MSc in Web Technologies

2013/2014

Deployment

Mark Ryder

09107037

[mark.ryder@student.ncirl.ie](mailto:mark.ryder@student.ncirl.ie)

Project in Deployment

Contents

***Introduction.....................................................................................................Page 2***

***Approach for Project.....................................................................................Page 2 User Requirements........................................................................................Page 4 Functional and Design Specification.......................................................Page 5 Functionality...................................................................................................Page 5 Detailed Design...................................................................................................................Page 7*** ***Test Plan............................................................................................................Page 8 Unit Tests.........................................................................................................Page 8 Functional tests...................................................................................................................Page 9***

***ITIL......................................................................................................................Page 10***

***Conclusion........................................................................................................Page 12***

***Execution Plan................................................................................................Page 13***

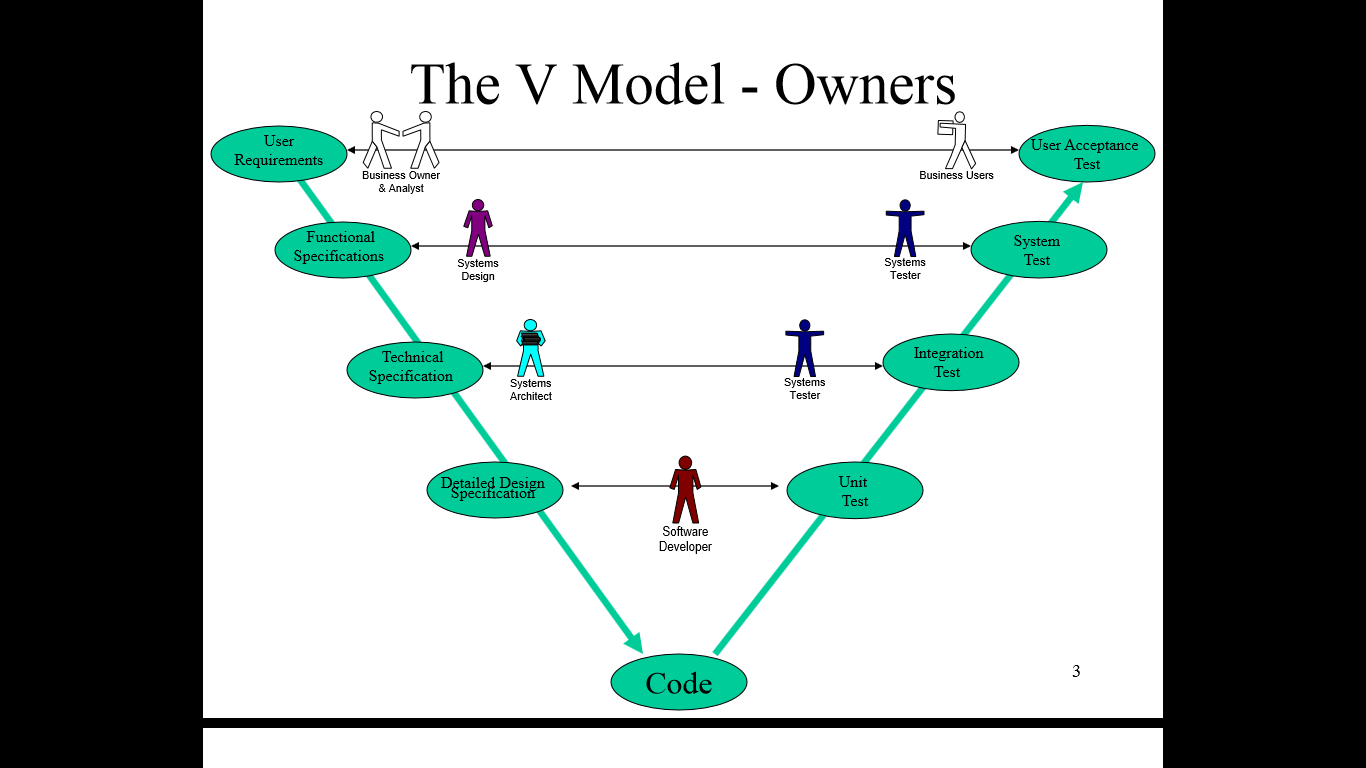
***Appendix...........................................................................................................Page 14***

**Introduction**

When IT services first started to be implemented by businesses, deployment required building of physical servers and configuration of routers. These days, deployment is much more about controlling the process programmatically, moving away from a reliance on hardware technologies. This project represents the basic premise of how deployment would be implemented.

The first section, *Approach for Project,* describes the V Model method which was implemented in this project. It describes each sequence in the method and the roles involved. *User Requirements* is the first step in the V model. This section gives the requirements the business owner or user requires for this project to be deemed a success. Using the inputs from the user requirements, *the Functional and Design Specifications* are described in the section. The *Test Plan* of the V model (the right hand side of the model) needs to be mapped back or must be parallel with the sections on the left hand side of the V model. The Test Plan section shows how this was achieved. The *ITIL* section details the best practices used to complete and maintain the project. The *Conclusion* determines whether this project was a success and what could have been done to improve the project. There is also a *Execution Plan* detailing how to run the project and *Appendix* featuring screen shots of the different scripts used to complete the project.

