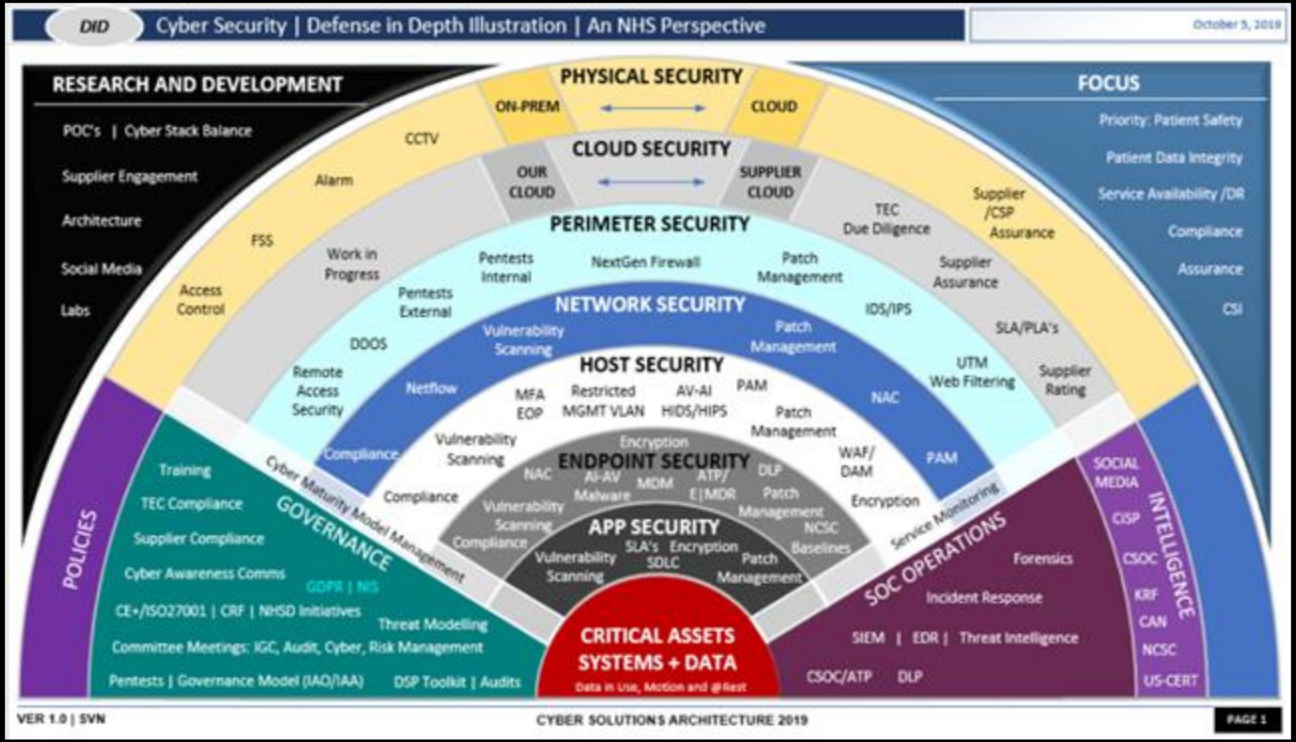
Green Pace

Security Policy Presentation

Developer: Walter Briones

OVERVIEW: DEFENSE IN DEPTH

In order to produce and maintain safe and secure software, the Green Pace security policy lays out our standards and best practices for ensuring our team delivers secure, high-quality systems to our customers. This strategy is core to our business in an ever-changing threat landscape. Green Pace is striving towards a paradigm shift in security as a core tenet within the development cycle.



THREATS MATRIX

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

	Likelihood		Prioritization
Likely:	High probability and low severity.	Low Priority:	Not probable and low severity.
Unlikely:	Low probability and high severity.	Priority:	High probability and high severity.

Green Pace

10 PRINCIPLES

	Principle	Associated Coding Standards
1.	Validate Input Data	STD-003-CPP, STD-004-CPP, STD-009-CPP
2.	Heed Compiler Warnings	STD-001-CPP, STD-005-CPP, STD-007-CPP, STD-008-CPP, STD-009-CPP
3.	Architect and Design for Security Policies	STD-004-CPP, STD-005-CPP
4.	Keep it Simple	STD-001-CPP, STD-002-CPP, STD-008-CPP, STD-010-CPP
5.	Default Deny	STD-004-CPP
6.	Adhere to the Principle of Least Privilege	STD-004-CPP
7.	Sanitize Data Sent to Other Systems	STD-003-CPP
8.	Practice Defense in Depth	STD-003-CPP, STD-004-CPP
9.	Use Effective Quality Assurance Techniques	STD-006-C, STD-007-CPP, STD-009-CPP, STD-010-CPP
10.	Adopt a Secure Coding Standard	STD-002-CPP, STD-006-C

