the field and `NewValue` is the field value
Type	Function
Returns	True or False

Availability	`AnalysisItem_FieldCanChange`
	`AnalysisItemFolder_FieldCanChange`
	`Baseline_FieldCanChange`
	`Bug_FieldCanChange`
	`BusinessModel_FieldCanChange`
	`BusinessModelActivity_FieldCanChange`

	BusinessModelFolder_FieldCanChange BusinessModelPath_FieldCanChange Component_FieldCanChange ComponentFolder_FieldCanChange ComponentStep_FieldCanChange Cycle_FieldCanChange DashboardFolder_FieldCanChange DashboardPage_FieldCanChange DesignStep_FieldCanChange Library_FieldCanChange LibraryFolder_FieldCanChange Release_FieldCanChange ReleaseFolder_FieldCanChange Req_FieldCanChange Resource_FieldCanChange ResourceFolder_FieldCanChange Run_FieldCanChange Step_FieldCanChange Test_FieldCanChange TestConfiguration_FieldCanChange TestFolder_FieldCanChange TestSet_FieldCanChange TestSetFolder_FieldCanChange TestSetTests_FieldCanChange

Example - Allow or deny a change

This function enables or denies certain user groups the permission to change the *Status* field in the defect according to its current and new value.

```
Function Bug_FieldCanChange(FieldName, NewValue)
On Error Resume Next
if FieldName = "BG_STATUS" then
       if User.IsInGroup("QATester") then
              if Bug_Fields.Field("BG_STATUS").value ="Fixed" then
                 Select Case NewValue
                    Case "Fixed", "Closed"
                        Bug_FieldCanChange  = true
```

```
                    Case else
                        Bug_FieldCanChange  = false
                        Exit function
                    End select
                End if
            End if
        End if

     On Error GoTo 0
    End Function
```

Entity_ FieldChange

This event is triggered when the value of the specified field changes. Every change of value triggers the field change event when the field loses focus.

You can add code to this event procedure to perform an action when the value of a particular field is changed. For example, you can hide or display one field depending on the value the user enters into another field.

Topic	Description
Syntax	`<entity>_FieldChange(FieldName)` where `FieldName` is the name of the field
Type	Sub
Returns	

Availability	AnalysisItem_FieldChange AnalysisItemFolder_FieldChange Baseline_FieldChange Bug_FieldChange BusinessModel_FieldChange BusinessModelActivity_FieldChange BusinessModelFolder_FieldChange BusinessModelPath_FieldChange Component_FieldChange ComponentFolder_FieldChange ComponentStep_FieldChange Cycle_FieldChange

	DashboardFolder_FieldChange DashboardPage_FieldChange DesignStep_FieldChange Library_FieldChange LibraryFolder_FieldChange Release_FieldChange ReleaseFolder_FieldChange Req_FieldChange Resource_FieldChange ResourceFolder_FieldChange Run_FieldChange Step_FieldChange Test_FieldChange TestConfiguration_FieldChange TestFolder_FieldChange TestSet_FieldChange TestSetFolder_FieldChange TestSetTests_FieldChange

Example - Dependency values

When changing a test status to the *To Automate* value, a template description is added.

```
Sub Test_FieldChange(FieldName)

On Error Resume Next

  if Test_Fields.Field("TS_STATUS").Value="To Automate"

     then if Test_Fields.Field("TS_DESCRIPTION").value=""

     then

       myComments="<html><body><b>TO AUTOMATE-" & Now & "/ Checked
by " & user.UserName & "</b><br></body></html>"

       Test_Fields.Field("TS_DESCRIPTION").value= myComments

     Else

       myComments="<br><b>TO AUTOMATE-" & Now & "/ Checked by "&
       user.UserName & "</b><br>"

       Test_Fields.Field("TS_DESCRIPTION").value =
       Test_Fields.Field("TS_DESCRIPTION").value & "<br> "&
       myComments

     End if
```

```
End if
```

```
 On Error GoTo 0
End Sub
```

Example – Update setup on change

Update the setup when a field, such as defect status, changes:

```
Sub Bug_FieldChange(FieldName)
On Error Resume Next
If FieldName=”BG_STATUS” then
   Select Case Bug_Fields.Field("BG_STATUS").value
          Case “New”
              Setup_Status_New
          Case “Open”
              Setup_Status_Open
          Case “Fixed”
              Setup_Status_Fixed
          Case “Closed”
              Setup_Status_Closed
  End select
End if
 On Error GoTo 0
End Sub
```

Also see the *Dependency List* example.