**Approach for Project**

****

V - Model is a Verification and Validation model. Just like the waterfall model, the V -shaped life cycle is a sequential path of execution of processes, as shown in the diagram. Each process must be completed before the next process begins. Testing of the project is planned in parallel with the corresponding process of the model. i.e. Detailed Design Specification is in parallel with Unit Test.

The advantages of the V - Model are :

1. Simple and easy to use.
2. Testing activities happen well before coding. This saves a lot of time and gives a better chance of success than the Waterfall Model.
3. Proactive defect tracking, that is defects are found at an early stage.
4. Avoids the downward flow of defects
5. Works well for small projects where requirements are easily understood.

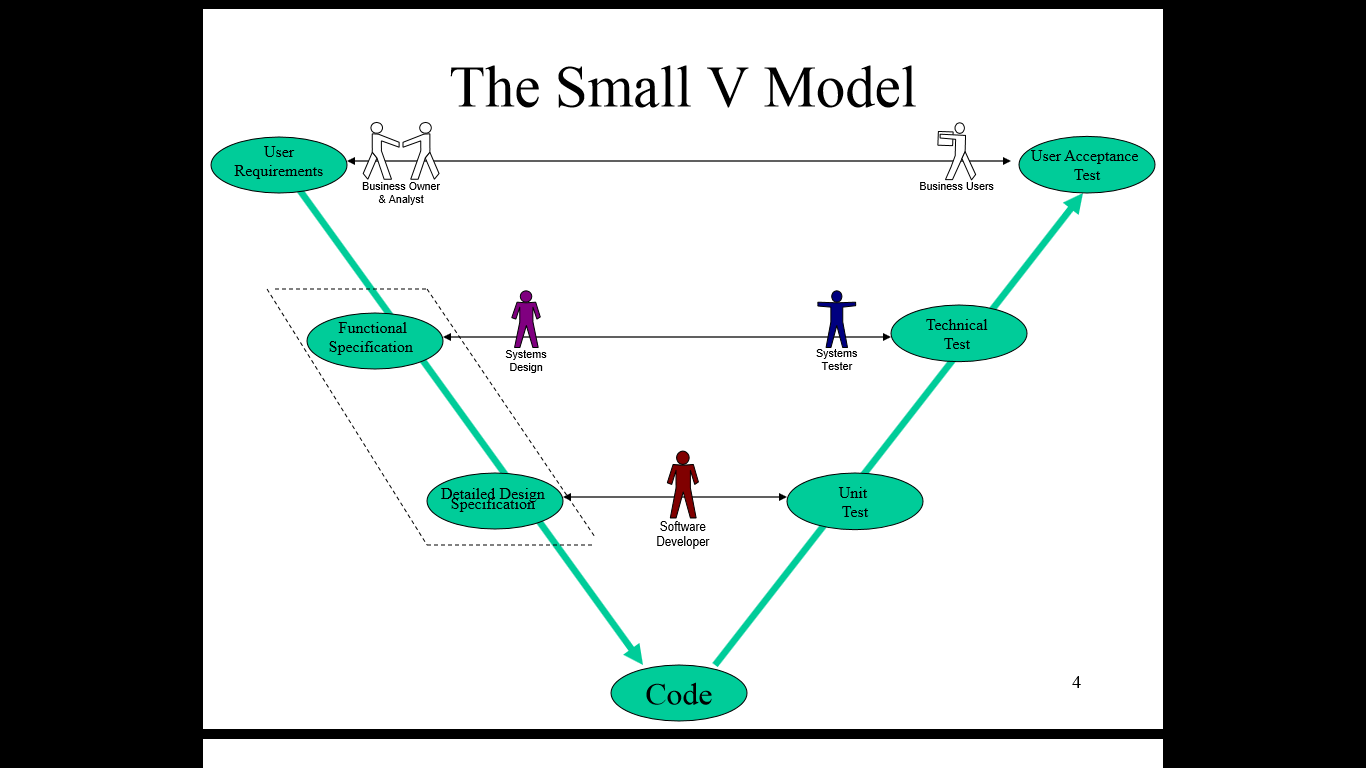
The disadvantages of the V - Model are:

1. Vary rigid and less flexible than Waterfall Model.
2. Software is developed during the implementation process, so no early prototypes of the project are produced.
3. If any changes happen midway through, then the test documentation along with requirement documentation has to be updated.

{sourced from istqbexamcertification.com, 22/1/14}

Before coding, all processes require signed off documents. The author of each process sign off documents are the people assigned to that particular process role.

|  |  |
| --- | --- |
| Document | Role |
| Functional Specification | System Design |
| Technical Specification | System Architect |
| Detail Design Specification | Software Developer |



# Since the is a small project, the small V - Model, as shown in the diagram, would be better suited then the normal V - Model. This model only requires one signed off document which covers the Functional Specification and Design Detail Specifications. This document is known as the Functional and Design Specification. This document may use flow charts, pseudo code and use cases to display the functions and design of the project. It is also used as the input for the Unit Tests and System Tests. Before you can write up the Functional and Design Specification, the User Requirements need to be completed by the business owners or end users. The User Requirements are used to create inputs for the Functional and Design Specification. It is also used to match back to the User Acceptance Test which is also completed by the end user. The User Requirements are displayed in MoSoCow format.

