Professional Software Development using Oracle Application Express

Ciber Nederland
Rob van Wijk
rob.van.wijk@ciber.com

March 2013
Version 1.0

Summary

Software development involves much more than just producing lines of code. It is also about version control, deployment to other environments, integrating code and unit testing. It is appropriate to the profession of software development to have a framework in place that handles all of this, in order for the developers to focus on the creative and fun part of their job: producing excellent and well-designed code. Without such a framework, you need to be cautious and deployments become more difficult and more error-prone. Which means more time and money needlessly spent and a development process which is less fun than it should be. Fortunately, these problems are well known and solutions are already widely adapted. However, in database application development in general and in APEX development specific, these practices are not so common, unfortunately. It is our belief that database and APEX development should not be treated different and deserve a similar framework. So that's what we set out to do.

This paper describes how we develop new APEX applications in such a way that most of the effort does gets spent on actually developing the code. If you can take advantage of version control, if you can build and deploy your entire application in one step, can make a daily build to continuously integrate all developed code, and can make sure your developers have their own self-contained development environment, then the benefits are many. You'll experience less errors, higher productivity and seamless software delivery from your team. Refactoring code becomes painless since you can easily be assured the changes won't break the application. Overall quality goes up, ensuring a much longer lifetime of the application.

This paper is the first in a series of three. The other two parts will focus on how to integrate back-end and front-end unit testing into this framework and on how to roll out incremental changes to your databases and your APEX application. This first part describes the base setup, needed before you can implement the other two setups. It provides enough information to get you started developing a new APEX application, albeit without the added benefits of unit testing yet. If you already have some setup at your own environment, this paper will hopefully point you to areas where you can improve.

Chapter 1: Version Control

“If you don't have source control, you're going to stress out trying to get programmers to work together. Programmers have no way to know what other people did. Mistakes can't be rolled back easily.” - Joel Spolsky [1]

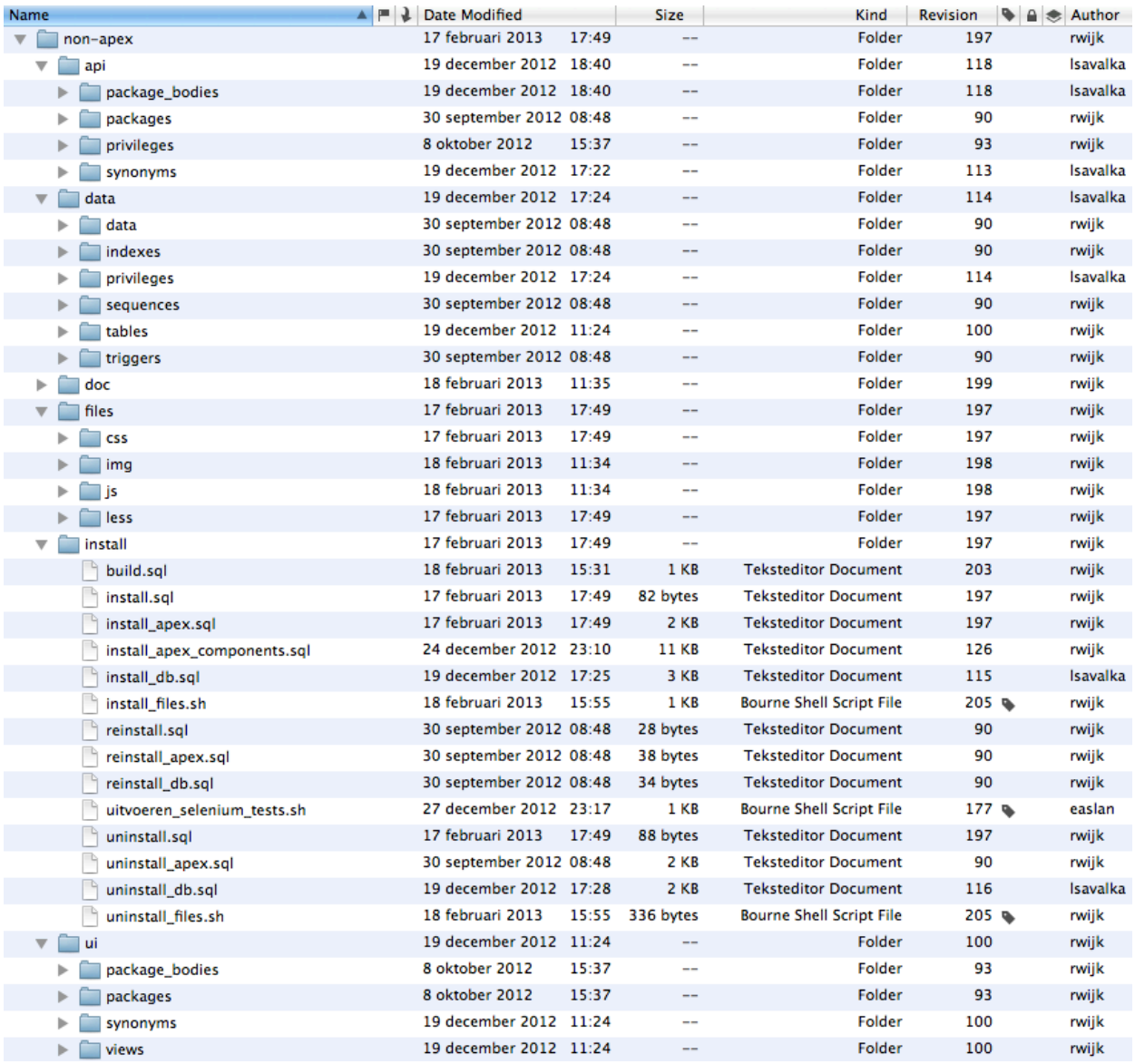
The first and probably the easiest step towards a more professional development environment, is to have version control. With version control in place, you have the history of all the files in your application. The other benefit of version control is that it makes sharing changes much easier. By frequently updating your checked out version of the application, you automatically incorporate the changes made by your colleagues. The chance that you will overwrite their code or vice versa, losing changes, is eliminated. Also, a welcome side effect of having source control is that all source code will be checked out on each developers computer, greatly reducing the risk of ever losing code.

