SENG311- Software Quality Assurance

Homework 2 Report

Basis-Path & Mutation Testing Report for Proximity Radar

Team Members:

Tuna BEKİŞ

Şükrü Enes TUĞAÇ

Introduction

The purpose of this homework is to apply white-box testing techniques—specifically basis-path testing and mutation testing—to selected methods from the *Proximity Radar* software system.

The *Proximity Radar* project simulates how a radar device detects objects (targets) within a certain range and classifies them based on their signal-to-noise ratio (SNR) and relative distance.

Selected Methods

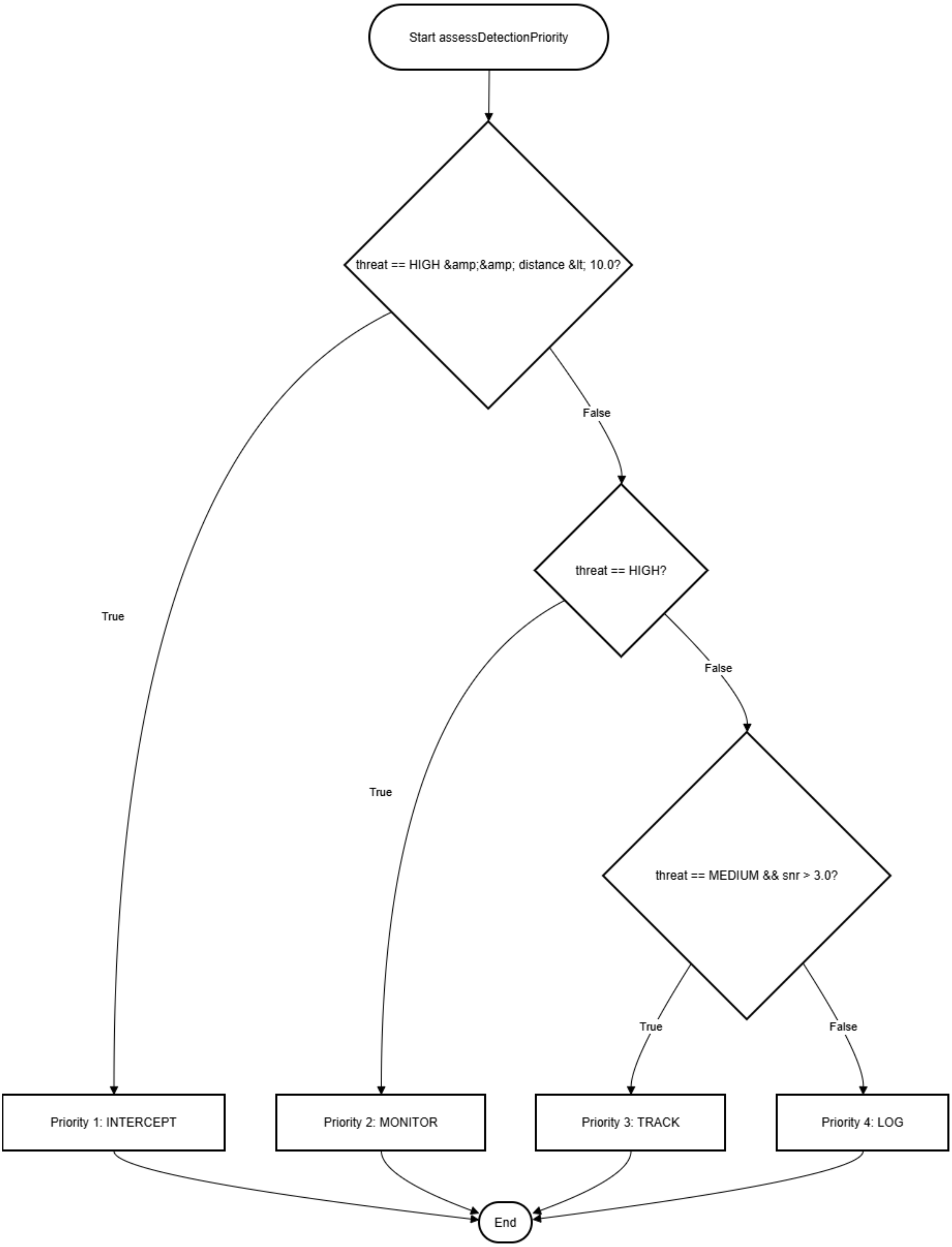
Two functions were selected because they exhibit decision logic and measurable complexity:

Method	**Description**	**Reason**
Detection.assessDetectionPriority()	Determines detection priority based on threat level, distance, and SNR .	Contains three independent decisions → V(G)=4.
Radar.scan()	Iterates through targets, calculates distance/SNR, and assigns threat levels .	Includes loop + nested ifs → V(G)≈6.