# User Requirements

The user requirements are laid out using the MoSoCow format and must be agreed upon with the business owners or end user. The MoSoCow format is a technique used in business analysis as well as software development to reach a common undertaking with the business owners on the importance they place on the delivery of each user requirement.

Mo - MUST - Describes a requirement that most be satisfied in the final solution to be considered a success. The must have requirements are critical to the project's success and have to be included within the project timeframe in order for the project to be a success. If any of the must have requirements are not met the project is a failure.

So - SHOULD - These are high priority requirements which should be included if possible. The should have requirements are as important as the must have requirements. The main difference is that they are not as time critical, allowing time for the must have requirements.

Co - COULD - This is a desirable requirement but is deemed not necessary. The requirement will be included if time and resources permit. The could have requirements could be considered extras or bonus requirements which add to the project's functionality but are not critical to the project's success.

Requirements Table

|  |  |  |
| --- | --- | --- |
| Requirement  Number | MoSoCow  type | Requirement Description |
| 1 | Mo | Content and components must be deployed with a single command. |
| 2 | Mo | Content and Components must be deployed using a clean environment. |
| 3 | Mo | Must be able to download content from repository . |
| 4 | Mo | Must be able to unpack content and move to proper location on the production server. |
| 5 | Mo | All static and dynamic content should be accessed from a browser. |
| 6 | Mo | The monitoring process must send reports to a log file. |
| 7 | Mo | If the deployment script fails, the old site must be still kept in place. |
| 8 | So | Each process should work from its own process directory, i.e. build process should run in the build directory. |
| 9 | So | Each process directory should be cleaned after each process execution. |
| 10 | So | The monitoring processes should send an email to an administrator if there is an error detected. |
| 11 | So | The monitoring process should have six key parameters, - Check mySql port, Apache http port, disk space, memory, network availability, network utilities. |
| 12 | Co | The build, integrate, test process could have a separate server (called pre-production) from the deploy and monitoring process. |
| 13 | Co | The deploy process could have its own server called production server. |
| 14 | Co | The monitoring process could have its own server called monitoring server. This server is monitoring both the pre-production and production servers. |

# Functional and Design Specification

# Functionality

# The use cases are used to specify the functionality of the project. A use case is a list of steps, typically defining interactions between a role and a system, required to achieve a goal. The use cases define the following processes:

Use Case 1 (UC1) - Process: Trigger

Use Case 2 (UC2) - Process: Debug

Use Case 3 (UC3) - Process: Monitor

Use Case 4 (UC4) - Process: Generate Report

|  |  |
| --- | --- |
| ID: | UC1. |
| Title: | Trigger Use Case. |
| Role: | End user. |
| Summary: | The end user executes the main script, myscript.sh. It pulls content from repository. Performs build, integrate, test and deploy processes. |
| Precondition: | End user executes myscript.sh |
| Sequence: | 1. End user runs myscript.sh. 2. Script cleans environment. 3. Pulls content from GitHub (build). 4. Manipulates database (integrate). 5. Tests manipulation takes place (test). 6. Moves files to proper location on server (deploy). 7. Removes clean environment. |
| Exceptions: | If error is detected by debug process (UC2), script stops preventing deployment. |
| Post Condition: | Successful deployment. |

|  |  |
| --- | --- |
| ID: | UC2. |
| Title: | Debug Use Case. |
| Role: | Main script, myscript.sh. |
| Summary: | Before a build, integrate, test, deploy process can be run, a script is run to test for errors. |
| Precondition: | Myscript.sh calls error scripts. |
| Sequence: | 1. Before the build code starts, GitTest.sh runs to make sure there is a connection to GitHub. 2. Before integration and test code starts, mySqlTest.sh runs to test mysql. 3. Before deploy code starts, apcaheTest.sh runs to test apache server. |
| Exceptions: | If there are no errors, myscript.sh will complete deployment. |
| Post Condition: | If there are errors, the script will stop, a message will echo to the console stating process where error occurred and email the administrator. The administrator will use bugzilla to debug system. |

|  |  |
| --- | --- |
| ID: | UC3. |
| Title: | Monitor Use Case. |
| Role: | Crontab. |
| Summary: | Monitor process script will monitoring server during and after production. |
| Precondition: | Crontab will be scheduled to run every minute to output the test results to a log file. |
| Sequence: | 1. myscript.sh runs clean.sh, which cleans environment. 2. This scripts stops the crontab schedule. 3. The next stage is to remove log file from pervious deployment cycle. 4. Finally the crontab is restarted, the next time it runs a new log is created and updated every minute till the next deployment is started. |
| Exceptions: | The only time the monitoring process does not run is during the clean.sh is executed. |
| Post Condition: | If an error occurs the administrator will be emailed, he/she can check the log file to see where the error occurred and the time/date. |

|  |  |
| --- | --- |
| ID: | UC4. |
| Title: | Generate Report Use Case. |
| Role: | Crontab. |
| Summary: | As described in previous use case, monitor process will generate a log file when it is first run. |
| Precondition: | Crontab will schedule to run every minute and output to a log file. |
| Sequence: | 1. After running the monitor.sh, crontab will create a log file. 2. If the log file exists, the crontab will update the file. |
| Exceptions: | As pervious use case stated, crontab always running except during clean.sh execution. |
| Post Condition: | Administrator can access log file at any time to generate report. |

