Appendix A

Table 1: U3 shirt assembly line time study

	Measurement of Observed Time (sec)										Avg (sec)
Operation Name	1	2	3	4	5	6	7	8	9	10	
Attach Back Yoke	27.5	29.5	27.2	28.5	28	26.7	30	31.1	31.4	31.7	30
	32.3	33.6	30.8	30.3	32	31.6	32.2	29.6	31.2	29.9	
Attaching and Topstitch/Closing collar size label	58.4	55.8	57.4	54.5	55.7	58.9	57.3	54.8	53.6	55.0	62
	55.1	58.8	64.0	66.0	69.0	57.7	61.0	70.0	62.0	68.0	
	56.0	56.0	67.0	64.0	56.0	57.0	57.0	60.0	68.0	57.5	
	74.0	71.0	77.0	69.0	61.0	67.0	60.0	68.0	63.0	57.0	
Joint Shoulder	35	35	35.3	33.2	35.1	34.7	32	33.4	33.8	30.2	34
	34.7	34.1	31.6	31.6	33.6	33.1	32	34.3	33.4	35	
Attach Epaulets	13	14.2	13.2	11.2	13.3	11.8	11.8	10.8	11.2	11	12
Attach Cuff	60.8	64.8	63	61	66.4	58	59.6	59.6	56.4	60.5	68
	79	78.4	81.6	83.2	82	78	78.2	77.2	80	87.2	
	71	74.6	71	75	71.6	74.6	75.2	75	74.2	70	
	58.4	60	60.2	55.4	55.6	56.6	55.4	55.8	56.2	60	
Topstitch Cuff	24.8	23.8	25.6	25	26.2	26.2	27.6	27.4	29.4	28.8	26
Attach Sleeve	34.8	35.3	35.6	37	35..4	34.2	34.8	34.5	34.2	37	36
	33	34	37.3	37.2	38.4	38.6	38.2	38.2	37.6	38.4	
Closed Side Seam	40.5	41.6	40	40.8	42.8	43.8	42.2	41.1	41.9	41.2	43
	41.7	42.2	44.3	41.9	42.7	44.1	43.7	45.1	44.7	44.1	
Bottom Hemming	28.1	27.2	27.6	26.8	27.5	26.1	25	25.9	27	25.9	29
	33.4	34.1	34.8	30.9	33.5	29.2	31.3	31.3	30	29.5	
Button Hole Collar x3/cuff	13.7	13.3	10.1	10.7	13.6	12.4	13.3	13.6	12.2	12	12
Attach Button x12	45	45.2	43.3	44.8	44.2	44.7	43.9	44.8	43.2	43.4	44
	44	45.6	41.1	40.5	44.8	44.4	40.5	44.1	43	43.2	
Marking Button Position x12	35.3	36.5	38.9	36.8	35.3	37.8	38.5	37.4	40.1	40.6	42
	48.6	49.6	46.8	45	46.6	45.2	49.5	46.7	49	50.2	
Thread Cleaning and Fasten Button	104	110	102	105	88	97	92	115	104	98	104
	99	108	82	118	102	98	118	112	107	122	
	113	110	115	106	106	112	114	120	109	105	
	96	120	103	97	95	100	85	95	89	102	
pressing	47	46	45	49	48	48	46	44	45	48	47
	49	46	45	46	45	47	48	48	45	48	
Quality Check	58	59	54	55	56	59	54	56	56	59	58
	60	59	58	59	56	60	59	58	60	59	

Table 2: U3 Shirt Assembly Line Precedence Relations

Operation No	Operation Name	Predecessor
1	Attach Back Yoke	-
2	Joint Shoulder	1
3	Attaching and Top stitch /Closing collar size label	2
4	Attach Epaulets	3
5	Attach Sleeve	4
6	Closed side seam	5
7	Attach Cuff	6
8	Top stitch Cuff 5mm	7
9	Button Hole Collar x3/Cuff	8
10	Marking Button Position x12	9
11	Attach Button (x12)	10
12	Bottom Hemming	11
13	Thread cleaning and fasten Button	12
14	Quality Check	13
15	pressing	14

