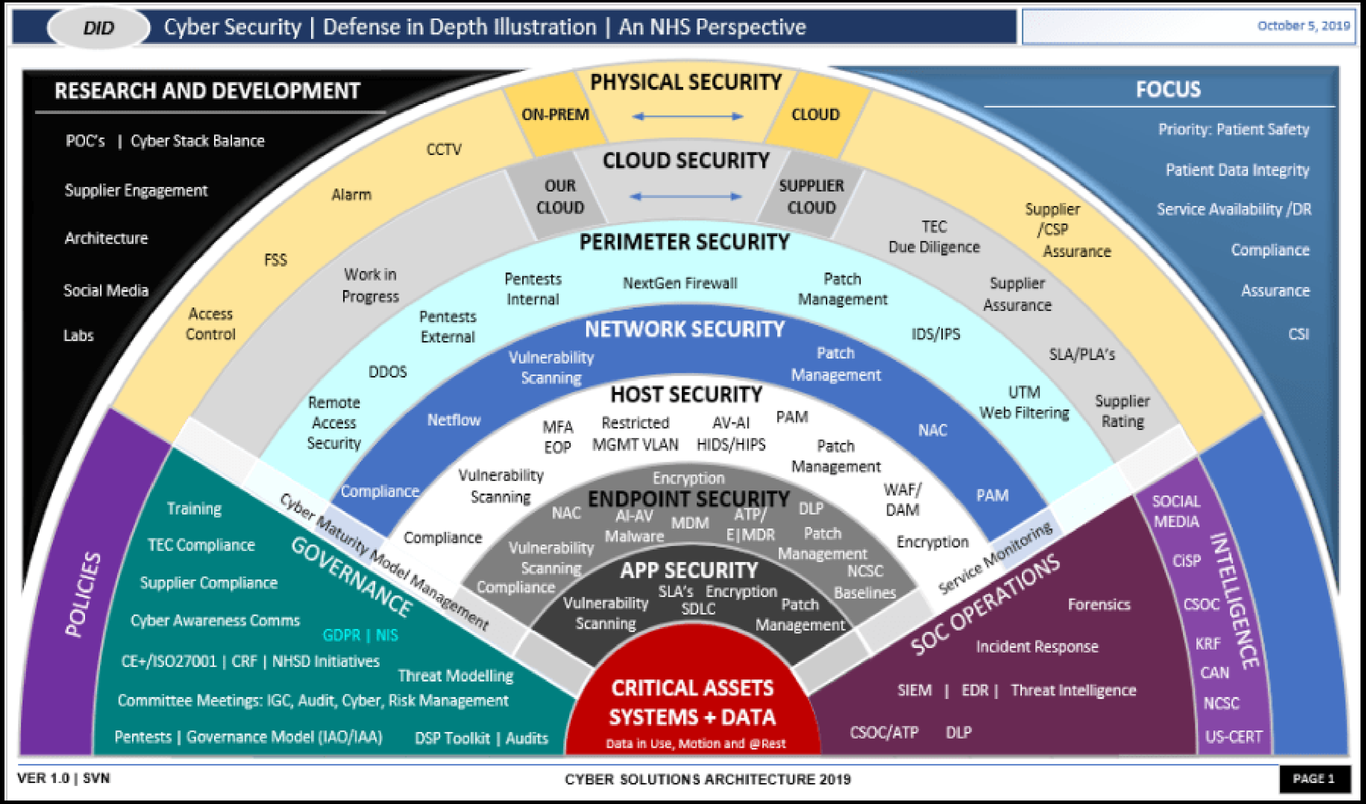
Green Pace

Security Policy Presentation

Developer: *Waeil Mikhaeil*

OVERVIEW: DEFENSE IN DEPTH

This security policy aims to establish a set of standards and best practices to ensure the safety of our software and data. Its various components form key layers within the Defense in Depth strategy.



DID Cyber Security | Defense in Depth Illustration | An NHS Perspective

October 5, 2019

RESEARCH AND DEVELOPMENT

POC's | Cyber Stack Balance

Supplier Engagement

Architecture

Social Media

Labs

PHYSICAL SECURITY

ON-PREM ⟷ CLOUD

CCTV, Alarm, FSS, Access Control

Supplier /CSP Assurance, Supplier Assurance, SLA/PLA's, Supplier Rating

CLOUD SECURITY

OUR CLOUD ⟷ SUPPLIER CLOUD

Work in Progress, DDOS, Remote Access Security

TEC Due Diligence, IDS/IPS, UTM Web Filtering

PERIMETER SECURITY

Pentests Internal, NextGen Firewall, Patch Management, Pentests External

NETWORK SECURITY

Vulnerability Scanning, Patch Management, Netflow, NAC, Compliance, PAM

HOST SECURITY

MFA EOP, Restricted MGMT VLAN, AV-AI HIDS/HIPS, PAM, Patch Management, Vulnerability Scanning, Compliance, WAF/ DAM, Encryption