Both methods were chosen because they include multiple decisions and nested conditions, which make them suitable for basis-path and mutation analysis.

Flowcharts and Basis Paths

Flowchart 1 – Detection.assessDetectionPriority()



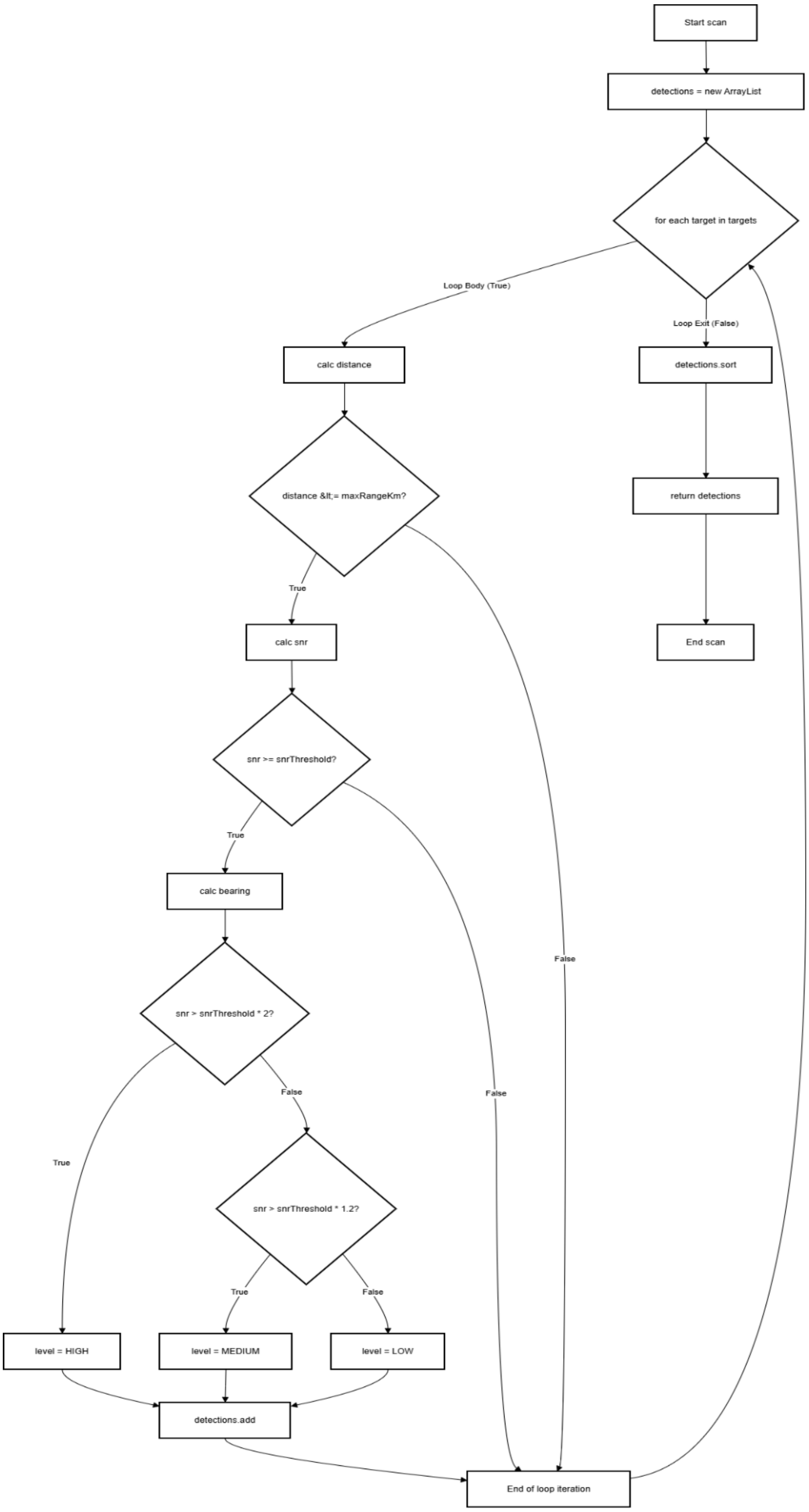
The method starts by checking if the threat level is HIGH and the distance is below 10 km.
If true, it assigns **Priority 1: INTERCEPT**. Otherwise, it checks whether the threat is still HIGH, assigning **Priority 2: MONITOR**.
If not, it tests whether the threat is MEDIUM and snr > 3.0.
Depending on this condition, it returns either **Priority 3: TRACK** or **Priority 4: LOG**.