There are lots of version control tools to choose from. We chose Subversion over git, CVS and Serena Version Manager, simply because that’s the one with which most of us are already familiar. And at the client side -on our laptops- we chose TortoiseSVN for Windows and Cornerstone on the Mac. It doesn’t really matter which version control tool you choose for your own project, as long as you choose one.

In our strategy, version control is more than just history and having control over changes. Version control is the single point of truth. It doesn’t matter how source code looks like in any database schema; the code in version control is what counts. PL/SQL code in the database and even data in the development database doesn’t matter. You can happily throw away database objects, data or even perform “drop user X cascade” commands. The developer can always restore the database without a DBA.

The other way round, if code is not in version control, it doesn’t exist. As a result, everything in your application should be under version control. Not only APEX object definitions, table definitions and packages, but also privileges, master data, images, CSS, installation scripts and so on. In short, everything that is needed to fully create the application at the site of the client. If you have a database with APEX installed and the APEX listener running, you can install the entire application from version control.

The structure of files and folders in version control
As adviced by Apache and as followed by many, we have the three standard directories in our root: trunk, tags and branches. All our applications reside inside the trunk directory. We aim for exactly the same folder structure for every APEX application. We split the application in Subversion between two folders: apex and non-apex, for a reason explained later in this chapter. The non-apex folder contains all database-objects and files which support the APEX application. The structure of that folder is shown in figure 1:



Name	Date Modified	Size	Kind	Revision	Author
non-apex	17 februari 2013 17:49	--	Folder	197	rwijk
api	19 december 2012 18:40	--	Folder	118	lsavalka
package_bodies	19 december 2012 18:40	--	Folder	118	lsavalka
packages	30 september 2012 08:48	--	Folder	90	rwijk
privileges	8 oktober 2012 15:37	--	Folder	93	rwijk
synonyms	19 december 2012 17:22	--	Folder	113	lsavalka
data	19 december 2012 17:24	--	Folder	114	lsavalka
data	30 september 2012 08:48	--	Folder	90	rwijk
indexes	30 september 2012 08:48	--	Folder	90	rwijk
privileges	19 december 2012 17:24	--	Folder	114	lsavalka
sequences	30 september 2012 08:48	--	Folder	90	rwijk
tables	19 december 2012 11:24	--	Folder	100	rwijk
triggers	30 september 2012 08:48	--	Folder	90	rwijk
doc	18 februari 2013 11:35	--	Folder	199	rwijk
files	17 februari 2013 17:49	--	Folder	197	rwijk
css	17 februari 2013 17:49	--	Folder	197	rwijk
img	18 februari 2013 11:34	--	Folder	198	rwijk
js	18 februari 2013 11:34	--	Folder	198	rwijk
less	17 februari 2013 17:49	--	Folder	197	rwijk
install	17 februari 2013 17:49	--	Folder	197	rwijk
build.sql	18 februari 2013 15:31	1 KB	Teksteditor Document	203	rwijk
install.sql	17 februari 2013 17:49	82 bytes	Teksteditor Document	197	rwijk
install_apex.sql	17 februari 2013 17:49	2 KB	Teksteditor Document	197	rwijk
install_apex_components.sql	24 december 2012 23:10	11 KB	Teksteditor Document	126	rwijk
install_db.sql	19 december 2012 17:25	3 KB	Teksteditor Document	115	lsavalka
install_files.sh	18 februari 2013 15:55	1 KB	Bourne Shell Script File	205	rwijk
reinstall.sql	30 september 2012 08:48	28 bytes	Teksteditor Document	90	rwijk
reinstall_apex.sql	30 september 2012 08:48	38 bytes	Teksteditor Document	90	rwijk
reinstall_db.sql	30 september 2012 08:48	34 bytes	Teksteditor Document	90	rwijk
uitvoeren_selenium_tests.sh	27 december 2012 23:17	1 KB	Bourne Shell Script File	177	easlan
uninstall.sql	17 februari 2013 17:49	88 bytes	Teksteditor Document	197	rwijk
uninstall_apex.sql	30 september 2012 08:48	2 KB	Teksteditor Document	90	rwijk
uninstall_db.sql	19 december 2012 17:28	2 KB	Teksteditor Document	116	lsavalka
uninstall_files.sh	18 februari 2013 15:55	336 bytes	Bourne Shell Script File	205	rwijk
ui	19 december 2012 11:24	--	Folder	100	rwijk
package_bodies	8 oktober 2012 15:37	--	Folder	93	rwijk
packages	8 oktober 2012 15:37	--	Folder	93	rwijk
synonyms	19 december 2012 11:24	--	Folder	100	rwijk
views	19 december 2012 11:24	--	Folder	100	rwijk

Figure 1: Folder structure for non-APEX objects

The first thing you'll notice, is the split up of our database objects in three parts: data, api and ui. These folders correspond with three physically separate database schemas in which the database objects reside. This layered approach is a choice we've made to enhance security and flexibility in our applications. The three schemas only have a minimal set of system privileges, just enough to create the object types needed for that layer. The schemas do not even have a CREATE SESSION privilege. And their passwords are generated with dbms_random.string('a',30). So objects need to be created from an appropriately privileged schema that issues an "ALTER SESSION SET CURRENT_SCHEMA = ..." command. Because of the schemas lack of a CREATE SESSION privilege, we ensure that all installations are done via install scripts.

