**Green Pace Developer: Security Policy Guide Template**



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

[C/C++ Ten Coding Standards 3](#_Toc52464058)

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

[Defense-in-Depth Illustration 15](#_Toc52464069)

[Project One 15](#_Toc52464070)

[1. Revise the C/C++ Standards 15](#_Toc52464071)

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

[5. Summary of Risk Assessments 16](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 16](#_Toc52464076)

[7. Map the Principles 17](#_Toc52464077)

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

[Approved C/C++ Language Acronyms 19](#_Toc52464085)

## Overview

Software development at Green Pace requires consistent implementation of secure principles to all developed applications. Consistent approaches and methodologies must be maintained through all policies that are uniformly defined, implemented, governed, and maintained over time.

## Purpose

This policy defines the core security principles; C/C++ coding standards; authorization, authentication, and auditing standards; and data encryption standards. This article explains the differences between policy, standards, principles, and practices (guidelines and procedure): [Understanding the Hierarchy of Principles, Policies, Standards, Procedures, and Guidelines](https://www.linkedin.com/pulse/understanding-hierarchy-principles-policies-standards-wally-beddoe/).

## Scope

This document applies to all staff that create, deploy, or support custom software at Green Pace.

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | Verifying incoming data from both trusted and untrusted sources to ensure no malicious data is going through the system. |
| 1. Heed Compiler Warnings | Compiler warnings could potentially lead to a more serious problem and must be taken seriously. Compile and test the code frequently to ensure the code is working and performing as expected. |
| 1. Architect and Design for Security Policies | Design and build code while meeting the standards and with security in mind. This will reduce the time it takes to find and fix any vulnerabilities in the code if they are found. |
| 1. Keep It Simple | Code should be easy and simple to understand. Keeping code simple means that if vulnerabilities are found they will be easier to trace and fix. |
| 1. Default Deny | Denying access to systems by default makes the system more security allow only those who need access to certain systems to have access to those systems when they go through the proper channels to gain access. |
| 1. Adhere to the Principle of Least Privilege | Least privilege goes along with default denial. Only those who need access to higher systems can gain access to those systems and only access to information within those systems will be given to those that truly need access to it. |
| 1. Sanitize Data Sent to Other Systems | Making sure that any data that is sent only contains the data that is needed, and that data is void of any vulnerabilities or sensitive information. |
| 1. Practice Defense in Depth | Add layers of redundant and varying security and protection to help safeguard a system from a breach. |
| 1. Use Effective Quality Assurance Techniques | Quality Assurance is the process to find as many vulnerabilities within the code as possible before any others can find them. Such as hackers or customers. |
| 1. Adopt a Secure Coding Standard | Secure coding standards envelope all above principles to the point that having them within the forefront of the policy reduces the overall time and money to test code and prevents future problems from arising. |

### C/C++ Ten Coding Standards

Complete the coding standards portion of the template according to the Module Three milestone requirements. In Project One, follow the instructions to add a layer of security to the existing coding standards. Please start each standard on a new page, as they may take up more than one page. The first seven coding standards are labeled by category. The last three are blank so you may choose three additional standards. Be sure to label them by category and give them a sequential number for that category. Add compliant and noncompliant sections as needed to each coding standard.

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Do not define a C-style variadic function. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example uses a C-style variadic function to add a series of integers together. The function reads arguments until the value 0 is found. Calling this function without passing the value 0 as an argument (after the first two arguments) results in undefined behavior. Furthermore, passing any type other than an int also results in undefined behavior. |
| [#include <cstdarg>    **int** add(**int** first, **int** second, ...) {  **int** r = first + second;  **va\_list** va;  **va\_start**(va, second);  **while** (**int** v = **va\_arg**(va, **int**)) {      r += v;    }  **va\_end**(va);  **return** r;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, a variadic function using a function parameter pack is used to implement the add() function, allowing identical behavior for call sites. Unlike the C-style variadic function used in the noncompliant code example, this compliant solution does not result in undefined behavior if the list of parameters is not terminated with 0. Additionally, if any of the values passed to the function are not integers, the code is ill-formed rather than producing undefined behavior. |
| #include <type\_traits>    **template** <**typename** Arg, **typename** std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr>  **int** add(Arg f, Arg s) { **return** f + s; }    **template** <**typename** Arg, **typename**... Ts, **typename** std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr>  **int** add(Arg f, Ts... rest) {  **return** f + add(rest...);  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** Use Effective Quality Assurance Techniques |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| HIGH | Probable | Medium | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Function-ellipsis | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++ | [Insert text.] |
| Clang | 3.9 | Cert | Checked by clang-tidy |
| CodeSonar | 7.0p0 | LANG.STRUCT.ELLIPSIS | Ellipsis |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not declare or define a reserved identifier |

