

# 7

## Development and Testing Strategies for SugarCRM

b81825581a3cb156cdf227fb7517bbd  
ebruary

By now you must be confident about customizing SugarCRM, after all over the last six chapters we've looked at:

- Changing the look and feel of SugarCRM
- Adding simple modules
- Custom fields and logic hooks
- The architecture of the application and its supporting database

So, you'll be champing at the bit to move on and start building your own complex modules and processes, and that's what the remaining chapters of this book will help you do. However, before we jump in with both feet, it will be worth spending some time looking at development and testing strategies. By the end of Chapter 7 you will:

- Understand why development and testing strategies are important
- Understand how to set up development and testing servers
- Understand how to migrate code from development to testing to live

b81825581a3cb156cdf227fb7517bbd  
ebruary

And *then* you'll be ready to start building your own modules.

### Why Use Development and Testing Strategies?

Let's imagine a scenario — you've carried out all of your customizations, and everything is working beautifully. Pygoscelis, Korora, and all the other Penguin P.I. staff are really happy with your work, and they're now able to use SugarCRM to carry out their day-to-day tasks. In fact, they've even started to rely on the

b81825581a3cb156cdf227fb7517bbd  
ebruary

application completely. So, you decide to upgrade to the newest version of SugarCRM— and overnight all of your modifications disappear. Suddenly you become the most unpopular person in the organization.

Or let us think about a second scenario— you've carried out all your customizations, and everything is working beautifully. Pygoscelis is really happy with your work, and tells Korora, and all of the other Penguin P.I. staff, to start using it immediately. However, they turn round and say 'Pygoscelis really doesn't know how we do our work— this is all useless!'. Again, you become the most unpopular person in the organization.

And one final scenario— after the customizations have been completed, and everyone is using the application Korora comes to you and says that she just needs a minor modification carrying out. So, you carry out the changes, only to find that it's affected some of the other modules. And for the third time you become the most unpopular person in the organization.

You don't really want to be that unpopular, so we'll spend some time looking at how you can carry out SugarCRM customizations in a professional and safe manner— so that you will *always* be popular.

## The Unbreakable Rule: Thou Shalt Not Do Any Development on a Live Server

The best way to ensure the failure of your project is to start messing about with the live application that people are using. So, don't do it. This may seem common sense, but there is always a temptation just to do a *quick change*, but as Pippin says in *The Lord of the Rings* 'Short cuts make long delays'.

So, if you're not going to do any development to the live SugarCRM application, then where are you going to do it? The answer is to set up two servers:

- Server 1— The live server that people use in their normal daily activities
- Server 2— The development server

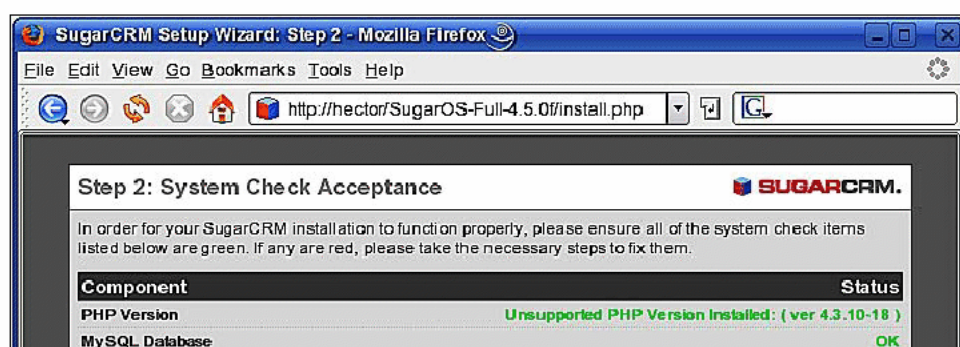
By doing this you can safely develop new modules and new functionality without affecting the users of your application. If something does go wrong then no one will be affected, and, in fact, no one but you need to know about it.

## Setting up a Development Server

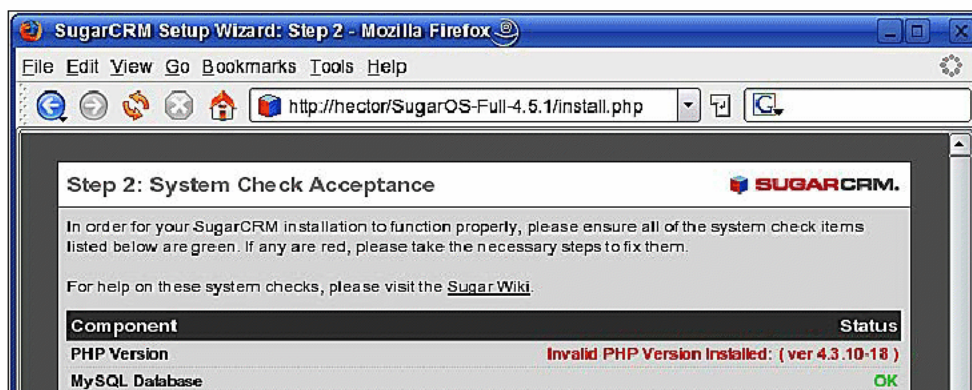
Having decided that we're *not* going to do any development directly on the live server, then obviously we need to set up a second server on which we can carry out the customizations. Now, at this stage you may be tempted to set up a second version of SugarCRM on your existing web server; after all it is possible to do that. However, there is a very good reason for not doing that.

Let's imagine that you've got an existing SugarCRM implementation based on SugarOS-Full-4.5.0f. Looking on the SugarCRM website you will find that the current version (at the time of writing) is SugarOS-Full-4.5.1, and so, you may decide that an upgrade is needed.

Now, remember from your initial installation that the second stage checks for SugarCRM dependencies. If you've currently got a working instance of SugarCRM then you would have seen something like:



However, if you download the newest version and try to install it on your existing web server then you may come across an immediate show stopper (have a look at the line **PHP version**):

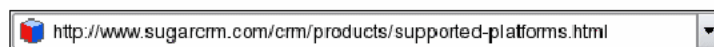


So, in this case the server has a PHP version suitable for SugarOS-Full-4.5.0f, but not SugarOS-Full-4.5.1. We could install a newer version of PHP, but then that goes against rule 1 – thou shalt not do any development on a live server. If you do upgrade PHP, then can you *really* guarantee that the existing (live) implementation of SugarCRM will continue working with the newer version of PHP?

The answer is, therefore, to set up a new, clean server – one that you can play with, and which won't affect your live users.

In the Penguin P.I. example the servers have had Debian Linux as the operating system, and by default it comes with PHP version 4.3.10. Therefore, the question is – which version of PHP *do* you need on the server? Fortunately, you can find this out directly from the SugarCRM website:

b81825581a3cb156cdf227fb7517bbd  
ebruary



And once there we can look at the PHP section:

PHP	4.3.11
	4.4.1 - 4.4.4
	5.0.1 - 5.0.5
	5.1.0, 5.1.2, 5.1.4

As you can see, the current version of SugarCRM does not support PHP 4.3.10. However, because we're going to be using two separate servers, we can upgrade one of them to another version of PHP and see what the effect is.

