Enabling Application Lifecycle Development in JDeveloper

Susan Duncan
Oracle Corporation
United Kingdom

Keywords:

Application Lifecycle Management,ALM, JDeveloper, Team Productivity Center, SCM, Versioning

Introduction

Application Lifecycle Management has been defined as "The process of delivering software as a continuously repeating cycle of inter-related steps." It can be described as the management of the lifecycle development practices (requirements, build, test, change control, defect management etc) integrated together through application of process, reporting, traceability and collaboration. The better this integration, the better the software. However, in the past many of these practices worked in isolation - in functional silos that did not lend themselves to collaboration. But collaboration has become increasingly necessary in the software development world where teams work across organizations, time zones, cultures and business functions.

Oracle Fusion Middleware encompasses a number of features to facilitate this mode of distributed working where collaboration is paramount including remote deployment and debugging, a shared resource catalog, SCM system integration, built-in testing with JUnit, functional, load and test management with Oracle Application Quality Management Suite and lifecycle management of SOA composites with ClearApp.

This paper concentrates on some of the practical ALM features that are available to the application developer using JDeveloper in her day-to-day working environment

Versioning

One of the most obvious areas of ALM is versioning or source code management. It would be obvious to say that every development team's process includes SCM.

Generally there are two main types of versioning systems:

Lock – Modify – Unlock
Although a very safe and reliable system this is not necessarily best suited to a distributed development team. Files are locked by the developer modifying them, this can prevent other developers from working and the need to 'steal' locks back, leading to a loss in productivity. Systems that operate in this fashion include Perforce.

Copy – Modify – Merge
Tools that use this approach include the open source system Subversion, considered by many as the def-facto standard for Java development. Each developer works on a local copy of the files and merges any modifications back to the main development trunk once those changes have been tested.

Although this means that all developers can continue working it does rely on good communication between developers (after all, no tool or system is any good without good team processes in place). Conflicts can occur when multiple developers modify the same file. These should be resolved through communication between developers and at the local copy level, before the file in conflict is merged back into the main development trunk.

Note: for a full understanding of Subversion see http://svnbook.red-bean.com/

Branching

Branching is available in both types of versioning systems and is a feature of SCM that all organizations expect to make extensive use of. Branching is good for longer tasks – where a developer is able to isolate the changes that she is making to code on her 'branch' until such time as she decides to pull them in.

However, branches should always be used sparingly and teams should beware of over-branching. Simple quick tasks are better merged directly back into the main development track

Given that branching is used to isolate and develop new code, the merging back of this code will inevitably lead to conflicts with existing code at some point. Your tool should be able to successfully merge in some code (for instance if the tool is XML-aware and can distinguish between new properties or whitespace). For those conflicts that cannot be automatically resolved your tool is only as good as your team's development process. There is no substitute for developers communicating with each other to resolve conflicts.

Once a branch has been merged it is conceivable that the branch can be removed. Although it might seem obvious, there are many teams where this practice is not followed as they feel the need to keep every piece of code. But please consider it.

SCM Support in JDeveloper

Support for multiple SCM systems is integrated into JDeveloper including CVS, Perforce, Serena Dimensions and TFS. This paper briefly summarises the Subversion (SVN) support.

Development teams using JDeveloper can benefit from using the integrated SVN tooling available. Using the Versioning Navigator users can browse, edit and create their SVN repository structures as well as checkout code, create branches and tags and view individual files without the need to checkout.

Once a folder hierarchy (most often an application or perhaps a project) is checked out this becomes the 'working copy' for a developer, stored locally. This working copy can then be manipulated as an entity using the Commit Working Copy and Update Working Copy menu options. This ensures that all the folders/files in that working copy are collectively worked on. This is especially helpful if the developer is using Oracle's Application Development Framework (ADF) or another metadata framework where XML and Java files may be inter-dependent.

JDeveloper also provides a Pending Changes window that shows the developer which files have been added (Candidates) or edited (Outgoing) and are therefore candidates for committing to the repository. It also includes an Incoming window to show the developer any files that have been committed to the SVN repository by other users.