Schematically, the schema structure looks figure 2 below:

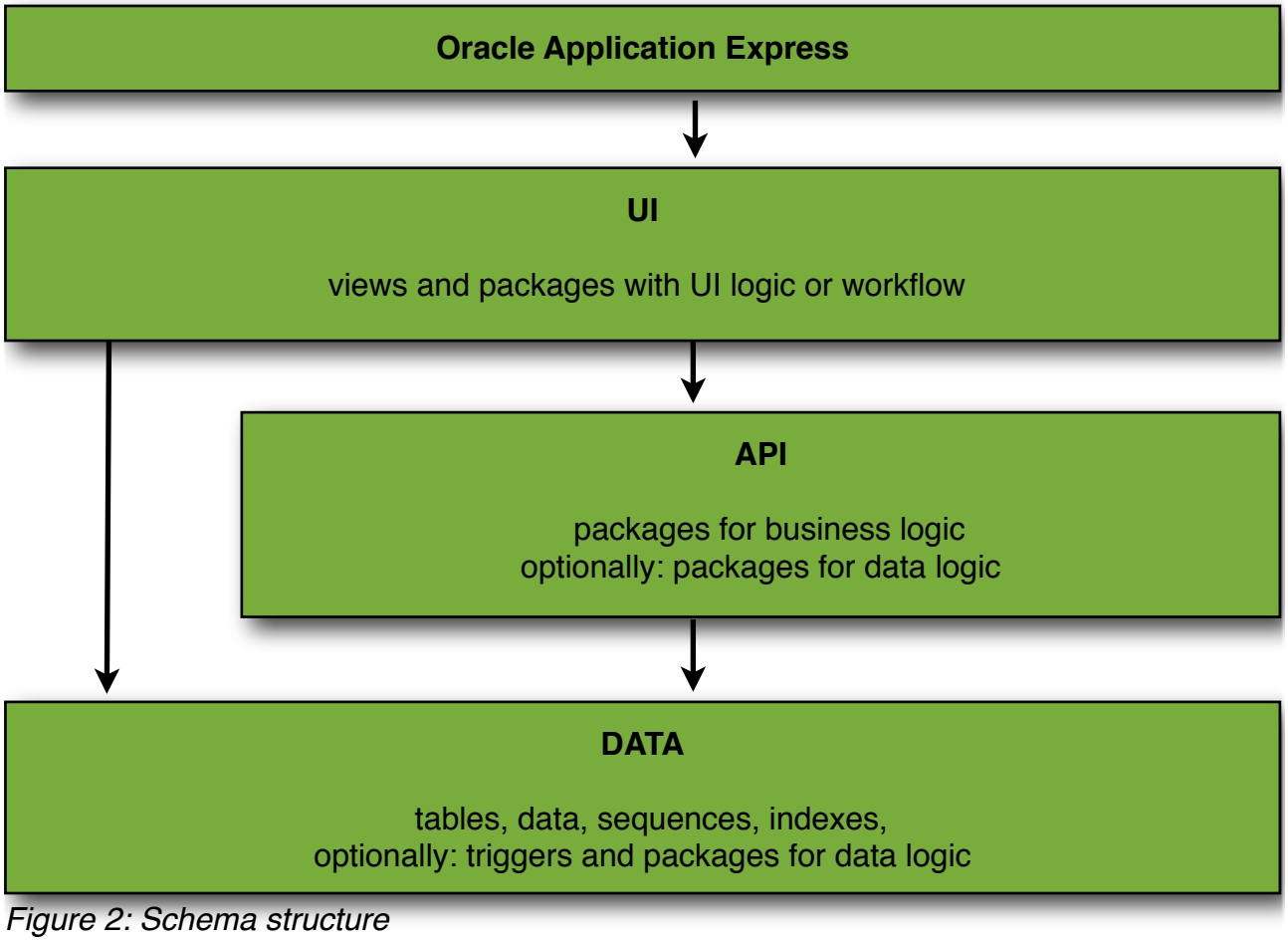


Figure 2: Schema structure

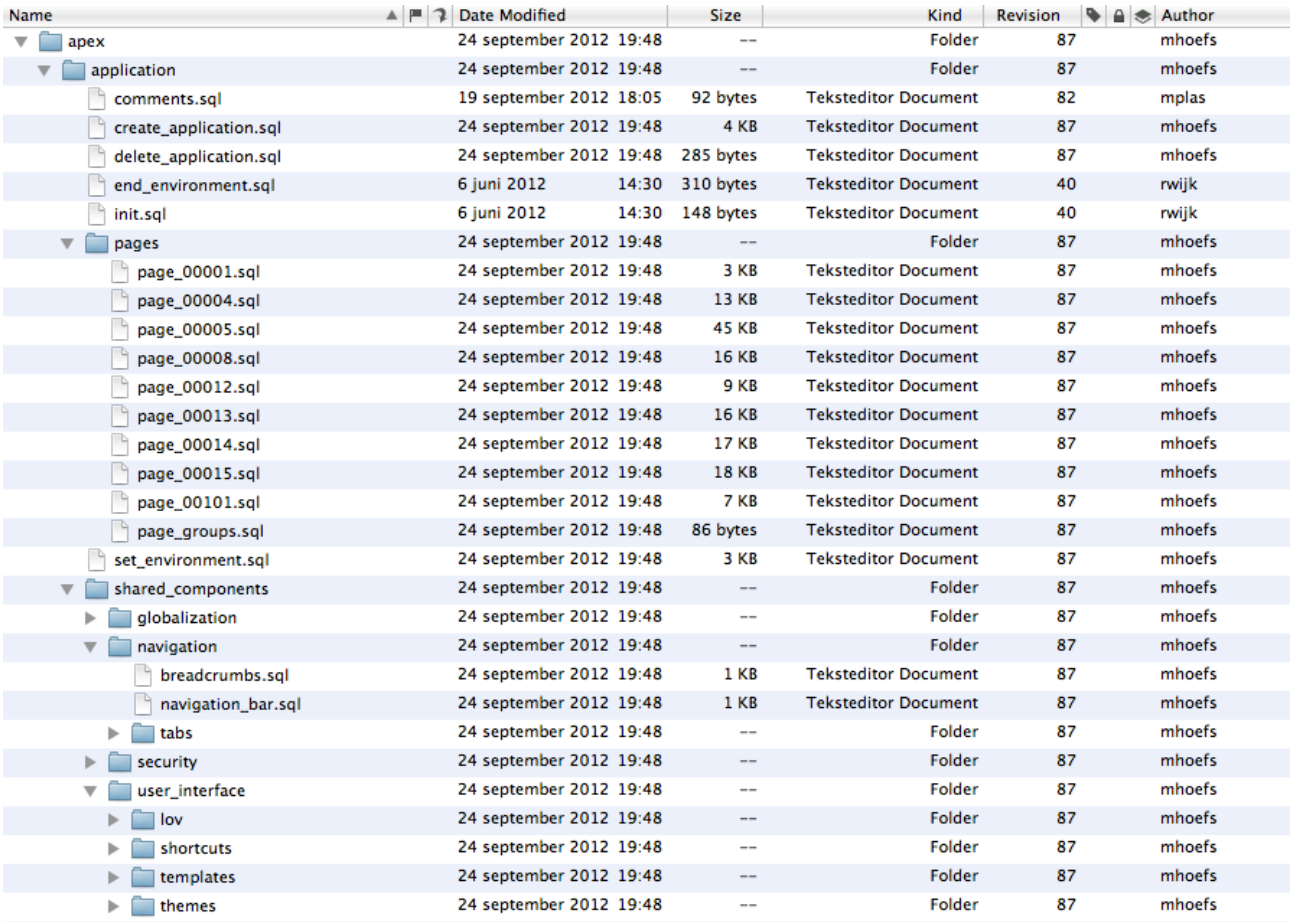
The UI schema is the parsing schema of the APEX application. It contains just the objects needed for the APEX application to function. There are views, directly on top of tables in the DATA layer and packages, either packages with UI logic or UI workflow packages orchestrating business logic in the API layer. Data constraints are implemented declaratively as much as possible. The data constraints that cannot be implemented declaratively, are either implemented by database triggers and data logic packages in the DATA layer, or as data logic packages in the API layer. There is quite a lot of debate around the subject of where and how to implement that data logic. The idea is to ask the client if he has preferences and -if not- leave the decision to our lead developer.