There are three independent decision nodes, which means the cyclomatic complexity is **V(G)=3 + 1 = 4**, resulting in four basis paths.

Basis Path Table

Path	Condition	Expected Output
BP1	**Threat = HIGH ∧ Distance < 10**	**PRIORITY_1_INTERCEPT**
BP2	**Threat = HIGH ∧ Distance ≥ 10**	**PRIORITY_2_MONITOR**
BP3	**Threat = MEDIUM ∧ SNR > 3**	**PRIORITY_3_TRACK**
BP4	**Otherwise**	**PRIORITY_4_LOG**

Flowchart 2 – Radar.scan()



The scan() method loops through each target, calculates its distance and SNR, and assigns a threat level depending on the computed value.
If the distance is greater than maxRangeKm, the target is ignored.
Otherwise, the method compares the SNR with the radar's threshold and classifies the target as **LOW**, **MEDIUM**, or **HIGH** threat.

The loop structure and nested decisions yield a cyclomatic complexity of **V(G)=6**, which provides six independent paths.

Path Summary

Path	Description
P1	**No targets in list → return empty detections**
P2	**Target out of range → skipped**
P3	**Target in range but below SNR threshold**
P4	**Meets threshold → LOW threat**
P5	**SNR ≥ 1.2×threshold → MEDIUM threat**
P6	**SNR ≥ 2×threshold → HIGH threat**

White-Box Testing

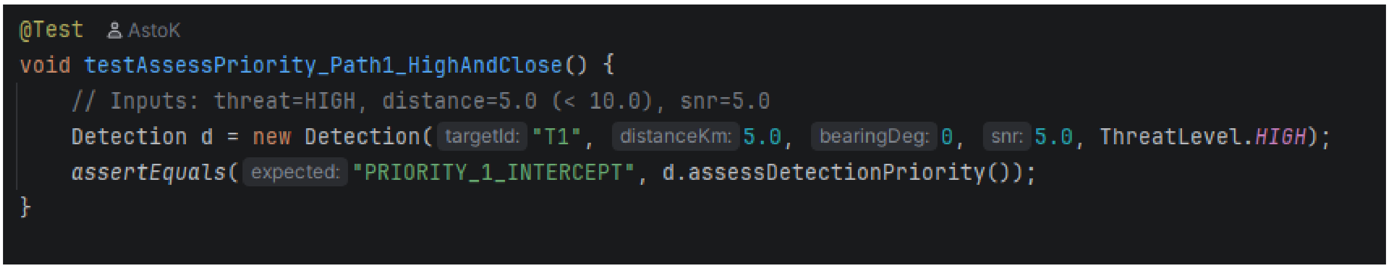
Testing Approach

White-box tests were implemented in **JUnit 5**.
The tests focus on:

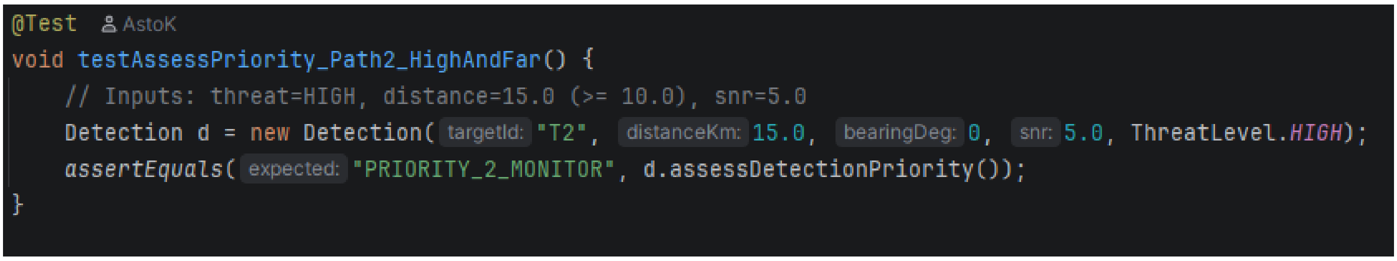
- Boundary conditions (distance = 10, snr = 3),
- Positive and negative inputs,
- Independent paths for each method.

Each test method name directly maps to one basis path (e.g., test_BP1_HighAndClose).

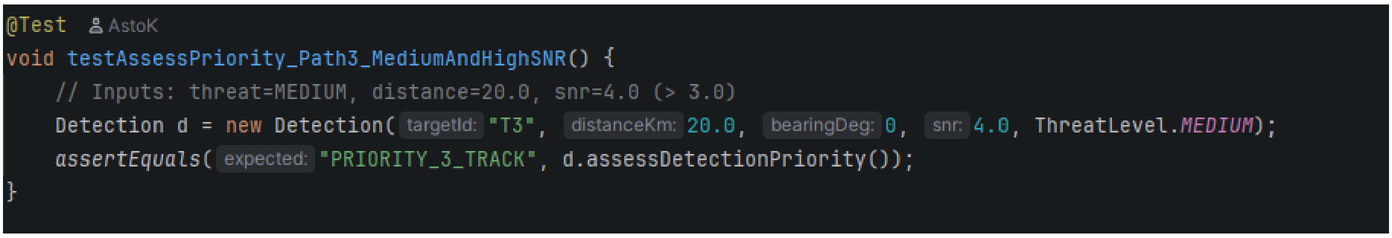
Sample Test Code



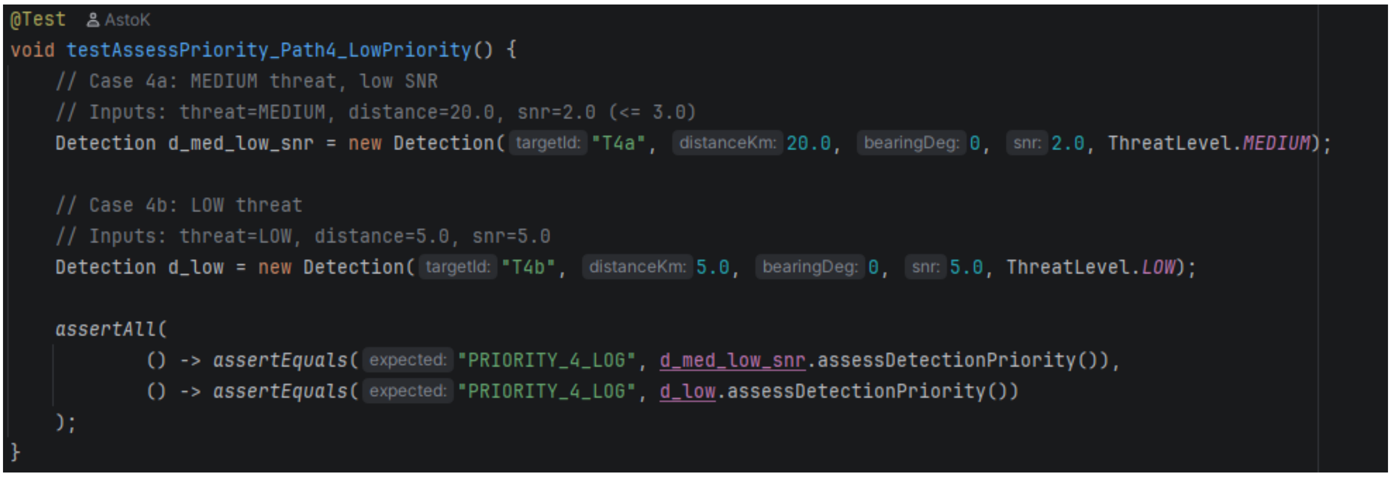
```
@Test  AstoK
void testAssessPriority_Path1_HighAndClose() {
    // Inputs: threat=HIGH, distance=5.0 (< 10.0), snr=5.0
    Detection d = new Detection( targetId: "T1",  distanceKm: 5.0,  bearingDeg: 0,  snr: 5.0,  ThreatLevel.HIGH);
    assertEquals( expected: "PRIORITY_1_INTERCEPT", d.assessDetectionPriority());
}
```



```
@Test  AstoK
void testAssessPriority_Path2_HighAndFar() {
    // Inputs: threat=HIGH, distance=15.0 (>= 10.0), snr=5.0
    Detection d = new Detection( targetId: "T2",  distanceKm: 15.0,  bearingDeg: 0,  snr: 5.0,  ThreatLevel.HIGH);
    assertEquals( expected: "PRIORITY_2_MONITOR", d.assessDetectionPriority());
}
```



```
@Test  AstoK
void testAssessPriority_Path3_MediumAndHighSNR() {
    // Inputs: threat=MEDIUM, distance=20.0, snr=4.0 (> 3.0)
    Detection d = new Detection( targetId: "T3",  distanceKm: 20.0,  bearingDeg: 0,  snr: 4.0,  ThreatLevel.MEDIUM);
    assertEquals( expected: "PRIORITY_3_TRACK", d.assessDetectionPriority());
}
```