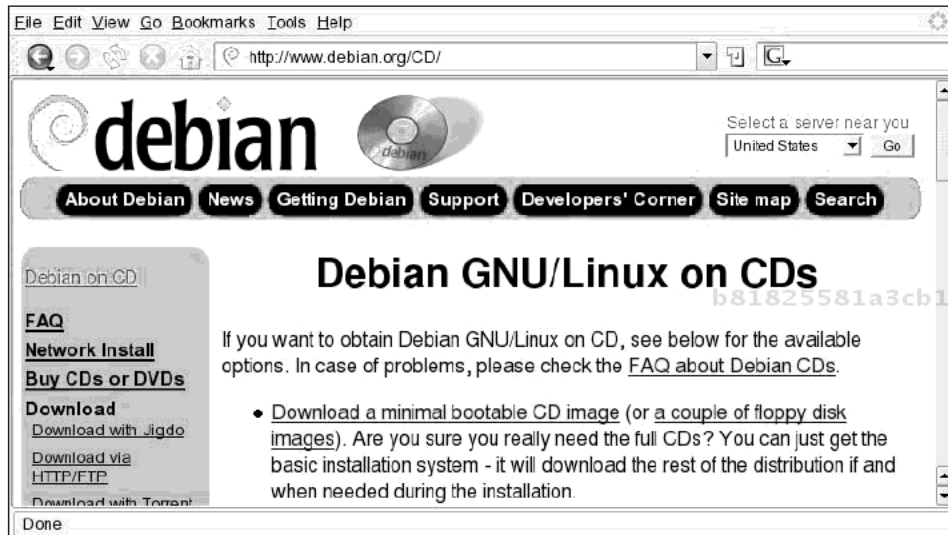
## Creating a Server

If you're already confident in setting up your own server then just move on to the next section – where we'll look at migrating files from one server to another. If not then let's see just how easy it is to create a server using Debian (Debian GNU/Linux if you want to be exact).

b81825581a3cb156cdf227fb7517bbd  
ebruary

The first thing that you need is a computer (obviously). Fortunately you won't need a new state of the art (and expensive) box – Debian, like many versions of Linux, will work on most machines – regardless of age, although a reasonable disk speed and plenty of memory won't do any harm. However, your computer *will* need a network card.

Next, you will need the installation disk, and this can be downloaded directly from Debian at <http://www.debian.org/CD/>:



Once you've created your installation disk then it's just a matter of connecting your machine to the network, inserting your disk and rebooting. After that, just follow the on-screen instructions. The process will:

- Format the machine for you
- Install the core Linux files
- Allow you to select one of the online application sources (choose one of the HTTP sources near to your location)
- Create a root user account (so that you can log on to do further work)

If you're wondering why the computer needs a network connection at the moment— it allows you to download most of the required files directly from the Internet, rather than having to install them from a number of disks.

When the process has finished you'll have a minimal setup— just enough to log on and start turning the base install into a working server. In order to do this you need to log on using your new root account, and then use the `apt-get` command to download all of the applications that you're going to need.

## Installing Software

Using `apt-get` is simple – all you need to know is the name of the application that you are going to install and then type:

```
apt-get install <package name>
```

For example:

```
apt-get install apache2
```

For your server you're going to need:

- SSH
- nfs-common
- nfs-kernel-server
- Sudo
- Apache2
- MySQL-Server
- PHP4
- php4-mysql
- libapache2-mod-php4
- Unzip

b81825581a3cb156cdff227fb7517bbd  
ebruary

These are just a tiny proportion of all of the Debian packages that are available, but they're all that you'll need in order to manage your SugarCRM server. You will have to modify Apache's config files so that it recognizes PHP, but apart from that all you need to do now is to set up the server's IP address.

## Setting the Server's IP Address

By default your new server would have been given a dynamic IP address – meaning that every time that there's a reboot then it will (potentially) be given a different IP address. And I'm sure that you will agree that this is not really of any use to the people trying to access SugarCRM. You'll be pleased to know that there is an easy remedy. All that you have to do is edit a file called `/etc/network/interfaces`. You need to find a line that says:

```
iface eth0 inet dhcp
```

Either comment out or delete the line and then add:

```
iface eth0 inet static
    address 192.168.1.3
    netmask 255.255.255.0
    gateway 192.168.1.1
```

b81825581a3cb156cdff227fb7517bbd  
ebruary

The address and gateway will, of course, depend on your network—address is the IP address that you want your machine to have, and gateway is your network's gateway to the Internet.

Finally, reboot the machine, and you've got a server ready for SugarCRM—in fact you'll already be able to access it from other computers on your network:



With the server in place, and accessible from anywhere on your network, we can now install SugarCRM. However, we're not going to do that from scratch—we're going to use the files from our existing SugarCRM setup.

## Migrating SugarCRM Files and Databases Between Servers

As you may well have worked out, we're working with two Debian Linux servers in the Penguin P.I scenario:

- Server 1—hector—which is to be used as the live server, and already has some minor customizations
- Server 2—acamas—the development server

Both servers have Apache, MySQL, and PHP installed – the only difference being that hector has a working version of SugarCRM. Our aim now is to:

- Set the servers up so that server 2 can see all of the files on server 1
- Copy all of the SugarCRM files from server 1 onto server 2
- Ensure that SugarCRM is running correctly on server 2

This can be achieved quite easily in Linux by setting up the appropriate exports and mount points.

## Setting Up the Export on Server 1

The plan is to export data from server 1 (hector) to server 2 (acamas), and the first step is add the IP address of acamas into hector's `/etc/hosts` file:

```
192.168.1.3  acamas
```

We do this so that we just have to refer to acamas in other files, rather than having to repeat the IP address all over the place. For example we next have to add an entry to hector's `/etc/exports` file:

```
/          acamas(ro)
```

The entry tell the network that acamas has authorized read access to hector's top directory.

Finally we need to export the information:

```
sudo /usr/sbin/exportfs -a
```

With that done you can access information on hector from acamas – once we've set up acamas, of course.