In the three layers, each database object is stored in its own file. For example, if the application has 10 tables, we have 10 files with a *.tab* extension in the tables folder. There are some decisions to make here about which object deserves its own file. For example, do you place CREATE INDEX statements in its own .idx file or do they go along with the table definition. These decisions are of minor importance, as long as you are consistent.

Then there is the files folder, where all images, cascading style sheets (CSS), javascript libraries and less [8] source files are located that are needed for the files on the application server.

APEX source files, under the apex folder, require some extra attention. Version control is file based, but an APEX application exists inside the database as rows inside the APEX repository tables. APEX helps to move your application to files by providing two undocumented Java programs: APEXExport and APEXExportSplitter. Both are command

line versions of actions you can perform inside the Application Builder as well. APEXExport produces one large file which you can use to import the entire application. The name of that file is *f[application ID].sql*, for example *f10001.sql*. The second Java program, APEXExportSplitter, creates a directory structure where each APEX component is in a separate file, ideal for version control. Figure 3 below depicts a directory structure created by APEXExportSplitter.



Name	Date Modified	Size	Kind	Revision	Author
apex	24 september 2012 19:48	--	Folder	87	mhoefs
application	24 september 2012 19:48	--	Folder	87	mhoefs
comments.sql	19 september 2012 18:05	92 bytes	Teksteditor Document	82	mplas
create_application.sql	24 september 2012 19:48	4 KB	Teksteditor Document	87	mhoefs
delete_application.sql	24 september 2012 19:48	285 bytes	Teksteditor Document	87	mhoefs
end_environment.sql	6 juni 2012 14:30	310 bytes	Teksteditor Document	40	rwijk
init.sql	6 juni 2012 14:30	148 bytes	Teksteditor Document	40	rwijk
pages	24 september 2012 19:48	--	Folder	87	mhoefs
page_00001.sql	24 september 2012 19:48	3 KB	Teksteditor Document	87	mhoefs
page_00004.sql	24 september 2012 19:48	13 KB	Teksteditor Document	87	mhoefs
page_00005.sql	24 september 2012 19:48	45 KB	Teksteditor Document	87	mhoefs
page_00008.sql	24 september 2012 19:48	16 KB	Teksteditor Document	87	mhoefs
page_00012.sql	24 september 2012 19:48	9 KB	Teksteditor Document	87	mhoefs
page_00013.sql	24 september 2012 19:48	16 KB	Teksteditor Document	87	mhoefs
page_00014.sql	24 september 2012 19:48	17 KB	Teksteditor Document	87	mhoefs
page_00015.sql	24 september 2012 19:48	18 KB	Teksteditor Document	87	mhoefs
page_00101.sql	24 september 2012 19:48	7 KB	Teksteditor Document	87	mhoefs
page_groups.sql	24 september 2012 19:48	86 bytes	Teksteditor Document	87	mhoefs
set_environment.sql	24 september 2012 19:48	3 KB	Teksteditor Document	87	mhoefs
shared_components	24 september 2012 19:48	--	Folder	87	mhoefs
globalization	24 september 2012 19:48	--	Folder	87	mhoefs
navigation	24 september 2012 19:48	--	Folder	87	mhoefs
breadcrumbs.sql	24 september 2012 19:48	1 KB	Teksteditor Document	87	mhoefs
navigation_bar.sql	24 september 2012 19:48	1 KB	Teksteditor Document	87	mhoefs
tabs	24 september 2012 19:48	--	Folder	87	mhoefs
security	24 september 2012 19:48	--	Folder	87	mhoefs
user_interface	24 september 2012 19:48	--	Folder	87	mhoefs
lov	24 september 2012 19:48	--	Folder	87	mhoefs
shortcuts	24 september 2012 19:48	--	Folder	87	mhoefs
templates	24 september 2012 19:48	--	Folder	87	mhoefs
themes	24 september 2012 19:48	--	Folder	87	mhoefs

Figure 3: Folder structure of APEX application objects

When updating your working copy using *svn update*, you are incorporating the committed changes of your colleagues into your working copy. Normally, when you have made some changes to the source files as well, Subversion will nicely merge the two deltas. And when the changes cannot be merged successfully -because they involve the same lines of code- you'll get a conflict, which you resolve manually. But with APEX, we -the mere mortal ones- are not modifying the APEX source files, but we modify the database contents of the APEX repository. So, when you are now performing an *svn update*, followed by an export and an export split, you'll overwrite the modifications of your colleagues in your working copy. Meaning that a check in (*svn commit*) will now lose the changes of your colleague(s) will be lost in the current version. And you'll have to perform a tediously accurate manual merge of the two deltas to restore the situation.