ENDPOINT SECURITY

Encryption, NAC, AI-AV Malware, MDM, ATP/ E|MDR, DLP, Patch Management, Vulnerability Scanning, Compliance, NCSC Baselines

APP SECURITY

Vulnerability Scanning, SLA's SDLC, Encryption, Patch Management

CRITICAL ASSETS SYSTEMS + DATA

Data in Use, Motion and @Rest

FOCUS

Priority: Patient Safety

Patient Data Integrity

Service Availability /DR

Compliance

Assurance

CSI

POLICIES

GOVERNANCE

Training

TEC Compliance

Supplier Compliance

Cyber Awareness Comms

GDPR | NIS

CE+/ISO27001 | CRF | NHSD Initiatives

Threat Modelling

Committee Meetings: IGC, Audit, Cyber, Risk Management

Pentests | Governance Model (IAO/IAA)

DSP Toolkit | Audits

Cyber Maturity Model Management

Service Monitoring

SOC OPERATIONS

Forensics

Incident Response

SIEM | EDR | Threat Intelligence

CSOC/ATP DLP

INTELLIGENCE

SOCIAL MEDIA, CiSP, CSOC, KRF, CAN, NCSC, US-CERT

VER 1.0 | SVN

CYBER SOLUTIONS ARCHITECTURE 2019

PAGE 1

Green Pace

THREATS MATRIX

Unlikely/Low: Minor issues like reference qualifiers (STD-001-CPP) or assert behavior (STD-006-CPP).

Unlikely/High: File closure (STD-009-CPP) or constructor order (STD-010-CPP)—rare but serious if ignored.

Likely/Low: Return values (STD-002-CPP) or exceptions (STD-007-CPP)—frequent but controllable.

Likely/High: Critical risks like range checks (STD-003-CPP), sanitization (STD-004-CPP), memory leaks (STD-005-CPP), and namespace mods (STD-008-CPP).

I use automation—unit tests and static analysis—to detect these early

Likely STD-002-CPP STD-007-CPP	Priority STD-004-CPP STD-005-CPP
STD-001-CPP STD-006-CPP STD-009-CPP STD-010-CPP	Unlikely STD-003-CPP STD-008-CPP

Green Pace

10 PRINCIPLES

Here’s how I tie essential security principles to specific standards:

Principle	Description	Related Standard
Ensure Data Validation	Confirm all incoming data is clean and safe	STD-004-CPP (data sanitization)
Respect Compiler Alerts	Address warnings to guarantee proper outputs	STD-002-CPP (return values)
Build with Security in Mind	Plan systems to resist threats	STD-003-CPP (range validation)
Simplify Design	Avoid complexity to maintain clarity	STD-008-CPP (avoid namespace changes)
Block by Default	Deny access unless explicitly allowed	STD-007-CPP (exception management)
Limit Access Rights	Grant only necessary permissions	STD-001-CPP (reference restrictions)
Clean External Inputs	Process outside data to remove risks	STD-004-CPP (data sanitization)
Layered Protection	Use multiple defenses for resilience	STD-005-CPP (memory handling)
Focus on Quality Checks	Test thoroughly to catch issues	STD-006-CPP (assert/abort behavior)
Follow Secure Coding Rules	Adhere to established safety practices	STD-009-CPP, STD-010-CPP (file management, constructor order)

CODING STANDARDS

I’ve ordered these based on their effect and how often they apply:

Standard	Purpose	Benefit
STD-004-CPP	Cleanse data to stop malicious inputs	Prevents injection exploits
STD-005-CPP	Free up allocated resources properly	Avoids memory leaks
STD-003-CPP	Verify ranges to prevent oversteps	Stops buffer overflow risks
STD-008-CPP	Restrict namespace alterations	Maintains system consistency
STD-002-CPP	Guarantee functions provide outputs	Ensures dependable behavior
STD-007-CPP	Manage errors to keep systems running	Reduces crash potential
STD-001-CPP	Avoid unnecessary qualifiers on references	Clarifies code intent
STD-006-CPP	Grasp assert and abort mechanics	Enhances debugging safety
STD-009-CPP	Release file handles when done	Frees up system resources
STD-010-CPP	Sequence constructor elements correctly	