**Risks and Controls**

For every software development project, there are risks involved. Controls should be implemented to reduce the impact of any risks to a project.

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Impact | Risk | Control |
| 1 | High | Build process may not be able to get content from repository. | GetTest.sh prevents build process from running if there is a problem. |
| 2 | High | Integrate process may not be able to manipulate the database. | mySql.sh prevents integrate process from running if there is a problem. |
| 3 | High | Test process may not be able to test manipulation of the database. | mySql.sh prevents test process from running if there is a problem. |
| 4 | High | Deploy process may not be able to move the files into proper location. | apacheTest.sh prevents deploy process from running if there is a problem. |

# 

# Detailed Design

Run by crontab

monitor.sh

build.sh

integrate.sh

test.sh

clean.sh

deploy.sh

# 

Run by myscript.sh

# The main script, myscript.sh, calls clean.sh, build.sh, integrate.sh, test.sh and deploy.sh in sequence to complete the deployment process. Monitor.sh is scheduled to run every minute by the crontab. The scripts are broken down step by step as follows.

**Clean.sh**

1. Stops Apache, mySql and crontab.
2. Removes Apache, mySql and the log file.
3. Reinstalls Apache and mySql.
4. Restarts Apache, mySql, crontab.

**Build.sh**

1. After myscript.sh has created sandbox and the process directories, build.sh downloads content from repository.
2. Zips content.
3. Moves zipped content to the integrate process directory.
4. Cleans directory.

**Integrate.sh**

1. Unzips content sent from build.sh.
2. Opens mySql.
3. Creates new table custdetails in database called dbtest.
4. Inserts values in to the table.
5. Zips content.
6. Moves zipped content to test process directory.
7. Cleans directory.

**Test.sh**

1. Unzips content send from integrate process directory.
2. Opens mySql.
3. Checks to see if content inserted in custdetails is there.
4. Zips content.
5. Moves zipped content to deploy process directory.
6. Cleans directory.

**Deploy.sh**

1. Unzips content send from test process directory.
2. Moves html files into their proper location on the server.
3. Moves script files into their proper location on the server.
4. Permissions to execute are given to the script files.
5. Cleans directory.
6. Removes Sandbox.

# Test Plan

Rather than trying to get all the processes to run in big script, it was easier to test each process individually by putting them into their own script. The main script would than call the process script at the right time. Each line of code or command can be executed on its own in the Ubuntu terminal. If there is a problem with the syntax of the command, Ubuntu would give an error and point to the problem. Each line was tested this way until it ran with no errors and then put in to the process script. Before each process script could be executed a functional test needed to be created for each process script. The functional tests will link back to the risks and controls table in the functionality section of the function and design specification. The functional tests are made up of level 1 and level 0 functions. The level 1 functions are essentially unit tests. The unit tests map back to the detailed design of the function and design specification. Another way to look at this the unit tests are in parallel with the detail design section of the V- model and functional tests are in parallel with the functional section of the V - model.

**Unit Tests**

**isApacheListening test:**

Will return true if Apache TCP is listening.

Will return false if not.

**isApacheRunning test:**

Will return true if Apache is running.

Will return false if not running.

**isMysqlRunning test:**

Will return true if Mysql is running.

Will return false if not.

**isApacheRemoteUp test:**

Will return true if Apache port is open.

Will return false if not.

**isMysqlRemoteUp test:**

Will return true if Mysql port is open.

Will return false if.

**isMysqlListening test:**

Will return true if Mysql TCP is listening.

Will return false if not.

**isIpGitAlive test:**

Will return true if GitHub is pinged at least once.

Will return false if not.

**Functional tests**

**mySqlTest.sh:**

Checks if mySql is running.

Checks if mySql TCP is listening.

Checks if MySql port is open.

If all checks are successful the test is passed.

Will not pass if any check fails and increments the ERRORCOUNT.

**ApaceTest.sh:**

Checks if Apache is running.

Checks if Apache TCP is listening.

Checks if Apache port is open.

If all checks are successful the test is passed.

Will not pass if any check fails and increments the ERRORCOUNT.

# 

**GitTest.sh:**

Pings GitHub.com 9 times (site might ping first time round)

If a ping is successfully received the test is passed.

Will not pass if no pings are received, increments ERRORCOUNT.