This overwriting cannot be fully prevented, but you can do something about it. We implemented the easiest method:

* warning everyone for this scenario,
* choosing a directory structure that allows an easy *svn update* on all non-APEX objects. That's why we have two directories, directly under the project directory: apex and non-apex.

* a special apexupdate.sh script which correctly incorporates the changes of colleagues into your working copy and your APEX workspace. This script ensures an export of your APEX application and an *svn update* are always performed together in the right order. The structure of this script looks like this:

```
rm -r $HOME/ciber/apexsofa/$1/apex/.
cd $HOME/ciber/apexsofa/$1
java oracle.apex.APEXExport\
 -db ourserver:1521:APEXSOFAO\
 -user apex_040200\
 -password secret\
 -applicationid $2\
 -skipExportDate
java oracle.apex.APEXExportSplitter f$2.sql
mv f$2 apex
cd apex
svn status | grep ^\? | awk '{print $2}' | xargs svn add
svn status | grep ^\! | awk '{print $2}' | xargs svn delete --force
svn update
sed s_@application_@$1/apex/application_\
 <install.sql\
 >../non-apex/install/install_apex_components.sql
cd ..
. non-apex/install/reinstall_apex
```

In this script, $1 equals the application code and name of the top directory, and $2 is the APEX application id of the application that is to be exported. It removes all files from the apex directory in your working copy and replaces them with freshly exported data from your APEX workspace, and it registers the new files (*svn add*) and deletes the obsolete files (*svn delete*). And only now that the working copy is up to date with your delta, we perform the *svn update*, incorporating the delta of your colleagues into your up-to-date working copy. The last step is to reinstall the APEX application in the database, by deleting the application in your workspace and installing it back.

The script works perfectly when *svn update* doesn't produce conflicts. But when it does produce conflicts, the installation of the APEX application in your workspace, won't work, because files with conflicts have conflict markers in them that won't compile/run. In this case, you just have to resolve the conflicts like you would normally do and run reinstall_apex again.

A nice enhancement, would be to make the script check the revision number your APEX application is based on, against the revision number of your working copy. If you have done an *svn update* on your apex folder, the revision number of your working copy would be higher than the one on which your APEX application is based on. In that case the script could be stopped, allowing the developer to revert the *svn update* by doing a *svn update -r [revision number APEX is based on]*.

Chapter 2: Self-contained Development Environments

"*The developer should be free to experiment as much as possible, safe in the knowledge that the worst that could happen is they destroy only their own environment and not impact the productivity of others*." - Nick Ashley [2]

The next step addresses a very common issue, which most Oracle database developers have likely experienced more than once: working with several developers under the same database schema. The troubles with this are many. For example, not being able to test your code when a colleague has just invalidated the schema while developing his latest enhancement. Or losing test data because someone has dropped and recreated a table. Or when a colleague accidently reinstalls the entire schema, while not having updated his working copy and thereby losing your changes and corrupting your test results. Problems increase with the number of developers and with diminishing communication. I have found the scenario of working together in one schema troublesome even in a small RUP team of just 2-3 developers. So, for us, an absolute must for our environment is to have a completely separate working environment for each developer. The integration issues that may arise from working separately, are discussed in chapter 4.

A separate working environment in an APEX environment means that every developer has his own APEX workspace which holds the applications that he works on. And, keeping in mind that in our way of working, database application needs three schemas DATA, API and UI, we give each developer three database schemas as well. We named these developer schemas after the username, with a suffix "_DATA", "_API" and "_UI". The username can follow any convention you like -within the boundaries of what Oracle allows-, but should not exceed 25 characters because Oracle doesn't allow usernames longer than 30 characters. In my case for example, the three developer schemas are called RWIJK_DATA, RWIJK_API and RWIJK_UI.

The developer schemas have the same minimal set of privileges as the application schemas described in chapter 1. The APEX application gets coupled to the user schemas by setting up the developers UI schema as the parsing schema in their APEX workspace. As with the application schemas, you cannot use these schemas to log in; they lack the CREATE SESSION privilege and (superfluous) the password is unknown because it was generated with "dbms_random.value('a',30)". So, each developer also needs a user he can use to log on and to use for installing objects in the other schemas. This username is without any suffix, so in my case: RWIJK. This user has the DBA role. The development environment is their environment, after all. And it's impossible to do any real harm, since version control is leading. Using the "*ALTER SESSION SET CURRENT_SCHEMA = ...*" mechanism, this user can install in the three developer schemas.

To be able to use the same scripts to install in either the application schema or in one of the developers schemas, the synonym and privilege scripts in Subversion are parameterized. For example, a script for granting select privileges on a table to the UI layer contains a "*grant select on [tablename] to &SCHEMAPREFIX._ui;*". And a synonym script contains a "*create synonym [tablename] for &SCHEMAPREFIX._data.[tablename]*". Both scripts start with a "*define SCHEMAPREFIX='&1'* ".

The object ID challenge

Having a separate APEX workspace for each developer imposes an extra challenge since you cannot import the same APEX application into two different workspaces in the same

