GARMIN

Changing Legacy Code with Confidence:

Practical Advice for Maintaining and Migrating Working Code

Brian Ruth (he/him) brian.ruth@garmin.com

Part 0: Legacy Code

GARMIN®

Legacy code has value

- It works*
- It may not have tests, but has been tested
- Documents all the decisions and problems encountered in its lifetime
- It is currently making the company money
- One person's bug is another's critical feature

*well enough, for some definition of "works"

GARMIN.

The Legacy of Code

- Best practices and languages change
- Developers cycle through projects
- Features added/removed/re-added changed
- New developers may not have a firm understanding of project structure*
- Supporting tools change

*for some definition of "structure"

GARMIN.

Goals of this talk

- Preserve working code
- Introduce testing
- Gradual improvement
- Make code self documenting
- Instill confidence in changes
- Develop for maintenance

*for some definition of "structure"

GARMIN.

The Cost of Testing

GARMIN®

Whether or not you write an automated test, you need to make sure the code you are writing works.

GARMIN.

Testing Strategies

Test Everything
on every feature

GARMIN®

Testing Strategies

Test Everything
on every feature

GARMIN®

Testing Strategies

Test Everything
on every feature

Test the change
you just made

GARMIN.

Testing Strategies

Test Everything
on every feature

Test the change
you just made

Write Automated
tests and run them
continuously

GARMIN.

Testing Strategies

Test Everything
on every feature

Test the change
you just made

Write Automated
tests and run them
continuously

GARMIN.

Testing Strategies

Test Everything
on every feature

Test the change
you just made

Write Automated
tests and run them
continuously

GARMIN.

Testing Strategies

Test Everything on every feature

Test the change you just made

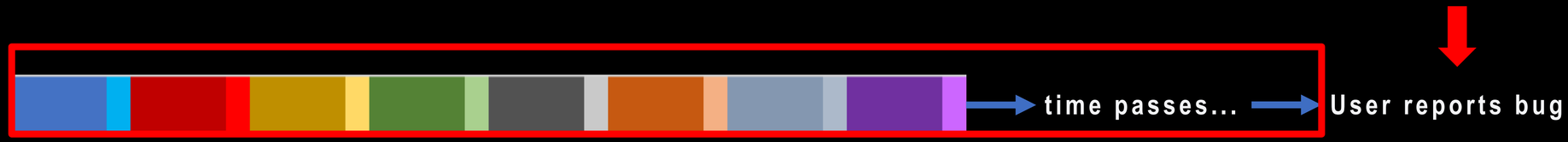
Write Automated tests and run them continuously

GARMIN.

Testing Strategies

Test Everything
on every feature

Test the change
you just made



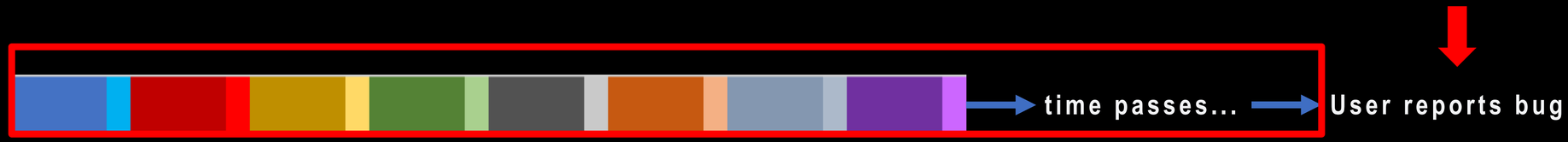
Write Automated
tests and run them
continuously

GARMIN.

Testing Strategies

Test Everything on every feature

Test the change you just made



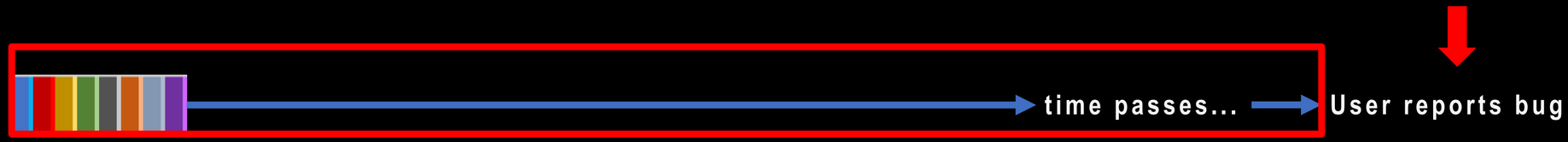
Write Automated tests and run them continuously

GARMIN.