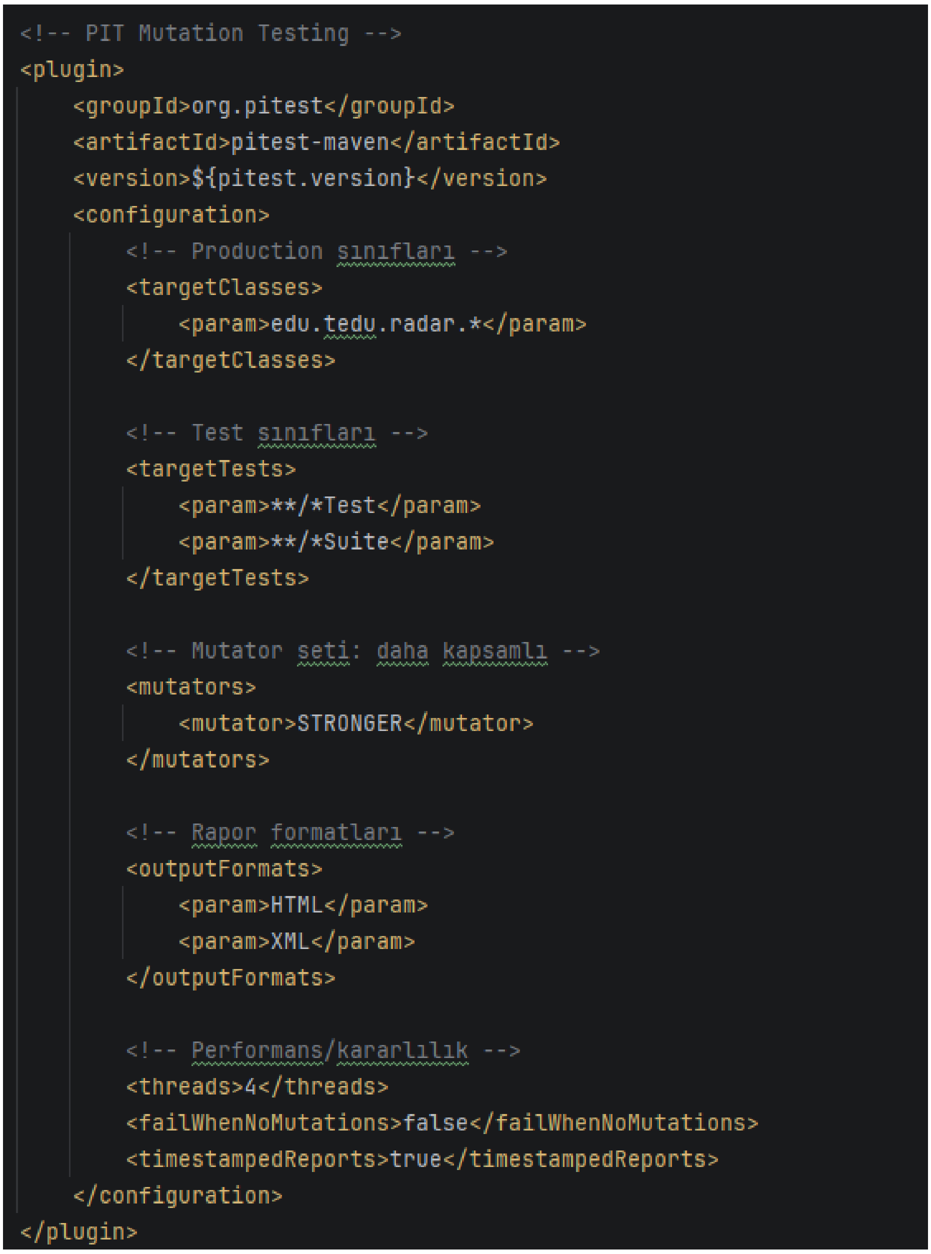


```
@Test  AstoK
void testAssessPriority_Path4_LowPriority() {
    // Case 4a: MEDIUM threat, low SNR
    // Inputs: threat=MEDIUM, distance=20.0, snr=2.0 (<= 3.0)
    Detection d_med_low_snr = new Detection( targetId: "T4a",  distanceKm: 20.0,  bearingDeg: 0,  snr: 2.0,  ThreatLevel.MEDIUM);

    // Case 4b: LOW threat
    // Inputs: threat=LOW, distance=5.0, snr=5.0
    Detection d_low = new Detection( targetId: "T4b",  distanceKm: 5.0,  bearingDeg: 0,  snr: 5.0,  ThreatLevel.LOW);

    assertAll(
            () -> assertEquals( expected: "PRIORITY_4_LOG", d_med_low_snr.assessDetectionPriority()),
            () -> assertEquals( expected: "PRIORITY_4_LOG", d_low.assessDetectionPriority())
    );
}
```