Entity_ CanPost

This event is triggered before ALM posts an object to the server, to check whether the object can be posted.

You can add code to this event procedure to prevent an object from being posted in specific cases.

Topic	Description
Syntax	`<entity>_CanPost`
Type	Function

Returns	True or False
Availability	AnalysisItem_CanPost AnalysisItemFolder_CanPost Baseline_CanPost Bug_CanPost BusinessModel_CanPost BusinessModelFolder_CanPost BusinessModelPath_CanPost Component_CanPost ComponentFolder_CanPost Cycle_CanPost DashboardFolder_CanPost DashboardPage_CanPost Library_CanPost LibraryFolder_CanPost Release_CanPost ReleaseFolder_CanPost Req_CanPost Resource_CanPost ResourceFolder_CanPost Run_CanPost Step_CanPost Test_CanPost TestConfiguration_CanPost TestFolder_CanPost TestSet_CanPost TestSetFolder_CanPost

Example - Disable update

If a requirement is completed without a comment, the user is not allowed to submit the requirement.

```
Function Req_CanPost
On Error Resume Next
if Req_Fields.Field("RQ_REQ_PRIORITY").IsModified then
   if Req_Fields.Field("RQ_DEV_COMMENTS").IsModified=false
      then Req_CanPost=false
```

```
        MsgBox "The priority was updated, you have to add a comment"
        Exit function
    End if
End if
On Error GoTo 0
End Function
```

Entity_CanDelete

This event is triggered before ALM deletes an object from the server, to check if the object can be deleted.

Topic	Description
Syntax	`<entity>_CanDelete`
Type	Function
Returns	True or False

Availability	`AnalysisItem_CanDelete`
	`AnalysisItemFolder_CanDelete`
	`Baseline_CanDelete`
	`Bug_CanDelete`
	`BusinessModel_CanDelete`
	`BusinessModelFolder_CanDelete`
	`BusinessModelPath_CanDelete`
	`Component_CanDelete`
	`ComponentFolder_CanDelete`
	`Cycle_CanDelete`
	`DashboardFolder_CanDelete`
	`DashboardPage_CanDelete`
	`Library_CanDelete`
	`LibraryFolder_CanDelete`
	`Release_CanDelete`
	`ReleaseFolder_CanDelete`
	`Req_CanDelete`
	`Resource_CanDelete`
	`ResourceFolder_CanDelete`
	`Test_CanDelete`

	TestConfiguration_CanDelete TestFolder_CanDelete TestSet_CanDelete TestSetFolder_CanDelete

Entity_AfterPost

This event is triggered after an object has been posted to the server. Project fields should not be changed after they have been posted, because the new value is not stored in the database.

Topic	Description
Syntax	<entity>_AfterPost
Type	Sub
Returns	
Availability	AnalysisItem_AfterPost AnalysisItemFolder_AfterPost Baseline_AfterPost Bug_AfterPost BusinessModel_AfterPost BusinessModelFolder_AfterPost BusinessModelPath_AfterPost Component_AfterPost ComponentFolder_AfterPost Cycle_AfterPost DashboardFolder_AfterPost DashboardPage_AfterPost Library_AfterPost LibraryFolder_AfterPost Release_AfterPost ReleaseFolder_AfterPost Req_AfterPost Resource_AfterPost ResourceFolder_AfterPost Run_AfterPost Step_AfterPost Test_AfterPost

	TestConfiguration_AfterPost TestFolder_AfterPost TestSet_AfterPost TestSetFolder_AfterPost

Example - Send Mail

A notification mail will be sent to the Requirement author if the Target Release field was modified. In order to send a mail, we need to add a customized function called sendreqmail.

```
Sub Req_AfterPost

If Req_Fields.field("RQ_TARGET_RCYC").IsModified Then

   Sendreqmail Req_Fields.field("RQ_REQ_ID").Value,
   Req_Fields.field("RQ_REQ_AUTHOR").Value, "", "Target Cycle has
   changed", "Please Review"

End if

End sub

Sub sendreqmail(ReqId,Mto,cc,msubject,mcomment)

Dim tdc, bgf, bg

       Set tdc = TDConnection

       Set rf =  tdc.ReqFactory

       Set req = rf.Item(ReqId)

       req.Mail  mto , cc, 2, mSubject, mComment


       Set req = Nothing

       Set rf  = Nothing

       Set tdc = Nothing

End sub
```

Workflow Sample - Define a Setup

This workflow code updates field properties: visibility, mandatory, read-only, and order.

Add these functions to the Defect Module node.