ENCRYPTION POLICIES

Data Protection Policies
Here’s how I secure data in different states:

State	Approach	Protection
In Flight	Employ TLS 1.3 for data in transit	Shields against eavesdropping
At Rest	Apply AES-256 encryption to stored data	Secures data if storage is compromised
In Use	Use secure enclaves (e.g., Intel SGX) for processing	Limits exposure during operations

Green Pace

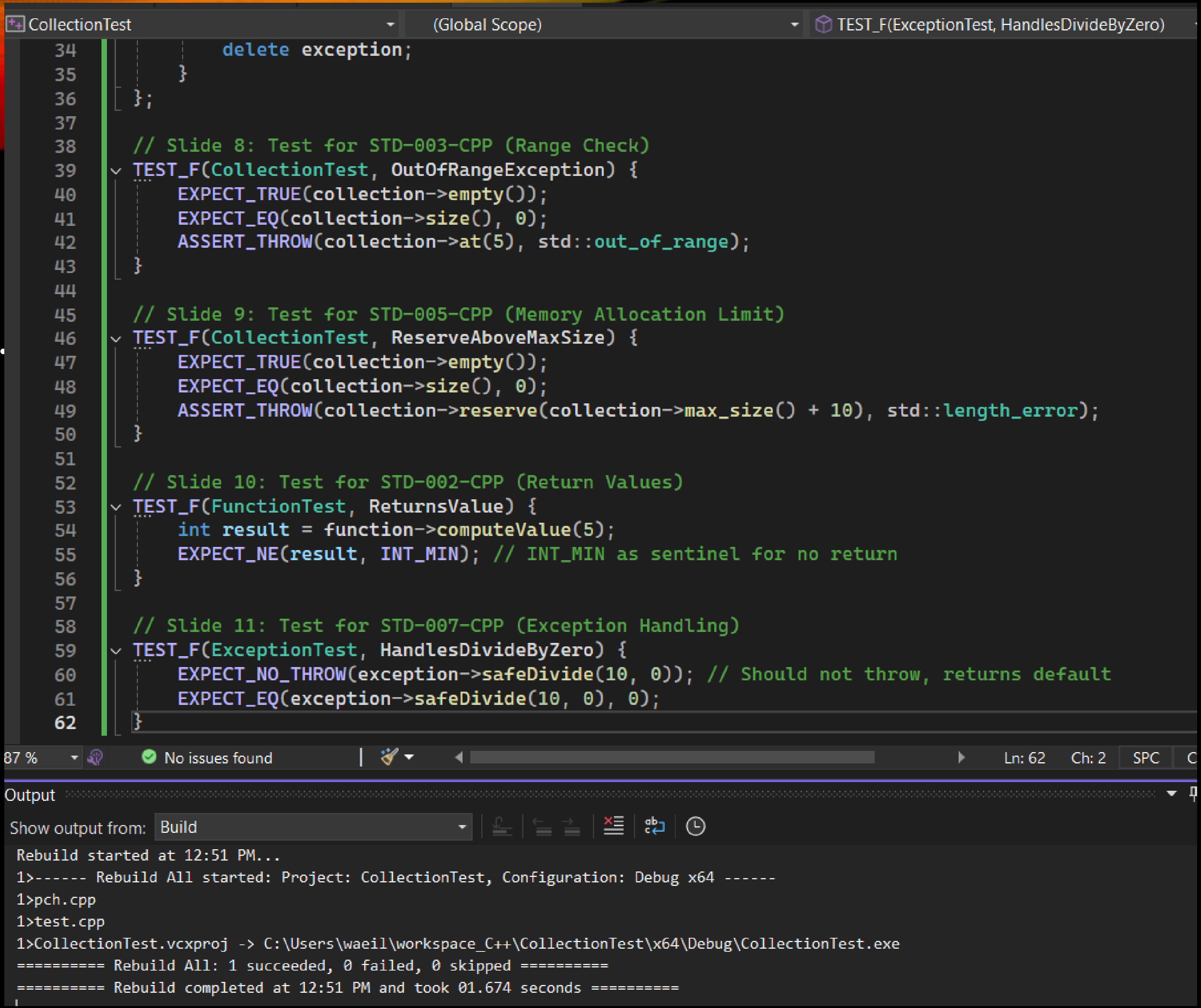
TRIPLE-A POLICIES

Access Control Policies
Here’s how I handle authentication, authorization, and accounting:

Component	**Policy**	**Outcome**
Authentication	Mandate MFA and robust passwords (16+ chars)	Confirms user identities
Authorization	Use RBAC tied to STD-001-CPP (minimal privileges)	Restricts access to essentials
Accounting	Record all actions with timestamps, stored safely	Enables tracking and audits