## Setting Up a Mount Point on Server 2

Turning our attention to acamas, we need to update its `/etc/hosts` file:

```
192.168.1.4          hector
```

Next we need to create a directory. This will be the mount point through which we will access all of the files on hector:

```
sudo mkdir /hector
```



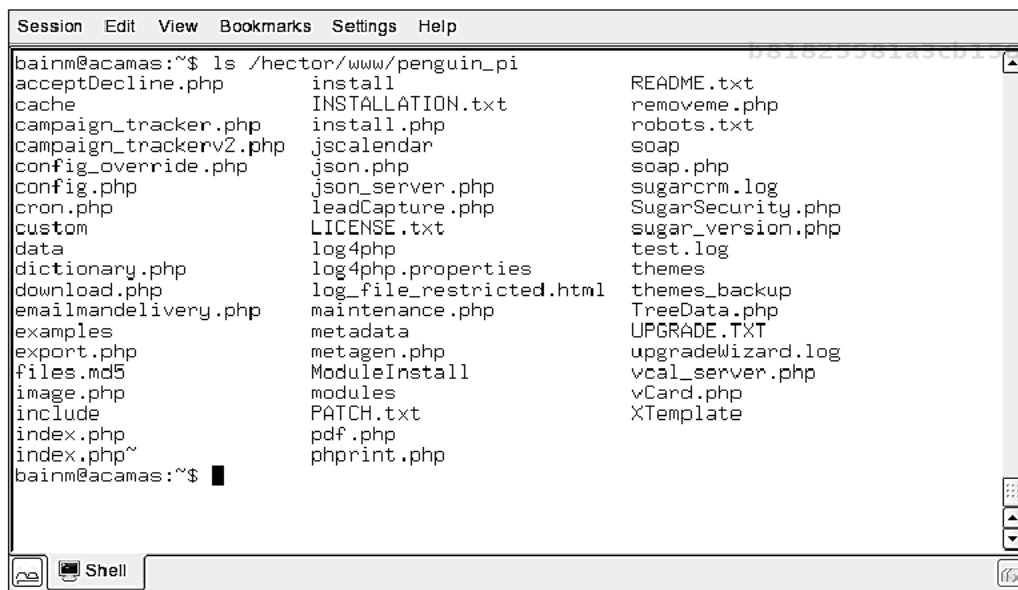
And then we need to tell acamas about this new mount point by updating its /etc/fstab file:

```
hector:/          /hector          nfs      ro 0 0
```

Finally we need to mount the mount point:

```
sudo mount /hector
```

Now we can access information on hector from acamas just as if it were in one of acamas' directories:



```
bainm@acamas:~$ ls /hector/www/penguin_pi
acceptDecline.php  install          README.txt
cache              INSTALLATION.txt removeme.php
campaign_tracker.php install.php      robots.txt
campaign_trackerV2.php jscalendar      soap
config_override.php json.php        soap.php
config.php         json_server.php sugarcrm.log
cron.php          leadCapture.php SugarSecurity.php
custom            LICENSE.txt     sugar_version.php
data              log4php         test.log
dictionary.php    log4php.properties themes
download.php      log_file_restricted.html themes_backup
emailmandelivery.php maintenance.php  TreeData.php
examples          metadata        UPGRADE.TXT
export.php        metagen.php     upgradeWizard.log
files.md5         ModuleInstall   vcal_server.php
image.php         modules         vCard.php
include          PATCH.txt       XTemplate
index.php        pdf.php
index.php~      phprint.php
```

## Migrating Files from Server 1 to Server 2

We've actually done the hardest part of the migration process, and that wasn't exactly difficult. All that's left to do is to transfer the files that we need from server 1 to server 2. So, to migrate the SugarCRM files we need to log onto server 2 (in this case acamas), and type:

```
sudo cp --preserve -r /hector/www/penguin_pi /www
```

Remembering, of course, to use your own directory location. Then we need to transfer the MySQL files:

```
sudo cp --preserve -r /hector/var/lib/mysql/penguinpi /var/lib/mysql
```

You'll need to restart MySQL:

```
sudo /etc/init.d/mysql restart
```

But once you've done that you'll be able to log on to the database and view the SugarCRM tables and their contents. However, you won't be able to access them from your web browser yet. Before you're able to do that you'll need to create a user on the database – the SugarCRM user account that's created during the normal installation process. Chances are you won't be able to remember the details that you originally entered. If you can't then don't worry – they are stored in the SugarCRM config file `/var/www/penguin_pi/config.php`. If you look through the file you'll find something like:

```
array
(
    'db_host_name' => 'localhost',
    'db_host_instance' => '',
    'db_user_name' => 'penguinpi_user',
    'db_password' => 'penguinpi_go',
    'db_name' => 'penguinpi',
    'db_type' => 'mysql',
),
```

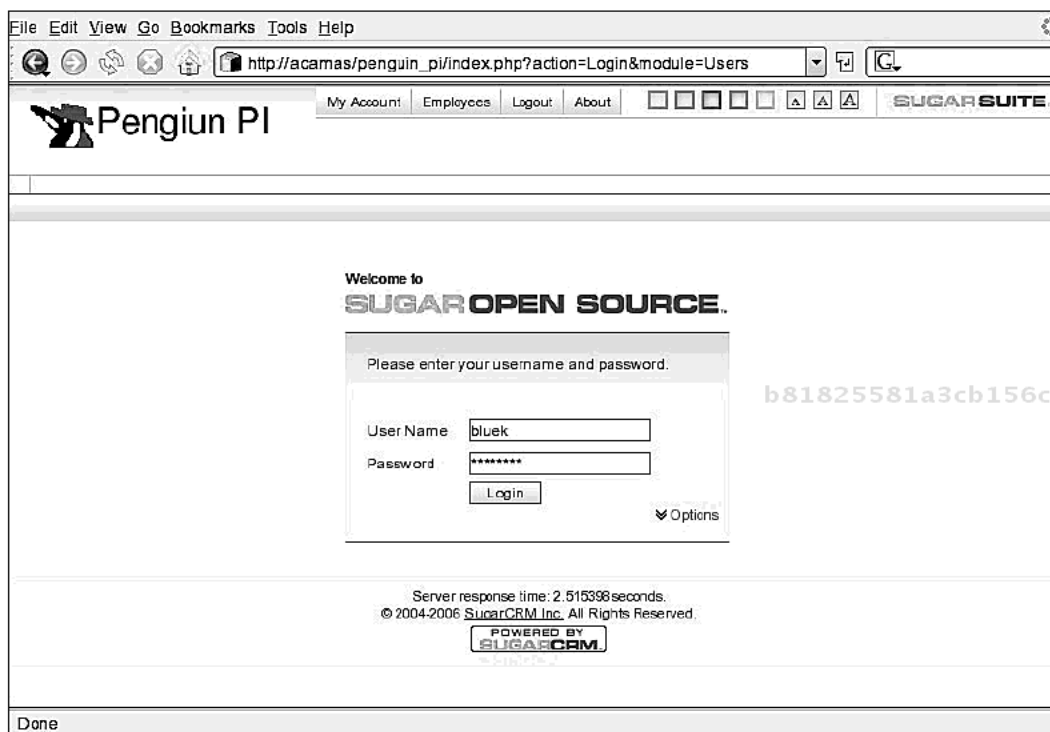
b81825581a3cb156cdff227fb7517bbd  
ebrary

You can now log onto the database and create the SugarCRM account from these details:

```
GRANT SELECT,UPDATE,INSERT,DELETE
ON penguinpi.* TO 'penguinpi_user'@'localhost'
IDENTIFIED BY 'penguinpi_go';
GRANT SELECT,UPDATE,INSERT,DELETE
ON penguinpi.* TO 'penguinpi_user'@'acamas'
IDENTIFIED BY 'penguinpi_go';
flush privileges;
```

b81825581a3cb156cdff227fb7517bbd  
ebrary

And so, with all the files in place, and having given SugarCRM access to the database, you can now use a web browser to view your new SugarCRM implementation (which, at the moment, will be identical to your old one):



You can now customize and upgrade to your heart's content, knowing that any changes you make will not affect the live users at all – well, not until you're ready that is.