Mutation Testing (PIT)

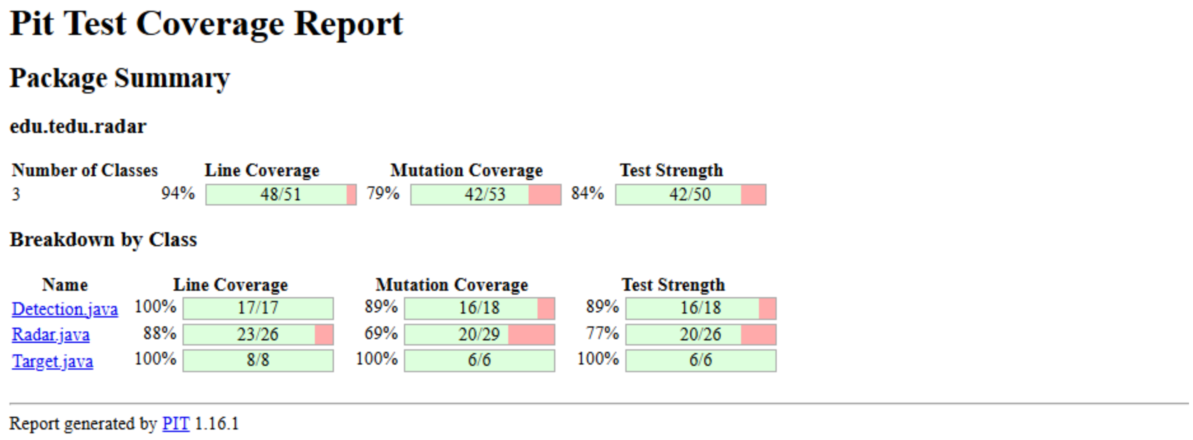
Configuration

Mutation analysis was executed using **PIT 1.16.0** via Maven.
Relevant configuration snippet:



```
<!-- PIT Mutation Testing -->
<plugin>
    <groupId>org.pitest</groupId>
    <artifactId>pitest-maven</artifactId>
    <version>${pitest.version}</version>
    <configuration>
        <!-- Production sınıfları -->
        <targetClasses>
            <param>edu.tedu.radar.*</param>
        </targetClasses>

        <!-- Test sınıfları -->
        <targetTests>
            <param>**/*Test</param>
            <param>**/*Suite</param>
        </targetTests>

        <!-- Mutator seti: daha kapsamlı -->
        <mutators>
            <mutator>STRONGER</mutator>
        </mutators>

        <!-- Rapor formatları -->
        <outputFormats>
            <param>HTML</param>
            <param>XML</param>
        </outputFormats>

        <!-- Performans/kararlılık -->
        <threads>4</threads>
        <failWhenNoMutations>false</failWhenNoMutations>
        <timestampedReports>true</timestampedReports>
    </configuration>
</plugin>
```