Green Pace

Unit Testing



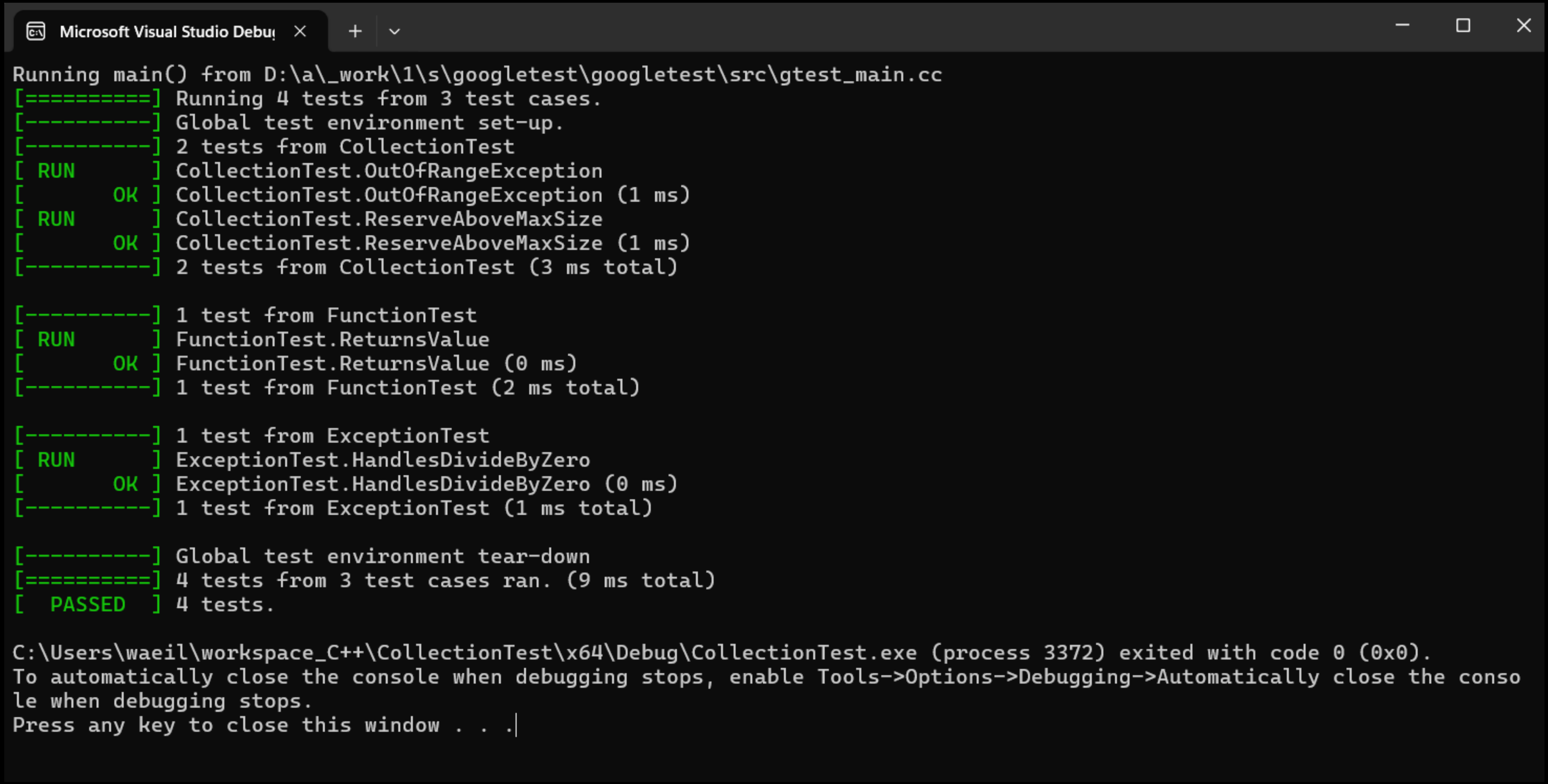
```
    delete exception;
  }
};

// Slide 8: Test for STD-003-CPP (Range Check)
TEST_F(CollectionTest, OutOfRangeException) {
  EXPECT_TRUE(collection->empty());
  EXPECT_EQ(collection->size(), 0);
  ASSERT_THROW(collection->at(5), std::out_of_range);
}

// Slide 9: Test for STD-005-CPP (Memory Allocation Limit)
TEST_F(CollectionTest, ReserveAboveMaxSize) {
  EXPECT_TRUE(collection->empty());
  EXPECT_EQ(collection->size(), 0);
  ASSERT_THROW(collection->reserve(collection->max_size() + 10), std::length_error);
}

// Slide 10: Test for STD-002-CPP (Return Values)
TEST_F(FunctionTest, ReturnsValue) {
  int result = function->computeValue(5);
  EXPECT_NE(result, INT_MIN); // INT_MIN as sentinel for no return
}

// Slide 11: Test for STD-007-CPP (Exception Handling)
TEST_F(ExceptionTest, HandlesDivideByZero) {
  EXPECT_NO_THROW(exception->safeDivide(10, 0)); // Should not throw, returns default
  EXPECT_EQ(exception->safeDivide(10, 0), 0);
}
```