## An Example Upgrade

You'll remember that earlier in the chapter we saw that we couldn't upgrade to SugarOS-Full-4.5.0h because:

- SugarOS-Full-4.5.0h needs a newer version of PHP than Debian supplies.
- We can't be sure of the effects that upgrading PHP will have on SugarOS-Full-4.5.0f.
- We don't want to do anything on the live server that might affect our users. In fact, we don't even want to do anything on the live server that *shouldn't* affect our users.

However, now that we've got a development server we can safely carry out the upgrade and see what effect it does have.

## Upgrading PHP

Obviously, you need to check how to carry out the upgrade for your own operating system, but on Debian it's just a matter of carrying out two steps:

- Select an appropriate download source
- Install the software

So, the first step is to find a download source. One such source is [dotdeb.org](http://packages.dotdeb.org)—this is a repository for many current applications, and so it's just a matter of updating `/etc/apt/sources.list` with the source details:

```
deb http://packages.dotdeb.org stable all
deb-src http://packages.dotdeb.org stable all
```

b81825581a3cb156cdf227fb7517bbd  
ebruary

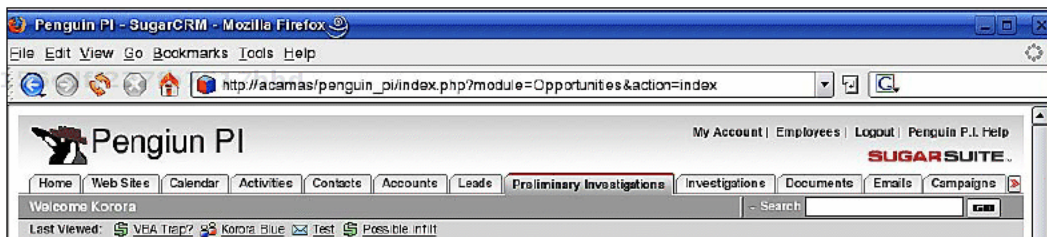
Next install the new version of PHP:

```
sudo apt-get update
sudo apt-get install php4
sudo apt-get install php4-mysql
```

And then you can check what version is now installed:

```
php4 -v
PHP 4.4.4-0.dotdeb.3 with Suhosin-Patch 0.9.6 (cli) (built: Nov 16
2006 11:21:12)
Copyright (c) 1997-2004 The PHP Group
Zend Engine v1.3.0, Copyright (c) 1998-2004 Zend Technologies
```

With the correct version of PHP installed, you can login to ensure that everything is working correctly:

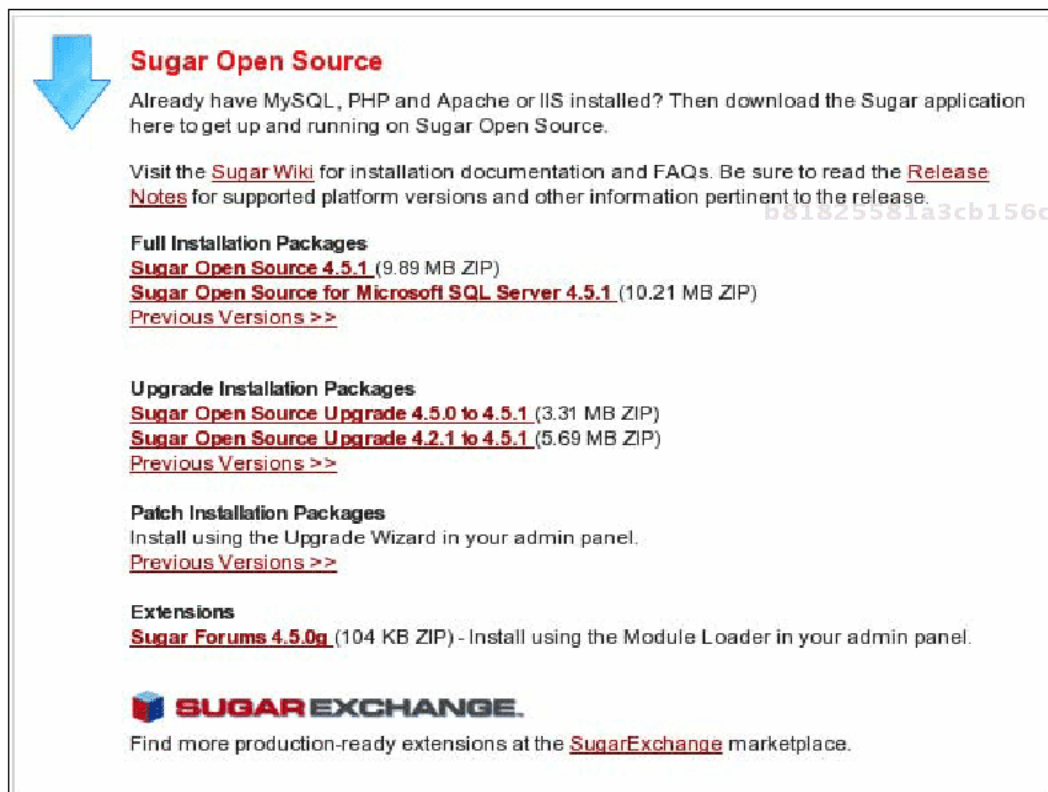


b81825581a3cb156cdf227fb7517bbd  
ebruary

In theory everything should work perfectly, but if it doesn't then at least it won't affect your users—and you'll be able to correct any problems in isolation. And now you can think about upgrading to a newer version of SugarCRM.

## Upgrading SugarCRM

As we've now got a suitable version of PHP on our development server we can think about looking at a more current version of SugarCRM. If you take a look at the SugarCRM website then you can see that you have a few options:



The screenshot shows the 'Sugar Open Source' section of the SugarCRM website. It features a large blue downward-pointing arrow on the left. The text includes instructions for downloading the application if MySQL, PHP, and Apache or IIS are installed. It provides links to the Sugar Wiki for documentation and FAQs, and the Release Notes for supported platform versions. Under 'Full Installation Packages', it lists 'Sugar Open Source 4.5.1' (9.89 MB ZIP) and 'Sugar Open Source for Microsoft SQL Server 4.5.1' (10.21 MB ZIP), with a link to 'Previous Versions >>'. Under 'Upgrade Installation Packages', it lists 'Sugar Open Source Upgrade 4.5.0 to 4.5.1' (3.31 MB ZIP) and 'Sugar Open Source Upgrade 4.2.1 to 4.5.1' (5.69 MB ZIP), also with a link to 'Previous Versions >>'. Under 'Patch Installation Packages', it instructs to use the Upgrade Wizard in the admin panel and provides a link to 'Previous Versions >>'. Under 'Extensions', it lists 'Sugar Forums 4.5.0g' (104 KB ZIP) and instructs to install using the Module Loader in the admin panel. At the bottom, there is a 'SUGAR EXCHANGE' logo and a link to find more production-ready extensions at the SugarExchange marketplace.