Developer productivity is enhanced through the use of the Pending Changes window along with the declarative interfaces to help with tagging, creating and merging branches, resolving any merge conflicts, adding properties and standardizing commit processes through the ability to create templates and standard comments.

But SCM is only one aspect of ALM and the remainder of this paper discusses Oracle's strategy to ALM for JDeveloper users'

Oracle's ALM Strategy
As stated earlier ALM can be described as (requirements, build, test, change control, defect management etc) integrated together through application of process, reporting, traceability and collaboration. Oracle's strategy is to continue its aim of being complete, integrated, best of breed and hot pluggable when it comes to ALM. In many organizations the different tools, repositories and procedures that have are being followed are disparate and not integrated. Oracle's aim is to provide a collaborative environment in JDeveloper where teams can integrate with their ALM repositories of choice and share information through centralized services that allow for customization, reporting, relationship management and process automation.

This is achieved through Oracle Team Productivity Center, an Application Lifecycle Management tool that enables software development teams to collaborate and work productively together when developing applications using JDeveloper.

Introducing Oracle Team Productivity Center
Oracle Team Productivity Center provides a framework to enable third party Application Lifecycle Management tools to be integrated into Oracle JDeveloper. These repositories include task and project management, version control, document management, software bug reporting, and build and management systems. The integration of repositories in JDeveloper enables users to directly interact with existing ALM artifacts whilst working in their IDE. Additionally, Oracle TPC provides configurable team management facilities designed to improve productivity and communication among team members.

Oracle JDeveloper supports the complete development lifecycle with integrated features for modeling, coding, debugging, testing, profiling, tuning and deploying applications. With the addition of Oracle Team Productivity Center developer productivity is improved and functional silos are minimized through flexible user and team maintenance; query and update of third party ALM repositories; linking of disparate ALM artifacts through extensible tagging and relationship and context management facilities; content management at artifact and team level; enhanced developer collaboration through integrated chat client

Installation
Oracle Team Productivity Center server install is downloadable from Oracle Technology Network (OTN) as a platform-independent JAR file. Full details are contained in the Oracle Fusion Middleware Installation Guide for Oracle Team Productivity Center Server.

The client software is a JDeveloper extension. It is available by selecting Help ->Check for Updates, and is installed like any other JDeveloper extension.

Connectors to enable the integration of ALM repositories require both a server install (as detailed in the installation guide) and installation in JDeveloper as extensions. The Oracle JDeveloper Help Center contains detailed help on installing connectors and using Oracle Team Productivity Center. This help is available once the client software has been installed

Architecture

The basic architecture of the Team Productivity Center product is comprised of three main components illustrated in Figure 1

- JDeveloper Client Model
- Team Productivity Center Server
- Team Productivity Center Connectors

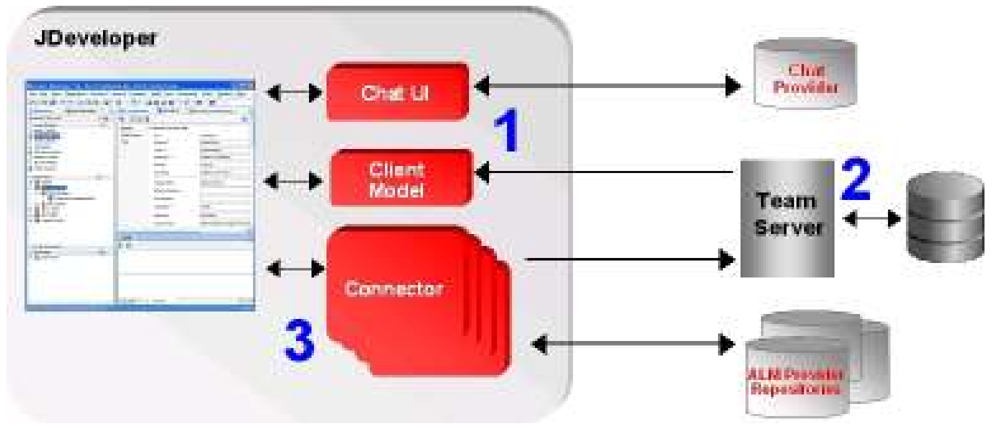

Illustration. 1: TPC Architecture

The JDeveloper Team Productivity Center extension acts as the TPC client framework (1). The Team Navigator drives all team related features. Part of the TPC extension package also includes a chat interface. JDeveloper users can collaborate with team members and others connected to the chat provider directly in JDeveloper without having to leave their IDE.