Table 3: No. of Workers on U3 Assembly Line

Operation No	Operation Name	No. of Operator
1	Attach Back Yoke	2
2	Joint Shoulder	2
3	Attaching and Top stitch /Closing collar size label	4
4	Attach Epaulets	1
5	Attach Sleeve	2
6	Closed side seam	2
7	Attach Cuff	4
8	Top stitch Cuff 5mm	1
9	Button Hole Collar x3/Cuff	1
10	Marking Button Position x12	2
11	Attach Button (x12)	2
12	Bottom Hemming	2
13	Thread cleaning and fasten Button	4
14	Quality Check	2
15	pressing	2
Total Workers = 33		

Table 4: Assembly line Machine Type

Op No	Operation Name	Machine Type
1	Attach Back Yoke	Single Needle Lockstitch Machine W/Thread Trimmer
2	Joint Shoulder	Single Needle Lockstitch Machine W/Thread Trimmer
3	Attaching and Top stitch /Closing collar size label	Single Needle Lockstitch Machine W/Thread Trimmer
4	Attach Epaulets	Single Needle Lockstitch Machine W/Thread Trimmer
5	Attach Sleeve	5Threads Overlock Machine
6	Closed side seam	5Threads Overlock Machine
7	Attach Cuff	Single Needle Lockstitch Machine W/Thread Trimmer
8	Top stitch Cuff 5mm	Single Needle Lockstitch Machine W/Thread Trimmer
9	Button Hole Collar x3/Cuff	Automatic Button Holer Machine w/ Thread Trimmer
10	Marking Button Position x12	Table
11	Attach Button (x12)	Computer Controlled Lockstitch Button Sewing Machine
12	Bottom Hemming	
13	Thread cleaning and fasten Button	Cleaning Table
14	Quality Check	manual
15	pressing	press