APEX instance. APEX uses instance-wide unique internal object ID's. When you import the application for the second time, you'll violate unique constraints on these object ID's. You can see the object ID's in the export file, where you can recognize them as the long numbers, that are always followed by " *+ wwv_flow_api.g_id_offset*". See for example the excerpt from an export file below:

```
declare
    h varchar2(32767) := null;
begin
wwv_flow_api.create_page_item(
  p_id=>1922330433247936 + wwv_flow_api.g_id_offset,
  p_flow_id=> wwv_flow.g_flow_id,
  p_flow_step_id=> 101,
  p_name=>'P101_USERNAME',
  p_data_type=> '',
  p_is_required=> false,
  p_accept_processing=> 'REPLACE_EXISTING',
  p_item_sequence=> 10,
  p_item_plug_id => 1922223885247934+wwv_flow_api.g_id_offset,
  p_use_cache_before_default=> '',
  p_prompt=>'Username',
  p_display_as=> 'NATIVE_TEXT_FIELD',
  p_lov_display_null=> 'NO',
  p_lov_translated=> 'N',
  p_cSize=> 40,
  p_cMaxlength=> 100,
  p_cHeight=> null,
  p_begin_on_new_line=> 'YES',
  p_begin_on_new_field=> 'YES',
  p_colspan=> 2,
  p_rowspan=> 1,
  p_label_alignment=> 'RIGHT',
  p_field_alignment=> 'LEFT',
  p_field_template=> 1920401041247919+wwv_flow_api.g_id_offset,
  p_is_persistent=> 'Y',
  p_attribute_01 => 'N',
  p_attribute_02 => 'N',
  p_attribute_03 => 'N',
  p_item_comment => '');

end;
/
```

In this example you see wwv_flow_api.g_id_offset being added to the ID's at parameters p_item_plug_id and p_field_template.

Since version 4.0, APEX provides the package APEX_APPLICATION_INSTALL, which allows modifications to application attributes during installation. For any exported application, you can dictate in which workspace under which application ID an import should take place. You can circumvent using the same object ID's during import, by using the *Apex_Application_Install.Generate_Offset* procedure. This procedure sets the offset value to some arbitrary large value, which is returned by the wwv_flow_api.g_id_offset public global variable. This way, you ensure that the metadata for the application definition does not collide with other metadata on the instance.

The trouble with the Generate_Offset procedure is, that once a developer is done with his enhancements and starts exporting and splitting the application that was imported with Generate_Offset, almost the entire application will be seen as modified by version control, just because all the object ID's have changed. So Generate_Offset is not really suited for

our job. We need a way to transform the export files back to the original ID's. So, instead of using Generate_Offset, we use the procedure Set_Offset, and we store a specific unique offset number for each developer. For the offset value, we used 10,000,000,000 and 20,000,000,000 and so on. This way, collisions of ID's among developers won't occur.

In a comment on his blog [6], Joel Kallman discloses the SQL that shows the three parts by which an offset id is generated:

```
select to_number
       ( to_char(wwv_seq.nextval) ||
         lpad( substr( abs(wwv_flow_random.rand), 1, 5 ),5, '0' ) ||
         ltrim(to_char(mod(abs(hsecs),1000000),'000000'))
       )
  into g_curr_val
  from sys.v_$timer;
```

A new id will always have a new sequence value in the first part (wwv_seq.nextval) and the developers offsets differ not more than 99,999,999,999. So they only differ in the second part of the number, thereby making collisions for new ID's impossible.

When importing an application we issue a apex_application_install.set_offset call with the stored offset number. The piece of code we run under the parsing schema looks like this:

```
declare
  cn_schemaprefix constant varchar2(25) := '&SCHEMAPREFIX';
  cn_applicatie   constant varchar2(30) := '&APPLICATIE';
begin
  apex_application_install.set_workspace_id
  ( meta.mta_admin.apex_werkruimte_id(cn_schemaprefix)
  );
  apex_application_install.set_application_id
  ( meta.mta_admin.apex_applicatie_id(cn_applicatie,cn_schemaprefix)
  );
  apex_application_install.set_offset
  ( p_offset => meta.mta_admin.apex_id_offset(cn_schemaprefix)
  );
  apex_application_install.set_schema(upper(cn_schemaprefix) || '_UI');
  apex_application_install.set_application_alias
  ( case lower(cn_schemaprefix)
    when lower(cn_applicatie) then
      cn_applicatie
    else
      cn_applicatie || '_' || cn_schemaprefix
    end
  );
end;
/
@@install_apex_components
```

After all application attributes have been set, we run the *install_apex_components.sql*, which is the *install.sql* generated by APEXExportSplitter, that calls all splitted files in succession. In Figure 4 we saw this line:

```
sed s_@application_@$1/apex/application_\
 <install.sql\
 >../non-apex/install/install_apex_components.sql
```

which sets the relative directory structure of the called files according to our chosen structure.

When exporting an application, inside the apexupdate.sh script, between the export and the export split, we use the following extra command on the export file to subtract the stored offset from the id's

```
sed -E 's/([0-9]+)([ ]*\+[ ]*wwv_flow_api.g_id_offset)/^\1^\2/'
<f$2.sql | awk -F^ '{if(length($3)>0) {print $1 $2-ENVIRON["offset"] $3} else
{print $0}}' >f$2.sql
```

The sed command places the id between two tilde symbols (^) and the awk command subtracts the value in environment variable "offset" from the id. Lines that do not contain a wwv_flow_api.g_id_offset, will go through unchanged. Now, an application export file will only contain the real differences. This is important because now version control will exactly show you which parts of your APEX application have changed with each revision number.

When each developer has their own self-contained development environment, he can now do whatever he feels is necessary, including dropping and recreating his entire environment. No longer will this hinder your colleagues. And if you mess up your environment you should be able to easily reinstall everything, which leads us to the next step.

Chapter 3: One-Step Build

"On good teams, there's a single script you can run that does a full checkout from scratch, rebuilds every line of code, makes the EXEs, in all their various versions, languages, and #ifdef combinations, creates the installation package, and creates the final media -- CDROM layout, download website, whatever." - Joel Spolsky [1]

According to wikipedia, a software build is the process of converting source code files into standalone software artifacts. In a database environment, the standalone software artifact is not an executable, but deployed database schemas and a fully working APEX application behind a chosen URL. Note that by this definition you can also speak of build-and-deploy, but for sake of simplicity it is simply called a build here.

If a build takes more than one step, you are giving the developers a list to memorize for building the application. And by that, you have given them a chance to fail. It's not knowledge worth remembering that you should first run a backup, then clean up three database schemas, import the APEX application, load your images and restart the HTTP server. Or some other arbitrary sequence unique to your situation, but you get the idea. It should be one step that does it all.