Check User

Check if the user is in a certain group to decide on the next action.

```
If User.IsInGroup("Developer") then
       Mygroup=”DEV”
End if
```

Set Field Appearance

This subroutine sets field appearance – visibility, mandatory status, page number and order on the screen.

```
Sub SetFieldApp( FieldName, Vis, Req, PNo, VOrder )
    With Bug_Fields(FieldName)
           .IsVisible = Vis
           .IsRequired = Req
           .PageNo = PNo
           .ViewOrder = VOrder
    End With
End Sub
```

Reset to Initial

Add the following subroutine to hide all defect fields.

```
Sub ResetMetadata
       For i=0 to Bug_Fields.Count
           Bug_Fields.FieldById(i).IsVisible = false
       Next
End sub
```

Set Status

The following subroutine sets status New according to user permissions.

You have to write this subroutine for each status.

```
Sub Setup_Status_New
If User.IsInGroup("Developer") then
   Mygroup=”DEV”
ElseIf User.IsInGroup("QATester") then
   Mygroup=”QA”
ElseIf User.IsInGroup("Documentation") then
   Mygroup=”DOC”
End if

Call ResetMetadata ‘set to initial status
Select case Mygroup
  Case ”DEV”
     SetFieldApp "BG_ACTUAL_FIX_TIME", True, False, 0, 0
     SetFieldApp "BG_CLOSING_DATE", True, False, 0, 1
     SetFieldApp "BG_CLOSING_VERSION", True, False, 0, 2
     SetFieldApp "BG_DETECTED_BY", True, True, 0, 3
     SetFieldApp "BG_DETECTED_IN_RCYC", True, False, 0, 4
     SetFieldApp "BG_DETECTED_IN_REL", True, False, 0, 5
     SetFieldApp "BG_DETECTION_DATE", True, True, 0, 6
     SetFieldApp "BG_DETECTION_VERSION", True, False, 0, 7
     SetFieldApp "BG_ESTIMATED_FIX_TIME", True, False, 0, 8
     SetFieldApp "BG_PLANNED_CLOSING_VER", True, False, 0, 8
     SetFieldApp "BG_PRIORITY", True, False, 0, 10
Case ” QA”
     SetFieldApp "BG_ACTUAL_FIX_TIME", True, False, 0, 0
     SetFieldApp "BG_CLOSING_DATE", True, False, 0, 1
     SetFieldApp "BG_CLOSING_VERSION", True, False, 0, 2
     SetFieldApp "BG_DETECTED_BY", True, True, 0, 3
     SetFieldApp "BG_DETECTED_IN_RCYC", True, False, 0, 4
     SetFieldApp "BG_DETECTED_IN_REL", True, False, 0, 5
```

```
            SetFieldApp "BG_DETECTION_DATE", True, True, 0, 6
            SetFieldApp "BG_DETECTION_VERSION", True, False, 0, 7
       Case ”DOC”
            SetFieldApp "BG_ACTUAL_FIX_TIME", True, False, 0, 0
            SetFieldApp "BG_CLOSING_DATE", True, False, 0, 1
            SetFieldApp "BG_CLOSING_VERSION", True, False, 0, 2
            SetFieldApp "BG_DETECTED_BY", True, True, 0, 3
           SetFieldApp "BG_DETECTED_IN_RCYC", True, False, 0, 4
    End Select
End sub
```

4 Conclusions

The demand for relevant, well-performing software drives business innovation and success. The increasing business criticality of software, combined with the emergence of complex, disruptive trends such as virtualization and cloud, continue to drive the need for process improvement.

ALM meets the needs of the modern application lifecycle by providing increased alignment between teams, including integration into strategy and planning teams, an offering of best practices to spur innovation and prevent tactical delays, and a bridge to the critical last mile of the operations organization. ALM is extensible and dynamic — ready to adapt to the dynamic nature of ALM. Its flexibility allows for covering various industries, from pharmaceutical to car manufacturing, various types of development, from classic waterfall to modern agile, various organizational structures, from flat to hierarchical to matrix, and the list can go on.

In many ways, this the result of the many customization capabilities built into the product that provide the tools to differentiate the business processes unique to each organization that adopts ALM. Workflow scripting gives the power to the project administrator to adjust standard procedures and screens to the project's specific needs. This document provides insights into the usage patterns, shows benefits and disadvantages of different coding approaches, gives details on the most useful events, and is full of practical examples that assist in writing the code.

We believe that the best practices listed in this document help in the proper adoption of ALM workflow in your organization.