Testing Strategies

Test Everything
on every feature

Test the change
you just made



Write Automated
tests and run them
continuously

GARMIN.

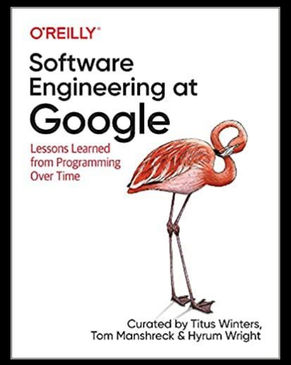
Automated testing...

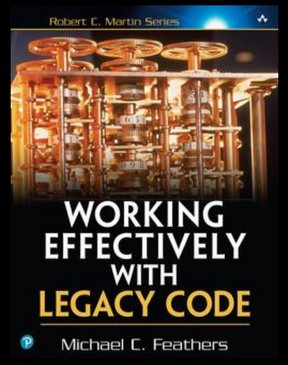
- reduces development time[1]
- reduces field defects[2]
- reduces costs over project lifetime[3]

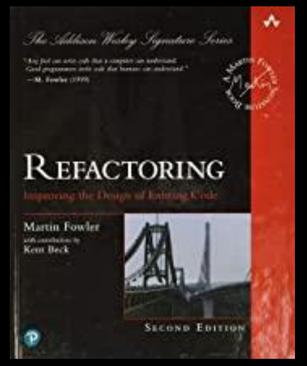
1. https://tinyurl.com/vkj8tbxu
2. https://tinyurl.com/4nkrt5vv
3. https://tinyurl.com/3ccxscrs

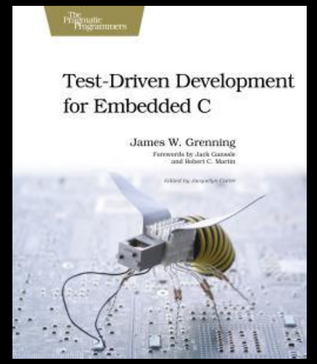
GARMIN®

There are many resources available









accu
professionalism in programming

IEEE

acm Association for Computing Machinery

GARMIN.

Keep in mind...

- This requires determination
- Code performance may be sacrificed
- Best practices will be violated
- You will fix other people's code
- You will own the changes you make
- Soft skills required
- Everything is a judgement call

GARMIN.

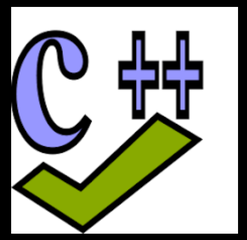
- This requires determination
- Code performance may be sacrificed
- Best practices will be violated
- You will fix other people's code
- You will own the changes you make
- Soft skills required
- Everything is a judgement call
- This never ends

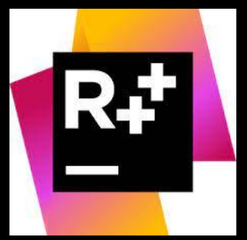
GARMIN.

Can't we just use tools?

GARMIN®

Many Tools Exist





GARMIN.

However...

- Not everyone will use them
- Not all code will work with them
- They require maintenance too
- Can be intrusive or leave artifacts in code
- Not available on all platforms
- Understand what the tools are automating

GARMIN®

Part I: Getting Started

Add A Test

GARMIN®

Add A Test: Hijack a Function

GARMIN®

Complicated Dependencies

```
class ActiveAssetsPage : public UI::Page {
 public:
   ActiveAssetsPage(UI::Page& parent, SQLDB dbId);
   void OnShow() override;
 private:
   UIList mAssetList
   std::vector<Asset> mFilteredAssets;
   AssetDatabase mDB;
};
```

GARMIN.