**Sugar Open Source**

Already have MySQL, PHP and Apache or IIS installed? Then download the Sugar application here to get up and running on Sugar Open Source.

Visit the [Sugar Wiki](#) for installation documentation and FAQs. Be sure to read the [Release Notes](#) for supported platform versions and other information pertinent to the release.

**Full Installation Packages**

[Sugar Open Source 4.5.1](#) (9.89 MB ZIP)

[Sugar Open Source for Microsoft SQL Server 4.5.1](#) (10.21 MB ZIP)

[Previous Versions >>](#)

**Upgrade Installation Packages**

[Sugar Open Source Upgrade 4.5.0 to 4.5.1](#) (3.31 MB ZIP)

[Sugar Open Source Upgrade 4.2.1 to 4.5.1](#) (5.69 MB ZIP)

[Previous Versions >>](#)

**Patch Installation Packages**

Install using the Upgrade Wizard in your admin panel.

[Previous Versions >>](#)

**Extensions**

[Sugar Forums 4.5.0g](#) (104 KB ZIP) - Install using the Module Loader in your admin panel.

**SUGAR EXCHANGE**

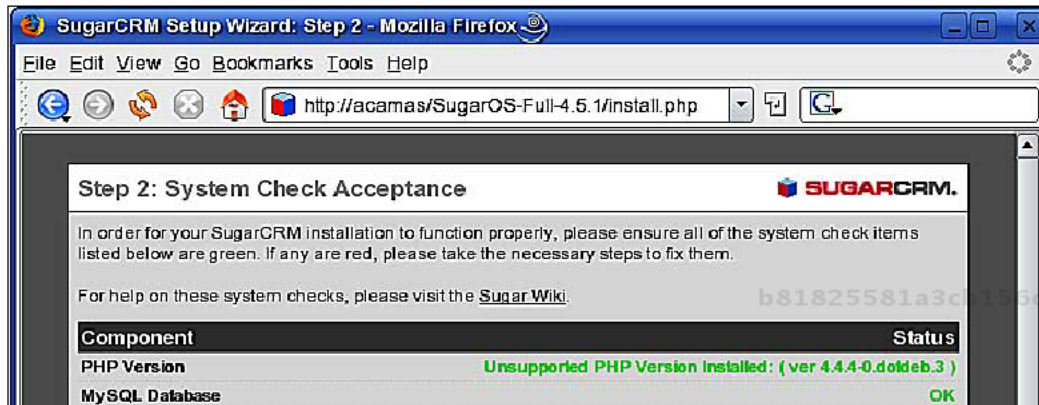
Find more production-ready extensions at the [SugarExchange](#) marketplace.

If you're certain that the upgrade will have no effect on your current implementation then you can download the files that will enable you to upgrade from 4.5.0 to 4.5.1. However, since we have already carried out some customizations then it is much safer to create a new installation, and then carry out a comparison of the two.

So, to get the new version of SugarCRM either download via the browser, or use widget, for example:

```
wget http://www.sugarforge.org/frs/download.php/2535/SugarOS-4.5.1.zip
```

Once you've unzipped the SugarCRM files then you'll be able to continue the installation process by opening up a web browser starting the installscript:



As you go through the process make sure that you use a new name for your database – we don't want to overwrite the existing one:

**Step 3: Database Configuration**

Please enter your database configuration information below. If you are unsure of what to fill in, we suggest that you use the default values.

\* Required field

**Database Configuration**

\* Host Name / Host Instance: local host

\* Database Name: sugarcrm\_new ☒ Create Database

\* Database Username: sugarcrm\_new ☒ Create User

Database Password: \*\*\*\*\*

Re-enter Database Password: \*\*\*\*\*

Drop and Recreate Existing Sugar tables? ☐  
Caution: All Sugar data will be erased if this box is checked.

Populate Database with Demo Data? ☐

Database Account Above Is a Privileged User? ☐

\* Privileged Database User Name: root  
This privileged database user must have the proper permissions to create a database, drop/create tables, and create a user. This privileged database user will only be used to perform these tasks as needed during the installation process. You may also use the same database user as above if that user has sufficient privileges.

Privileged Database User Password: \*\*\*\*\*

Help Back Next

Once you've followed all of the instructions, and completed the process then you're ready to start comparing installations.

## Comparing Database Files

Before we look at the PHP application side of SugarCRM we'll look at the database. The question that we need to ask first is 'Are the tables in 4.5.0 the same as in 4.5.1?'. We can answer this with a quick bit of Linux scripting:

```
#Define the databases to used
DATABASES[0]="penguinpi"
DATABASES[1]="sugarcrm_new"

#Loop through the databases
for DATABASE in ${DATABASES[*]}
do
    #Count the tables
    TABLES=$(echo "show tables" |
mysql -s -uroot -ppassword $DATABASE |
wc -l )

    #Output the result
    echo $DATABASE $TABLES
done
```

b81825581a3cb156cdff227fb7517bbd  
ebruary

To which you'll get the output:

```
penguinpi 93
sugarcrm_new 92
```

So, at first glance it would appear that there are different tables involved—the new version loses a table. This means that we need to know the differences between the lists of tables. Again, a little Linux scripting will tell us:

```
#Define the databases to used
DATABASES[0]="penguinpi"
DATABASES[1]="sugarcrm_new"

#Loop through the databases
for DATABASE in ${DATABASES[*]}
do
    #output the stucture of the database to files
    echo "show tables" |
    mysql -s -uroot -ppassword $DATABASE > $DATABASE
done

#compare the contents of the files
diff ${DATABASES[*]}
```

b81825581a3cb156cdff227fb7517bbd  
ebruary

This time the output is:

```
< opportunities_cstm
```

If you remember, this table was created automatically by SugarCRM when we introduced our own custom fields – meaning that the default list of tables is the same for both 4.5.0 and 4.5.1. Of course, that doesn't mean that the table structures are the same. So, let's look at that next:

```
#Define the databases
DATABASES[0]="penguinpi"
DATABASES[1]="sugarcrm_new"
#Loop through the databases
for DATABASE in ${DATABASES[*]}
do
    #Obtain the list of files in table in each database
    TABLES="$(cat $DATABASE)"
    for TABLE in $TABLES
    do
        echo desc $TABLE |
        mysql -s -uroot -ppassword $DATABASE |
        awk '{print $1}'> $DATABASE.$TABLE
    done
done

Get the list of table files for one database
TABLES="$(cat ${DATABASES[1]})"

#Compare the field list for each table
for TABLE in $TABLES
do
    DIFF="$(diff ${DATABASES[0]}.$TABLE ${DATABASES[1]}.$TABLE | wc -l)"
    if [ $DIFF -gt 0 ]
    then
        diff ${DATABASES[0]}.$TABLE ${DATABASES[1]}.$TABLE |
        grep ">" | awk '{print $2}' > $TABLE.new
    fi
done

#Output the results
FILES=*.new
for FILE in $FILES
do
    basename $FILE .new
    echo "_____ "
    cat $FILE
    echo
done
```

b81825581a3cb156cdff227fb7517bbd  
ebruaryb81825581a3cb156cdff227fb7517bbd  
ebruaryb81825581a3cb156cdff227fb7517bbd  
ebruary