Green Pace

CODING STANDARDS

Label	Coding Standard
STD-006-C	Incorporate diagnostic tests using assertions
STD-005-CPP	Properly deallocate dynamically allocated resources
STD-009-CPP	Ensure that unsigned integer operations do not wrap
STD-004-CPP	Sanitize data passed to complex subsystems
STD-003-CPP	Guarantee that storage for strings has sufficient space for character data and the null terminator
STD-010-CPP	Do not declare variables inside a switch statement before the first case label
STD-002-CPP	Value-returning functions must return a value from all exit paths
STD-007-CPP	Handle all exceptions
STD-008-CPP	Do not attempt to modify string literals
STD-001-CPP	Never qualify reference types with const or volatile

*Coding standards ranked in order of priority

Green Pace

ENCRYPTION POLICIES

Encryption at rest	Ensures encryption of data "at rest." Applies to data in storage such as on mobile devices, desktop and/or laptop computers, and any other medium holding data that is not currently in motion or in use. Encryption algorithms should be in place when storing such data, as well as recalling the data for use. This ensures data manipulation is unlikely and that the data at rest is secured.
Encryption in flight	Applies to encrypting data that is in motion. Examples include file transfers to and from cloud storage, network data transfers, emails, messaging, client-to-server communications, and more. Proper encryption schemes and security features such as firewalls, TLS/SSL, IPsec, and asymmetric encryption, among others, ensures that data in motion is as secure as possible.
Encryption in use	Refers to data actively being modified, processed, or otherwise categorized as "in use." Strategies such as applying authentication and authorization as a prerequisite for using such data minimizes security risks and assures only the parties allowed to use said data is doing so. Applying encryption prior-to and post-modification of such data also bolsters their security.

Green Pace

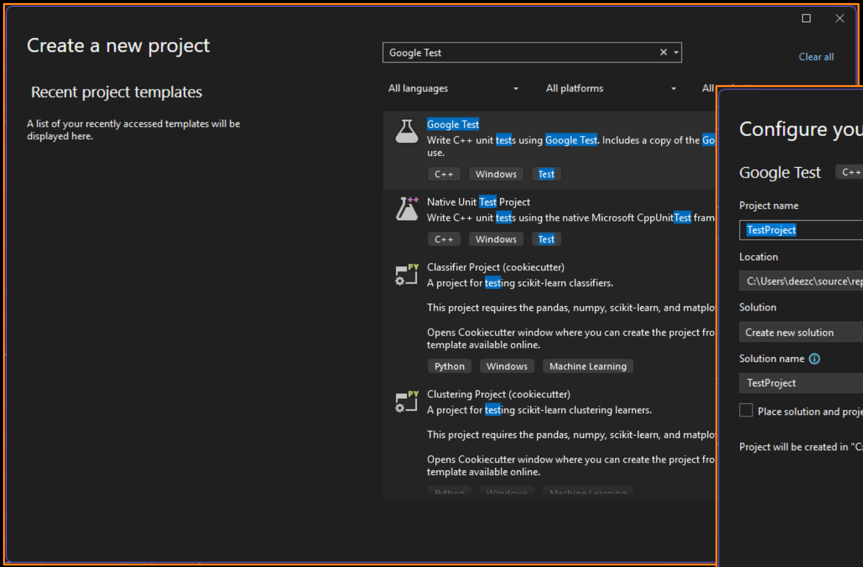
TRIPLE-A POLICIES

Authentication	Refers to verifying a user’s identity prior to allowing access to system components. Techniques such as 2-factor authentication, biometric authentication, strong password enforcement, one-time or single-use passwords, and even dedicated physical authentication tokens are used to properly authenticate users and secure systems.
Authorization	Allows for the specification of what a given user of a system is allowed access to. Can apply to system components, data, and functionality. Limits access based on principle of least privilege to secure system assets against unwarranted accesses to potentially sensitive data. Techniques such as role definitions, tiered accesses, and permission-management solutions ensure parties are only given access to data that is needed to complete a particular task – further securing the system.
Accounting	Tracks system activity for potential threats or lapses in security. Helps to gain insight as to when, where, and potentially why a vulnerability exists. Accounting also helps to identify if a vulnerability has been utilized to the system’s possible detriment. Strategies such as logging system activities, monitoring of data transfers, scanning data for potential risk, and applying timestamps where possible allow for close auditing of system security as well as traceability if a security event takes place.

Unit Testing

Applying the Unit Testing Framework

Create a new project in Microsoft Visual Studio. In the "recent projects" template section, search for "Google Test."