| **Noncompliant Code** |
| --- |
| A common practice is to use a macro in a preprocessor conditional that guards against multiple inclusions of a header file. While this is a recommended practice, many programs use reserved names as the header guards. Such a name may clash with reserved names defined by the implementation of the C++ standard template library in its headers or with reserved names implicitly predefined by the compiler even when no C++ standard library header is included. |
| #ifndef \_MY\_HEADER\_H\_  #define \_MY\_HEADER\_H\_    // Contents of <my\_header.h>    #endif // \_MY\_HEADER\_H\_ |

| **Compliant Code** |
| --- |
| This compliant solution avoids using leading or trailing underscores in the name of the header guard. |
| #ifndef MY\_HEADER\_H  #define MY\_HEADER\_H    // Contents of <my\_header.h>    #endif // MY\_HEADER\_H |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** Use Effective Quality Assurance Techniques |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | reserved-identifier | Partially checked |
| Clang | 3.9 | -Wreserved-id-macro  -Wuser-defined-literals | The -Wresered-id-macro flag is nor enabled by default or with -Wall, but is enabled with -Weverything. This flag does not catch all instances of this rule, such as redefining reserved names. |
| CodeSonar | 7.0p0 | LANG.ID.NU.MK  LANG.STRUCT.DECL.RESERVED | Macro name is C keyword  Declaration of reserved name |
| Helix QAC | 2022.2 | C++5003 |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Guarantee that storage for strings has sufficient space for character data and the null terminator. |