Appendix B

Source Code

```
package com.app.site.cases;


import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.LinkedHashMap;

import java.util.LinkedList;

import java.util.Queue;

import java.util.Stack;


import javax.servlet.http.HttpServletRequest;


import com.app.core.CoreMgr;

import com.app.core.smartyglobals;

import com.app.db.mysql;


public class CaseSimModuleV2 extends CoreMgr{

	private int noOfPieces;

	private LinkedHashMap<Integer, Double> opTime = new
LinkedHashMap<Integer, Double>();
```

```
	private LinkedHashMap<Integer, Double>scenario1WaitingTime
= new LinkedHashMap<Integer, Double>();

	private LinkedHashMap<Integer, Double>scenario2WaitingTime
= new LinkedHashMap<Integer, Double>();

	private LinkedHashMap<Integer, Double> scenario1FinishTime =
new LinkedHashMap<Integer, Double>();

	private LinkedHashMap<Integer, Double> scenario2FinishTime =
new LinkedHashMap<Integer, Double>();

	private int totNoOfWorkers1 = 0;

	private int totNoOfWorkers2 = 0;

	private double finishTime1=0;

	private double finishTime2=0;


	public String getScenario1FinishTime() {

		String xAxis ="";

		boolean first = true;

		for (Integer opid : scenario1FinishTime.keySet()) {

			if (!first )

				xAxis +=",";

			xAxis +=
""+Math.round(scenario1FinishTime.get(opid));

			first = false;

		}

		return xAxis;

	}
```

```
	public String getScenario2FinishTime() {
		String xAxis ="";
		boolean first = true;
		for (Integer opid : scenario2FinishTime.keySet()) {
			if (!first )
				xAxis +=",";
			xAxis +=
""+Math.round(scenario2FinishTime.get(opid));
			first = false;
		}
		return xAxis;
	}

	public String getSecnario1WaitingTime() {
		String xAxis ="";
		boolean first = true;
		for (Integer opid : scenario1WaitingTime.keySet()) {
			if (!first )
				xAxis +=",";
			xAxis +=
""+Math.round(scenario1WaitingTime.get(opid));
			first = false;
		}
		return xAxis;
	}
```

```
	public String getSecnario2WaitingTime() {
		String xAxis ="";
		boolean first = true;
		for (Integer opid : scenario2WaitingTime.keySet()) {
			if (!first )
				xAxis +=",";
			xAxis +=
""+Math.round(scenario2WaitingTime.get(opid));
			first = false;
		}
		return xAxis;
	}

	double stagePerc = 10;

	double initTime=0;



	public String getXAxisLabels() {
		String xAxis ="";
		boolean first = true;
		int i =1;
		for (Integer mth : opTime.keySet()) {
```

```
                if (!first )
                        xAxis +=",";
                xAxis += "'OP "+i+"'";
                first = false;
                i++;
        }
        return xAxis;
}

public CaseSimModuleV2 () {
        MainSql = "select op_name, op_t_avg,op_id,
dtl_scenario1worker , dtl_scenario2worker,
ifnull(dtl_scenario2finishtime,0) as dtl_scenario2finishtime,
ifnull(dtl_scenario1finishtime,0) as dtl_scenario1finishtime
,ifnull(dtl_scenario1idletime,0) as dtl_scenario1idletime "
                + " , ifnull(dtl_scenario2idletime,0) as
dtl_scenario2idletime  "
                + "    from kboperation left join c_casesimdtls
on (dtl_opid = op_id and dtl_cid= {c_id}) ";
        //userModifyTD.put("itemno",
"modfiythis({itemno},{op_id}, {op_t_avg} )");
        canEdit = true;
        mainTable = "kboperation";
        keyCol = "op_id";
        userDefinedGridCols.add("op_name");
        userDefinedGridCols.add("op_t_avg");
        userDefinedGridCols.add("dtl_scenario1worker");
```

```
                userDefinedGridCols.add("dtl_scenario1idletime");
                userDefinedGridCols.add("dtl_scenario1finishtime");
                userDefinedGridCols.add("dtl_scenario2worker");
                userDefinedGridCols.add("dtl_scenario2idletime");
                userDefinedGridCols.add("dtl_scenario2finishtime");

                userDefinedColLabel.put("dtl_scenario1worker", "Scenario
1,Workers");
                userDefinedColLabel.put("dtl_scenario1idletime", "Scenario
1, Idle Time");
                userDefinedColLabel.put("dtl_scenario1finishtime",
"Scenario 1 Finish Time");
                userDefinedColLabel.put("dtl_scenario2worker", "Scenario
2,Workers");
                userDefinedColLabel.put("dtl_scenario2idletime", "Scenario
2, Idle Time");
                userDefinedColLabel.put("dtl_scenario2finishtime",
"Scenario 2 Finish Time");
                userDefinedColLabel.put("op_name", "Operation");
                userDefinedColLabel.put("op_t_avg", "Time (sec) per
worker");

                userDefinedEditCols.add("dtl_scenario1worker");
                userDefinedEditCols.add("dtl_scenario2worker");
                setDisplayMode("GRIDEDIT");
                userDefinedCaption = "";
```

```
            userModifyTD.put("dtl_scenario1idletime",
"showIdleTime1({dtl_scenario1idletime},{op_id},{op_t_avg})");

            userModifyTD.put("dtl_scenario2idletime",
"showIdleTime2({dtl_scenario2idletime},{op_id},{op_t_avg})");


            userModifyTD.put("dtl_scenario1finishtime",
"showFinishTime1({dtl_scenario1worker},{dtl_scenario1finishtime},{op
_id},{op_t_avg})");

            userModifyTD.put("dtl_scenario2finishtime",
