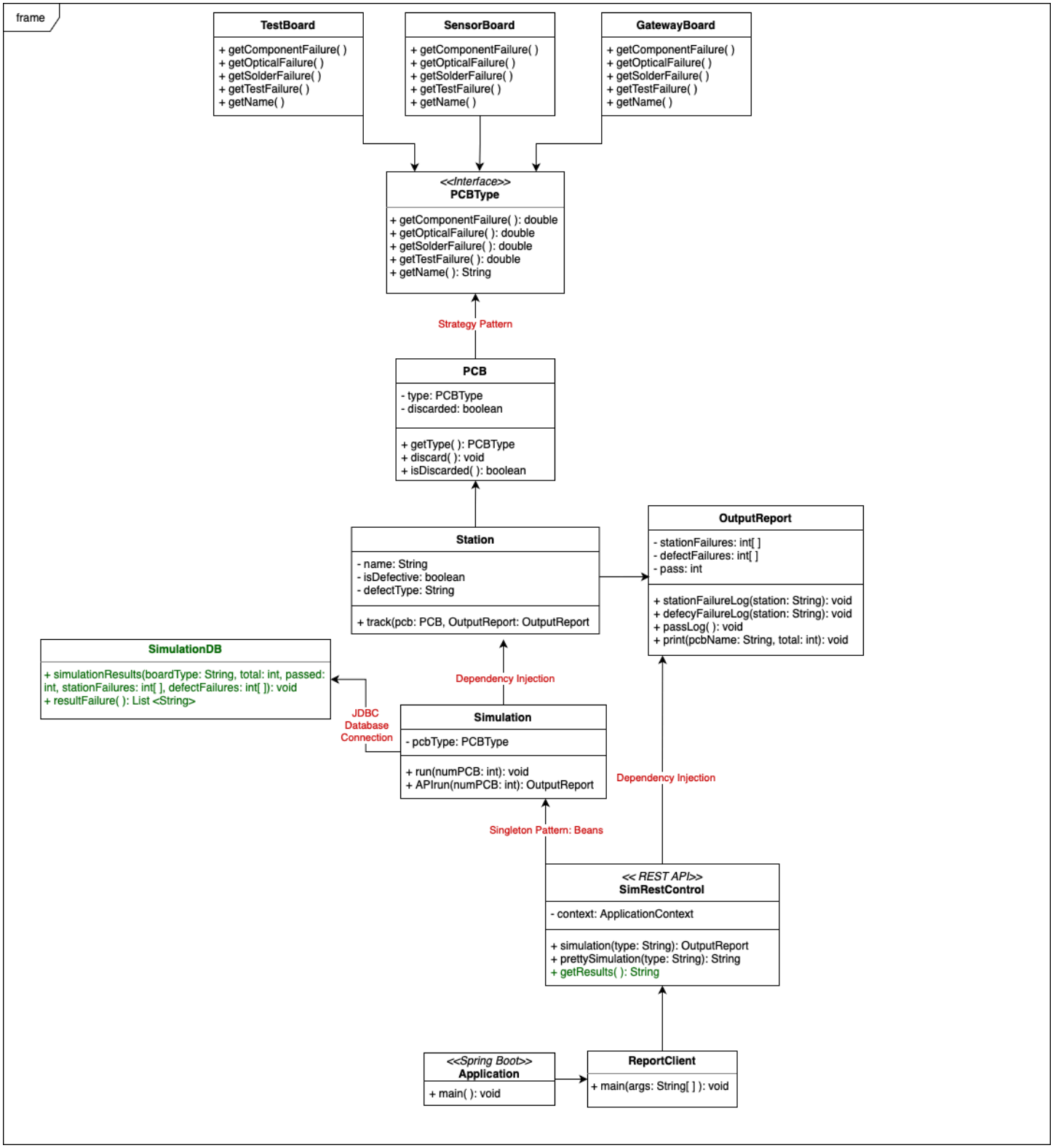
UML Class Diagram:



Java Code with Strategy Pattern, Singleton Pattern, Dl, API, AND SQLite DB:

Strategy Pattern:

```
/*************************************************/
//////          PCBType.java          /////
/*************************************************/

package pcb;

public interface PCBType {

   /*************************************************/
   //////         Strategy Pattern          //////
   //////        instantiate different       //////
   //////    failure detection strategies    //////
   /*************************************************/

   double getComponentFailure();
   double getOpticalFailure();
   double getSolderFailure();
   double getTestFailure();
   String getName();

}


/*************************************************/
//////          TestBoard.java          /////
/*************************************************/
package pcb;

public class TestBoard implements PCBType {
   /*************************************************/
   //////         Strategy Pattern          /////
   /*************************************************/

   @Override
   public double getComponentFailure() {
      return 0.05;
   }

   @Override
   public double getOpticalFailure() {
      return 0.10;
   }
```

```
    @Override
    public double getSolderFailure() {
        return 0.05;
    }

    @Override
    public double getTestFailure() {
        return 0.10;
    }

    @Override
    public String getName() {
        return "Test Board";
    }
}


/***********************************************/
//////          SensorBoard.java          /////
/***********************************************/
package pcb;

public class SensorBoard implements PCBType {
    /***********************************************/
    //////          Strategy Pattern          /////
    /***********************************************/
    @Override
    public double getComponentFailure() {
        return 0.002;
    }

    @Override
    public double getOpticalFailure() {
        return 0.002;
    }

    @Override
    public double getSolderFailure() {
        return 0.004;
    }

    @Override
    public double getTestFailure() {
        return 0.004;
    }

    @Override
    public String getName() {
```

```
        return "Sensor Board";
    }
}


/*********************************************/
//////          GatewayBoard.java          /////
/*********************************************/
package pcb;

public class GatewayBoard implements PCBType {
    /*********************************************/
    //////          Strategy Pattern          /////
    /*********************************************/
    @Override
    public double getComponentFailure() {
        return 0.004;
    }

    @Override
    public double getOpticalFailure() {
        return 0.004;
    }

    @Override
    public double getSolderFailure() {
        return 0.008;
    }

    @Override
    public double getTestFailure() {
        return 0.008;
    }

    @Override
    public String getName() {
        return "Gateway Board";
    }
}



/*********************************************/
//////              PCB.java              /////
/*********************************************/
package pcb;

public class PCB {
```

```
    private boolean discarded = false;
    /*************************************************/
    //////          Strategy Pattern          /////
    /*************************************************/
    private final PCBType type;

    public PCB(PCBType type) {
        this.type = type;
    }

    public void discard() {
        this.discarded = true;
    }

    public boolean isDiscarded() {
        return discarded;
    }

    public PCBType getType() {
        return type;
    }
}


/*************************************************/
//////          Station.java          /////
/*************************************************/
package pcb;

public class Station {

    private final String name;
    private final boolean isDefective;
    private final String defectType;
    private static final double FailureChance = 0.002;

    public Station(String name, boolean isDefective, String defectType){
        this.name = name;
        this.isDefective = isDefective;
        this.defectType = defectType;
    }

    public void track(PCB pcb, OutputReport OutputReport){
        if(Math.random() < FailureChance){
            pcb.discard();
            OutputReport.stationFailureLog(name);
            return;
        }
        /*************************************************/
```

```
        //////          Strategy Pattern          /////
        /*********************************************/
        if(isDefective && defectType != null){
            double failureRate = 0;
            switch(defectType){
                case "Place Components":
                    failureRate = pcb.getType().getComponentFailure();
                    break;
                case "Optical Inspection":
                    failureRate = pcb.getType().getOpticalFailure();
                    break;
                case "Hand Soldering/Assembly":
                    failureRate = pcb.getType().getSolderFailure();
                    break;
                case "Test (ICT or Flying Probe)":
                    failureRate = pcb.getType().getTestFailure();
                    break;
            }
            if(Math.random() < failureRate){
                pcb.discard();
                OutputReport.defectFailureLog(name);
            }
        }
    }

    public String getName(){
        return name;
    }


}

package pcb;
/*********************************************/
//////        Application.java          /////
/*********************************************/
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.ImportResource;

@SpringBootApplication
@ImportResource("classpath:beans.xml")
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
package pcb;
```

```
/*********************************************/
//////          Views.java          /////
/*********************************************/
public class Views{
   public interface Public{}
}
plugins {
  id 'java'
  id 'org.springframework.boot' version '2.7.0'
  id 'io.spring.dependency-management' version '1.0.11.RELEASE'
  id 'application'
}
```