| **Noncompliant Code** |
| --- |
| Because the input is unbounded, the following code could lead to a buffer overflow. |
| #include <iostream>    void f() {  char buf[12];  std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| The best solution for ensuring that data is not truncated and for guarding against buffer overflows is to use std::string instead of a bounded array, as in this compliant solution. |
| #include <string>    void f() {  std::string input;  std::string stringOne, stringTwo;  std::cin >> stringOne >> stringTwo;  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** ValidateInput Data, Architect and Design for Security Policies |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.0p0 | MISC.MEM.NTERM  LANG.MEM.BO  LANG.MEM.TO | No space for null terminator  Buffer overrun  Type overrun |
| Helix QAC | 2022.2 | C++2835, C++2836, C++2839, C++5216 |  |
| Klocwork | 2022.2 | NNTS.MIGHT  NNTS.TAINTED  NNTS.MUST  SV.UNBOUND\_STRING\_INPUT.CIN |  |
| LDRA tool suite | 9.7.1 | 489 S, 66 X, 70 X, 71 X | Partially implemented |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Close files when they are no longer needed. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a std::fstream object file is constructed. The constructor for std::fstream calls std::basic\_filebuf<T>::open(), and the default std::terminate\_handler called by std::terminate() is std::abort(), which does not call destructors. Consequently, the underlying std::basic\_filebuf<T> object maintained by the object is not properly closed. |
| #include <exception>  #include <fstream>  #include <string>    void f(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }  // ...  std::terminate();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, std::fstream::close() is called before std::terminate() is called, ensuring that the file resources are properly closed. |
| #include <exception>  #include <fstream>  #include <string>    void f(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }  // ...  file.close();  if (file.fail()) {  // Handle error  }  std::terminate();  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** Architect and Design for Security Policies, Sanitize Data Sent to Other Systems |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.0p0 | ALLOC.LEAK | Leak |
| Helix QAC | 2022.2 | C++4786,C++4787, C++4788 |  |
| Klocwork | 2022.2 | RH.LEAK |  |
| Parasoft Insure++ |  |  | Runtime detection |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Do not access freed memory |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, s is dereferenced after it has been deallocated. If this access results in a write-after-free, the vulnerability can be exploited to run arbitrary code with the permissions of the vulnerable process. Typically, dynamic memory allocations and deallocations are far removed, making it difficult to recognize and diagnose such problems. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  delete s;  // ...  s->f();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the dynamically allocated memory is not deallocated until it is no longer required. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  s->f();  delete s;  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** Use Effective Quality Assurance Techniques, Architect and Design for Security Policies |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Dangling\_pointer\_use |  |
| Clang | 3.9 | clang-analyzer-cplusplus.NewDelete  clang-analyzer-alpha.security.ArrayBoundV2 | Checked by clang-tidy, but does not catch all violations of this rule. |
| CodeSonar | 7.0p0 | ALLOC.UAF | Use after free |
| Coverity | V7.5.0 | USE\_AFTER\_FREE | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Prefer special member functions and overloaded operators to C Standard Library functions |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, std::memcmp() is used to compared two objects of nonstandard-layout type. Because std::memcmp() performs a bytewise comparison of the object representations, if the implementation uses a vtable pointer as part of the object representation, it will compare vtable pointers. If the dynamic type of either c1 or c2 is a derived class of type C, the comparison may fail despite the value representation of either object. |
| #include <cstring>    class C {  int i;    public:  virtual void f();    // ...  };    void f(C &c1, C &c2) {  if (!std::memcmp(&c1, &c2, sizeof(C))) {  // ...  }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, C defines an equality operator that is used instead of calling std::memcmp(). This solution ensures that only the value representation of the objects is considered when performing the comparison. |
| class C {  int i;    public:  virtual void f();    bool operator==(const C &rhs) const {  return rhs.i == i;  }    // ...  };    void f(C &c1, C &c2) {  if (c1 == c2) {  // ...  }  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** Use Effective Quality Assurance Techniques |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | High | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | stdlib-use-ato  stdlib-use  stdlib-use-getenv  stdlib-use-system  include-time  stdlib-use-string-unbounded | Partially checked |
| Helix QAC | 2022.2 | C++5017, C++5038 |  |
| LDRA tool suite | 9.7.1 | 44 S | Enhanced Enforcement |
| PRQA QA-C++ | 4.4 | 5017, 5038 |  |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Do not abruptly terminate the program |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the call to f(), which was registered as an exit handler with std::at\_exit(), may result in a call to std::terminate() because throwing\_func() may throw an exception. |
| #include <cstdlib>    void throwing\_func() noexcept(false);    void f() { // Not invoked by the program except as an exit handler.  throwing\_func();  }    int main() {  if (0 != std::atexit(f)) {  // Handle error  }  // ...  } |