"showFinishTime2({dtl_scenario2worker},{dtl_scenario2finishtime},{op
_id},{dtl_scenario1finishtime})");


            opTime = new LinkedHashMap<Integer, Double>();

            scenario1WaitingTime = new LinkedHashMap<Integer,
Double>();

            scenario2WaitingTime = new LinkedHashMap<Integer,
Double>();

            userDefinedPageFooterFunction = "playThisFooter()";

        }


        public String showFinishTime1(HashMap<String, String>hashy) {

            setTotNoOfWorkers1(getTotNoOfWorkers1() +
Integer.parseInt(hashy.get("dtl_scenario1worker")));

            String html ="<td>";

            scenario1FinishTime.put(Integer.parseInt(
hashy.get("op_id")),
Double.parseDouble(hashy.get("dtl_scenario1finishtime")));
```

```
		html +=hashy.get("dtl_scenario1finishtime");
		html+="</td>";
		return html;
	}

	public String showFinishTime2(HashMap<String, String>hashy) {
		setTotNoOfWorkers2(getTotNoOfWorkers2() +
Integer.parseInt(hashy.get("dtl_scenario2worker")));
		double time2=
Double.parseDouble(hashy.get("dtl_scenario2finishtime"));
		double time1=
Double.parseDouble(hashy.get("dtl_scenario1finishtime"));
		String arrow = "";
		setFinishTime1(time1);
		setFinishTime2(time2);
		String tdColor = "color:black";
		if (time2<time1) {
			arrow ="<i class=\"fa fa-arrow-down\"
style='padding-left:10px;'></i>";
			tdColor = "color:green";
		}else if (time1<time2) {
			arrow ="<i class=\"fa fa-arrow-up\" style='padding-
left:10px;'></i>";
			tdColor = "color:red";
		}
		String html ="<td style="'+tdColor+'">";
```

```
                scenario2FinishTime.put(Integer.parseInt(
hashy.get("op_id")),
Double.parseDouble(hashy.get("dtl_scenario2finishtime")));

                html +=hashy.get("dtl_scenario2finishtime");

                html += arrow;

                html+="</td>";

                return html;

        }


        public String playThisFooter(String colName) {

                System.out.println(colName);

                return "<td>"+colName+"</td>";

        }

        public String showIdleTime1(HashMap<String, String>hashy) {

                String html ="<td>";

                opTime.put(Integer.parseInt(hashy.get("op_id")),
Double.parseDouble(hashy.get("op_t_avg")));


        scenario1WaitingTime.put(Integer.parseInt(hashy.get("op_id")),
Double.parseDouble(hashy.get("dtl_scenario1idletime")));

                html +=hashy.get("dtl_scenario1idletime");

                html+="</td>";

                return html;

        }


        public String showIdleTime2(HashMap<String, String>hashy) {
```

```
String html ="<td>";


scenario2WaitingTime.put(Integer.parseInt(hashy.get("op_id")),
Double.parseDouble(hashy.get("dtl_scenario2idletime")));
html +=hashy.get("dtl_scenario2idletime");
html+="</td>";
return html;


}


@Override
public void initialize(HashMap smartyStateMap){
super.initialize(smartyStateMap);
String cid = replaceVarsinString("{c_id}",
arrayGlobals).trim();


Connection conn = null;
PreparedStatement pst = null;
ResultSet rs = null;
System.out.println("c_id==>"+cid);
try {
conn = mysql.getConn();
pst = conn.prepareStatement("select c_totalpieces
from c_cases where c_id=?");
pst.setString(1, cid);
rs = pst.executeQuery();
```

```
                if(rs.next())
                        noOfPieces = rs.getInt(1);
                try {rs.close();}catch(Exception e) {}
                userDefinedCaption = "Total number of items to
produce: "+noOfPieces;
        }catch(Exception e) {
                e.printStackTrace();
        }finally {
                try {rs.close();}catch(Exception e) {}
                try {pst.close();}catch(Exception e) {}
                try {conn.close();}catch(Exception e) {}
        }

    }

    public String modfiythis(HashMap<String ,String> hashy) {
                String HTMLButton="<td align='center'><div
id='op_"+hashy.get("op_id")+"' style='background-
color:red'>"+hashy.get("itemno")+"</div>"
                        + " <input type='hidden'
value='"+hashy.get("op_id")+"' id ='opid_"+hashy.get("op_id")+"'/>"
                                + " <input type='hidden'
value='"+hashy.get("op_t_avg")+"' id
='opt_avg_"+hashy.get("op_id")+"'/> </td>";
                return HTMLButton;
    }
```

```
@Override
public String doUpdate(HttpServletRequest rqs, boolean commit) {
    PreparedStatement pst = null;
    ResultSet rs = null;
    Connection conn = null;
    String cid = replaceVarsinString("{c_id}",
arrayGlobals).trim();
    parseUpdateRqs(rqs);
    int rowsNo = 0;
    if (inputMap_ori.get("smartyhiddenmultieditrowsno")!=null)
        rowsNo =
Integer.parseInt(inputMap_ori.get("smartyhiddenmultieditrowsno")[0]);



    LinkedHashMap<Integer, Integer>scenario1WorkersList =
new LinkedHashMap<Integer, Integer>();
    LinkedHashMap<Integer, Double> opTime = new
LinkedHashMap<Integer, Double>();
    LinkedHashMap<Integer, Double>scenario1WaitingTime =
new LinkedHashMap<Integer, Double>();
    LinkedHashMap<Integer, Double>scenario2WaitingTime =