Complicated Dependencies

```
class ActiveAssetsPage : public UI::Page {
 public:
   ActiveAssetsPage(UI::Page& parent, SQLDB dbId);
   const Asset& GetNextAsset(AssetId id, AssetFilter filter) const;
   void OnShow() override;
 private:
   UIList mAssetList
   std::vector<Asset> mFilteredAssets;
   AssetDatabase mDB;
};
```

GARMIN.

Complicated Dependencies

```
class ActiveAssetsPage : public UI::Page {
 public:
   ActiveAssetsPage(UI::Page& parent, SQLDB dbId);
   const Asset& GetNextAsset(AssetId id, AssetFilter filter) const;
   void OnShow() override;
 private:
   UIList mAssetList
   std::vector<Asset> mFilteredAssets;
   AssetDatabase mDB;
};
```

GARMIN.

Complicated Dependencies

```
class ActiveAssetsPage : public UI::Page {
 public:
   ActiveAssetsPage(UI::Page& parent, SQLDB dbId);
   const Asset& GetNextAsset(AssetId id, AssetFilter filter) const;
   void OnShow() override;
 private:
   UIList mAssetList
   std::vector<Asset> mFilteredAssets;
   AssetDatabase mDB;
};
```

```
void ActiveAssetsPage::OnShow() {
 #if ENABLE_TEST
 TestGetNextAsset(*this, mDB);
 #else
  //normal UI OnShow stuff
 #endif
}
```

GARMIN

Complicated Dependencies

```
#if ENABLE_TEST
static void TestGetNextAsset(ActiveAssetPage& page, AssetDatabase& db) {
  const auto print_assert = ()(bool test, int line) {
    std::cout << (test ? "PASSED" : "FAILED) << " on line " << line;
  };
```

GARMIN®

Complicated Dependencies

```
#if ENABLE_TEST
static void TestGetNextAsset(ActiveAssetPage& page, AssetDatabase& db) {
  const auto print_assert = ()(bool test, int line) {
    std::cout << (test ? "PASSED" : "FAILED) << " on line " << line;
  };

  db.clear();
  print_assert(INVALID_ID == page.GetNextAssetInList( 0_id, AT::All ).id, __LINE__ );
```

GARMIN.

Complicated Dependencies

```
#if ENABLE_TEST
static void TestGetNextAsset(ActiveAssetPage& page, AssetDatabase& db) {
  const auto print_assert = ()(bool test, int line) {
    std::cout << (test ? "PASSED" : "FAILED) << " on line " << line;
  };

  db.clear();
  print_assert(INVALID_ID == page.GetNextAssetInList( 0_id, AT::All ).id, __LINE__ );

  Asset a1(0_id);
  a.type = AT::Dog;
  db.AddAsset(a1);
  print_assert(0_id == page.GetNextAssetInList( 0_id, AT::All ).id, __LINE__ );
  print_assert(INVALID_ID == page.GetNextAssetInList(0_id, AT::Cat), __LINE__);
  print_assert(0_id == page.GetNextAssetInList(0_id, AT::Dog), __LINE__);
```