**ITIL Processes**

ITIL stands for Information Technology Infrastructure Library. It is used as a set of practices for IT service management that focuses on aligning IT services with the needs of business. ITIL describes processes, procedures, tasks and checklists that are not organisation-specific, used by an organisation for establishing integration with the organisation's strategy, delivering value and maintaining a minimum level of competence

**ITIL**

**Service Design**

1. Service Level Agreement
2. Availability
3. Capacity
4. Information Security

**Service Transition**

1. Configuration Management
2. Asset Management

**Service Operation**

1. Incident Management
2. Problem Management
3. Identity Management

**Service Design**

Service Design is used to provide good practice guidance on the design of IT services and addresses how a planned service solution interacts with the larger business and technical environments. The processes used in Service Design in this project are:

1. Service Level Agreement - This agreement ensures that the agreed IT services are delivered when and where they are supposed to be. In this project the Apache 2 server and mySql are the agreed services that need to be running. Without this Service Level Agreement, Availability Management, Capacity Management and Information Security cannot operate.
2. Availability Management - Makes sure Apache and mySql are available to perform aver the agreed time set out in the Service Level Agreement. The functional tests make sure that the Apache and mySql are reliable and resilient.
3. Capacity Management - would involve making sure the system can run the project. The memory test in the monitoring script alerts the administrator if the system memory becomes an issue.
4. Information Security - To run the integration and test process in pre production, access to mySql is required. The password needs to be hardcoded into the integration and test scripts. This process falls under Information Security.

**Service Transition**

Service Transition relates to the delivery of services required for live/operational use, and often encompasses the project side of IT rather than business as usual (BAU).The processes involved with Service Transition in the is project are:

1. Configuration Management - At the start of the build process in pre production, a sandbox is required to hold the content downloaded from the repository. The sandbox holds a clean environment sanitizes the content preventing the likely hood of any errors.
2. Assets Management - As part of creating the clean environment, apt-get remove and install Apache and mySql. This process ensures that the project assets, Apace and mySql, are up to date.

**Service Operation**

After the project has run the deployment process, Service Operation processes take over to make sure the project keeps running. The Service Operation processes in this project are:

1. Problem/Incident Management - These are actually two separate processes but are very closely linked. An Incident is a catastrophe error that brings down the system, the Incident Management process has to be in place to deal with the incident. It would be considered a reactive process. The Problem Management is considered to be a proactive process. It is designed to find and resolve the problem. In effect should prevent an incident. Both management processes use the crontab logger.
2. Identity Management - Full user privileges are required for some of the code in the processes in the project. As a testuser does not have these privileges, a sudo command is required to move the content to the appropriate locations in the server.

**Conclusion**

The project is deemed a success as it meets the Must have requirements set out by the business owners. Three out four of the Should have requirements were also met. Unfortunately due to time constraints , I could not complete the last Should have requirement - The monitoring process should have six key parameters, - Check mySql port, Apache http port, disk space, memory, network availability, network utilities. The only key monitoring process I could not find in time was disk space.

As a result, there was not enough time to complete the Could have requirements. In a real world environment, The project would have been completed over three servers - pre production, production and monitoring.

Monitoring server - uses crontab

Production - deploy processes

Pre production - build, integrate, test processes

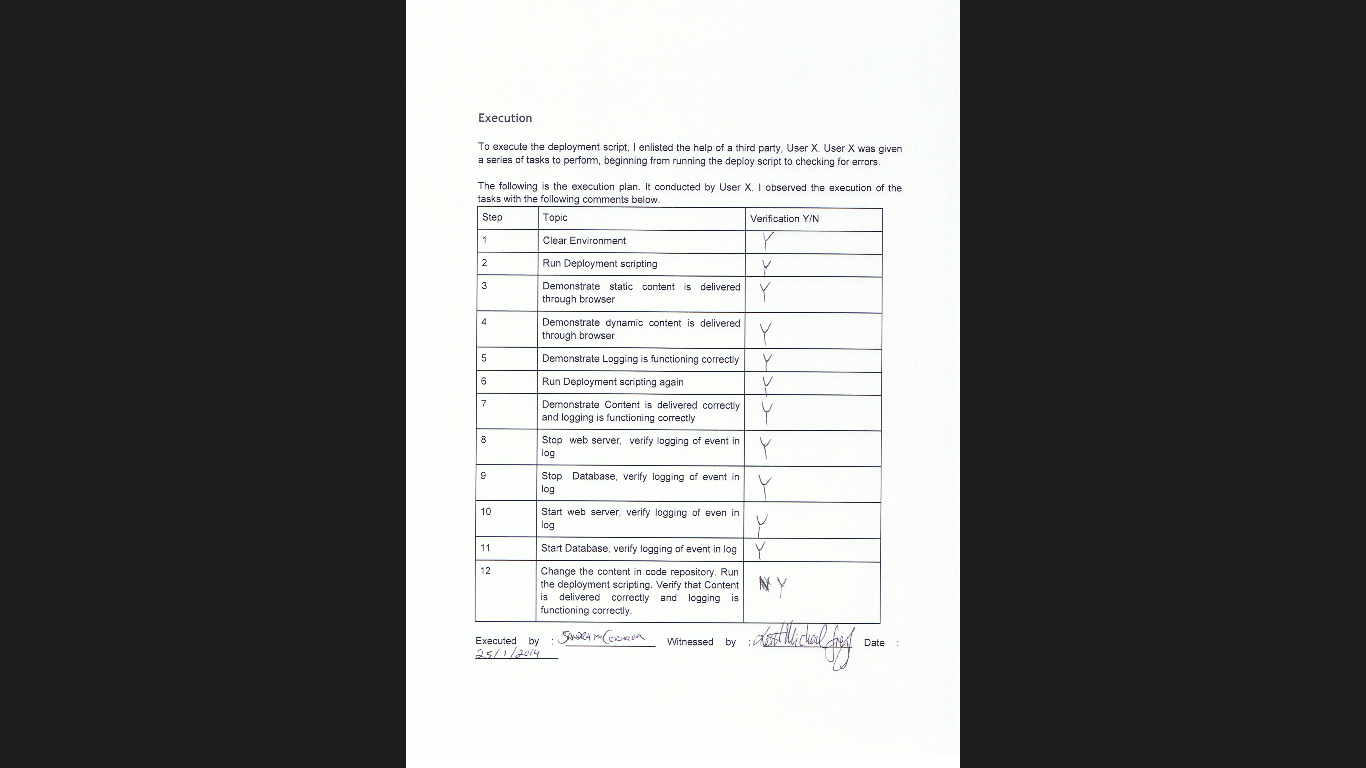
However the essence of the project could be shown on one server once it has met all the Must have requirements.