new LinkedHashMap<Integer, Double>();
    LinkedHashMap<Integer, Integer>scenario2WorkersList =
new LinkedHashMap<Integer, Integer>();



    for (int i = 1; i <= rowsNo ; i++) {
```

```
                    scenario1WorkersList.put(
Integer.parseInt(inputMap_ori.get("smarty_op_id_hidden_smartyrow_"+i
)[0]),


Integer.parseInt(inputMap_ori.get("dtl_scenario1worker_smartyrow_"+i)
[0]));
                    scenario2WorkersList.put(
Integer.parseInt(inputMap_ori.get("smarty_op_id_hidden_smartyrow_"+i
)[0]),


Integer.parseInt(inputMap_ori.get("dtl_scenario2worker_smartyrow_"+i)
[0]));
                    if
(Integer.parseInt(inputMap_ori.get("dtl_scenario1worker_smartyrow_"+i
)[0])<=0) {


                        return "Number of workers must be more than
0";
                    }
                    if
(Integer.parseInt(inputMap_ori.get("dtl_scenario2worker_smartyrow_"+i
)[0])<=0) {


                        return "Number of workers must be more than
0";
                    }
                }
                try {
                    conn = mysql.getConn();
```

```
pst = conn.prepareStatement("select op_t_avg from
kboperation where op_id = ?");
for (int opid : scenario1WorkersList.keySet()) {
    pst.setInt(1, opid);
    rs = pst.executeQuery();
    rs.next();
    opTime.put(opid, rs.getDouble("op_t_avg"));

    try {rs.close();}catch(Exception e) {}
    pst.clearParameters();
}
try {pst.close();}catch(Exception e) {}
//System.out.println(scenario1WorkersList);
//System.out.println(scenario2WorkersList);
LinkedHashMap<Integer, ArrayList<Double>>
opActualTimeFinishA= doCalc (opTime, scenario1WorkersList ,
noOfPieces, scenario1WaitingTime);
LinkedHashMap<Integer, ArrayList<Double>>
opActualTimeFinishB= doCalc (opTime, scenario2WorkersList ,
noOfPieces, scenario2WaitingTime);
```

```
pst = conn.prepareStatement("update c_casesimdtls "
        + "   set dtl_scenario1worker=?
,dtl_scenario2worker= ?, dtl_scenario1idletime=?,
```

```
dtl_scenario2idletime=?, dtl_scenario1finishtime=? ,
dtl_scenario2finishtime=? "

                                + "   where dtl_cid=? and dtl_opid=?");

                    ArrayList<Double> actualTimeA = new
ArrayList<Double>();

                    ArrayList<Double> actualTimeB = new
ArrayList<Double>();

                    for (Integer opid :opTime.keySet()) {

                            actualTimeA =
opActualTimeFinishA.get(opid);

                            actualTimeB = opActualTimeFinishB.get(opid);


                            pst.setInt(1, scenario1WorkersList.get(opid));

                            pst.setInt(2, scenario2WorkersList.get(opid));

                            pst.setDouble(3,
scenario1WaitingTime.get(opid));

                            pst.setDouble(4,
scenario2WaitingTime.get(opid));

                            pst.setDouble(5,
actualTimeA.get(actualTimeA.size()-1));

                            pst.setDouble(6,
actualTimeB.get(actualTimeB.size()-1));

                            pst.setString(7, cid);

                            pst.setInt(8, opid);

                            pst.executeUpdate();

                            pst.clearParameters();

                    }
```

```
                conn.commit();
            }catch(Exception e) {
                e.printStackTrace();
                try {conn.rollback();}catch(Exception eRoll) {}
            }finally {
                try {rs.close();}catch(Exception e) {}
                try {pst.close();}catch(Exception e) {}
                try {conn.close();}catch(Exception e) {}
            }


            return "Calculations Made";
        }


        public LinkedHashMap<Integer, Double>
calcWaitingTime_wrong
                (LinkedHashMap<Integer, ArrayList<Double>>
opActualTimeFinish,
                LinkedHashMap<Integer, Double> opTime){
            LinkedHashMap<Integer, Double> waitingTime = new
LinkedHashMap<Integer, Double>();
            double waitingTimePerProcess =0.0;
            double previousFinishTime =0.0;
            double processTimePerWorker=0.0;
```

```
int i =0;
// this first item there is not waiting process
boolean isFirstItemInBucket = true;
for (Integer opid : opActualTimeFinish.keySet()) {
    i++;
    waitingTimePerProcess =0.0;
    previousFinishTime  =0.0;
    processTimePerWorker = opTime.get(opid);
    isFirstItemInBucket = true;
    for (double finishTime:
opActualTimeFinish.get(opid)) {
        if (isFirstItemInBucket) {
            isFirstItemInBucket = false;
            waitingTimePerProcess = 0.0;
        }else {
            waitingTimePerProcess += (finishTime -
previousFinishTime - processTimePerWorker);
        }
        previousFinishTime = finishTime;
    }

    if (waitingTimePerProcess<0.0)
        waitingTimePerProcess = 0.0;
```

```
                System.out.println("for process "+opid+", waiting
time is "+Math.round(waitingTimePerProcess*100.0)/100.0);

                waitingTime.put(opid,
Math.round(waitingTimePerProcess*100.0)/100.0);

        }

        return waitingTime;

    }


