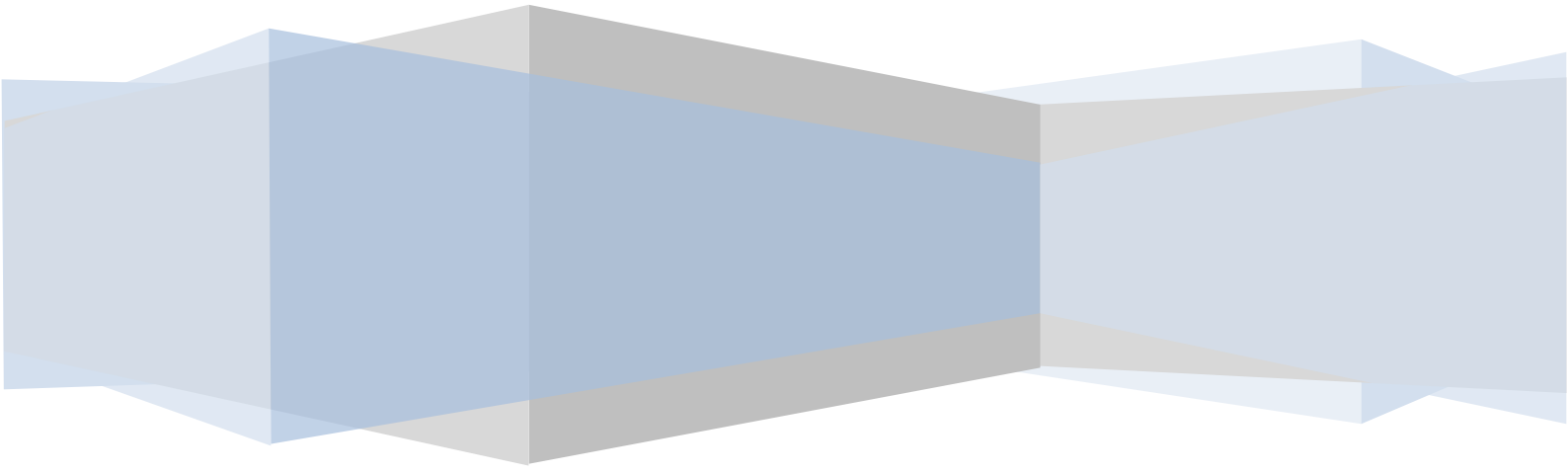
Case Study Report

EQUITY PULSE

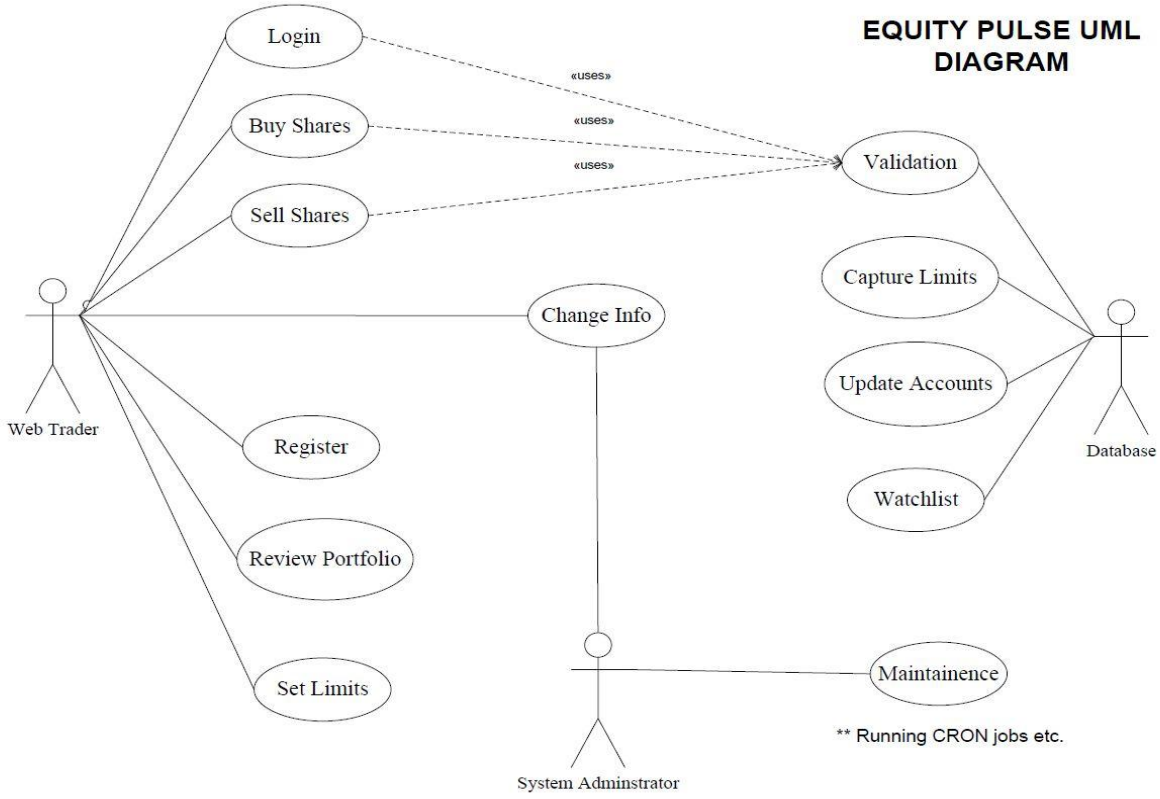
A virtual Stock Market trading system

Group 7: Rajesh Pandey, SVV Ravikumar, Namra Maheshwari, Rahul Gupta



Introduction:

Equity Pulse is an electronic marketplace where clients can connect and post orders to buy and sell stocks. The system is responsible for processing new orders received, placing them in an order book and executing a matching function on the order book. These functions occur whenever the trading server receives a new order. The trading system accepts orders from clients, sending acknowledgements, ACK's, or negative acknowledgements NACK's, in response to communicate receipt of the order or rejection of the order to a given client. If the match function results in actual matches of buyers and sellers, the trading system communicate fill messages back to the parties involved in the transaction. Normally this would be a secure communication between the server and the individual clients involved in the transaction, but this specification only requires that the fill message contain unique identifiers for the buyer and seller (and that the buyer and seller client programs process only the fill information that is related to their unique identity). Whenever information passes into the trading system those results in material change in the state of the order book, the trading system publishes market data messages. These messages are anonymous and represent aggregated data. The last message includes the price, quantity and time of a fill or trade, the book message contains up to the top twenty buy and sell price levels (after all matching has resolved). Trading systems have extensive audit functionality, for this specification the only required audit capability is two administrative web pages that show 1) all messages sent and received organized by type and 2) all trades (fills) executed during the session and some aggregated measures.



Requirements:

Our system has a set of primary functional requirements:

- Client connectivity, implementing an authentication process when a client first connects.
- Managing client connectivity to the database and organizing client orders in an Order book.
- Executing a first-in-first-out matching algorithm when new orders enter the system.
- Transmitting order fills to clients and broadcasting real-market data to clients.
- Recording all incoming and outgoing messages and provide an interface that can show the data in an organized fashion including important real-market statistics.
- A process that notifies clients of market close.
- A well designed Web interface for order entry.
- The system should be portable and platform independent.
- The system must allow transactions from registered users only.
- All the transactions must be recorded.

Problems:

- As this is a dynamic system, data retrieval goes on for half a day. So we need to optimize the system.
- The back-end script is very vaguely written code with zero documentation, every year a new
- Developer comes; most of his time goes in just understanding the code.
- Degradation of system performance with increase in number of players.
- Another major problem is the 'time lag' between real and virtual market (approx. 3 secs), so a player can watch real market and trade in virtual market. This creates a significant doubt about the whole idea of the game.
- Website is not secure (Easy SQL injection is possible)
- usage of *hashing* and *mysql_real_escape_string()* for unsafe variables should be used.

Proposed Solutions:

- **Rewriting the code**
  - After studying the code it was decided by our team to rewrite the code.
  - This time with proper modularization and documentation.
  - Whole code sequence is divided into three parts
    - Data-retrieval
    - Data-parsing
    - Populating-database
    - CRON Job

- **Optimization of Resources**
  - Hardware Optimization
  - Page Optimization
  - Database Optimization

Hardwar e Optimization:

- In order to improve the performance of existing servers, we can add one or more mirror servers, and use the load-balancing algorithm to assign the multiple concurrent tasks to different servers.