Create a new project

Google Test

Clear all

Recent project templates

A list of your recently accessed templates will be displayed here.

All languages | All platforms | All

Google Test
Write C++ unit tests using Google Test. Includes a copy of the Go
use.
C++ | Windows | Test

Native Unit Test Project
Write C++ unit tests using the native Microsoft CppUnitTest fram
C++ | Windows | Test

Classifier Project (cookiecutter)
A project for testing scikit-learn classifiers.

This project requires the pandas, numpy, scikit-learn, and matplo

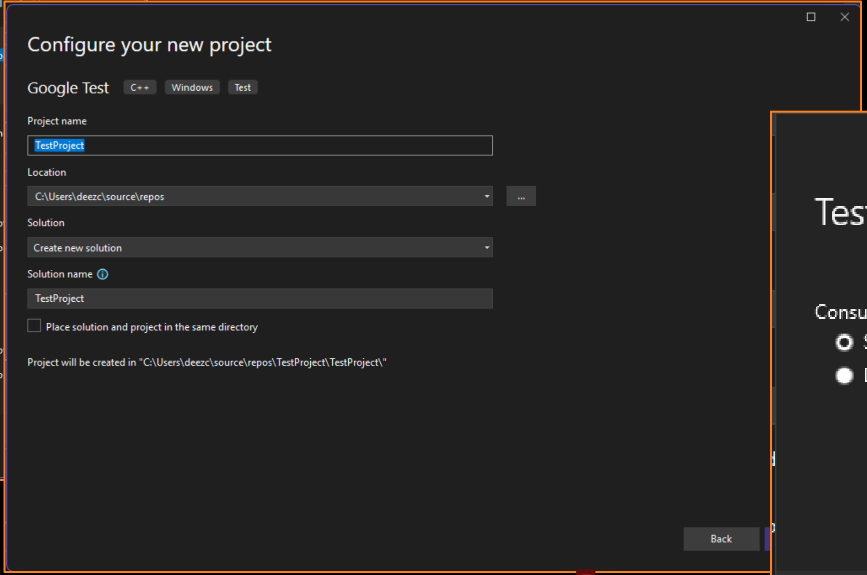
Opens Cookiecutter window where you can create the project fro
template available online.
Python | Windows | Machine Learning

Clustering Project (cookiecutter)
A project for testing scikit-learn clustering learners.

This project requires the pandas, numpy, scikit-learn, and matplo

Opens Cookiecutter window where you can create the project fro
template available online.

Next, name your project and designate a path for its associated files. Then click the "Create" button.



Configure your new project

Google Test | C++ | Windows | Test

Project name: TestProject

Location: C:\Users\deezc\source\repos

Solution: Create new solution

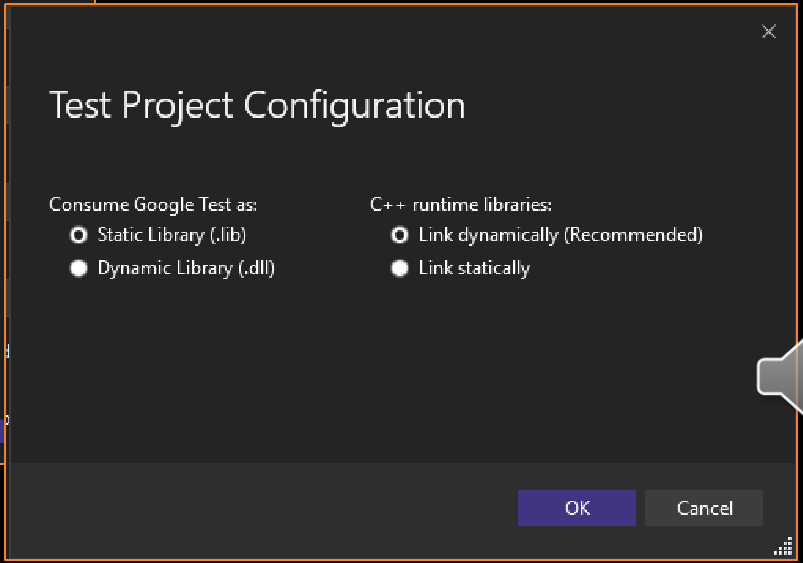
Solution name: TestProject

☐ Place solution and project in the same directory

Project will be created in "C:\Users\deezc\source\repos\TestProject\TestProject\"

Back

Finally, select Static Library (.lib) and Link Dynamically options in "Test Project Configuration" page and click "OK"



Test Project Configuration

Consume Google Test as:
[✓ Static Library (.lib)]
Dynamic Library (.dll)