Dependency Injection:

```
package pcb;
/*********************************************/
//////          Simulation.java          /////
/*********************************************/
import java.util.List;

public class Simulation {
   private final PCBType pcbType;
   private final OutputReport report;
   /*********************************************/
   //////        Dependency Injection        /////
   /*********************************************/
   public Simulation(PCBType pcbType) {
      this.pcbType = pcbType;
      this.report = new OutputReport();
   }

   public void run(int count) {
      List<Station> stations = List.of(
         new Station("Apply Solder Paste", false, null),
         new Station("Place components", true, "Place Components"),
         new Station("Reflow Solder", false, null),
         new Station("Optical Inspection", true, "Optical Inspection"),
         new Station("Hand Soldering/Assembly", true, "Hand Soldering/Assembly"),
         new Station("Cleaning", false, null),
         new Station("Depanelization", false, null),
         new Station("Test (ICT or Flying Probe)", true, "Test (ICT or Flying Probe)")
      );

      for (int i = 0; i < count; i++) {
         PCB pcb = new PCB(pcbType);
         for (Station station : stations) {
            station.track(pcb, report);
            if (pcb.isDiscarded()) {
```

```
                break;
            }
        }
        if (!pcb.isDiscarded()) {
            report.passLog();
        }
    }
    report.print(pcbType.getName(), count);
}
/*********************************************/
//////      Simulation Runner for API       /////
//////  No longer prints, returns report  //////
/*********************************************/
public OutputReport APIrun(int count){
        List<Station> stations = List.of(
        new Station("Apply Solder Paste", false, null),
        new Station("Place components", true, "Place Components"),
        new Station("Reflow Solder", false, null),
        new Station("Optical Inspection", true, "Optical Inspection"),
        new Station("Hand Soldering/Assembly", true, "Hand Soldering/Assembly"),
        new Station("Cleaning", false, null),
        new Station("Depanelization", false, null),
        new Station("Test (ICT or Flying Probe)", true, "Test (ICT or Flying Probe)")
    );
    OutputReport report = new OutputReport();

    for (int i = 0; i < count; i++) {
        PCB pcb = new PCB(pcbType);
        for (Station station : stations) {
            station.track(pcb, report);
            if (pcb.isDiscarded()) {
                break;
            }
        }
        if (!pcb.isDiscarded()) {
            report.passLog();
        }
    }
    /*****************************************************/
    ////// Database Integration to Fetch Data from API /////
    /*****************************************************/
    SimulationDB.simulationResults(
    pcbType.getName(),
    count,
    report.getPass(),
    report.getStationFailures(),
    report.getDefectFailures()
    );
    return report;
```

```
    }
}
```

REST API's and Singleton Pattern with DB:

```
package pcb;
import java.util.List;

// https://spring.io/guides/gs/spring-boot
// https://spring.io/guides/tutorials/rest
/*********************************************/
//////        SimRestControl.java         /////
/*********************************************/
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationContext;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import com.fasterxml.jackson.annotation.JsonView;

/*********************************************/
//////              REST API              /////
/*********************************************/
@RestController
public class SimRestControl{
   @Autowired
   /*********************************************/
   //////        Dependency Injection        /////
   /*********************************************/
   private ApplicationContext context;
   /*********************************************/
   //////         JSON REST Endpoint         /////
   /*********************************************/
   @GetMapping("/simulation")
   @JsonView(Views.Public.class)
   public OutputReport simulation(@RequestParam("type") String type){
      Simulation sim;
      /*********************************************/
      //////         Singleton Pattern           /////
      ////// Uses my Spring Beans as Singletons  //////
      /*********************************************/
      switch(type.toLowerCase()){
         case "test":
            sim = (Simulation) context.getBean("testSim");
            break;
         case "sensor":
            sim = (Simulation) context.getBean("sensorSim");
            break;
```

```
            case "gateway":
                sim = (Simulation) context.getBean("gatewaySim");
                break;
            default:
                throw new IllegalArgumentException("Invalid");
        }
        return sim.APIrun(1000);
    }
    /************************************************/
    //////   Jackson Formatted REST Endpoint    /////
    /************************************************/
    @GetMapping("/pretty-simulation")
    public String prettySimulation(@RequestParam("type") String type){
        Simulation sim;
        /************************************************/
        //////          Singleton Pattern           /////
        ////// Uses my Spring Beans as Singletons  //////
        /************************************************/
        switch(type.toLowerCase()){
            case "test":
                sim = (Simulation) context.getBean("testSim");
                break;
            case "sensor":
                sim = (Simulation) context.getBean("sensorSim");
                break;
            case "gateway":
                sim = (Simulation) context.getBean("gatewaySim");
                break;
            default:
                throw new IllegalArgumentException("Invalid");
        }
        OutputReport report = sim.APIrun(1000);
        return report.prettyPrint(type + " Board", 1000);
    }
    /************************************************************/
    ////// Database Integration to Fetch Results from API /////
    /************************************************************/
    @GetMapping("/results")
    public String getResults() {
        List<String> resultStrings = SimulationDB.resultFailure();
        return String.join("<br/><br/>", resultStrings).replace("n", "<br/>");
    }

}
```