    public LinkedHashMap<Integer, ArrayList<Double>>
doCalc(LinkedHashMap<Integer, Double> opTime ,
LinkedHashMap<Integer, Integer> scenarioWorkersList , int noOfItems,

                LinkedHashMap<Integer,
Double>scenarioWaitingTime ) {


        LinkedHashMap<Integer, Integer> opBucket = new
LinkedHashMap<Integer, Integer>();

        LinkedHashMap<Integer, ArrayList<Double>>
opActualTimeFinish = new LinkedHashMap<Integer,
ArrayList<Double>>();

        LinkedHashMap<Integer, ArrayList<Double>>
opActualTimeStart = new LinkedHashMap<Integer,
ArrayList<Double>>();

        LinkedHashMap<Integer, ArrayList<Integer>> buckets =
new LinkedHashMap<Integer, ArrayList<Integer>>();

        for (int opid : scenarioWorkersList.keySet()) {

            opBucket.put(opid , noOfItems);

            break;

        }
```

```
int itemsPerBucket = 0;
int prevopid = 0;
int workersNo= 0;
ArrayList<Double> finishTime = new ArrayList<Double>();
ArrayList<Double> startTime;
double timeToStartWith = 0;
double timePrevItem =0;
double timePrevOp = 0;
boolean firstItem = true;
int reduceBucketBy =0;
ArrayList<Integer> bucketPerOp  = new ArrayList<Integer>();
int i =0;

for (int opid: opBucket.keySet()) {
    itemsPerBucket = opBucket.get(opid);
    finishTime = new ArrayList<Double>(); // to hold the finish time for each op
    workersNo = scenarioWorkersList.get(opid);
    firstItem = true;
    timeToStartWith = 0;
    i = 0;
    bucketPerOp = new ArrayList<Integer>();
    while (itemsPerBucket>0) {
```

```
                    if (prevopid ==0) {


                        if (firstItem) {
                            finishTime.add(opTime.get(opid));
                        }else {
                            finishTime.add(opTime.get(opid)+
(finishTime.get(finishTime.size()-1)));
                        }


                        if
(itemsPerBucket<scenarioWorkersList.get(opid))
                            bucketPerOp.add(itemsPerBucket);
                        else

        bucketPerOp.add(scenarioWorkersList.get(opid));



                            //System.out.println(finishTime);


                    }
                    reduceBucketBy = bucketPerOp.get(i);

        //System.out.println("reduceBucketBy==>"+reduceBucketBy);
                    firstItem = false;
                    //no decrease the items in bucket
```

```
                        itemsPerBucket = itemsPerBucket-
reduceBucketBy;

        //System.out.println("itemsPerBucket==>"+itemsPerBucket);
                        i++;


                    }
                    buckets.put(opid,bucketPerOp);
                    opActualTimeFinish.put(opid, finishTime);
                    scenarioWaitingTime.put(opid, 0.0);
                    break;


                }


                boolean firstop = true;
                ArrayList<Integer> newBucket = new ArrayList<Integer>();
                ArrayList<Double> newBucketFinishTime  = new
ArrayList<Double>();
                int prevNoOfItemsInBucket=0 , newNoOfItemsInBucket = 0
, remainingItems =0;
                double prevtimeToFinishItems=0 , newTimeToFinishBucket
=0 ;
                int noOfSpareWorkers =0;
                int UnprocessedItems = noOfItems;
                int processedItems = 0;
                boolean firstBucket= true;
```

```
LinkedList<HashMap<Integer, Double>> freeWorkers =
new LinkedList<HashMap<Integer, Double>>();

LinkedList<HashMap<Integer, Double>> busyWorkers =
new LinkedList<HashMap<Integer, Double>>();

ArrayList<Double> newfinishTime = new
ArrayList<Double>();

int bucketid = 0;

for (int opid: scenarioWorkersList.keySet()) {

if (firstop) {

firstop = false;

continue;

}

UnprocessedItems = noOfItems;

noOfSpareWorkers = scenarioWorkersList.get(opid);

newBucket = new ArrayList<Integer>();

newBucketFinishTime = new ArrayList<Double>();

freeWorkers =  new LinkedList<HashMap<Integer,
Double>>();


for (int w = 1 ; w <= noOfSpareWorkers ; w++) {

HashMap<Integer, Double> worker = new
HashMap<Integer, Double>();

worker.put(w, 0.0);

freeWorkers.add(worker);

}

busyWorkers = new LinkedList<HashMap<Integer,
Double>>();
```

```
                    bucketid = 0;
                    double waitingTime = 0;
                    for (int index = 0 ; index <bucketPerOp.size(); index
++) {
                        if (UnprocessedItems<=0)
                            break;

                        prevNoOfItemsInBucket =
bucketPerOp.get(index);
                        prevtimeToFinishItems =
finishTime.get(index);

                        while (prevNoOfItemsInBucket>0) {
                            processedItems =0;

        System.out.println("prevNoOfItemsInBucket=>"+prevNoOfItemsI
nBucket+", Free Workers =>"+freeWorkers);
                            System.out.println("Free Workers
=>"+freeWorkers);
                            System.out.println("Busy Workers
=>"+busyWorkers);
                            if (freeWorkers.size()>0) {
                                for (HashMap<Integer, Double>
worker : freeWorkers) {