| **Compliant Code** |
| --- |
| In this compliant solution, f() handles all exceptions thrown by throwing\_func() and does not rethrow. |
| #include <cstdlib>    void throwing\_func() noexcept(false);    void f() { // Not invoked by the program except as an exit handler.  try {  throwing\_func();  } catch (...) {  // Handle error  }  }    int main() {  if (0 != std::atexit(f)) {  // Handle error  }  // ...  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** Use Effective Quality Assurance Techniques |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | Medium | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | stdlib-use | Partially checked |
| CodeSonar | 7.0p0 | BADFUNC.ABORT  BADFUNC.EXIT | Use of abort  Use of exit |
| Helix QAC | 2022.2 | C++5014 |  |
| LDRA tool suite | 9.7.1 | 122 S | Enhanced Enforcement |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Concurrency** | [STD-008-CPP] | Wrap functions that can spuriously wake up in a loop. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example monitors a linked list and assigns one thread to consume list elements when the list is nonempty.  This thread pauses execution using wait() and resumes when notified, presumably when the list has elements to be consumed. It is possible for the thread to be notified even if the list is still empty, perhaps because the notifying thread used notify\_all(), which notifies all threads. Notification using notify\_all() is frequently preferred over using notify\_one(). (See STD-009-CPP. Preserve thread safety and liveness when using condition variables for more information.)  A condition predicate is typically the negation of the condition expression in the loop. In this noncompliant code example, the condition predicate for removing an element from a linked list is (list->next != nullptr), whereas the condition expression for the while loop condition is (list->next == nullptr).  This noncompliant code example nests the call to wait() inside an if block and consequently fails to check the condition predicate after the notification is received. If the notification was spurious or malicious, the thread would wake up prematurely. |
| #include <condition\_variable>  #include <mutex>    struct Node {  void \*node;  struct Node \*next;  };    static Node list;  static std::mutex m;  static std::condition\_variable condition;    void consume\_list\_element(std::condition\_variable &condition) {  std::unique\_lock<std::mutex> lk(m);    if (list.next == nullptr) {  condition.wait(lk);  }    // Proceed when condition holds.  } |

| **Compliant Code** |
| --- |
| This compliant solution calls the wait() member function from within a while loop to check the condition both before and after the call to wait(). |
| #include <condition\_variable>  #include <mutex>    struct Node {  void \*node;  struct Node \*next;  };    static Node list;  static std::mutex m;  static std::condition\_variable condition;    void consume\_list\_element() {  std::unique\_lock<std::mutex> lk(m);    while (list.next == nullptr) {  condition.wait(lk);  }    // Proceed when condition holds.  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** Architect and Design for Security Policies, Use Effective Quality Assurance Techniques |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.0p0 | LANG.STRUCT.ICOL  CONCURRENCY.BADFUNC.CNDWAIT | Inappropriate Call Outside Loop  Use of Condition Variable Wait |
| Helix QAC | 2022.2 | C++5019 |  |
| Klocwork | 2022.2 | CERT.CONC.WAKE\_IN\_LOOP |  |
| PRQA QA-C++ | 4.4 | 5019 |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Concurrency** | [STD-009-CPP] | Preserve thread safety and liveness when using condition variables. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example uses five threads that are intended to execute sequentially according to the step level assigned to each thread when it is created (serialized processing). The currentStep variable holds the current step level and is incremented when the respective thread completes. Finally, another thread is signaled so that the next step can be executed. Each thread waits until its step level is ready, and the wait() call is wrapped inside a while loop, in compliance with STD-008-CPP. Wrap functions that can spuriously wake up in a loop. |
| #include <condition\_variable>  #include <iostream>  #include <mutex>  #include <thread>    std::mutex mutex;  std::condition\_variable cond;    void run\_step(size\_t myStep) {  static size\_t currentStep = 0;  std::unique\_lock<std::mutex> lk(mutex);    std::cout << "Thread " << myStep << " has the lock" << std::endl;    while (currentStep != myStep) {  std::cout << "Thread " << myStep << " is sleeping..." << std::endl;  cond.wait(lk);  std::cout << "Thread " << myStep << " woke up" << std::endl;  }    // Do processing...  std::cout << "Thread " << myStep << " is processing..." << std::endl;  currentStep++;    // Signal awaiting task.  cond.notify\_one();    std::cout << "Thread " << myStep << " is exiting..." << std::endl;  }    int main() {  constexpr size\_t numThreads = 5;  std::thread threads[numThreads];    // Create threads.  for (size\_t i = 0; i < numThreads; ++i) {  threads[i] = std::thread(run\_step, i);  }    // Wait for all threads to complete.  for (size\_t i = numThreads; i != 0; --i) {  threads[i - 1].join();  }  } |