GARMIN®

Complicated Dependencies

```
#if ENABLE_TEST
static void TestGetNextAsset(ActiveAssetPage& page, AssetDatabase& db) {
  const auto print_assert = ()(bool test, int line) {
    std::cout << (test ? "PASSED" : "FAILED) << " on line " << line;
  };

  db.clear();
  print_assert(INVALID_ID == page.GetNextAssetInList( 0_id, AT::All ).id, __LINE__ );

  Asset a1(0_id);
  a.type = AT::Dog;
  db.AddAsset(a1);
  print_assert(0_id == page.GetNextAssetInList( 0_id, AT::All ).id, __LINE__ );
  print_assert(INVALID_ID == page.GetNextAssetInList(0_id, AT::Cat), __LINE__);
  print_assert(0_id == page.GetNextAssetInList(0_id, AT::Dog), __LINE__);

  for(AssetId i = 1_id; i < 20_id; i++) {
    Asset a(i);
    a.type = AT::Cat;
    db.AddAsset(i);
  }

  print_assert(1_id == page.GetNextAssetInList( 0_id, AT::All ).id, __LINE__ );
  print_assert(1_id == page.GetNextAssetInList( 0_id, AT::Cat ).id, __LINE__ );
  print_assert(2_id == page.GetNextAssetInList( 1_id, AT::Cat ).id, __LINE__ );
  print_assert(1_id == page.GetNextAssetInList( 19_id, AT::Cat ).id, __LINE__ );
  print_assert(0_id == page.GetNextAssetInList( 19_id, AT::All ).id, __LINE__ );
  print_assert(0_id == page.GetNextAssetInList( 19_id, AT::Dog ).id, __LINE__ );
  print_assert(0_id == page.GetNextAssetInList( 1_id, AT::Dog ).id, __LINE__ );
}
#endif
```

GARMIN®

Discovery Testing

What does this function do?

```
proc_rf_msg(buff, true, false, 7);
```

```
proc_rf_msg(uint8* buff, bool is_full, bool is_actvty, size_t sz);
```

GARMIN®

Discovery Testing

```
void hndl_new_pkt(rf_pkt* pkt, size_t size) {

...

proc_rf_msg(buff, true, false, 7);
```

GARMIN®

Discovery Testing

```
void hndl_new_pkt(rf_pkt* pkt, size_t size) {

...

//proc_rf_msg(buff, true, false, 7);
```

GARMIN®

Discovery Testing

```
void hndl_new_pkt(rf_pkt* pkt, size_t size) {

...

//proc_rf_msg(buff, true, false, 7);
proc_rf_msg(nullptr, false, false, 0);
```

GARMIN®

Discovery Testing

```
void hndl_new_pkt(rf_pkt* pkt, size_t size) {

...

//proc_rf_msg(buff, true, false, 7);
proc_rf_msg(nullptr, false, false, 0);

activity activity;
activity.start_loc = {32.2226, 110.9747};
activity.end_loc = {39.8175, 104.7509};
proc_rf_msg(&activity, true, true, sizeof(activity)};
...
}
```

GARMIN®

Discovery Testing

```
void hndl_new_pkt(rf_pkt* pkt, size_t size) {

...

//proc_rf_msg(buff, true, false, 7);
proc_rf_msg(nullptr, false, false, 0);

activity activity;
activity.start_loc = {32.2226, 110.9747};
activity.end_loc = {39.8175, 104.7509};
proc_rf_msg(&activity, true, true, sizeof(activity)};
...
}

void proc_rf_msg(void* buf, bool is_verified, bool is_activity, size_t buf_size) {
...

if(is_activity) {
 activity* activ = (activity)buf;
 int activity_distance = 0;
 //convoluted code doing something with distance
 //complicated code to calculate elevation
 // incomprehensible code to calculate calories burned
 activ.calories = kcals;
 }
 ...
}
```

GARMIN®

Discovery Testing

```
static int test_distance = 0;

void hndl_new_pkt(rf_pkt* pkt, size_t size) {
...
//proc_rf_msg(buff, true, false, 7);
proc_rf_msg(nullptr, false, false, 0);

activity activity;
activity.start_loc = {32.2226, 110.9747};
activity.end_loc = {39.8175, 104.7509};
proc_rf_msg(&activity, true, true, sizeof(activity)};
...
}

void proc_rf_msg(void* buf, bool is_verified, bool is_activity, size_t buf_size) {
...

if(is_activity) {
 activity* activ = (activity)buf;
 int activity_distance = 0;
 //convoluted code doing something with distance
 test_distance = dist;
 //complicated code to calculate elevation
 // incomprehensible code to calculate calories burned
 activ.calories = kcals;
 }
 ...
}
```

GARMIN.