                                    for (int workerid :
worker.keySet()) {
```

```
                                        double workerFreeAt
= worker.get(workerid);

        HashMap<Integer,Double> busyWorker = new
HashMap<Integer,Double>();
                                        if
(workerFreeAt<=prevtimeToFinishItems) {
                                            if
(workerFreeAt>0)

        waitingTime += (prevtimeToFinishItems-workerFreeAt);

        busyWorker.put(workerid,
prevtimeToFinishItems+opTime.get(opid));
                                        }else {

        busyWorker.put(workerid, workerFreeAt+opTime.get(opid));
                                        }



        busyWorkers.add(busyWorker);

        prevNoOfItemsInBucket--;
                                        processedItems++;
                                        if
(prevNoOfItemsInBucket == 0) {
                                            break;
                                        }
                                    }
```

```
                                        if (prevNoOfItemsInBucket
== 0)
                                            break;
                                    }
                                    //remove free workers
                                    int freeWorkerIndex = 0;
                                    boolean found = false;
                                    for (HashMap<Integer, Double>
busyworker : busyWorkers) {
                                        for (int busyworkerid :
busyworker.keySet()) {
                                            freeWorkerIndex = 0;
                                            found = false;
                                            for
(HashMap<Integer, Double> freeWorker : freeWorkers) {
        freeWorkerIndex = freeWorkers.indexOf(freeWorker);
                                                for (int
freeWorkerid : freeWorker.keySet()) {
                                                    if
(freeWorkerid == busyworkerid) {
        found = true;
        break;
                                                    }
                                                }
                                                if (found)
```

```
                                                    break;
                                                }
                                                if (found)

        freeWorkers.remove(freeWorkerIndex);
                                                }
                                            }
                                        }else { // if we don't have free workers
then get them from the busy workers
                                            for (HashMap<Integer, Double>
worker : busyWorkers) {
                                                freeWorkers.add(worker);
                                            }
                                            busyWorkers.clear();
                                        }
                                        //System.out.println("Free Workers
=>"+freeWorkers);
                                        //System.out.println("Busy Workers
=>"+busyWorkers);

                                        if (processedItems>0) {
                                            newBucket.add(processedItems);
                                            for (Double bucketWorker:
busyWorkers.getLast().values()) {

        newBucketFinishTime.add(bucketWorker);
                                            }
```

```
                    }
                }
                //System.out.println("======ENd OF BUCKET====================");
            }
            System.out.println("======ENd of op=====waitingTime===>"+waitingTime);
            bucketPerOp = newBucket;
            finishTime = newBucketFinishTime;
            //System.out.println("opid==>"+opid+" no of buckets="+bucketPerOp.size()+", bucket per op ==>"+bucketPerOp+", finishTime=>"+finishTime);

        //System.out.println("============================opid finished=============================================="+opid);
            opActualTimeFinish.put(opid,finishTime);
            scenarioWaitingTime.put(opid, Math.round(waitingTime*100.0)/100.0);
        }
        System.out.println("End of Simulation==============>"+opActualTimeFinish);
        return opActualTimeFinish;
    }

    public int getTotNoOfWorkers1() {
        return totNoOfWorkers1;
```

```
}

public void setTotNoOfWorkers1(int totNoOfWorkers1) {
        this.totNoOfWorkers1 = totNoOfWorkers1;
}

public int getTotNoOfWorkers2() {
        return totNoOfWorkers2;
}

public void setTotNoOfWorkers2(int totNoOfWorkers2) {
        this.totNoOfWorkers2 = totNoOfWorkers2;
}

public double getFinishTime1() {
        return finishTime1;
}

public void setFinishTime1(double finishTime1) {
        this.finishTime1 = finishTime1;
}

public double getFinishTime2() {
        return finishTime2;
```

```
	}

	public void setFinishTime2(double finishTime2) {
		this.finishTime2 = finishTime2;
	}
}
```