Page Optimization:

- Add Expires Header to the page. Expires Header value is set correctly, the page elements can be cached in the client. Clients can use cache data to reduce the number of HTTP requests, making web loading speed faster.
- To build independent external file to store the JavaScript and CSS, and to share these files as much as possible. Use an external file will speed up page display speed, because the external file will be cached in client.

Database Optimization:

- Instead of setting up a single database, A cluster of databases can provide a more reasonable approach. Cluster includes separate databases for logDB , CRON-Job-database (or primary DB), static-database (contains all the information about the system, which does not changes frequently.)
  Reason: Decreases query time due to light weight primary database, also reduces cache load.
- SQL statement optimization, that is to improve the efficiency of data query by writing more efficient SQL statements.

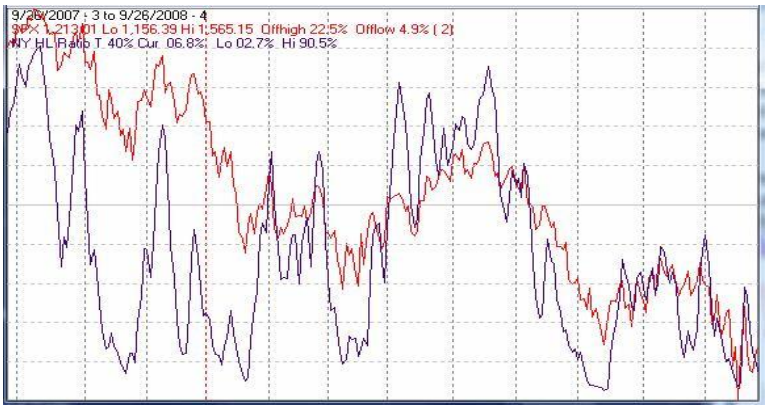
Solution for 'Time Lag'

As discussed above this is one of the major problems of this system. So our team came with the solution of randomizing the prices of stock based on their **52 weeks range**. It is said in the stock market that price of a stock , more or less lies between its last 52 weeks range (with fewer exceptions like market crash etc.) .Using this range we can calculate the average variance of a stock price. Now given the current price of stock in real market and average variance it can have, we can generate a random number i.e.

new_stock_price ~ current_stock_price + RANDOM (average_variance)

What is RANDOM?

Randomizing the variance using Monte Carlo Algorithm or Eigen Faces algorithm.



Real Market Price
Monte Carlo Randomization

Verification & Validation:

Our new system is designed, now we are going to apply some techniques to verify and validate system components. Techniques thus used are as follows:

- *Basic process*: Evaluation of the artifacts.
- *Software inspection and peer reviews:*
  Software inspection and peer reviews are used to analyze and the system representations such as the design diagrams, the source code of the program. The inspections can also be replaced by some automatic works of the text documents. This is basically a static verification and validation technique it does not require to run software.
- *Code walkthrough and Inspection:*
  checking whether the written comments and documents are correct.
- *Testing:*
  Testing is a dynamic technique of verification and validation where it involves the running software by supplying some test cases. We examine the outputs of the software and check whether it has met the requirements.

After completing first two steps successfully we are moving towards Testing, We will be discussing the same briefly in next section

Testing

The various types of testing which should be done with respect to the requirements are:

Interface testing:

Check if all the interactions between these servers are executed properly. Errors are handled properly. If database or web server returns any error message for any query by application server then application server should catch and display these error messages appropriately to users.

Compatibility testing:

Compatibility of your web site is very important testing aspect. See which compatibility test to be executed:

- *Browser compatibility*
- *Operating system compatibility*
  All new technologies used in web development like graphics designs, interface calls like different API's may not be available in all operating systems.

Performance testing:

Web application should sustain to heavy load. Web performance testing should include:

- *Web Load Testing*
- *Web Stress Testing*

Web Load Testing
Test application performance on different internet connection speed. In web load testing test if many users are accessing or requesting the same page. Can system sustain in peak load times? Site should handle many simultaneous user requests, large input data from users, Simultaneous connection to DB, heavy load on specific pages etc.