The TPC team server (2) allows JDeveloper clients to connect and retrieve team related information. The TPC server also has a password vault for each user to enter credentials in order to seamlessly connect to their 3rd party repositories. The team server stores centralized information (such as tags, relationships, team information) in a database schema. This database also provides storage of the built-in Task repository that teams can choose to install.

The TPC connector infrastructure (3) allows ALM repositories to be instantiated and a UI exposed directly inside of JDeveloper. Through the published interface built on open standards, the connector can expose repository fields using a meta-driven user interface and support basic create/update/delete operations on those objects. Each connector can support a number of artifact object types each with multiple user interface definitions. The connector also defines what fields in each artifact type are queryable by the user and how to query them. By implementing the connector interface and supplying some XML metadata to define a connector, a developer can quickly integrate their own repository into the Team Productivity Center framework, allowing any 3rd party repository to easily be exposed inside of JDeveloper. The exception is the TPC task repository, It is accessible through the Task connector and is shipped as part of TPC. This connector exposes fields and operations on the task repository; the only difference is that this repository is hosted in the TPC database. For teams that do not have access to other ALM repositories this allows them to start defining tasks and working with TPC with no additional infrastructure

Team Navigator

Once Oracle Team Productivity Center is installed the Team Navigator menu appears in the Team menu. The Team Navigator is the access point for Oracle Team Productivity Center in JDeveloper. Users can view their assigned teams, initiate chat conversations, browse their ALM repositories and also, with the correct privileges, carry out administration tasks such as editing users, setting repository connections etc.

Chat can be used regardless of whether the user is logged into the Team Server or not. The chat functionality is provided as part of the Oracle Team Productivity Center client install.

Work Items

In Oracle Team Productivity Center the term 'Work Item' is applied to any artifact from a connected ALM repository, for instance a JIRA issue. Repository queries can be defined and run against each repository. Queries can be visible either at the team or the user level..

It should be noted that the work items returned by a query are not stored by Team Productivity Center. Instead, the query criteria are saved and the query is performed against the relevant repository when the query is run.

The integration of Application Lifecycle Repositories into Oracle JDeveloper is the lifeblood of Oracle Team Productivity Center. This integration enables JDeveloper users to work more productively with these systems and take advantage of the additional centralized services that Team Productivity Center provides to link work items from disparate repositories together.

Versioning

The versioning accordion allows a user to login and interact with their team's versioning system using the Team Navigator. The functionality provided mirrors that of the standalone Versioning Navigator. At the team level, the team administrator can specify the repository location of the team's versioned items. One advantage of this is that a new team member can be provided with their Team Productivity Center login and the Team Navigator will provide them with the information they need to review the team's repositories and documents from within JDeveloper,

Working With Work Items

Oracle Team Productivity Center allows users to work with individual work items from within the JDeveloper IDE.

Detail

For each work item (i.e. for each repository type: JIRA, Task, Bugzilla etc) the fields that are displayed, which fields are updateable, the type of validation that is carried out, and the interactions available with the repository are all determined by that repository's API.

Relationships

Oracle Team Productivity Center provides a mechanism for contextual linking between work items from the same or different repositories. For instance a relationship between a requirement in a JIRA repository and the associated task in the Task project can be created. This allows any user browsing either end of the relationship to open the associated work item(s), provided they have access to that repository. This is one of the major benefits of Oracle Team Productivity Center – the ability to integrate disparate repositories together

Tags
A less formal method grouping work items together is to use tags. Tags differ from relationships in that they apply only within a team. It's possible to filter a list of tagged items by repository by querying by tag on a specific repository. The team server stores the tag list details.