C++ runtime libraries:
[✓ Link dynamically (Recommended)]
Link statically

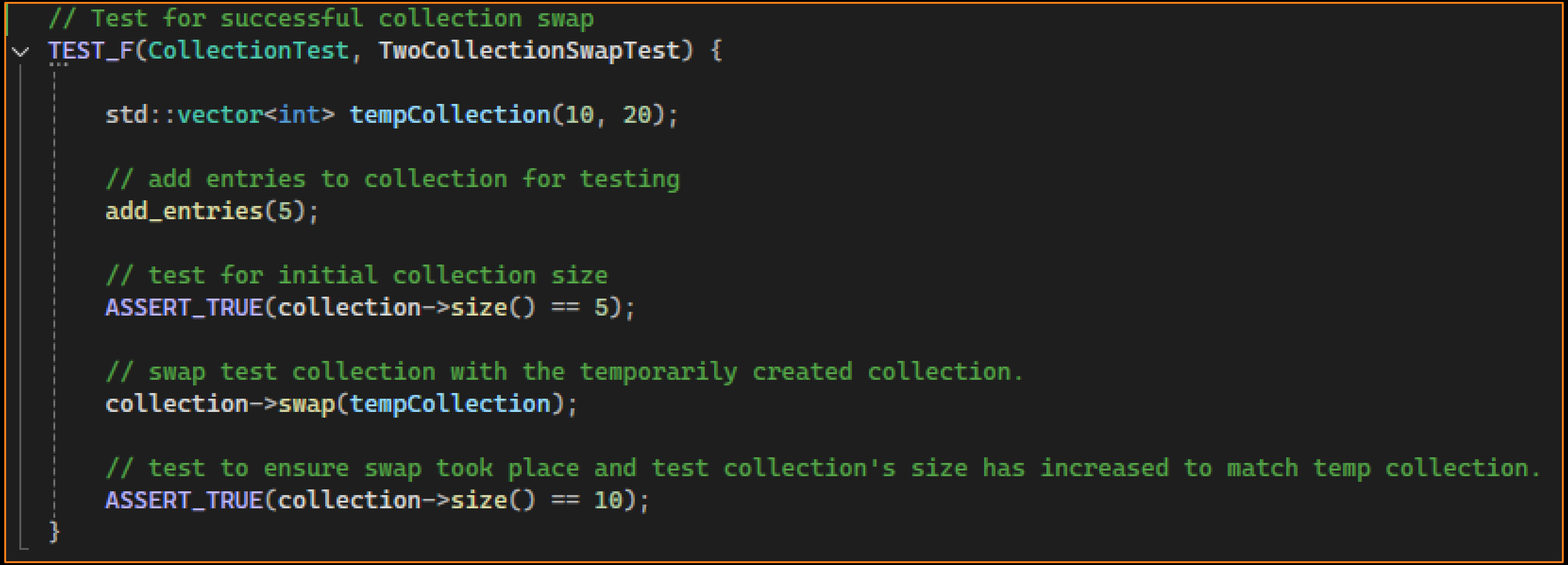
OK | Cancel

Unit Testing

Purpose: Testing for successful swap of data from one vector to another.

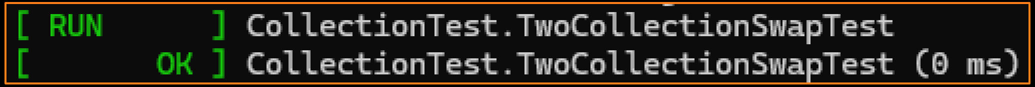
Coding vulnerability being tested: Movement of data between vector objects and ensuring storage of date into vector objects is taking place successfully.

Test Definition:



```
// Test for successful collection swap
TEST_F(CollectionTest, TwoCollectionSwapTest) {

    std::vector<int> tempCollection(10, 20);

    // add entries to collection for testing
    add_entries(5);

    // test for initial collection size
    ASSERT_TRUE(collection->size() == 5);

    // swap test collection with the temporarily created collection.
    collection->swap(tempCollection);

    // test to ensure swap took place and test collection's size has increased to match temp collection.
    ASSERT_TRUE(collection->size() == 10);
}
```

Result:



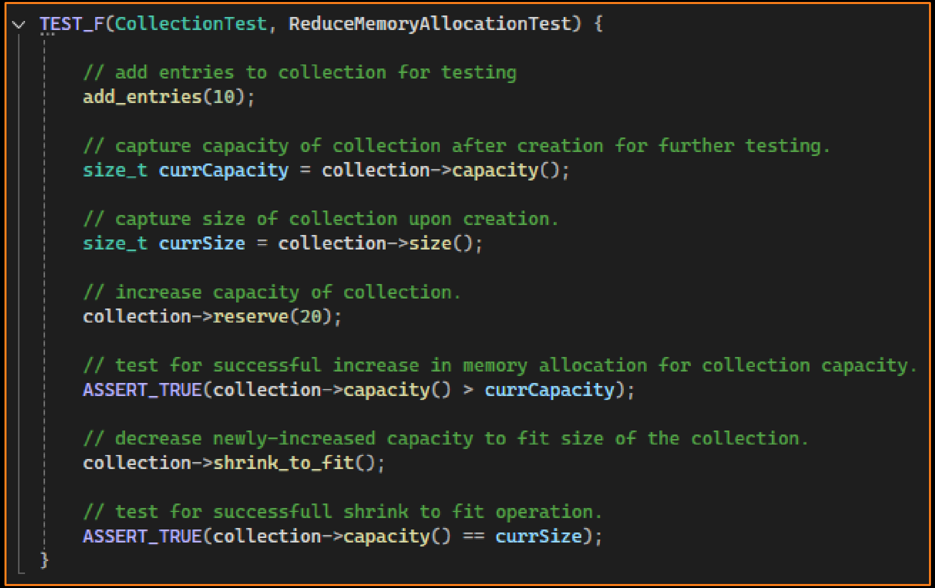
```
[ RUN      ] CollectionTest.TwoCollectionSwapTest
[       OK ] CollectionTest.TwoCollectionSwapTest (0 ms)
```

Unit Testing

Purpose: Testing for successful reduction of memory allocation.

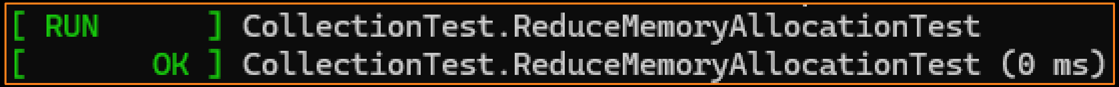
Coding vulnerability being tested: Managing resources and allocation to prevent from unexpected behaviors through decreases in resources allotted.

Test Definition:



```
TEST_F(CollectionTest, ReduceMemoryAllocationTest) {

    // add entries to collection for testing
    add_entries(10);

    // capture capacity of collection after creation for further testing.
    size_t currCapacity = collection->capacity();

    // capture size of collection upon creation.
    size_t currSize = collection->size();

    // increase capacity of collection.
    collection->reserve(20);

    // test for successful increase in memory allocation for collection capacity.
    ASSERT_TRUE(collection->capacity() > currCapacity);

    // decrease newly-increased capacity to fit size of the collection.
    collection->shrink_to_fit();

    // test for successfull shrink to fit operation.
    ASSERT_TRUE(collection->capacity() == currSize);
}
```

Result:



```
[ RUN      ] CollectionTest.ReduceMemoryAllocationTest
[       OK ] CollectionTest.ReduceMemoryAllocationTest (0 ms)
```