Discovery Testing

```
static int test_distance = 0;

void hndl_new_pkt(rf_pkt* pkt, size_t size) {
...
//proc_rf_msg(buff, true, false, 7);
proc_rf_msg(nullptr, false, false, 0);

activity activity;
activity.start_loc = {32.2226, 110.9747};
activity.end_loc = {39.8175, 104.7509};
proc_rf_msg(&activity, true, true, sizeof(activity)};
if(test_distance != 1012) {
 printf("FAILED test_distance: %d", test_distance);
}
...
}

void proc_rf_msg(void* buf, bool is_verified, bool is_activity, size_t buf_size) {
...

if(is_activity) {
 activity* activ = (activity)buf;
 int activity_distance = 0;
 //convoluted code doing something with distance
 test_distance = dist;
 //complicated code to calculate elevation
 // incomprehensible code to calculate calories burned
 activ.calories = kcals;
 }
 ...
}
```

GARMIN.

Add A Test: A New main

GARMIN.

Create a new build target

- Create a location to store tests, something like PRJ/TESTS
- Create a test package and new build manifest based on the simulator manifest

```
<manifest
  name="unit-test-only"
  version="1.0"
  format-version="7">
    <include path="./manifest-simulator.pkg.xml"/>
    <include path="../../PRJ/TESTS/tests.pkg.xml"/>
</manifest>
```

- Create a build flag like PRD_UNIT_TEST_BUILD to enable testing

GARMIN®

Create a new build target (continued)

- Enable the PRD_UNIT_TEST_BUILD flag and copy all the flags from the simulator
- Add a new build configuration copied from the simulator to the project's INI file and point to the test files

```
[simulator]
sys.cobra          = configs/simulator.flags.cfg

[unit-tests]
parent             = simulator
sys.target         = test
sys.config         = configs/test.flags.cfg
sys.pkgtool        = PRD//manifest-test.pkg.xml
```

- At this point, test that we can build the test product

GARMIN®

Add the test framework

- Download your favorite test framework
- And to the unit test package manifest

```
<manifest
  name="unit-test-only"
  version="1.0"
  format-version="7">
    <include path="./manifest-simulator.pkg.xml"/>
    <include
      path="../../modules/external/googletest/googletest.pkg.xml"
      scope="public"/>
    <include path="../../PRJ/TESTS/tests.pkg.xml"/>
</manifest>
```

- And set appropriate configuration flags

```
PARENT CONFIG simulator.flags.cfg
OUTPUT PATH
XML PATH ..\..\PRJ\TESTS\tests.config.xml
XML PATH ..\..\modules\external\googletest\googletest.config.xml

GOOGLEMOCK_CONFIG_ENABLED                          true          FALSE
GOOGLETEST_CONFIG_ENABLED                          false         TRUE
GOOGLETEST_CONFIG_USE_ENTRY_POINT                  false         FALSE
PRD_UNIT_TEST_BUILD                                false         TRUE
```

GARMIN®

Hijack the Simulator's main

- Conditionally remove the simulator's main from the test build's manifest

```
<!-- need to exclude the main function from the simulator since the test suite has its own -->
<if condition="not defined PRD_UNIT_TEST_BUILD or PRD_UNIT_TEST_BUILD == FALSE">
  <source path='main.cpp' />
</if>
```

- Add a new main for the testing framework

```
#include <gtest/gtest.h>

// This class is setup and torn down once for the
// entire test suite
class TestEnvironment : public ::testing::Environment {
 public:
  void SetUp() override {}

  void TearDown() override {}
};

int main(int argc, char** argv) {
  ::testing::InitGoogleTest(&argc, argv);
  ::testing::AddGlobalTestEnvironment(new TestEnvironment);
  return RUN_ALL_TESTS();
}
```

GARMIN.

Our First Test

- **Add a test file with a failing test**

PRJ/TESTS/FirstTest.cpp

```
#include <gtest/gtest.h>

TEST(FirstTest, DoesThisCompile) {
  EXPECT_TRUE(false);
}
```

- **Build our test**

```
$> waf --product=unit_tests --mode=development
```

- **Run our test**

```
$> out/unit-tests/development/test.exe
```

GARMIN.