Show output from: Build

```
Rebuild started at 12:51 PM...
1>------ Rebuild All started: Project: CollectionTest, Configuration: Debug x64 ------
1>pch.cpp
1>test.cpp
1>CollectionTest.vcxproj -> C:\Users\waeil\workspace_C++\CollectionTest\x64\Debug\CollectionTest.exe
========== Rebuild All: 1 succeeded, 0 failed, 0 skipped ==========
========== Rebuild completed at 12:51 PM and took 01.674 seconds ==========
```

Green Pace

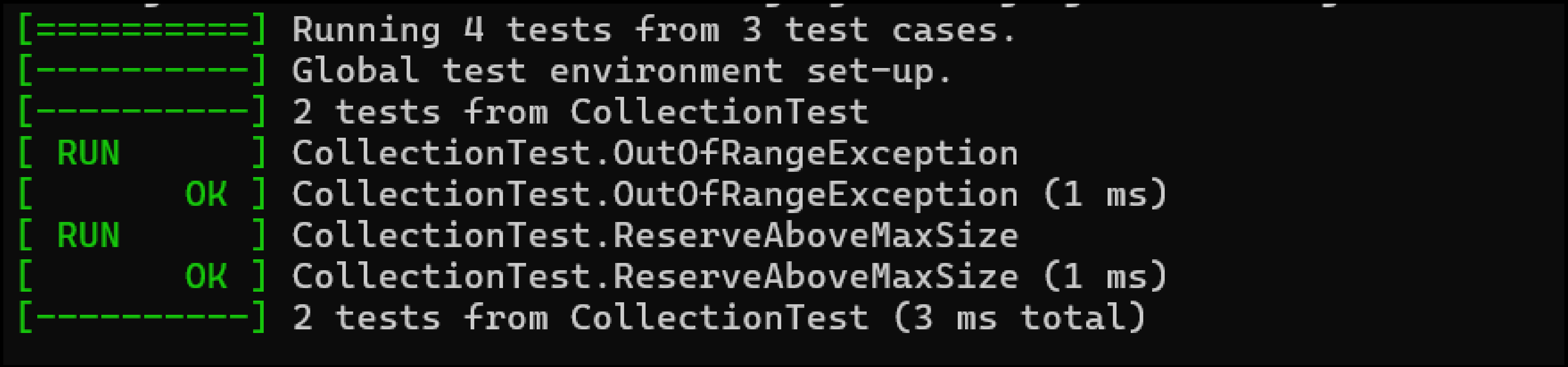
Unit Testing



```
Running main() from D:\a\_work\1\s\googletest\googletest\src\gtest_main.cc
[==========] Running 4 tests from 3 test cases.
[----------] Global test environment set-up.
[----------] 2 tests from CollectionTest
[ RUN      ] CollectionTest.OutOfRangeException
[       OK ] CollectionTest.OutOfRangeException (1 ms)
[ RUN      ] CollectionTest.ReserveAboveMaxSize
[       OK ] CollectionTest.ReserveAboveMaxSize (1 ms)
[----------] 2 tests from CollectionTest (3 ms total)

[----------] 1 test from FunctionTest
[ RUN      ] FunctionTest.ReturnsValue
[       OK ] FunctionTest.ReturnsValue (0 ms)
[----------] 1 test from FunctionTest (2 ms total)

[----------] 1 test from ExceptionTest
[ RUN      ] ExceptionTest.HandlesDivideByZero
[       OK ] ExceptionTest.HandlesDivideByZero (0 ms)
[----------] 1 test from ExceptionTest (1 ms total)

[----------] Global test environment tear-down
[==========] 4 tests from 3 test cases ran. (9 ms total)
[  PASSED  ] 4 tests.

C:\Users\waeil\workspace_C++\CollectionTest\x64\Debug\CollectionTest.exe (process 3372) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

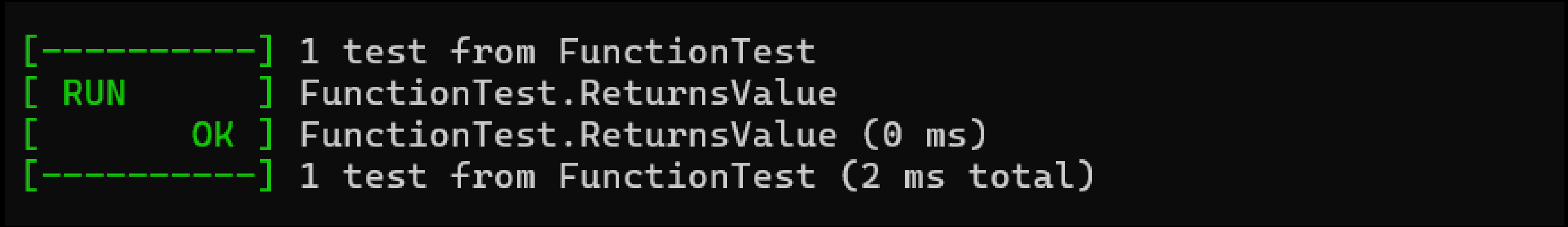
Green Pace

Unit Testing

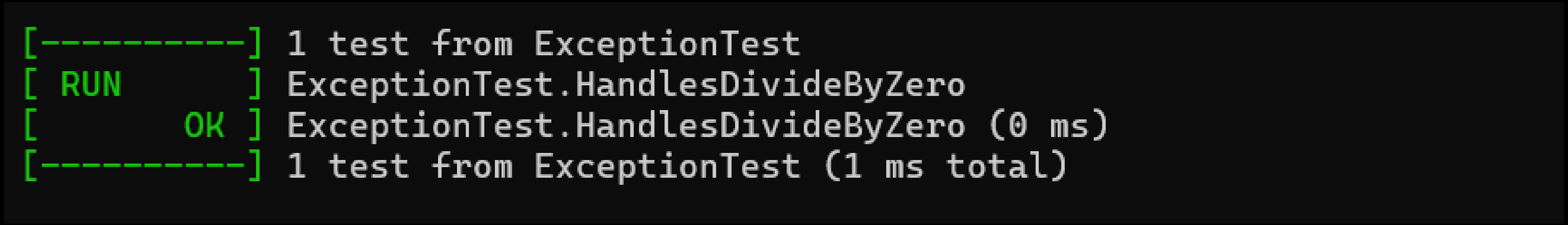