**Execution Plan**

1. Edit crontab by typing crontab -e, at the end of editor type \*/1 \* \* \* \* /home/testuser/monitor.sh >> /home/testuser/test2.log
2. Run main script by typing in: sudo ./myscript.sh and type password for sudo password in the Ubuntu terminal, the script will clear environment before running processes.
3. To check static content open browser, Morzilla Firefox or Google Chrome, and type localhost:8089/index.html.
4. To check dynamic content, type in localhost:8089/form.html and add values to the form and submit, accept\_form.pl will run and return value plus pervious values (including values inputted during integration process.
5. Open test2.log to view logging by typing cat test2.log in Unbuntu terminal.
6. Type sudo ./myscript.sh to run deployment scripting again.
7. Check /var/www to see html is delivered to server and check /usr/lib/cgi-bin to see if scripts are delivered to server.
8. Type sudo /etc/init.d/Apache2 stop to stop Apache2, after a minute type cat test2.log.
9. Type sudo stop mysql, after a minute type cat test2.log.
10. Type sudo /etc/init.d/Apache2 start, after a minute type cat test2.log.
11. Type sudo start mysql, after a minute type cat test2.log.

# Type sudo cp page\_4\_no\_12.html /var/www. Then check URL localhost:8089/page\_4\_no\_12.html in the browser to see the new page deloyed.

Appendix



Record of Demonstration

myscript.sh

#!/bin/bash

#start clean environment using external script - clean.sh

source /home/testuser/clean.sh

cd /tmp

SANDBOX=sandbox\_$RANDOM

mkdir $SANDBOX

cd $SANDBOX

# Make the process directories

mkdir build

mkdir integrate

mkdir test

mkdir deploy

#BUILD

#run Git error script will not build if errorcount greater than 0

source /home/testuser/GitTest.sh

if [ $ERRORCOUNT -eq 0 ]; then

#runs build process from external script - build.sh

source /home/testuser/build.sh

else

echo Cannot continue, Error count is higher than 0 at build process

exit 0

fi

#INTEGRATE

#run mySql error script will not integrate if error count higher than 0

source /home/testuser/mySqlTest.sh

if [ $ERRORCOUNT -eq 0 ]; then

#runs integrate process from external script - integrate.sh

source /home/testuser/integrate.sh

else

echo Cannot continue, Error count is higher than 0 at integrate process

exit 0

fi

#TEST

#run mySql error script will not test if error count higher than 0

source /home/testuser/mySqlTest.sh

if [ $ERRORCOUNT -eq 0 ]; then

#runs test process from external script - test.sh

source /home/testuser/test.sh

else

echo Cannot continue, error count is higher than 0 at test process

exit 0

fi

#DEPLOY

#run Apache error script will not deploy if error count higher than 0

source /home/testuser/apacheTest.sh

if [ $ERRORCOUNT -eq 0 ]; then

#runs deploy process from external script - deploy.sh

source /home/testuser/deploy.sh

else

echo cannot continue, errorcount is higher than 0 at deploy process

exit 0

fi

monitor.sh

#!/bin/bash

#level 1 functions

function isApacheRunning {

isRunning apache2

return $?

}

function isApacheListening {

isTCPlisten 80

return $?

}

function isMysqlListening {

isTCPlisten 3306

return $?

}

function isApacheRemoteUp {

isTCPremoteOpen 127.0.0.1 80

return $?

}

function isMysqlRemoteUp {

isTCPremoteOpen 127.0.0.1 3306

return $?

}

function isMysqlRunning {

isRunning mysqld

return $?