| **Compliant Code** |
| --- |
| This compliant solution uses notify\_all() to signal all waiting threads instead of a single random thread. Only the run\_step() thread code from the noncompliant code example is modified. |
| #include <condition\_variable>  #include <iostream>  #include <mutex>  #include <thread>    std::mutex mutex;  std::condition\_variable cond;    void run\_step(size\_t myStep) {  static size\_t currentStep = 0;  std::unique\_lock<std::mutex> lk(mutex);    std::cout << "Thread " << myStep << " has the lock" << std::endl;    while (currentStep != myStep) {  std::cout << "Thread " << myStep << " is sleeping..." << std::endl;  cond.wait(lk);  std::cout << "Thread " << myStep << " woke up" << std::endl;  }    // Do processing ...  std::cout << "Thread " << myStep << " is processing..." << std::endl;  currentStep++;    // Signal ALL waiting tasks.  cond.notify\_all();    std::cout << "Thread " << myStep << " is exiting..." << std::endl;  }    // ... main() unchanged ... |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** Architect and Design for Security Policies, Use Effective Quality Assurance Techniques |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.0p0 | CONCURRENCY.BANFUNC.CNDSIGNAL | Use of Condition Variable Signal |
| Helix QAC | 2022.2 | C++1778, C++1779 |  |
| Klocwork | 2022.2 | CERT.CONC.UNSAFE\_COND\_VAR |  |
| PRQA QA-C++ | 4.4 | 5020 |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exception** | [STD-010-CPP] | Catch handlers should order their parameter types from most derived to least derived. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the first handler catches all exceptions of class B, as well as exceptions of class D, since they are also of class B. Consequently, the second handler does not catch any exceptions. |
| // Classes used for exception handling  class B {};  class D : public B {};    void f() {  try {  // ...  } catch (B &b) {  // ...  } catch (D &d) {  // ...  }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the first handler catches all exceptions of class D, and the second handler catches all the other exceptions of class B. |
| // Classes used for exception handling  class B {};  class D : public B {};    void f() {  try {  // ...  } catch (D &d) {  // ...  } catch (B &b) {  // ...  }  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** Use Effective Quality Assurance Techniques |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Likely | Low | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Exception-caught-by-earlier-handler | Fully checked |
| Clang | 3.9 | -Wexceptions |  |
| CodeSonar | 7.0p0 | LANG.STRUCT.UCTCH | Unreachable Catch |
| Helix QAC | 2022.2 | C++4030, C++4639 |  |

### Defense-in-Depth Illustration

This illustration provides a visual representation of the defense-in-depth best practice of layered security.



## Project One

There are seven steps outlined below that align with the elements you will be graded on in the accompanying rubric. When you complete these steps, you will have finished the security policy.

### Revise the C/C++ Standards

You completed one of these tables for each of your standards in the Module Three milestone. In Project One, add revisions to improve the explanation and examples as needed. Add rows to accommodate additional examples of compliant and noncompliant code. Coding standards begin on the security policy.

### Risk Assessment

Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority, and level using the values provided in the appendix.

### Automated Detection

Complete this section of each table on the coding standards to show the tools that may be used to detect issues. Provide the tool name, version, checker, and description. List one or more tools that can automatically detect this issue and its version number, name of the rule or check (preferably with link), and any relevant comments or description—if any. This table ties to a specific C++ coding standard.

### Automation

Provide a written explanation using the image provided.



Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Define guidance on where and how to modify the existing DevOps process to automate enforcement of the standards in this policy. Use the DevSecOps diagram and provide an explanation using that diagram as context.

### DevSecOps should be standard practice during all stages of the SDLC. Assess and plan are the first step. Security should always be at the forefront and a plan should be created. Design is the second step, where security tools are selected. The build phase focuses on automated security analysis. Developers need to be aware of third-party code dependencies and untrusted sources during this phase. The test phase uses dynamic testing to detect user authentication, authorization, and SQL injection to detect live flaws. The transition and health check phase, is where the code should be tested, configured, and deployed with security protocols. The Principle of Least Privilege concern needs to be addressed during this phase to ensure access is limited to owners and designated users. Respond phase blocks attacks and neutralize threats. In the event of any attack, use logs and assess the damage, and roll back to a state prior to the attack.