Web Stress Testing
Generally stress means stretching the system beyond its specification limits. Web stress testing is performed to break the site by giving stress and checked how system reacts to stress and how system recovers from crashes.
Stress is generally given on input fields, login and sign up areas.

Security testing:

- Test by pasting internal URL directly into browser address bar without login. Internal pages should not open.
- Try some invalid inputs in input fields like login username, password, and input text boxes. Check the system reaction on invalid inputs.(static checking)
- All transactions, error messages, security breach attempts should get logged in log files somewhere on web server.
- Web directories or files should not be accessible directly unless given download option.

Database testing:

This Application involves complex transaction which are performed both at UI level and Database level, Therefore Database testing is as important as functional testing. Database in itself is an entirely separate layer hence it is carried out by database specialists and it uses techniques like:

- Data loading
- Database Migration
- Testing DB Schema and Data types
- Rules testing
- Testing Stored Procedures and Functions
- Testing Triggers, Data Integrity.

Smoke testing:

Testing the system modules with limited scenarios of happy path.

Q: Why Security Testing is necessary, there's no 'Return on Investment' in it?

A: *By performing a quality 3rd party Security Test, you can verify the need for resources, justify a budget request, or even assist management in understanding the methodologies and technologies used in Security.*

On the Active side of ROI, Security Testing can point out areas for improvement that can improve efficiency and reduce downtime, allowing for the maximum throughput of information allowed by the technology and processes utilized.

Testing which should be performed:

- *Regression testing:*
  Regression testing cannot be done since the basic system is working fine and we need to test only on the enhancements that are made to the system. Moreover we have already done smoke testing so this time of testing is mere a waste of time and resources. (Regression testing means testing the whole system again after fixing some defects.)

- *Unit testing:*
  Testing each and every module, by supplying various boundary test cases. No generic script can be written, hence no automation.

- *Acceptance testing:*
  As there is no client for this system to whom we have to handover the system after development, so user has to accept this in its form.

Code Walk through:

After going through step 4 of our testing guidelines we moved to step 3 and prepared a checklist of code testing by reviewing the given code. We divided the code into 3 parts as discussed above.

- *Data-retrieval*

```
import urllib
file=urllib.urlopen('http://www.nseindia.com/content/equities/niftywatch.htm')
dat=file.read()
file.close()
a=ex()
a.feed(dat)
a.close()
list=a.output()
sharedir={}
for elem in list:
        if elem[0]=='M':
                elem[0]='M&M'
        sharedir[elem[0]]=elem[1:]
```

- *Data-parsing*

```
from sgmllib import SGMLParser
class ex(SGMLParser):
        def reset(self):
                self.temp=[]
                self.finallist=[]
                self.count=0
                self.flag=0
                SGMLParser.reset(self)
        def unknown_starttag(self,tag,attrs):
                if tag=='td' and len(attrs)>0 and 'class' in attrs[0] and attrs[0][1] in
['t0','t1','t2']:
                        self.flag=1
                        self.count=self.count+1
        def handle_data(self,text):
                if self.flag==1:
                        self.flag=0
                        self.temp.append(text)
```

- *Populating-database*

```
import MySQLdb
conn=MySQLdb.connect(host='#########',user='#########',passwd='######',db='########')
cursor=conn.cursor()
qr='select company,last_price from data';
cursor.execute(qr)
answer=cursor.fetchall()
for ans in answer:
        if ans[1]==float(sharedir[ans[0]][3]):
                qr="""update data set
flag=0,prev_close='%s',open='%s',high='%s',low='%s',last_price='%s',per_change_prev='%
s',total_trad_quantity='%s',turnover='%s' where
company='%s'"""%(float(sharedir[ans[0]][4]),float(sharedir[ans[0]][0]),float(sharedir[
ans[0]][1]),float(sharedir[ans[0]][2]),float(sharedir[ans[0]][3]),float(sharedir[ans[0
]][5]),float(sharedir[ans[0]][6]),float(sharedir[ans[0]][7]),ans[0])
                cursor.execute(qr)
        else:
                qr="""update data set
flag=1,prev_close='%s',open='%s',high='%s',low='%s',last_price='%s',per_change_prev='%
s',total_trad_quantity='%s',turnover='%s' where
company='%s'"""%(float(sharedir[ans[0]][4]),float(sharedir[ans[0]][0]),float(sharedir[
ans[0]][1]),float(sharedir[ans[0]][2]),float(sharedir[ans[0]][3]),float(sharedir[ans[0
]][5]),float(sharedir[ans[0]][6]),float(sharedir[ans[0]][7]),ans[0])
                cursor.execute(qr)
#print qr
from time import time
now=time()
now=int(now)
for elem in list:
        if elem[0]=='M':
                elem[0]='M&M'
        qr="""insert into NewPrices
values('%s','%d','%f')"""%(elem[0],now,float(elem[4]))
#       print qr
        cursor.execute(qr)
cursor.close()
conn.commit()
conn.close()
```