So, this one-step build means that we should have a script in place which can get us from an empty database to a fully working APEX application. We call this script *install.sql*. And vice versa, to get to an empty database (for the application), we have a script *uninstall.sql*. Since the application consists of three parts, the APEX part, the supporting-database-objects-part and the files-on-the-application-server-part, we also developed intermediate scripts called *install_apex.sql*, *install_db.sql and install_files.sh* and their counterparts *uninstall_apex.sql*, *uninstall_db.sql* and *uninstall_files.sh*. This means that *install.sql* does nothing but call *install_db.sql*, *install_apex.sql* and *install_files.sh*, and likewise, *uninstall.sql* calls only *uninstall_apex.sql*, *uninstall_db.sql* and *uninstall_files.sh*.

The individual scripts explained
An *install_db.sql* is a handcrafted script in which you install each database object in the right order. Tools exist to generate such a script, but then you'd have to either prefix the names of the scripts to ensure the right order, or the tools rather randomly installs everything and finishes with compiling the entire schema. In the latter case, the install will look messy with intermediate compilation errors, which we do not want to occur when we give an application to a client. Below is a trimmed down version of one of our *install_db.sql* scripts to give you an idea:

```
whenever sqlerror exit failure
column current_schema new_value curschema
select sys_context('userenv','current_schema') current_schema
  from dual
/
define SCHEMAPREFIX='&1'
define APPLICATION='sca'
prompt ********************************************************************
prompt  Install db-part of &APPLICATION in schemas &SCHEMAPREFIX._data,
&SCHEMAPREFIX._api and &SCHEMAPREFIX._ui
prompt ********************************************************************

define tables_path='&APPLICATION./non-apex/data/tables/'
define sequences_path='&APPLICATION./non-apex/data/sequences/'
define indexes_path='&APPLICATION./non-apex/data/indexes/'
define data_path='&APPLICATION./non-apex/data/data/'
define privs_path='&APPLICATION./non-apex/data/privileges/'
```

```
define view_path='&APPLICATION./non-apex/ui/views/'

set verify off

alter session set current_schema = &SCHEMAPREFIX._data
/
@@&tables_path.SCA_OPNAMES.sql
@@&tables_path.SCA_METERSTANDEN.sql
@@&sequences_path.SCA_MSD_SEQ1.sql
@@&sequences_path.SCA_ONE_SEQ1.sql
@@&indexes_path.MSD_ONE_FK1_I.sql
@@&privs_path.privileges.sql &SCHEMAPREFIX

alter session set current_schema = &SCHEMAPREFIX._api
/
remark This application doesn't have an API layer yet.

alter session set current_schema = &SCHEMAPREFIX._ui
/
@@&view_path.sca_v_meterstanden.vw &SCHEMAPREFIX
@@&view_path.sca_v_opnames.vw &SCHEMAPREFIX

alter session set current_schema = &CURSCHEMA
/
set verify on
undefine SCHEMAPREFIX
undefine APPLICATION
undefine CURSCHEMA
whenever sqlerror continue
```

The SCHEMAPREFIX substitution variable allows this script to be executed in the application schema, as well as in the developer schemas.

The *uninstall_db.sql* script simply drops all database objects. It is the inverse of *install_db.sql*. One difference with the *install.sql* is also that the *uninstall.sql* is allowed to produce errors. In other words, it does not contain the "whenever sqlerror exit failure" statement. Errors may occur when a colleague has created database objects, which he added to the *uninstall_db.sql* script. When you bring your working copy up to date, you'll get the new uninstall_db.sql, which will try to uninstall the database object that doesn't exist in your schema yet. Or in general an error during uninstall may occur, when something unexpected happens during the install and you're left with some half installed application. In such a case, the uninstall should always be able to remove all that's left.

The *install_apex.sql* script has already been discussed in Chapter 2. The *uninstall_apex.sql* uses the four scripts that are generated during the APEXExportSplitter program and looks like this:

```
declare
  cn_schemaprefix constant varchar2(25) := '&SCHEMAPREFIX';
  cn_applicatie   constant varchar2(10) := '&APPLICATIE';
begin
  apex_application_install.set_workspace_id
  ( meta.mta_admin.apex_werkruimte_id(cn_schemaprefix)
  );
  apex_application_install.set_application_id
  ( meta.mta_admin.apex_applicatie_id(cn_applicatie,cn_schemaprefix)
  );
end;
/
@sca/apex/application/init.sql
@sca/apex/application/set_environment.sql
@sca/apex/application/delete_application.sql
@sca/apex/application/end_environment.sql
```

Finally, the *install_files.sh* copies all images, css files and javascript libraries to the docroot of the application server. We created separate directories next to the /i/ folder where APEX puts its own files. Each application gets its own directory and so does each developer. Within each developers directory, there is an application directory for the applications he's working on. Using an APEX substitution variable, which we called APP_IMAGE_PREFIX, we can reference the files from APEX. The *install_apex.sql* uses a little post import script to set the APP_IMAGE_PREFIX to the right value.

```
ssh -t -t $USER@$HOST <<END_SCRIPT
cd $DOCROOT_HOST
if [ ! "$SCHEMAPREFIX" == "$APPLICATIE" ]; then
  if [ ! -d "$SCHEMAPREFIX" ]; then
    mkdir $SCHEMAPREFIX
  fi
  cd $SCHEMAPREFIX
fi
if [ ! -d "$APPLICATIE" ]; then
  mkdir $APPLICATIE
fi
cd $APPLICATIE
if [ ! -d img ]; then
  mkdir img
fi
if [ ! -d css ]; then
  mkdir css
fi
if [ ! -d js ]; then
  mkdir js
fi
exit
END_SCRIPT

if [ "$SCHEMAPREFIX" == "$APPLICATIE" ]; then
  SUBDIR=$APPLICATIE
else
  SUBDIR=$SCHEMAPREFIX/$APPLICATIE
fi

scp -r $APPLICATIE/non-apex/files/img/* $USER@$HOST:$DOCROOT_HOST$SUBDIR/img/
scp -r $APPLICATIE/non-apex/files/css/* $USER@$HOST:$DOCROOT_HOST$SUBDIR/css/
scp -r $APPLICATIE/non-apex/files/js/* $USER@$HOST:$DOCROOT_HOST$SUBDIR/js/
```

The *uninstall_files.sh* simply does a "rm -r" on the application server directory. For these scripts to work, each developer has to have ssh connection to the application server in place, with private and public keys, so no passwords need to be typed in when issuing an ssh or scp command.