The output of this shows us the additional fields that are required in 4.5.1 (and you may remember these from Chapter 6):

Table	Additional Fields Required
accounts	campaign_id
campaign_log	marketing_id
campaigns	impressions
	frequency
contacts	campaign_id
email_templates	text_only
emails_contacts	campaign_data
emails_leads	campaign_data
emails_prospects	campaign_data
emails_users	campaign_data
notes	embed_flag
opportunities	campaign_id
prospects	campaign_id
upgrade_history	name
	description
	id_name
	manifest

In fact, if you log onto the database you will find that there are also some fundamental changes to the tables themselves:

```
mysql -uroot -ppassword mysql
mysql> desc penguinpi.cases;
```

Field	Type	Null	Key	Default	Extra
id	varchar(36)	NO	PRI		
case_number	int(11)	NO	MUL	NULL	auto_increment
date_entered	datetime	NO		0000-00-00 00:00:00	
date_modified	datetime	NO		0000-00-00 00:00:00	
modified_user_id	varchar(36)	NO			
assigned_user_id	varchar(36)	YES		NULL	
created_by	varchar(36)	YES		NULL	
deleted	tinyint(1)	NO		0	
name	varchar(255)	YES	MUL	NULL	
account_id	varchar(36)	YES		NULL	
status	varchar(25)	YES		NULL	
priority	varchar(25)	YES		NULL	
description	text	YES		NULL	
resolution	text	YES		NULL	

*Development and Testing Strategies for SugarCRM*

```
14 rows in set (0.01 sec)
```

```
mysql> desc sugarcrm_new.cases;
```

Field	Type	Null	Key	Default	Extra
id	char(36)	NO	PRI		
case_number	int(11)	NO	MUL	NULL	auto_increment
date_entered	datetime	NO			
date_modified	datetime	NO			
modified_user_id	char(36)	NO			
assigned_user_id	char(36)	YES		NULL	
created_by	char(36)	YES		NULL	
deleted	tinyint(1)	NO		0	
name	varchar(255)	YES	MUL	NULL	
account_id	char(36)	YES		NULL	
status	varchar(25)	YES		NULL	
priority	varchar(25)	YES		NULL	
description	text	YES		NULL	
resolution	text	YES		NULL	

```
14 rows in set (0.01 sec)
```

```
mysql>
```

You'll notice that the ID, modified\_user\_id, assigned\_user\_id, and created\_by fields have been changed from varchar to char, and the default values for the date\_entered and date\_modified fields have been removed. This means, therefore, that we can't just add the missing fields to our old tables – we must load our existing data into the new tables.

## Migrating Database Files

If all of the tables contained the same fields then the migration from one database to another would be very simple, for example for cases the SQL would be:

```
insert into sugarcrm_new.cases select * from penguinpi.cases;
```

However, as we've already seen, some of the new database tables contains additional fields, and you mustn't forget the tables and fields that you may have added to the old database during your customizations. So, let's look at the custom tables first.

Earlier in the chapter we identified the one table that we were using, and that is missing from the new one – that's opportunities\_cstm. You'll remember that this was created in Chapter 2 when we introduced custom fields into the SugarCRM application. Now, if you manually created the table then you can just apply the SQL to the new database. If not (i.e. SugarCRM created the table for you), then you won't have the SQL. Obviously you could write your own SQL to do the job, but there is an easier way – just copy the MySQL database files from the old database to the new one:

```
sudo cp /var/lib/mysql/penguinpi/opportunities_cstm.* /var/lib/  
mysql/sugarcrm_new
```

That's fine for one table, but what if you have introduced a number of tables? If that's the case then the same Linux scripting that told us `opportunities_cstm` was missing can also migrate any other missing tables:

```
#Define the databases  
DATABASES[0]="penguinpi"  
DATABASES[1]="sugarcrm_new"  
  
#Loop through the databases  
for DATABASE in ${DATABASES[*]}  
do  
    #Write the data structure to a file  
    echo "show tables" |  
    mysql -s -uroot -ppassword $DATABASE > $DATABASE  
done  
  
#Obtain a list of missing tables  
diff ${DATABASES[*]} |  
awk '{print $2}' |  
grep -v ^$ |  
while read TABLE  
do  
    #Copy the missing tables from one database to the other  
    sudo cp /var/lib/mysql/penguinpi/${TABLE}.* /var/lib/mysql/sugarcrm_  
new/  
done
```

b81825581a3cb156cdff227fb7517bbd  
ebrary

I'm sure that you will agree that table migration is very straightforward; however, if you've created custom fields on your tables then things could be a little more involved – and this time you're going to have to do some manual checking, unless, of course you've been careful in the way that you've added the fields.

b81825581a3cb156cdff227fb7517bbd  
ebrary

Let's say, for example, that you added a field called `campaign_id` to the accounts table. If that's the case then you've got a problem, because that's exactly what the new version of SugarCRM has done. However, if you name your fields sensibly then you can avoid this problem – for example if you'd named your field `ppi_campaign_id` (for example) then you can minimize the chances of this type of conflict happening.

Assuming that you've not been tripped up by any field naming problems then the next step is to add your fields to the new database. Now, the process of migrating fields should actually start with the initial creation of the custom field. When you create the custom field don't just log onto the database and type the SQL directly. Instead write your SQL into a file, and then apply the file to the database. Now all

b81825581a3cb156cdff227fb7517bbd  
ebrary

you have to do is apply this file to your new database. For example, for our correctly named `ppi_campaign_id` field we would store a file called `ppi_campaign_id.sql` which would contain:

```
alter table accouts add ppi_campaign_id char(36);
```

Now we just need to do:

```
mysql -uroot -ppassword sugarcrm_new < ppi_campaign_id.sql
```