- *CRON Job*

```
#!/usr/bin/sh
while [ 1 ]
do
        python extractstock.py
        php update_portfolios.php
          php update_rank.php
        sleep 5
done
```

Code Review Checklist:

- ✓ **Data fault checks:**
  - Variables used before initialisation.
  - Possible array bound violations.
  - Undeclared variables.
- ✓ **Control faults**
- ✓ **Unreachable code**
- ✓ **Unconditional branches into loops.**
- ✓ **Interface faults check lists:**
  - Parameter type mismatches.
  - Parameter number mismatches.
  - Non usage of results of functions.
  - Uncalled functions or procedures.
- ✓ **Storage management faults.**
- ✓ **Unassigned pointers.**

Test Cases for System:

Test case id:	TC#1
Objective of test:	To check whether the password is sent to the users account During "forgot password".
Unit to test:	Equity Pulse
Assumptions:	The user needs his password since he forgot his password
Pre-requisites:	The user has entered his email id correctly.
Test data:	E-mail id string
Steps to be executed:	1) the user enters his email id. 2) Submit query to the server.
Expected result:	The user receives his password in the database to his mail Account.
Actual result:	The user does not receives his password in the database to his mail account.
Pass/Fail:	Fail.
Comments:	There is some problem in sending the password, a problem might be with the SMTP server.

Test case id:	TC#2
Objective of test:	Checking the response time of the system.
Unit to test:	When 1000 users are logged into the system then the response time of a particular query.
Assumptions:	All the users are properly connected to the system.
Pre-requisites:	The user is logged in and has entered a particular query.
Test data:	Internet connectivity/network speed.
Steps to be executed:	1) Fork 1000 HTTP requests. 2) Note system Load. 3) Note error Log. 4) Every user enters a query.
Expected result:	The system handles all the queries properly.
Actual result:	The system does not handle all the queries properly.
Pass/Fail:	Fail.
Comments:	The system has to be developed in a larger scale.

Test Case ID:	TC#3
Test Case Objective:	After selecting the shares to buy the user needs to pay. The transaction has to happen in a span of less than 2 seconds on an average (Throughput).
Test Unit:	Equity Pulse
Assumption:	payment gateway works successfully.
Sequence of steps:	1) click on the payment mode 2) Click on submit button.
Test Data:	password of the form (a-z, 0-9, special characters)
Expected Result:	success message should come after clicking the submit button this transaction should be completed within 2 sec for every such operation.
Actual Result:	**--**
Comments:	Number of transactions executed per sec should be 0.5.

Test Case ID:	TC#4
Test Case Objective:	To test system stability when all the users are logged in and playing.
Test Unit:	Equity Pulse
Assumption:	The site has all the users logged in.
Sequence of steps:	1) A 1000 users logs in to the system. 2) Start entering random queries at random spaces 3) Submit queries. 4) Check CPU usage, memory consumption of server.
Test Data:	Queries entered by users
Expected Output:	The site should work without crashing and the users should be able to work with all the functionality of the site.
Actual Output:	The pages are working properly.
Pass/Fail:	Pass.