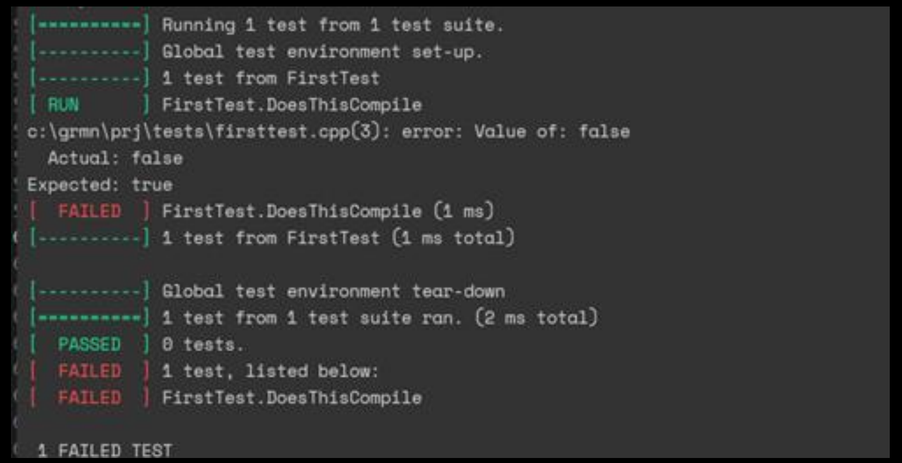
Our First Test

- **Add a test file with a failing test**

PRJ/TESTS/FirstTest.cpp

```
#include <gtest/gtest.h>

TEST(FirstTest, DoesThisCompile) {
  EXPECT_TRUE(false);
}
```



- **Build our test**

```
$> waf --product=unit_tests --mode=development
```

- **Run our test**

```
$> out/unit-tests/development/test.exe
```

GARMIN.

Add A Test: Bottom Up

GARMIN®

Start out similarly to A New Main

- Create an empty build configuration
- Add the PRD_UNIT_TEST_BUILD build flag
- Add the test framework

GARMIN®

Separate out Testable from Non-Testable

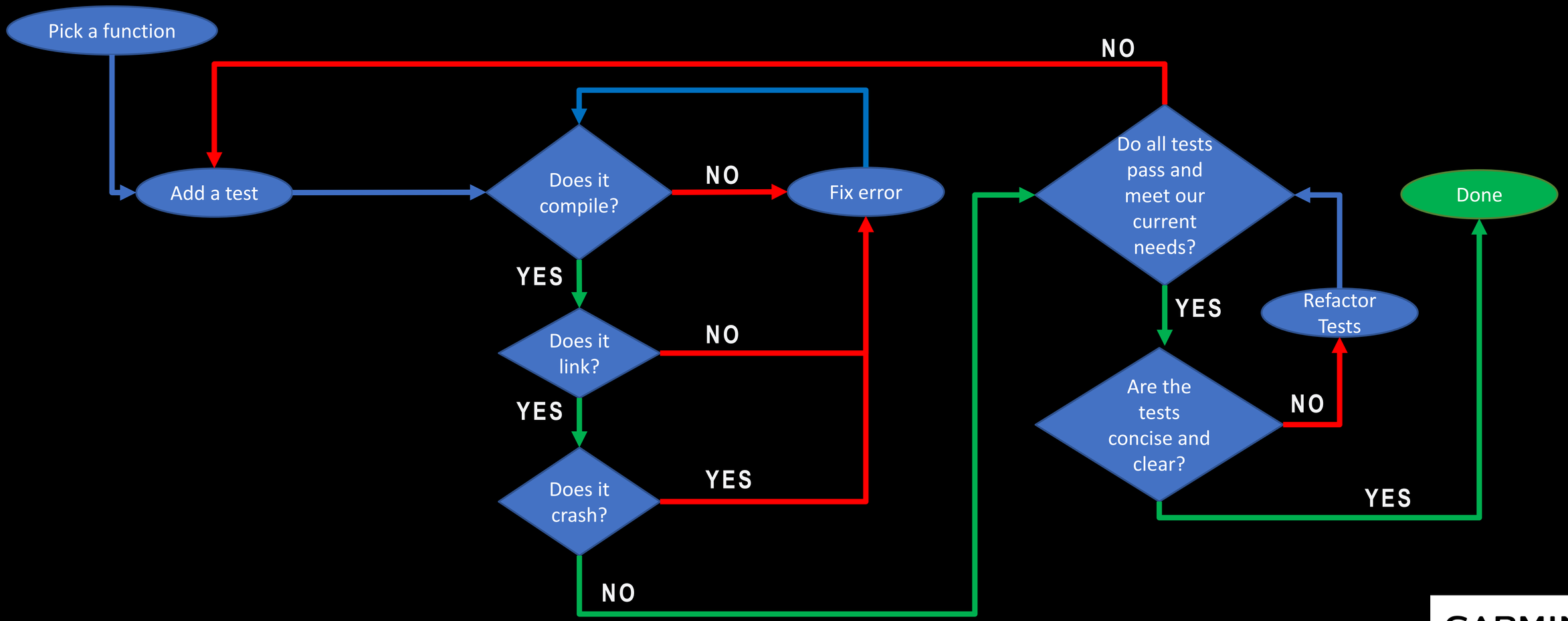
```
<!-- All testable translation units go here -->
<!-- UNIT_TEST_BUILD == TRUE -->

<source path ='share_code.cpp' />

<!-- All un-testable translation units go here -->
<ifeq name="UNIT_TEST_BUILD" value="FALSE">
 <!-- Everything else -->
</ifeq>
```