}

function isIpGitAlive {

isIPalive www.github.com

return $?

}

#level 0 functions

function isRunning {

PROCESS\_NUM=$(ps -ef | grep "$1" | grep -v "grep" | wc -l)

if [ $PROCESS\_NUM -gt 0 ] ; then

echo $PROCESS\_NUM

return 1

else

return 0

fi}

function isTCPlisten {

TCPCOUNT=$(netstat -tupln | grep tcp | grep "$1" | wc -l)

if [ $TCPCOUNT -gt 0 ] ; then

return 1

else

return 0

fi

}

function isUDPlisten {

UDPCOUNT=$(netstat -tupln | grep udp | grep "$1" | wc -l)

if [ $UDPCOUNT -gt 0 ] ; then

return 1

else

return 0

fi

}

function isTCPremoteOpen {

timeout 1 bash -c "echo >/dev/tcp/$1/$2" && return 1 || return 0

}

function isIPalive {

PINGCOUNT=$(ping -c 5 "$1" | grep "100%" | wc -l)

if [ $PINGCOUNT -ne 1 ] ; then

return 1

else

return 0

fi

}

function getCPU {

app\_name=$1

cpu\_limit="5000"

app\_pid=`ps aux | grep $app\_name | grep -v grep | awk {'print $2'}`

app\_cpu=`ps aux | grep $app\_name | grep -v grep | awk {'print $3\*100'}`

if [[ $app\_cpu -gt $cpu\_limit ]]; then

return 0

else

return 1

fi

}

#use functions to check for errors

ERRORCOUNT=0

isApacheRunning

if [ "$?" -eq 1 ]; then

echo 1. Apache process is Running $(date +"%c")

else

echo 1. Apache process is not Running $(date +"%c")

ERRORCOUNT=$((ERRORCOUNT+1))

fi

isApacheListening

if [ "$?" -eq 1 ]; then

echo 2. Apache is Listening $(date +"%c")

else

echo 2. Apache is not Listening $(date +"%c")

ERRORCOUNT=$((ERRORCOUNT+1))

fi

isApacheRemoteUp

if [ "$?" -eq 1 ]; then

echo 3. Remote Apache TCP port is up $(date +"%c")

else

echo 3. Remote Apache TCP port is down $(date +"%c")

ERRORCOUNT=$((ERRORCOUNT+1))

fi

isMysqlRunning

if [ "$?" -eq 1 ]; then

echo 4. Mysql process is Running $(date +"%c")

else

echo 4. Mysql process is not Running $(date +"%c")

ERRORCOUNT=$((ERRORCOUNT+1))

fi

isMysqlListening

if [ "$?" -eq 1 ]; then

echo 5. Mysql is Listening $(date +"%c")

else

echo 5. Mysql is not Listening $(date +"%c")

ERRORCOUNT=$((ERRORCOUNT+1))

fi

isMysqlRemoteUp

if [ "$?" -eq 1 ]; then

echo 6. Remote Mysql TCP port is up $(date +"%c")

else

echo 6. Remote Mysql TCP port is down $(date +"%c")

ERRORCOUNT=$((ERRORCOUNT+1))

fi

echo ERROR COUNT is: $ERRORCOUNT

isIpGitAlive

if [ "$?" -eq 1 ]; then

echo 7. Git.com is pinging, there is a connection to Git $(date +"%c")

else

echo 7. Git.com is not pinging, there is no connection to Git $(date +"%c")

ruby sengmail.rb "ryddlemethis@gmail.com" "Connection Problem" "No connection to GitHub for Deployment"

fi

MaxMem=$(top -n 1 -b | grep "Mem" | cut -c 7-14)

UsedMem=$(top -n 1 -b | grep "Mem" | cut -c 25-31)

UsedPer=$(echo "scale=3; $UsedMem/$MaxMem \* 100" | bc -l)

echo The total percentage of memory used is: $UsedPer%

if [ $(echo "$UsedPer > 90" | bc) -ne 0 ]; then

ruby sendgmail.rb "ryddlemethis@gmail.com" "Critical Error" "Too much memory in use"

fi

if [ $ERRORCOUNT -gt 0 ] ; then

ruby sendgmail.rb "ryddlemethis@gmail.com" "Error Notification" "Error in Deployment Project"

fi

build.sh clean.sh

#stop services

/etc/init.d/apache2 stop

stop mysql

service cron stop

#uninstall and reinstall services

apt-get -q -y purge apache2

apt-get -q -y purge mysql-server mysql-client

apt-get install apache2

apt-get install mysql-server mysql-client

rm test2.log

#start services

/etc/init.d/apache2 start

start mysql

service cron start

#!/bin/bash

cd build

git clone https://github.com/FSlyne/NCIRL.git

#tar NCIRL and move to integrate folder

tar -czvf pre\_integrate.tgz NCIRL

mv pre\_integrate.tgz -t /tmp/$SANDBOX/integrate #clean build folder

rm -rf NCIRL

cd ..

integrate.sh

#!/bin/bash

cd integrate

tar -zxvf pre\_integrate.tgz

cd NCIRL

#opens SQL and adds in data to custdetails table

cat<<FINISH | mysql -uroot -ppassword

drop database if exists dbtest;

CREATE DATABASE dbtest;

GRANT ALL PRIVILEGES ON dbtest.\* TO dbtestuser@localhost IDENTIFIED BY 'dbpassword';

use dbtest;

drop table if exists custdetails;

create table if not exists custdetails (

name VARCHAR(30) NOT NULL DEFAULT '',

address VARCHAR(30) NOT NULL DEFAULT '');

insert into custdetails (name,address) values ('John Doe','21 Jump Street');

FINISH

cd ..