Test Case id:	TC#5
Test Case Objective:	To test the system is taking the users input correctly.
Unit to test:	Equity Pulse
Assumption:	The site has shares transaction.
Pre-Requisite:	One user is online and is buying/selling stock.
Test Data:	Query provide by user
Steps Executed:	The user will select some stock and buy it.
Expected Result:	The product is added to the database.
Actual Result:	The product doesn't add to the database
Pass/Fail:	Fail

Test Case id:	TC#6
Test Case Objective:	To test if the system is updating the data.
Unit to test:	Equity Pulse
Assumption:	The site has the data retrieval on.
Pre – Requisite:	Site connected with the real market.
Test Data:	Stock prices of Morgan Stanley.
Steps Executed:	1) The site is connected with the real market and has the data retrieval on. 2) Check the variance with time of the stock prices.
Expected Result:	The price on the real market is same as retrieved.
Actual Result:	Error of 11.40 INR
Pass/Fail:	Fail
Comments:	There is a floating point error in the module.

Test Case id:	TC#7
Test Case Objective:	To test if the system has the archives accurately.
Unit to test:	Equity Pulse
Assumption:	The site has access to archives from the real market.
Pre – Requisite:	Site connected with the real market.
Test Data:	Trends of Morgan Stanley.
Steps Executed:	1) Login through Admin Panel. 2) Check the graphs and compare with the real market.
Expected Result:	The trends are same as that in the real market.
Actual Result:	The trends are same as that in the real market.
Pass/Fail:	Pass

Test Case id:	TC#8
Test Case Objective:	To test if the system has the algorithm running.
Unit to test:	Equity Pulse
Assumption:	The site has access to stocks from the real market.
Pre – Requisite:	Site connected with the real market.
Test Data:	Trends of Morgan Stanley.
Steps Executed:	1) Login through Admin Panel. 2) Feed the value into the algorithm and 3) Check if the value coming out is near to real value
Expected Result:	The prices have some sort of a relationship with the market and Rise and fall accordingly.
Actual Result:	The prices have some sort of a relationship with the market and Rise and fall accordingly.
Pass/Fail:	Pass
Test Case id:	TC#9

Test Case Objective:	To test if the system has the algorithm running correctly.
Unit to test:	Equity Pulse
Assumption:	The site has access to stocks from the real market.
Pre – Requisite:	Site connected with the real market.
Test Data:	Trends of Morgan Stanley.
Steps Executed:	1) Login through Admin Panel. 2) Retrieve the prices of the stock. 3) Feed the value into the algorithm and check if the variance is The one desired (with variance).
Expected Result:	The prices have some sort of a relationship with the market and rise and fall accordingly.
Actual Result:	The prices have some sort of a relationship with the market and rise and fall accordingly.
Pass/Fail:	Pass
Comments:	--

Issues with Checklist and Inspection:

After doing code review and inspection, team has yet to decide which method is more efficient, so we weighed both options.

Issues with check list:

- The cost of testing depends upon the amount of the bugs present in the system.
- Checklist can vary from language to language e.g. Java compilers check for exact number of parameters but c compiler does not check for it.
- As more bugs are surfaced the checklist has to be updated that will increase the cost.

Issues with the inspection:

- Conflict in the thoughts of the team members.
- Chance of not catching dependency related bugs.
- Cannot get the glimpse of actual operational behaviour of the system.(works on hypothetical model)

Thus our team decided that testing using checklist method is a better option than inspection.

Advantages of testing:

- It is the only way of finding operational defects, and to make sure that non-functional requirements are working as they are supposed to.
- For highly reliable and time critical systems, as there failure may lead to great loss of money and man power.

Automated Testing:

A way to playback pre-recorded and predefined actions, compare the results to the expected behaviour and report the success or failure of these manual tests.

- **Advantages:**
  - Conceptually simple.
  - Straight forward.
  - Resource savvy.

- **Disadvantages:**
  - One problem with automated tools is that we need to write additional code and we will not know if the defect is in the systems code or in the test code.
    Ex: **Windows NT 4** had **6 *million*** lines *of code*, and ***12 million*** lines of test code.

Middle way:

- Try to do automated testing only for generic modules, those which does not changes with newer versions.
- Use of already available API's and tools for automated testing
- Examples of Open source tools:
  - **Silk Monitor**:
    - 24x7 monitoring and reporting of Web, application and database servers.
  - **Silk Realizer**:
    - Scenario testing and system monitoring.
  - **Selenium**
    - It is a portable software testing framework for web applications.
  - **Ranorex**
    - It is a Windows GUI test automation framework for testing many different application types including **Web 2.0** applications.

==========