```
[==========] Running 4 tests from 3 test cases.
[----------] Global test environment set-up.
[----------] 2 tests from CollectionTest
[ RUN      ] CollectionTest.OutOfRangeException
[       OK ] CollectionTest.OutOfRangeException (1 ms)
[ RUN      ] CollectionTest.ReserveAboveMaxSize
[       OK ] CollectionTest.ReserveAboveMaxSize (1 ms)
[----------] 2 tests from CollectionTest (3 ms total)
```

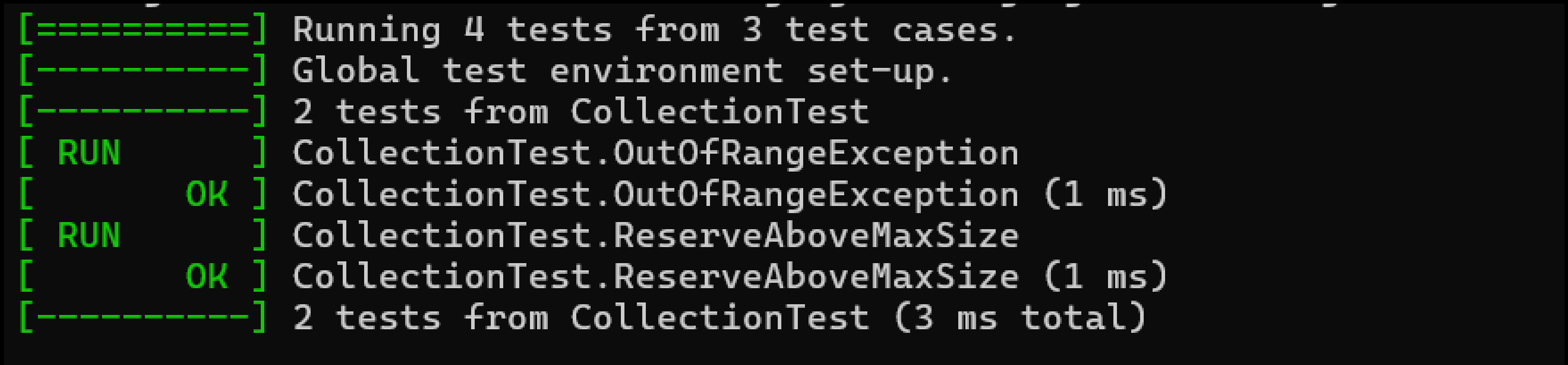
Unit Testing



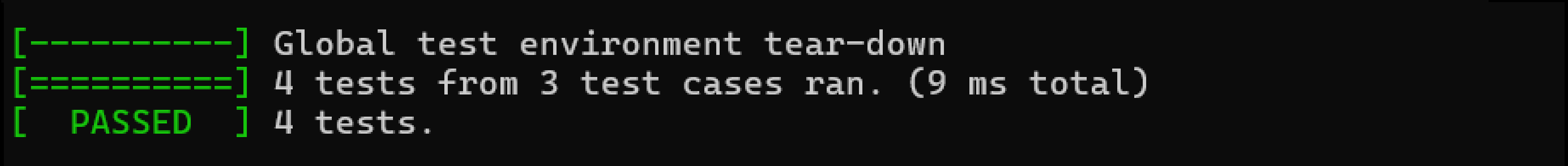
```
[----------] 1 test from FunctionTest
[ RUN      ] FunctionTest.ReturnsValue
[       OK ] FunctionTest.ReturnsValue (0 ms)
[----------] 1 test from FunctionTest (2 ms total)
```