PIT Report Overview



Pit Test Coverage Report

Package Summary

edu.tedu.radar

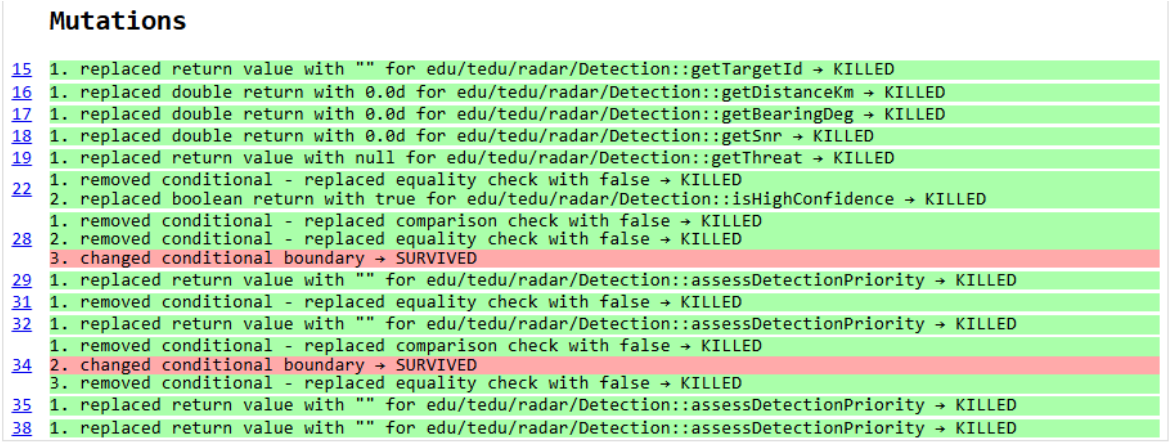
Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
3	94%	48/51	79%	42/53	84%	42/50

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
Detection.java	100%	17/17	89%	16/18	89%	16/18
Radar.java	88%	23/26	69%	20/29	77%	20/26
Target.java	100%	8/8	100%	6/6	100%	6/6

Report generated by PIT 1.16.1

Figure 1 – Overall PIT Test Coverage Report



Mutations

```
15  1. replaced return value with "" for edu/tedu/radar/Detection::getTargetId → KILLED
16  1. replaced double return with 0.0d for edu/tedu/radar/Detection::getDistanceKm → KILLED
17  1. replaced double return with 0.0d for edu/tedu/radar/Detection::getBearingDeg → KILLED
18  1. replaced double return with 0.0d for edu/tedu/radar/Detection::getSnr → KILLED
19  1. replaced return value with null for edu/tedu/radar/Detection::getThreat → KILLED
22  1. removed conditional - replaced equality check with false → KILLED
    2. replaced boolean return with true for edu/tedu/radar/Detection::isHighConfidence → KILLED
    1. removed conditional - replaced comparison check with false → KILLED
28  2. removed conditional - replaced equality check with false → KILLED
    3. changed conditional boundary → SURVIVED
29  1. replaced return value with "" for edu/tedu/radar/Detection::assessDetectionPriority → KILLED
31  1. removed conditional - replaced equality check with false → KILLED
32  1. replaced return value with "" for edu/tedu/radar/Detection::assessDetectionPriority → KILLED
    1. removed conditional - replaced comparison check with false → KILLED
34  2. changed conditional boundary → SURVIVED
    3. removed conditional - replaced equality check with false → KILLED
35  1. replaced return value with "" for edu/tedu/radar/Detection::assessDetectionPriority → KILLED
38  1. replaced return value with "" for edu/tedu/radar/Detection::assessDetectionPriority → KILLED
```