In production and user acceptance databases, you are not going to do full installs, except for the very first time. But when new developers enter the team and when someone messes up their environment, you want them to be able to install their own development environment from scratch. For this reason, you want to keep the full install and uninstall scripts right. And you do so, by constantly testing the full install from scratch, at least on the development environment and preferably also on test.

Chapter 4: Continuous Integration

"Communication is one of the key factors in software development and one of CI's most important features is that it facilitates human communication" - Martin Fowler [4]

The downside of every developer having its own APEX workspace and database schemas, is that multiple versions of the application will arise in each environment. These versions will have to be integrated at some point in time. And you do not want to postpone this integration for too long. The longer you wait, the harder integration will become. And a Big Scary Merge [4] is awaiting you.

Continuous integration addresses this issue. It is a software development practice where you avoid big integration phases at the end of a development effort, by doing numerous small merges of code. The issues will then be less complex, easier to fix and therefore less time consuming. Overall software quality will improve. The mantra here is to integrate and merge often, fail early and fix fast.

In our APEX environment, a typical workflow for a developer looks like this:

1. update working copy of the application with all changes of his colleagues, by issuing our *apexupdate.sh* script for the APEX objects and a regular *svn update* on the non-apex folder
2. make code changes
3. install the changes in the developers environment by running *apexupdate.sh* again if your code changes involves APEX, otherwise only *reinstall_db.sql* and/or *reinstall_files.sh* and test the changes
4. update working copy of the application with all the changes and test again if you have incorporated committed changes of one of your colleagues
5. *svn commit* the code changes

If you forget step 4 and a colleague has changed some files you were working on as well, Subversion will notify you during the commit that your version is out of date, and you are forced to do step 4 anyway.

Integrating the work of your colleagues in your working copy can be done as often as you wish, especially if it takes a little longer for you to make the code changes. A good practice is to start your day with a Subversion update.

Even though the application may have worked fine in your environment, this does not mean the application will work fine anywhere. For example, it's still possible that your database schema contains a package that you did not put under version control. So your version works fine, but if anyone integrates your changes, their version will break. You have broken the build. To detect such mistakes, we need an automated build which installs in the application schemas.

Implementing an automated build with Hudson

We have chosen for Hudson for the automated build. Hudson is an extensible continuous integration server. For each application we define a project in Hudson, which runs our reinstall.sql in the application schemas each night. If the automatic build fails, all developers receive an email. The break of the build should now be fixed before doing anything else. Hudson shows you the output of the reinstall.sql script, which will show you what went wrong. The figures below show how the console output looks like.

Figure 8a: Top of Hudson console output of the automated build

Figure 8b: End of Hudson console output of the automated build

Our developers can also refresh their environments together with the automated build. This only happens if they indicate that they want so. The developer can always install and uninstall his own environment, so this is just for convenience to keep the development environment up to date, should you want so. Automatically installing the latest version of the application in a development environment should be done with care, since it may just overwrite your work. A developer should only choose to automatically build his own environment each night if he has the discipline to always end the day committing his work. This way he can immediately begin working the next day. We have implemented the automatic building the development environments in our META schema. This schema contains three tables, one containing the developers, one containing the applications and an intersection table containing the application copies. A row in the application copies table means that a developer is working on that application. This table holds two indicators:

- an indicator whether the APEX application in his environment should be refreshed each night
- an indicator whether the database schemas in his environment should be refreshed each night

For any indicator that is set to 'Y', the build script also does a fresh install of that part (apex and/or non-apex) in the development environment.

Now that we have version control, self-contained development environments, a one-step-build and continuous integration fully setup, you are ready for the next steps: including unit tests for both the PL/SQL code in the database and the APEX application code and rolling out incremental changes to other environments.

References and further reading

[1] Joel Spolsky - The Joel Test: 12 Steps to Better Code
http://www.joelonsoftware.com/articles/fog0000000043.html

[2] Nick Ashley - Taking control of your database development
http://dbdeploy.com/documentation/taking-control-of-your-database-development-white-paper/

[3] Martin Fowler - Continuous Integration
http://martinfowler.com/articles/continuousIntegration.html

[4] Martin Fowler - FeatureBranch
http://martinfowler.com/bliki/FeatureBranch.html

[5] Scott W. Ambler - The Process of Database Refactoring: Strategies for Improving Database Quality
http://www.agiledata.org/essays/databaseRefactoring.html

[6] Joel R. Kallman - YABAOAE blogpost titled APEX_APPLICATION_INSTALL
http://joelkallman.blogspot.nl/2010/07/apexapplicationinstall.html

[7] Oracle Documentation - APEX_APPLICATION_INSTALL
http://docs.oracle.com/cd/E17556_01/doc/apirefs.40/e15519/apex_app_inst.htm

[8] Wikipedia - LESS (stylesheet language)
http://en.wikipedia.org/wiki/LESS_%28stylesheet_language%29

Acknowledgements

Thank you Marcel Hoefs, Erdal Aslan en Etienne Hamers for providing valuable feedback on this paper. Thanks to Michiel van Kessel for installing and configuring Red Hat Enterprise Linux, the Oracle databases, APEX, Glassfish and the APEX Listener, and Ronald Rood for patiently answering all my Linux and shell scripting questions. And last but not least thanks to the entire APEXSoFa team for exploring and testing all aspects of our APEXSoFa with me.

About the author

Rob van Wijk is an Oracle database developer who works with Oracle technology since 1995. He currently works as a principal consultant at Ciber Nederland. His main areas of expertise are the Oracle database, SQL, PL/SQL, performance and APEX. He writes about these subjects on his blog at http://rwijk.blogspot.com. He's a regular presenter at Oracle conferences, is an Oracle ACE and he teaches an Expert Seminar called "SQL Masterclass" for Oracle University.