```
[----------] 1 test from ExceptionTest
[ RUN      ] ExceptionTest.HandlesDivideByZero
[       OK ] ExceptionTest.HandlesDivideByZero (0 ms)
[----------] 1 test from ExceptionTest (1 ms total)
```



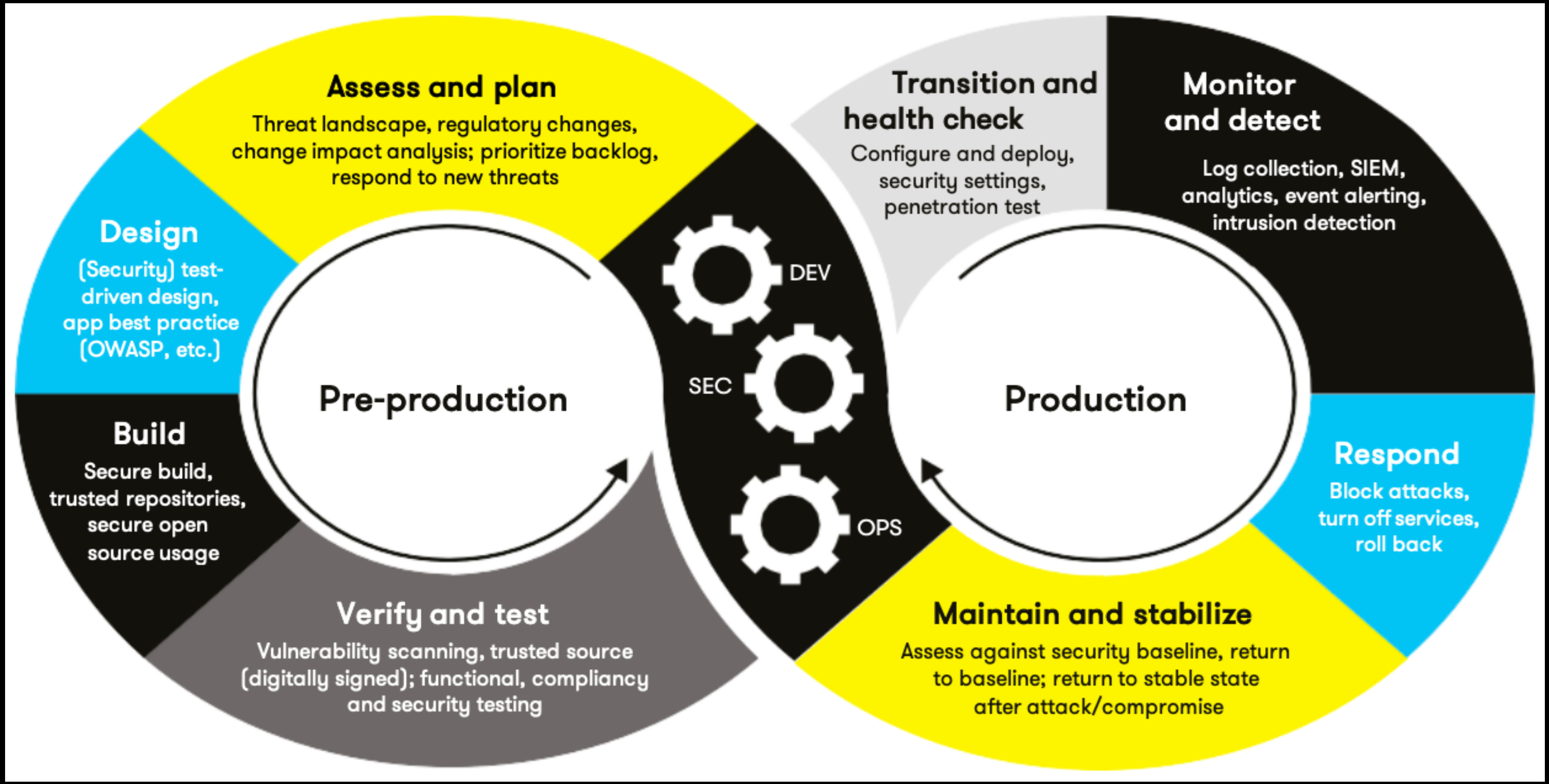
```
[==========] Running 4 tests from 3 test cases.
[----------] Global test environment set-up.
[----------] 2 tests from CollectionTest
[ RUN      ] CollectionTest.OutOfRangeException
[       OK ] CollectionTest.OutOfRangeException (1 ms)
[ RUN      ] CollectionTest.ReserveAboveMaxSize
[       OK ] CollectionTest.ReserveAboveMaxSize (1 ms)
[----------] 2 tests from CollectionTest (3 ms total)
```