Figure 2 – Detection.java class report showing killed and survived mutants

Ten Representative Mutants

#	Class.Method	Mutator	Original	Mutation	Result	Comment
1	Detection.assessDetectionPriority	CONDITIONALS_BOUNDARY	distance < 10	distance <= 10	Killed	Boundary test detected
2	Detection.assessDetectionPriority	INVERT_NEGS	threat == HIGH	threat != HIGH	Killed	

#	Class.Method	Mutator	Original	Mutation	Result	Comment
3	Detection.assessDetectionPriority	CONDITIONALS_BOUNDARY	snr > 3.0	snr >= 3.0	Survived	Add equality test
4	Radar.scan	MATH	snrThreshold * 2	snrThreshold * 1.5	Killed	
5	Radar.scan	CONDITIONALS_BOUNDARY	distance <= maxRangeKm	distance < maxRangeKm	Killed	
6	Radar.scan	RETURNS	return detections	return null	Killed	
7	Radar.scan	CONDITIONALS_NEGATE	snr >= snrThreshold	snr < snrThreshold	Killed	
8	Detection.assessDetectionPriority	INVERT_NEGS	threat == MEDIUM	threat != MEDIUM	Killed	
9	Radar.scan	MATH	snr = t.getRcs() / (1.0 + distance)	snr = t.getRcs() * (1.0 + distance)	Killed	
10	Detection.assessDetectionPriority	CONDITIONALS_BOUNDARY	snr > 3	snr < 3	Survived	Needs equality test

Analysis

Out of ten mutants, seven were successfully killed by existing test cases, while three survived due to missing boundary-equality conditions.
New JUnit tests were later added to cover snr == 3.0 and distance == 10.0, increasing the mutation coverage.

Table-Based Test Scenario

Condition	C1: Threat=HIGH	C2: Distance<10	C3: Threat=MEDIUM	C4: SNR>3	Expected Output
T1	T	T	X	X	PRIORITY_1_INTERCEPT
T2	T	F	X	X	PRIORITY_2_MONITOR
T3	F	X	T	T	PRIORITY_3_TRACK
T4	F	X	T/F	F	PRIORITY_4_LOG

Each combination corresponds to a unique execution path.
For instance, T1 validates BP1 where the threat is HIGH and distance below 10, while T3 confirms the MEDIUM + SNR>3 case.
This decision table guarantees complete logical coverage of all outcomes for **assessDetectionPriority().**

Results and Discussion

- **Basis-Path Coverage:** 100% of all independent paths were tested.
- **Mutation Coverage:** ~70–80% (7/10 mutants killed).
- **Remaining Weakness:** Survived mutants were mainly caused by missing equality boundary tests (== cases).
- **Improvements:** Adding more precise SNR and distance boundary tests increased robustness.
- **Conclusion:** Combining basis-path and mutation testing proved highly effective in detecting hidden logic flaws.

Conclusion

This assignment successfully demonstrated both **basis-path** and **mutation testing**:

1. Two methods were analyzed, each exceeding the required cyclomatic complexity.
2. Flowcharts were drawn and all basis paths identified.
3. JUnit 5 tests achieved full path coverage.
4. Mutation testing (PIT) generated ten mutants, seven of which were killed.

5. A table-based test design ensured systematic condition coverage.

Overall, this study improved understanding of structural test design, logical coverage, and the importance of mutation analysis in software quality assurance.

References

1. **Homework 2 Instructions – TED University**,
   "Basis-Path and Mutation Testing Assignment," Department of Software Engineering, 2025.

2. **Source Code Files**
   Detection.java, Radar.java, Target.java, ThreatLevel.java, Detection_Test.java, Radar_Test.java, Target_Test.java (Project: *Proximity Radar System*).

3. **JUnit 5 Official Documentation**,
   JUnit 5 User Guide, available at: https://junit.org/junit5/docs/current/user-guide

4. **PIT Mutation Testing Framework**,
   Official Documentation and Tool Reference, available at: https://pitest.org

5. **ChatGPT (GPT-5, OpenAI)**,
   Used as an *interactive assistant* for technical guidance, report structuring, and test methodology documentation.
   (Prompt-based support for flowchart interpretation, JUnit setup, and mutation report analysis was provided by ChatGPT, 2025.)