Attachments
Attachments provide a mechanism for uploading either documents or references (URL) to a pre-specified content management system associated with the work item repository. This functionality is dependent on the relevant API being available for the repository type. For instance, in the first release of Oracle Team Productivity Center this API is not available for either JIRA or Task repositories

Changes
At commit time users can record the details of files checked in to their SCM system against one or more work items.

When you install the TPC client an additional field is added to any installed SCM system integration that supports TPC (including SCM, CVS). The user has the opportunity to associate one or more work items with a check in and record the details of checked in files against that work item.

Context
Context is the method by which a user can save the current state of their JDeveloper IDE (window layout, open files) against a specific work item within a team. This allows the user to be more productive when having to switch between multiple tasks or bug fixes, for example, and quickly restore their IDE when ready to resume work on the task against which context has been stored.

Active *Work* item
The Active Work Item provides a way for a user to mark a specific work item as current. There are a number of reasons that setting your current work item is beneficial:

- The work item can be quickly opened directly from the link in the navigator for quick reading and updating
- The work item is automatically added to the 'associate check-in with work item' field on an SCM check-in dialog.

Administration
After initial installation, administration of Oracle Team Productivity Center is done from the administration dialog. This is found in the Administration menu as stated at the beginning of this paper.

The dialog consists of four tabs:

- Users – where user details are recorded
- Teams – where teams, team members and team repositories are configured
- Repositories – where ALM repository connections are setup and administered
- Roles – where roles and privileges are defined

Build and Deploy
JDeveloper deployment is built around deployment profiles and associated descriptors. The descriptor files required depend on the technology the application uses and also on the target application server.

An application can be deployed directly to various application server (Oracle WLS, JBoss, Tomcat, Websphere) from within the IDE. JDeveloper's help center provides full details of creating the appropriate deployment profile and how to connect to the application server of your choice and deploy. Alternatively, applications can be deployed indirectly to an archive file for later installation on an application server.

Many organizations prefer to deploy using a batch file or other script, perhaps to include in a continuous integration server. JDeveloper provides a command-line tool, OJDEPLOY, to allow deployment of Archive Profile(s) without invoking the JDeveloper IDE.

For more complex builds there is an Ant wrapper for OJDEPLOY that you can develop and run within JDeveloper if you wish.

Support for another build tool, Maven, is also available in JDeveloper as a Developer Preview extension available through the Update Center. Maven differs from Ant in that it enforces a common project structure as well as providing ways to compile and build applications.

Continuous Integration
As part of a complete application lifecycle development process, especially where teams are using any kind of agile approach or are distributed and users are to take ownership in quick bug fixing, continuous integration (CI) is becoming more and more important.

Essentially continuous integration involves keeping application code in a repository, integrating the work of developers frequently, automatically and frequently building the code. This ensures that errors are discovered more quickly thereby reducing fix time. It also makes the development available to everyone to install and take a stake a stake in its maintenance.

To work effectively a continuous integration server should run automatically, triggered by operations from the application's versioning repository and have integrated tests running with each build. Also essential is feedback to the team of build results (perhaps via email, IM, RSS feeds…) together with analysis of results through reporting and graphs over time.

Hudson
Hudson is a continuous integration server. It is an open source community project sponsored by Oracle who provides the infrastructure to the community. It is very well used in the Java community and provides the essentials and much more outlined above. It is very easy to install and configure with a web-based interface and through its extensive and ever-growing set of plugins provides users with tools for distributed building, unit test reporting, file fingerprinting, status notifications, SCM system integration, authentication and much more.

As part of the CI process there are basic questions that need to be addressed:

- Did it build?
  - Hudson accesses your SCM repository of choice as and when it is needed and extracts the required files
  - You configure how the application should be built – perhaps using OJDEPLOY, Ant, Maven
  - Hudson provides dashboards and details feedback on all builds (successful or not)
- Does it deploy?