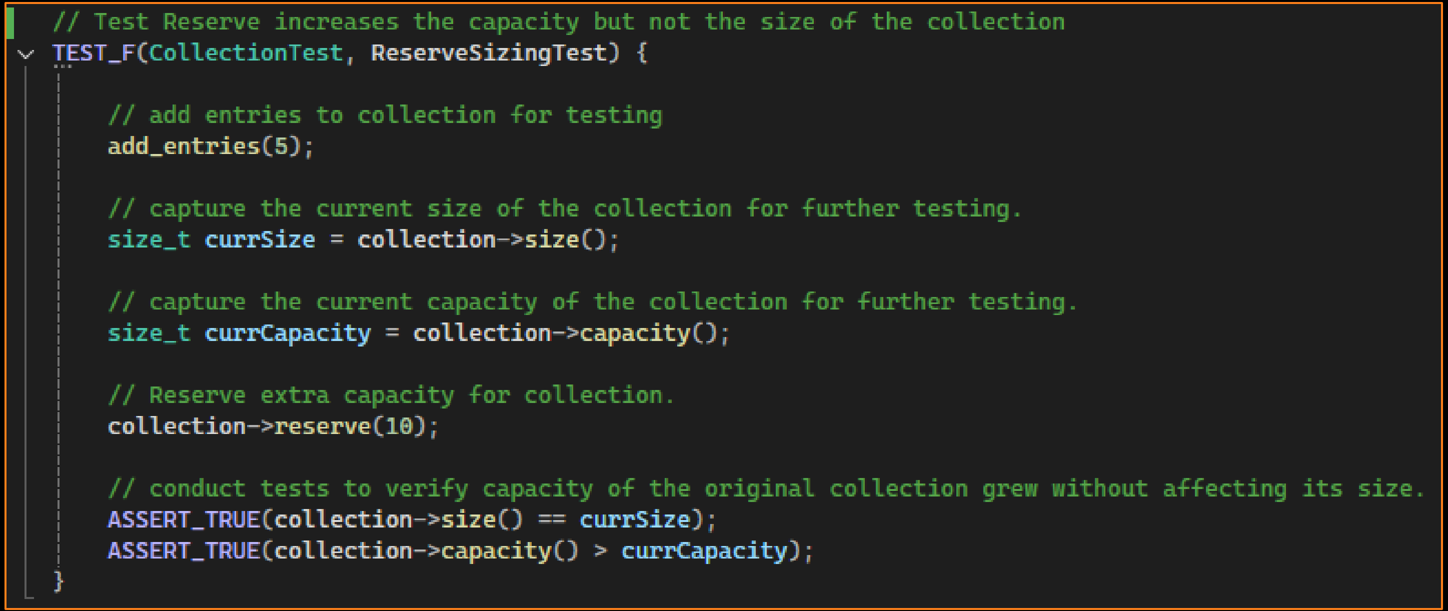
Green Pace

Unit Testing

Purpose: Testing for successful manipulation of vector capacity.

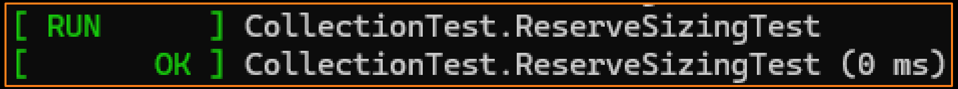
Coding vulnerability being tested: Vector object resource management and sizing/capacity manipulation.

Test Definition:



```
// Test Reserve increases the capacity but not the size of the collection
TEST_F(CollectionTest, ReserveSizingTest) {

    // add entries to collection for testing
    add_entries(5);

    // capture the current size of the collection for further testing.
    size_t currSize = collection->size();

    // capture the current capacity of the collection for further testing.
    size_t currCapacity = collection->capacity();

    // Reserve extra capacity for collection.
    collection->reserve(10);

    // conduct tests to verify capacity of the original collection grew without affecting its size.
    ASSERT_TRUE(collection->size() == currSize);
    ASSERT_TRUE(collection->capacity() > currCapacity);
}
```

Result:



```
[ RUN      ] CollectionTest.ReserveSizingTest
[       OK ] CollectionTest.ReserveSizingTest (0 ms)
```

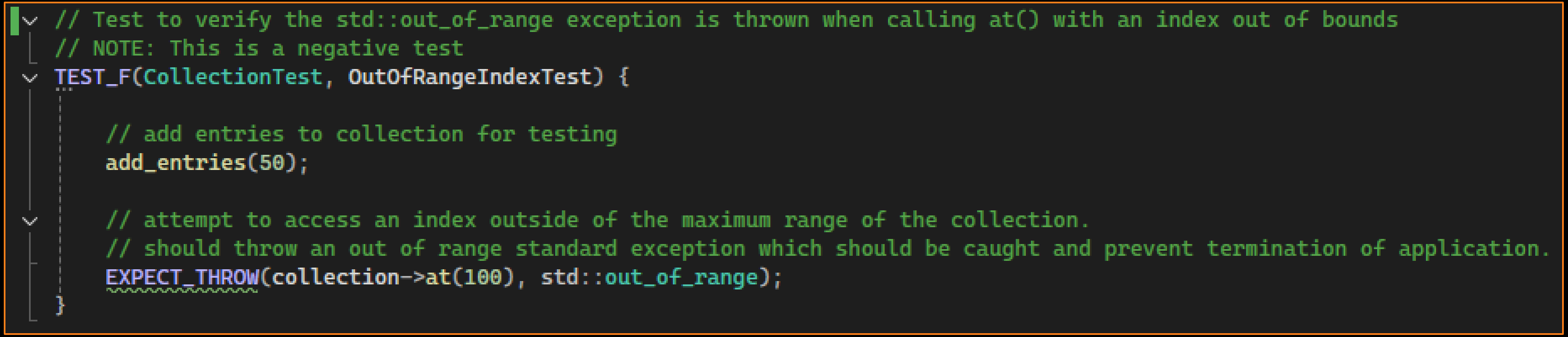
Green Pace

Unit Testing

Purpose: Testing for successful expansion of memory allocation.

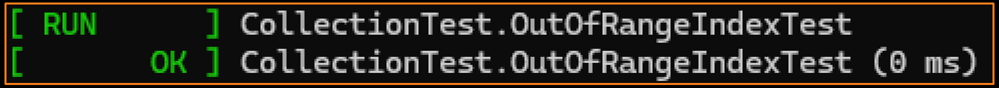
Coding vulnerability being tested: Test for appropriate behavior for error handling.