Appendix C

Published Papers

Two papers were published during conducting this research. They are:

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

Effective Way to Estimate the Standard Minute Value (SMV) of A U3 Shirt by Using Time Study Technique

Hanan O.A [1], Seedahmed A.I.[2]

[1]Department of Textile Engineering, [2]Department of Plastic Engineering, Sudan University of Science and Technology, Sudan

Abstract

There is no doubt that sewing section in an apparel industry is the most important department that plays a vital role in the whole firm .Time study is a method of measuring work for recording the time of performing a certain specific task or its elements carried out under specified conditions. To improve the existing situation of this section and increasing productivity, time study is a very effective technique. This study is based on calculations of standard minute value (SMV) of a U3 long sleeve shirt. For conducting time study, a traditional stop watch was used for measuring time of each operation. The U3 shirt (Uniform number 3) manufacturing has 43 operations, for each one 10 measurements were taken for each task and operator working on the line. Then the average of each task is calculated and the final SMV as well.

Paper citation:

Hanan, O. A., & Seedahmed, A. I. (2018). **Effective Way to Estimate the Standard Minute Value (SMV) of A U3 Shirt by Using Time Study Technique,** *INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY, 7*(7), 179-184. ***Impact factor 5.164***

International Journal of Scientific Engineering and Science

Garment Assembly Line Balancing Using Modeling and Simulation

Hanan O.A [1], Seedahmed A.I.[2]

[1]Department of Textile Engineering, [2]Department of Plastic Engineering,
Sudan University of Science and Technology, Sudan

Abstract

Assembly line balancing of garment industry became an important area for research. The objective of this study is to develop a simulation model which represents real production process scenarios based on balancing U3 Shirt assembly line in Sur Military Clothing Factory that helps to decrease the waiting time, finish time, and increase the efficiency of the line in addition to generating different alternative scenarios to utilize the assembly line. The methodology adopted includes firstly calculation of cycle time of U3 Shirt assembly line process by using time study, secondly, to set up a model and simulation of the line which was achieved by using different tools and technologies: MySQL Data Base Management System (DBMS), Java language, Hyper Text Markup Language (HTML), Cascades Style Sheets (CSS) and SMARTY J. Several scenarios were created to enhance the system. Three scenarios were proposed and the best one had a result of a finish time decremented to be 24113s, the second scenario was 26603s while the third gave 26622s compared to 29616 seconds for the real scenario finish time. The efficiency of the line is increased to 19%, 10%, and 10% for the three scenarios respectively. The cost of the three alternative scenarios was increased by 9%, 15% and 3% from the real system cost consecutively.

The study concluded that the real scenario case could be enhanced to be more effective by using the best scenario result that was created by using scenario 1 which saves more time, and had more efficiency.

Paper Citation:

Hanan, O. A., & Seedahmed, A. I. (2018). Garment Assembly Line Balancing Using Modeling and Simulation. *International Journal of Scientific Engineering and Science, Volume 3, Issue 3, pp. 4-7, 2019.*