Or if this is stored in a directory with any other custom field files then we could move to that directory and type:

```
FILES=*.sql
for F in $FILES
do
    mysql -uroot -ppassword sugarcrm_new < $F
done
```

b81825581a3cb156cdff227fb7517bbd  
ebrary

Once we've applied the structure to the new database, we need to think about transferring the existing data. However, as we've already identified, some of the tables contain different numbers of fields, and in these cases we have to say exactly which fields are to be used. For example, to transfer the data in accounts the SQL would be:

```
insert into sugarcrm_new.accounts
(id, date_entered, date_modified, modified_user_id, assigned_user_id,
created_by, name, parent_id, account_type, industry, annual_revenue,
phone_fax, billing_address_street, billing_address_city, billing_
address_state, billing_address_postalcode, billing_address_country,
description, rating, phone_office, phone_alternate, email1, email2,
website, ownership, employees, sic_code, ticker_symbol, shipping_
address_street, shipping_address_city, shipping_address_state,
shipping_address_postalcode, shipping_address_country, deleted)
select
id, date_entered, date_modified, modified_user_id, assigned_user_id,
created_by, name, parent_id, account_type, industry, annual_revenue,
phone_fax, billing_address_street, billing_address_city, billing_
address_state, billing_address_postalcode, billing_address_country,
description, rating, phone_office, phone_alternate, email1,
email2, website, ownership, employees, sic_code, ticker_symbol,
shipping_address_street, shipping_address_city, shipping_address_
state, shipping_address_postalcode, shipping_address_country, deleted
from penguinpi.accounts ;
```

b81825581a3cb156cdff227fb7517bbd  
ebrary

To do this by hand would be very time consuming, but again we can turn to a Linux script to do the jobs for us. First we'll create a set of SQL files that will load the data:

```
#Define the databases
DATABASES[0]="penguinpi"
DATABASES[1]="sugarcrm_new"

#Obtain the list of tables
TABLES="$(cat ${DATABASES[1]})"
#Loop through the tables
for TABLE in $TABLES
do
    #Define the SQL to empty the table
    SQL="delete from ${DATABASES[1]}.${TABLE};"

    #Define the SQL to load the new data
    SQL="$SQL insert into ${DATABASES[1]}.${TABLE}"
    FIELDS="$(cat ${DATABASES[0]}.${TABLE})"
    FC=$(cat ${DATABASES[0]}.${TABLE}|wc -l)
    FIELD_LIST=""
    FN=0
    for FIELD in $FIELDS
    do
        let FN=FN+1
        if [ $FN -lt $FC ]
        then
            FIELD="${FIELD},"
        fi
        FIELD_LIST="${FIELD_LIST}${FIELD}"
    done
    SQL="$SQL ($FIELD_LIST)"
    SQL="$SQL select"
    SQL="$SQL $FIELD_LIST"
    SQL="$SQL from ${DATABASES[0]}.${TABLE};"

    #Output the complete SQL to a file
    echo $SQL > ${TABLE}.sql
done
```

b81825581a3cb156cdff227fb7517bbd  
ebruaryb81825581a3cb156cdff227fb7517bbd  
ebruary

Next we'll need another piece of script to run each of the SQL files:

```
#Obtain the list of SQL files
SQLS=*.sql

#Loop through the SQL files
```

b81825581a3cb156cdff227fb7517bbd  
ebruary

```
for SQL in $SQLS
do
    #Display the table name
    basename $SQL .sql
    #Run the SQL
    mysql -uroot -ppassword mysql < $SQL
done
```

You may be wondering why we're using individual SQL files. That's just in case there is any problem with an individual table – the remainder of the data load will still continue, leaving you to look at the table's SQL file, and work out what the problem is.

One thing that you will have to do before you continue is to run a little more SQL on the new database:

```
update config set value='4.5.1' where name = 'sugar_version';
```

If you don't do this then SugarCRM will refuse to log on because it will believe that you're trying to use a 4.5.0 database.

At the end of the process you will have all of your data loaded into your brand new SugarCRM database, and it's time to turn your attention to the application files.

## Comparing and Migrating the SugarCRM Application Files

Having set up the new SugarCRM database we need to consider the PHP files that make up the application itself. We, of course, can't just copy our modified files over the top of the new ones – we have no idea what kind of affect this will have since we don't know where new functionality has been added to SugarCRM 4.5.1. The first step, therefore, is to find any changes that affect our customizations.

In order to find files that have changed, and what those changes are, we can turn back to a Linux command that we've already been using – `diff`. If you do want to find *every* file that has been changed then go to your web server's document root and type:

```
diff -q -r penguin_pi SugarOS-Full-4.5.1
```

After you've seen data scrolling up the screen for a minute or two then you'll realize that there are an awful lot of files that have been changed. In fact if you type:

```
diff -q -r penguin_pi SugarOS-Full-4.5.1 | wc -l
```

you'll find that there are 1251 differences between the two versions. Obviously we want to narrow it down a little bit. And this is where it becomes essential that you keep track of your customizations as you carry them out—if you do that then this next bit becomes really easy.

In our case, we haven't customized that much yet, so it's not a major problem, the only module that we've worked with is Opportunities—if you remember in Chapter 3 we added some custom fields, and one of the files that we edited was `EditView.html`. So let's compare that to the version in `SugarOS-Full-4.5.1` by making use of the `Linux diff` command:

```
diff penguin_pi/modules/Opportunities/EditView.html \
SugarOS-Full-4.5.1/modules/Opportunities/EditView.html
```

The output tells us that the only differences between the files are the ones that we made:

```
87,88c96,97
<      <td class="dataLabel"><span sugar='slot11'>{MOD.Surveillance_
Required_c_10}</span sugar='slot'></td>
<      <td class="dataField"><span sugar='slot11b'><select
title='{SURVEILLANCE_REQUIRED_C_HELP}' name="surveillance_required_
c">{OPTIONS_SURVEILLANCE_REQUIRED_C}</select></span sugar='slot'></td>
```

You can see that there are differences between line 87 of the first file and 88 of the second, as well as 96 in the first and 97 of the second. So then it's just a matter of migrating our modified file:

```
sudo cp penguin_pi/modules/Opportunities/EditView.html \
SugarOS-Full-4.5.1/modules/Opportunities/
```

Of course we mustn't forget to transfer our custom files (including the logic hooks):

```
sudo cp -r penguin_pi/custom SugarOS-Full-4.5.1
```

And, our nice new theme:

```
sudo cp -r penguin_pi/themes/PenguinPI SugarOS-Full-4.5.1/themes/
```

As well as the module that we created:

```
sudo cp -r penguin_pi/modules/TestApp SugarOS-Full-4.5.1/modules
```

Finally we need to check and then migrate any supporting files that we've had to change:

```
diff penguin_pi/include/modules.php \
SugarOS-Full-4.5.1/include/modules.php
< $moduleList[] = 'TestApp';
```

```
< $beanList['NewTab'] = 'TestApp';
< $beanFiles['NewTab'] = 'modules/TestApp/TestApp.php';
```

In this case we can copy our modified `modules.php` over the top of the new one; however, that's not so in the next situation:

```
diff penguin_pi/modules/Emails/vardefs.php \
SugarOS-Full-4.5.1/modules/Emails/vardefs.php
<
>         'massupdate'=>true,
>         'massupdate'=>false,
<         'type' => 'text',
>         'type' => 'longtext',
<         'type' => 'text',
>         'type' => 'longtext',
```

b81825581a3cb156cdff227fb7517bbd  
ebrary

Here, we can see that there are more changes than the ones that we made, and so you'll need to edit the new file rather than just copying the modified one.