Test Definition:



```
// Test to verify the std::out_of_range exception is thrown when calling at() with an index out of bounds
// NOTE: This is a negative test
TEST_F(CollectionTest, OutOfRangeIndexTest) {

    // add entries to collection for testing
    add_entries(50);

    // attempt to access an index outside of the maximum range of the collection.
    // should throw an out of range standard exception which should be caught and prevent termination of application.
    EXPECT_THROW(collection->at(100), std::out_of_range);
}
```

Result:



```
[ RUN      ] CollectionTest.OutOfRangeIndexTest
[       OK ] CollectionTest.OutOfRangeIndexTest (0 ms)
```

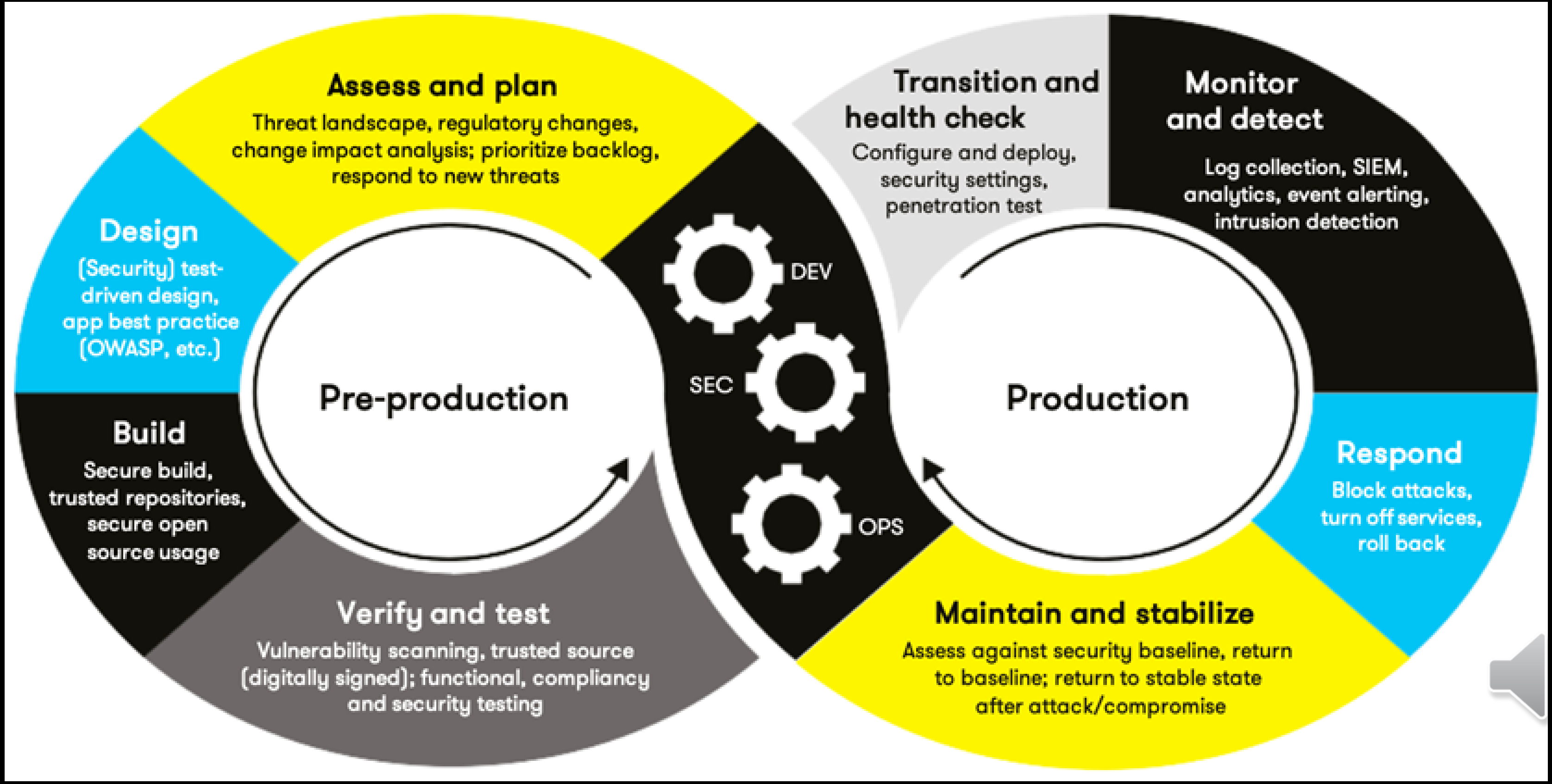
AUTOMATION SUMMARY

Security Tools and the DevSecOps Cycle:

Compiler: Used during the build and test phases where errors and warnings will assist in diagnosing code issues.