Simulation SQLite Database with JDBC Driver:

```
package pcb;
// https://www.tutorialspoint.com/jdbc/jdbc-exceptions.htm?utm_source=chatgpt.com
// https://www.w3schools.com/jsref/jsref_tostring_array.asp
// https://www.sqlitetutorial.net/sqlite-java/sqlite-jdbc-driver/
```

```java
/*********************************************/
//////        SimulationDB.java          /////
/*********************************************/
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class SimulationDB {
    private static final String URL = "jdbc:sqlite:simulation.db";
    /*************************************************/
    ////// SQLite JDBC Driver to Retrieve Results /////
    /*************************************************/
    public static void simulationResults(String boardType, int total, int passed, int[] stationFailures, int[]
defectFailures) {
        String sql = "INSERT INTO simulation_runs (board_type, total_pcb, passed_pcb, station_failures,
defect_failures) VALUES (?, ?, ?, ?, ?)";
        try(Connection connection = DriverManager.getConnection(URL);
            PreparedStatement statement = connection.prepareStatement(sql)){
                statement.setString(1, boardType);
                statement.setInt(2, total);
                statement.setInt(3, passed);
                statement.setString(4, arrayToString(stationFailures));
                statement.setString(5, arrayToString(defectFailures));
                statement.executeUpdate();
            } catch(SQLException e){
                e.printStackTrace();
            }
    }
    /*************************************************/
    ////// Text Conversion for Result Preparation /////
    /*************************************************/
    public static List<String> resultFailure(){
        List<String> result = new ArrayList<>();
        String sql = "SELECT * FROM simulation_runs ORDER BY id DESC LIMIT 3";
        try(Connection connection = DriverManager.getConnection(URL);
            Statement statement = connection.createStatement();
            ResultSet results = statement.executeQuery(sql)){
                while(results.next()){
                    StringBuilder queries = new StringBuilder();
                    queries.append("Board Type: ").append(results.getString("board_type")).append("n");
                    queries.append("Total PCBs: ").append(results.getInt("total_pcb")).append("n");
                    queries.append("Passed PCBs: ").append(results.getInt("passed_pcb")).append("n");
                    queries.append("Station Failures: ").append(results.getString("station_failures")).append("n");
                    queries.append("Defect Failures: ").append(results.getString("defect_failures")).append("n");
                    result.add(queries.toString());
                }
            } catch(SQLException e){
                e.printStackTrace();
            }
```

```
        return result;
    }
    /*************************************************/
    ////// Simple Method for String Fmatting  /////
    /*************************************************/
    private static String arrayToString(int[] arr){
        StringBuilder queries = new StringBuilder();
        for(int i = 0; i< arr.length; i++){
            queries.append(arr[i]);
            if(i!= arr.length-1){
                queries.append(", ");
            }
        }
        return queries.toString();
    }
}
```