```
[----------] Global test environment tear-down
[==========] 4 tests from 3 test cases ran. (9 ms total)
[  PASSED  ] 4 tests.
```

Green Pace

AUTOMATION SUMMARY



Assess and plan
Threat landscape, regulatory changes, change impact analysis; prioritize backlog, respond to new threats

Design
(Security) test-driven design, app best practice (OWASP, etc.)

Build
Secure build, trusted repositories, secure open source usage

Verify and test
Vulnerability scanning, trusted source (digitally signed); functional, compliancy and security testing

Pre-production

Transition and health check
Configure and deploy, security settings, penetration test

Monitor and detect
Log collection, SIEM, analytics, event alerting, intrusion detection

Respond
Block attacks, turn off services, roll back

Maintain and stabilize
Assess against security baseline, return to baseline; return to stable state after attack/compromise

Production

Green Pace

TOOLS

Development Workflow

The DevSecOps pipeline outlines the steps of the development process from beginning to end.

Pre-production focuses on designing and testing software to identify issues early.

Production involves deploying, monitoring, and maintaining software for smooth operation.

In the design phase, OWASP guidelines or the security policy shape a secure foundation.

During the build stage, an IDE and compiler handle code compilation and execution.

The verify and test phase uses static analysis and unit testing tools to uncover vulnerabilities.

In the monitor and detect stage, logging and application monitoring tools track issues in real time.

RISKS AND BENEFITS

Why Act Now?

Should security be addressed now or delayed? The answer is now without question!

Protecting applications and data is a critical priority that demands immediate attention.

Security needs to be considered right from the planning stage.

Benefits:

Embedding security early simplifies implementation.

Code becomes more robust and reliable

.Consistency improves across the codebase.

Risks:

Lack of full team buy-in can pose challenges.

Retroactively adding security is harder and less effective.

Delaying risks breaches and exposure of sensitive data.

Green Pace

RECOMMENDATIONS

Plan for Success

A security policy is useless unless it’s understood and followed.

DevSecOps training for developers ensures everyone stays aligned.

Automated monitoring of software dependencies detects vulnerabilities and applies patches to prevent supply chain attacks.

https://learn.microsoft.com/en-us/nuget/concepts/security-best-practices.

CONCLUSIONS

- Security must be a priority from the outset—no exceptions.
- Training on security and the policy equips the team for success.
- Enforcement mechanisms ensure compliance with the policy.
- Applications won’t deploy unless security requirements are met.
- Regular reviews and updates keep the policy ahead of emerging threats.

REFERENCES

Microsoft. (2022, October 11). Best practices for a secure software supply chain. Microsoft Learn. https://learn.microsoft.com/en-us/nuget/concepts/security-best-practices

SEI CERT. C++ coding standard. Carnegie Mellon University. https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046322

OWASP. OWASP top ten. https://owasp.org/www-project-top-ten/

Green Pace