  - If an application does not deploy the feedback and reports generated give you information as to why not
- Does it perform as expected?
  - Hudson can be integrated with various testing tools to provide headless and runtime testing

Testing
There are very many tools and approached that can be used both within and outside of a continuous integration process. This paper highlights just a very few of those options both integrated into JDeveloper and externally that the author has found useful

Auditing
JDeveloper provides auditing tools for the static analysis of code adherence to programming standards: the rules and metrics defined for the application. The rules are maintained using audit profiles. There are a number of predefined profiles in JDeveloper that teams can use or redefine their own to establish the rules, code assists and metrics that should be applied. Profiles can be imported and exported for sharing. Auditing can be invoked directly in the IDE or from the command line

Unit Testing with JUnit
JUnit is an open source regression testing framework for Java. Using JDeveloper you can create test fixtures, cases and suites using a wizard driven interface. In addition there are specialized wizards in JDeveloper for developing tests for ADF Business Components.

ADF Logger
ADF Logger extends the J2SE standard logging mechanism java.util.logging package. It is a very useful and adaptable logging utility for debugging and performance testing. It is a wrapper class that, if used by teams in a disciplined manner, can be added to the arsenal of weapons in testing. It is particularly helpful in debug and performance testing.

Many developers using System.out.println statements throughout their code to print information to the standard output stream. If they replace this practice for java.util.logging they will find there are many advantages:

- The log file can be filtered (either using Oracle Enterprise Manager 11g Fusion Middleware Control or the Oracle JDevloper 11g Oracle Diagnostic Log Analyzer) to view log messages at different levels depending on the granularity that is set
- Log entries can include time, class and method name etc for better interpretation and problem solving. All the related messages from a particular request can be viewed together, again helping debugging and problem solving and performance testing
- ADF Logger can be turned on and off. So statements can be left in the code for production with no impact on performance, but can even be turned on again to help problem solving in both a development and production environment

Profile Monitors
JDeveloper's CPU and Memory profiler monitors and logs a running application's use of processor and memory resources. Profiling can be done either locally or remotely and allows you to more easily identify both performance and coding inefficiency problems in the code.

The CPU profiler logs the processing time spent in each method in your application, as well as counting calls to those methods, generally you display these stats in the Hotspots view. In addition the Call Stacks view shows the call hierarchy of the methods called, perhaps sorted by thread group.

The Memory profiler helps you detect memory leaks by identifying which parts of the application use the most memory. It is available in three views:

- The Classes view shows new objects/garbage collection data organized by Java class allowing you to identify which classes are allocated the most memory
- The Allocators view shows the most memory intensive threadgroups, threads and methods
- The References view displays the application heap at the time of the snapshot using a hierarchical display of classes, objects and references

External Runtime testing
There are a number of invaluable tools to help with runtime testing.

Abbot
Abbot is an open source framework for the functional testing of Java GUIs. It is particularly suited to testing swing based clients such as JClient. Generally it is used in conjunction with Costello (itself built on Abbot) that can easily launch an application allowing you to control and explore it. You can record user actions directly into a script and add additional steps to control the event playback and testing.

Selenium
Selenium is a very simple extension to Firefox that allows you to record and playback web-based applications, again as part of your testing arsenal. The tests can be saved in a number of languages (HTML, Java C#, Python etc) and integrated into systems such as Hudson for automated testing of your running applications. The tests can be run locally or using Selenium's Remote Control or Grid configurations.

Firebug
Firebug is another extension to Firefox that is sometimes invaluable in the editing, debugging, logging and analysis of web-based applications

Conclusion
JDeveloper provides application lifecycle support in many different ways, from its multiple versioning system integration in the IDE to the team collaboration and external ALM repository integration available through Team Productivity Center. In addition it provides a wealth of testing capabilities from unit to system and performance testing through its built-in support for JUnit and its audit and profiling tools.

This paper has also identified a number of external tools available to enhance the UI system and performance testing of Java and web UIs

Contact address:

Susan Duncan
Oracle Corporation
Thames Valley Park
Reading, UK

Phone:	+44(0) 118 924 5896
Email	susan.duncan@oracle.com
Internet:	http://www.susanduncan.blogspot.com