Report Client with DB:

```
package pcb;
/*************************************************/
//////        ReportClient.java          /////
/*************************************************/
import org.springframework.web.client.RestTemplate;

public class ReportClient {
    public static void main(String[] args) {
        String[] pcbTypes = {"test", "sensor", "gateway"};
        RestTemplate restTemplate = new RestTemplate();

        for (String type : pcbTypes) {
            String url = "http://localhost:8080/simulation?type=" + type;
            OutputReport report = restTemplate.getForObject(url, OutputReport.class);

            String reportText = report.prettyPrint(type + " Board", 1000);

            System.out.println(reportText.replace("<br/>", "n"));
            System.out.println("n");

            /*************************************************/
            //////   Integrate Database to REST API   /////
            /*************************************************/
            String urlR = "http://localhost:8080/results";
            String resultText = restTemplate.getForObject(urlR, String.class);
            System.out.println(resultText);
        }
    }
}
plugins {
```

```
    id 'java'
    id 'org.springframework.boot' version '2.7.0'
    id 'io.spring.dependency-management' version '1.0.11.RELEASE'
    id 'application'
}


sourceSets {
    main {
        resources {
            srcDirs = ['.']
            includes = ['beans.xml']
        }
    }
}


repositories {
    mavenCentral()
}


dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-web:2.7.0'
    implementation 'com.fasterxml.jackson.core:jackson-databind:2.13.0'
    implementation 'org.xerial:sqlite-jdbc:3.36.0.3'
}


java {
    toolchain {
        languageVersion = JavaLanguageVersion.of(17)
    }
}


application {
    mainClass = 'pcb.ReportClient'
}


bootJar {
    archiveFileName = 'pcb.jar'
    mainClass = 'pcb.Application'
}


jar {
    enabled = true
```

```
  archiveClassifier = ''
}


task reportClientJar(type: Jar) {
  archiveBaseName = "pcb-report"
  archiveClassifier = ""
  archiveVersion = ""
  destinationDirectory = file("$buildDir/libs")


  manifest {
    attributes 'Main-Class': 'pcb.ReportClient'
  }


  duplicatesStrategy = DuplicatesStrategy.EXCLUDE


  from {
    configurations.runtimeClasspath.collect { it.isDirectory() ? it : zipTree(it) }
  }


  from sourceSets.main.output.classesDirs
}


build.dependsOn reportClientJar
<!-- // https://www.geeksforgeeks.org/advance-java/spring-dependency-injection-with-example/
// https://www.geeksforgeeks.org/java/spring-applicationcontext/ -->
<!-- /*********************************************/
//////            beans.xml            /////
/*********************************************/ -->
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    https://www.springframework.org/schema/beans/spring-beans.xsd">

  <bean id="testBoard" class="pcb.TestBoard" />
  <bean id="sensorBoard" class="pcb.SensorBoard" />
  <bean id="gatewayBoard" class="pcb.GatewayBoard" />
  <!-- /*********************************************/
  //////      Dependency Injection      /////
  /*********************************************/ -->
  <bean id="testSim" class="pcb.Simulation">
    <constructor-arg ref="testBoard" />
  </bean>
```

```
    <bean id="sensorSim" class="pcb.Simulation">
        <constructor-arg ref="sensorBoard" />
    </bean>

    <bean id="gatewaySim" class="pcb.Simulation">
        <constructor-arg ref="gatewayBoard" />
    </bean>
</beans>
```

Results:

Gateway Board DB and Output Results

Board Type: Gateway Board
Total PCBs: 1000
Passed PCBs: 948
Station Failures: 2, 8, 4, 1, 3, 1, 2, 3
Defect Failures: 0, 5, 0, 3, 12, 0, 0, 8

PCB type: Gateway Board
PCBs run: 1000
Station Failures
Apply Solder Paste: 2
Place components: 8
Reflow Solder: 4
Optical Inspection: 1
Hand Soldering/Assembly: 3
Cleaning: 1
Depanelization: 2
Test (ICT or Flying Probe): 3

PCB Defect Failures
Apply Solder Paste: 0
Place components: 5
Reflow Solder: 0
Optical Inspection: 3
Hand Soldering/Assembly: 12
Cleaning: 0
Depanelization: 0
Test (ICT or Flying Probe): 8

Final Results
Total failed PCBs: 52
Total PCBs produced:948

Sensor Board DB and Output Results

Board Type: Sensor Board
Total PCBs: 1000
Passed PCBs: 978
Station Failures: 0, 3, 1, 4, 2, 2, 0, 1
Defect Failures: 0, 3, 0, 2, 1, 0, 0, 3

PCB type: Sensor Board
PCBs run: 1000
Station Failures
Apply Solder Paste: 0
Place components: 3
Reflow Solder: 1
Optical Inspection: 4
Hand Soldering/Assembly: 2
Cleaning: 2
Depanelization: 0
Test (ICT or Flying Probe): 1

PCB Defect Failures
Apply Solder Paste: 0
Place components: 3
Reflow Solder: 0
Optical Inspection: 2
Hand Soldering/Assembly: 1
Cleaning: 0
Depanelization: 0
Test (ICT or Flying Probe): 3

Final Results
Total failed PCBs: 22
Total PCBs produced:978

Test Board DB and Output Results

Board Type: Test Board
Total PCBs: 1000
Passed PCBs: 734
Station Failures: 1, 0, 3, 0, 4, 0, 1, 1
Defect Failures: 0, 55, 0, 87, 40, 0, 0, 74

PCB type: Test Board
PCBs run: 1000
Station Failures
Apply Solder Paste: 1
Place components: 0
Reflow Solder: 3
Optical Inspection: 0
Hand Soldering/Assembly: 4
Cleaning: 0
Depanelization: 1
Test (ICT or Flying Probe): 1

PCB Defect Failures
Apply Solder Paste: 0
Place components: 55
Reflow Solder: 0
Optical Inspection: 87
Hand Soldering/Assembly: 40
Cleaning: 0
Depanelization: 0
Test (ICT or Flying Probe): 74

Final Results
Total failed PCBs: 266
Total PCBs produced:734