So, as you can see the process consists of:

1. Compare all of the customized files with the new ones.
2. If suitable copy the customized files over the new ones, if not then make the changes directly to the new files.

Once you've done all that then you'll have two identical versions of SugarCRM, except that one is version 4.5.0 and the other is 4.5.1. So, next we need to think about testing our application.

## Testing SugarCRM

You're now in a rather nice situation:

- Your users can work in the safe knowledge that their using a stable system that's not subject to random changes.
- You can work on customizations to the system knowing that you won't affect your users.

However, at some point you're going to want to release all of your changes to your users. At this time you'll need to think about testing. By this I don't mean the testing that you should be doing anyway; by testing I mean someone sitting down and replicating the normal day to day task that the users carry out. At this point there are two questions that you need to ask:

- Who is going to do the testing?
- Where is the testing to be carried out?



The answer to the first question should *not* be 'Myself' or 'one of the developers'. Why? Think about emails – when do you notice spelling mistakes? Invariably it's once you've pressed the send button. Bugs are just the same – you'll only find them after the application's been released. Plus, any user will only be confident if the software is tested by someone who understands their job – and that's not you, it's one of them. So, which one of *them* should do the testing? Let's look at the Penguin P.I. organization for some inspiration:

- Sphen – being the managing director he's the last person to ask. Not because he's too important – it's because he only *thinks* he knows how everything works.
- Robby Eudypates – the newbie – he's not doing anything essential at the moment, as so seems the obvious choice. However, he's also the most inexperienced person in the organization.
- Korora – the most experienced (and busiest) person around – now, that's who you want on board. If you can get Sphen to get Korora to pass some of her work onto Robby, then you've got someone who completely understands the daily workings, and will pick up any problems very quickly.

Next, having a willing volunteer, you're going to have to give them something to test. Obviously you could let them loose on the development server, but by definition that's an unstable environment – plus you may have some elements that you're not ready to release yet. The solution is to set up another server – a test server.

So, it's back to the start of the chapter for you and just follow the instructions for setting up a server. However, this time you won't be migrating from the live environment, you'll be migrating from the development environment. And, of course, in the scenario that we've been looking at then you need two migration and testing periods:

- Migrate the original (4.5.0) set up to the test server with the newer version of PHP. Once that's been tested then you'll be able to upgrade the version of PHP on the live server.
- Migrate your customized version of SugarCRM 4.5.1 to the test server.

Of course, there's one other thing that you may want to consider at this point – documentation. It's all very well Korora telling you that everything is OK, but imagine the embarrassment when you release the software and then a bug is found, and then Korora turns round and says 'Well, I didn't test *that* bit'. So, make sure either:

1. You document the areas to be tested, and Korora signs to say that they've been tested.
2. Korora records her testing.

Obviously, the best solution is a combination of the two.

So, assuming that there are no issues, then you're ready to pass the new version on to all of the other users.

## Releasing Your Customizations

You've now done all of the hard work, and you're on the final step of the process—releasing the application. All you have to do now is:

- Carry out any required upgrades to the live sever (such as PHP)
- Transfer the new SugarCRM directory to the live server
- Transfer the new SugarCRM database to the live server
- Migrate the live data into the new database
- Make the new SugarCRM application the default one

We've already covered the first three of these activities in detail, and so we'll just concentrate on the fourth and fifth ones—migrating the live data into the new database, and making the new SugarCRM application the default one.

In fact, we've already seen how to migrate the database data—you'll remember that we created a set of SQL files to copy the data from our live snapshot into the development database—we can use those same SQL files to transfer the data from the live database into the new database. Obviously you'll need to transfer the migration files to the live server, and you'll need to run the files at a time when there aren't any changes being made i.e. there are no users using the application.

And, of course, don't forget to set the value of `sugar_version` to '4.5.1' in the config table. The SQL (in case you can't remember) is:

```
update config set value='4.5.1' where name = 'sugar_version';
```

Next you can make the new application the default one—after all the users will want this change over to be as seamless as possible. If you looking in your web server's document root you should see two directories, and in our example these would be:

1. penguin\_pi
2. SugarOS-Full-4.5.1

All you have to do is:

```
sudo mv penguin_pi penguin_pi_old  
sudo mv SugarOS-Full-4.5.1 penguin_pi
```

Now, when your users next log on they'll be using your new version of SugarCRM. However, there's one little bit of tidying up that you want to do before they do that – the database name.

The database name has no affect on the operation at all, and so the users will be unaware of the fact that their database was named `penguinpi` when they last logged on, but is now named `sugarcrm_new`. However, it is easier for *you* to manage if you maintain some consistency. Therefore the final thing to do is to rename the database, and then tell your new version of SugarCRM about the change.

Renaming the database is easy:

```
cd /var/lib/mysql/  
sudo mv penguinpi penguinpi_old  
sudo mv sugarcrm_new penguinpi
```

b81825581a3cb156cdff227fb7517bbd  
ebruary

Now you'll need to edit the SugarCRM `config.php` file where you'll find something like:

```
'db_name' => 'sugarcrm_new',
```

Just change it to:

```
'db_name' => 'penguinpi',
```

Finally you'll need to restart MySQL and Apache:

```
sudo /etc/init.d/mysql restart  
sudo /etc/init.d/Apache2 restart
```

And now your users will be free to carry on with their day-to-day tasks, and you can get back to the next round of customizations.

## b81825581a3cb156cdff227fb7517bbd ebruary Summary

In this chapter we've looked at how to develop, test, and use SugarCRM in a safe environment. To do this we've seen that we need: a development server, a test server, and a live server.

By setting up a development server we can ensure that SugarCRM customizations can be carried out in isolation – without any danger of affecting users of the live application. The server should have: a copy of the live application, a snapshot of the live data, and all your development work.

---

*Development and Testing Strategies for SugarCRM*

---

You should set up a test server so that you can ensure that: users have a safe environment in which they can test your customizations – they don't have to worry about affecting live data; and testing doesn't have to affect ongoing customizations – you can carry on working towards the next release whilst users test the current one.

Remember the cardinal rule – Thou shalt not do any development on a live server. The only thing that should be placed on the live server is a thoroughly tested release from the test server.

Don't take shortcuts, and always ensure that you follow a well defined process: replicate your live environment on the development server; carry out any customizations on the development server. When you're ready migrate your changes to the test server and get a well respected user to do all of the testing on the test server; if testing is successful then agree a time to migrate your changes to the live server.

If you load your data into a new database ensure that the SugarCRM version is set to the correct value when you finish.

Now that you know that you can develop and test your application safely, it's time to move on to Chapter 8 and look at developing a completely new module.