### Summary of Risk Assessments

Consolidate all risk assessments into one table including both coding and systems standards, ordered by standard number.

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Probable | Medium | P12 | L1 |
| STD-002-CPP | Low | Unlikely | Low | P3 | L3 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CPP | High | Probable | High | P6 | L2 |
| STD-007-CPP | Low | Probable | Medium | P4 | L3 |
| STD-008-CPP | Low | Unlikely | Medium | P2 | L3 |
| STD-009-CPP | Low | Unlikely | Medium | P2 | L3 |
| STD-010-CPP | Medium, | Likely | Low | P18 | L1 |

### Create Policies for Encryption and Triple A

Include all three types of encryption (in flight, at rest, and in use) and each of the three elements of the Triple-A framework using the tables provided***.***

* 1. Explain each type of encryption, how it is used, and why and when the policy applies.
  2. Explain each type of Triple-A framework strategy, how it is used, and why and when the policy applies.

Write policies for each and explain what it is, how it should be applied in practice, and why it should be used.

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption at rest refers to encrypting data on the disk to prevent malicious access. |
| Encryption at flight | Encryption at flight refers to the data being transferred or transmitted. Transmitted data needs to be encrypted otherwise there is a risk of sensitive data being exposed |
| Encryption in use | Encryption in use refers to ensuring that all sensitive data is always encrypted. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is verifying the identity of the users and only granting them access. |
| Authorization | Authorization is granting an authenticated user the proper privileges. |
| Accounting | Accounting measures the resources the users consume while they have access to the system. This consists of time spent on the system and data sent or received during that session. |

**\***Use this checklist for the Triple A to be sure you include these elements in your policy:

* User logins
* Changes to the database
* Addition of new users
* User level of access
* Files accessed by users

### Map the Principles

Map the principles to each of the standards, and provide a justification for the connection between the two. In the Module Three milestone, you added definitions for each of the 10 principles provided. Now it’s time to connect the standards to principles to show how they are supported by principles. You may have more than one principle for each standard, and the principles may be used more than once. Principles are numbered 1 through 10. You will list the number or numbers that apply to each standard, then explain how each of these principles supports the standard. This exercise demonstrates that you have based your security policy on widely accepted principles. Linking principles to standards is a best practice.

**NOTE:** Green Pace has already successfully implemented the following:

* Operating system logs
* Firewall logs
* Anti-malware logs

The only item you must complete beyond this point is the Policy Version History table.

## Audit Controls and Management

Every software development effort must be able to provide evidence of compliance for each software deployed into any Green Pace managed environment.

Evidence will include the following:

* Code compliance to standards
* Well-documented access-control strategies, with sampled evidence of compliance
* Well-documented data-control standards defining the expected security posture of data at rest, in flight, and in use
* Historical evidence of sustained practice (emails, logs, audits, meeting notes)

## Enforcement

The office of the chief information security officer (OCISO) will enforce awareness and compliance of this policy, producing reports for the risk management committee (RMC) to review monthly. Every system deployed in any environment operated by Green Pace is expected to be in compliance with this policy at all times.

Staff members, consultants, or employees found in violation of this policy will be subject to disciplinary action, up to and including termination.

## Exceptions Process

Any exception to the standards in this policy must be requested in writing with the following information:

* Business or technical rationale
* Risk impact analysis
* Risk mitigation analysis
* Plan to come into compliance
* Date for when the plan to come into compliance will be completed

Approval for any exception must be granted by chief information officer (CIO) and the chief information security officer (CISO) or their appointed delegates of officer level.

Exceptions will remain on file with the office of the CISO, which will administer and govern compliance.

## Distribution

This policy is to be distributed to all Green Pace IT staff annually. All IT staff will need to certify acceptance and awareness of this policy annually.

## Policy Change Control

This policy will be automatically reviewed annually, no later than 365 days from the last revision date. Further, it will be reviewed in response to regulatory or compliance changes, and on demand as determined by the OCISO.

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

| Language | Acronym |
| --- | --- |
| C++ | CPP |
| C | CLG |
| Java | JAV |