Encryption & Triple-A Frameworks: Implementation begins during assess, plan, and design phases to ensure proper data handling and authentication measures are in place where needed.

Unit Testing: Takes place during Verify and Test phase to validate committed code.


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
DEV
SEC
OPS
Transition and health check
Configure and deploy, security settings, penetration test
Monitor and detect
Log collection, SIEM, analytics, event alerting, intrusion detection
Production
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

Green Pace

TOOLS

- **The DevSecOps Pipeline:** Development framework that incorporates security within the development process flow.
  - **Pre-production phase:** Defines process steps during period where a project is yet to be deployed into a production environment. Steps such as planning, design, build, and testing take place during this phase. Each step in the development flow introduces security-centric considerations such as designing for threat landscapes, best practices and using industry-standard security protocols.
  - **Production Phase:** Continuous improvement takes place during the production phase where the project has been deployed. Threat responses, mitigation, and management takes place during this phase.

- **External Tools:**
  - **OWASP:** Industry-standard security policies for implementation during planning and design phase.
  - **Integrated Development Environments:** Automated tools such as error checking, and early-warning detection.
  - **Unit Tests and Static Code Analysis:** Tools for further validating the code base. Used for quality assurance.
  - **Logging:** Tools for monitoring the state of the system. Ensures sufficient information is available for tracking down and mitigating threats in a timely manner.

Green Pace

RISKS AND BENEFITS

Problem	**Risk**	**Solution & Steps to take**	**Benefit**
Security Policy Limited in Scope for Brevity	Principles and standards outside of the policy's scope are not as easily identifiable. Requires continuous learning and research.	Green Pace's development teams are encouraged to seek continuous improvement opportunities as a further shift in culture toward security-first.	New challenges are addressed in a timely manner with dynamic and adequate responses.
Immediate implementation or "wait and see" approach	Constantly changing threat landscape can hinder project progress and put existing systems at risk.	Prioritize security. Approach projects by setting a foundation that includes security to deliver projects effectively and securely the first time. With room for further improvement.	Time savings due to early detection and threat mitigation. Safe and secure deliverables.
Modern trends in development contributing to "hands-off" approach	Inadequate testing of project components causing vulnerabilities throughout the development process.	Adopt a "SecDevOps" culture and remain vigilant about new tools that attempt to abstract thorough testing away from the developer.	Tools benefit time constraints which can be used to test for further vulnerabilities.

Green Pace

RECOMMENDATIONS

- Defined coding standards have limited coverage compared to the vast scope of multiple programming languages and their risk vectors. Green Pace's team are advised to be vigilant when deploying code not explicitly covered by the standards.
- While Green Pace's security policy explicitly states 10 principles, it is to be considered a prioritization for the sake of brevity. Many security principles exist, and as such, teams should also carefully consider further principles as they relate to the project at hand.
- Automated tools listed in the security policy are recommendations based on Green Pace validation practices. However, teams are reminded that more tools exist outside of the scope of the documented policy. In the pursuit of security, teams are encouraged to seek out more effective tools and submit recommendations for their subsequent addition to the policy after proper validation.
- Modern development environments assist teams via automatic unit testing definitions. While the Green Pace security policy outlines use of unit testing, it is best practice to explicitly monitor the unit tests and ensure criteria is being met. Automated tests may only address simple criteria while leaving fringe cases out. Exercise caution with auto-generated test cases.

Green Pace

CONCLUSIONS

- Adherence to the Green Pace security policy is paramount to delivering safe and secure systems to our stakeholders the first time. With further emphasis on continuous improvement during the production phase.
- The threat landscape is ever-dynamic. Green Pace is committed to take a security-first approach to address vulnerabilities as swiftly and concisely as possible.
- Security is a mindset. Therefore, It should constitute the foundation of any Green Pace project at its inception. Security principles are to be incorporated with prioritization on-par with all other facets of the software development lifecycle.
- Green Pace's security policy is a living document, meant to transform to meet new threats where they are. Green Pace teams are required to familiarize themselves with updates to the policy when changes are submitted.

Green Pace

REFERENCES

JetBrains. (2016). *C++ Unit Testing with Google Test Tutorial*. YouTube. https://www.youtube.com/watch?v=16FI1-d2P4E

SEI. (n.d.). *Sei cert C++ coding standard*. SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence. https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682

Top 10 secure coding practices. Top 10 Secure Coding Practices - CERT Secure Coding - Confluence. (n.d.). https://wiki.sei.cmu.edu/confluence/display/seccode/Top+10+Secure+Coding+Practices?focusedCommentId=88044413

Green Pace