#tar NCIRL and move to test folder

tar -czvf pre\_test.tgz NCIRL

mv pre\_test.tgz -t /tmp/$SANDBOX/test

#clean integrate folder

rm -rf NCIRL

cd ..

test.sh deploy.sh

#!/bin/bash

cd deploy

tar -zxvf pre\_deploy.tgz

rm pre\_deploy.tgz

cd NCIRL/Apache

#deploy html files to /var/www in Apache

cp -R www /var

#deploy scripts to /usr/lib/cgi-bin in Apache

cp -R cgi-bin /usr/lib

#changes permissions on script

chmod +x /usr/lib/cgi-bin/\*

cd ..

cd ..

cd ..

cd ..

rm -rf $SANDBOX

#!/bin/bash

cd test

tar -zxvf pre\_test.tgz

rm pre\_test.tgz

cd NCIRL

#opens SQL and shows what was entered in to table in integrtate section

cat<<FINISH | mysql -uroot -ppassword

use dbtest;

select\*from custdetails;

FINISH

cd ..

#tar NCIRL and move to deploy folder

tar -czvf pre\_deploy.tgz NCIRL

mv pre\_deploy.tgz -t /tmp/$SANDBOX/deploy

rm -rf NCIRL

cd ..

ApacheTest.sh

#!/bin/bash

#level 0 functions

function isRunning {

PROCESS\_NUM=$(ps -ef | grep "$1" | grep -v "grep" | wc -l)

if [ $PROCESS\_NUM -gt 0 ] ; then

echo $PROCESS\_NUM

return 1

else

return 0

fi

}

function isTCPlisten {

TCPCOUNT=$(netstat -tupln | grep tcp | grep "$1" | wc -l)

if [ $TCPCOUNT -gt 0 ] ; then

return 1

else

return 0

fi

}

#level 1 functions

function isApacheRunning {

isRunning apache2

return $?

}

function isApacheListening {

isTCPlisten 80

return $?

}

#error check myscript

ERRORCOUNT=0

isApacheRunning

if [ "$?" -eq 1 ]; then

echo Apache process is Running $(date +"%c")

else

echo Apache process is not Running $(date +"%c")

ERRORCOUNT=$((ERRORCOUNT+1))

fi

isApacheListening

if [ "$?" -eq 1 ]; then

echo Apache is Listening $(date +"%c")

else

echo Apache is not Listening $(date +"%c")

ERRORCOUNT=$((ERRORCOUNT+1))

fi

echo error count is: $ERRORCOUNT

mySqlTest.sh

#!/bin/bash

#level 0 function

function isRunning {

PROCESS\_NUM=$(ps -ef | grep "$1" | grep -v "grep" | wc -l)

if [ $PROCESS\_NUM -gt 0 ] ; then

echo $PROCESS\_NUM

return 1

else

return 0

fi

}

function isTCPlisten {

TCPCOUNT=$(netstat -tupln | grep tcp | grep "$1" | wc -l)

if [ $TCPCOUNT -gt 0 ] ; then

return 1

else

return 0

fi

}

#level 1 function

function isMysqlRunning {

isRunning mysqld

return $?

}

function isMysqlListening {

isTCPlisten 3306

return $?

}

#checks errors for myscript

ERRORCOUNT=0

isMysqlRunning

if [ "$?" -eq 1 ]; then

echo Mysql process is Running $(date +"%c")

else

echo Mysql process is not Running $(date +"%c")

ERRORCOUNT=$((ERRORCOUNT+1))

fi

isMysqlListening

if [ "$?" -eq 1 ]; then

echo Mysql is Listening $(date +"%c")

else

echo Mysql is not Listening $(date +"%c")

ERRORCOUNT=$((ERRORCOUNT+1))

fi

echo error count is: $ERRORCOUNT

GitTest.sh

#!/bin/bash

#level 0 function

function isIPalive {

PINGCOUNT=$(ping -c 9 "$1" | grep "100%" | wc -l)

if [ $PINGCOUNT -ne 1 ] ; then

return 1

else

return 0

fi

}

#level 1 function

function isIpGitAlive {

isIPalive www.github.com

return $?

}

ERRORCOUNT=0

isIpGitAlive

if [ "$?" -eq 1 ]; then

echo Git.com is pinging, there is a connection to Git $(date +"%c")

else

echo Git.com is not pinging, there is no connection to Git $(date +"%c")

ERRORCOUNT=$((ERRORCOUNT+1))

fi

echo ERROR COUNT is: $ERRORCOUNT