GARMIN®

Bottom Up Testing Workflow



GARMIN

Part II: Dealing with Dependencies

GARMIN®

Add them to the test project

- Isolated
  - avoid cascading dependencies
- Light-weight
  - avoid long running stuff like file or network access

GARMIN.

Create Type Only Headers

Header_to_test.hpp

```
#include "MLB_pub.hpp"

class CodeToTest {
  public:
    void SetModel( MLB::Model aModel);
    MLB::Model GetModel() const;

  private:
    MLB::Model mModel;
};
```

MLB_pub.hpp

```
#include "GPSReceiver.hpp"
#include "XMLParser.hpp"
// many more includes

namespace MLB{
// 200 lines of definitions, function declarations

  enum Model {
    AB_2020,
    TT_2251
    ...
  };

// and more code...
}
```

GARMIN®

Create Type Only Headers

Header_to_test.hpp

```
#include "MLB_pub_types.hpp"

class CodeToTest {
  public:
    void SetModel( MLB::Model aModel);
    MLB::Model GetModel() const;

  private:
    MLB::Model mModel;
};
```

MLB_pub_types.hpp

```
namespace MLB
 enum Model {
   AB_2020,
   TT_2251
   ...
 };
}
```

MLB_pub.hpp

```
#include "GPSReceiver.hpp"
#include "XMLParser.hpp"
// many more includes
#include "MLB_pub_types.hpp"

namespace MLB{
// all the stuff we don't need right now
}
```

GARMIN®

Stub out what you need

Settings.cpp

```
#include "FileSystem.hpp"


...

void Settings::SaveBacklightTimeout( Milliseconds ms) {
 //...

 if(WRITE_FAIL ==
   mSettingsFile.write(Setting::BackLight, ms.to_int()) {
  //error handling
 }
 // ...
}
```

FileSystem.hpp

```
#include "DiskDriver.h"
#include "JSONParser.hpp"
// many more includes

enum error_code { WRITE_FAIL, READ_FAIL, OK };

struct DataTag {
  Id id;
  size_t data_size;
};

class FileSystem : public DBInterface {
 //...
 error_code write( DataTag tag, const void* data);
 error_code read(DataTag tag, void* data) const;
 error_code scan(size_t offset);
 //...
};
```

GARMIN.

Stub out what you need

Settings.cpp

```
#include "FileSystem.hpp"


...

void Settings::SaveBacklightTimeout( Milliseconds ms) {
 //...

 if(WRITE_FAIL == mSettingsFile.write(Setting::BackLight,
ms.to_int()) {
  //error handling
 }
 // ...
}
```

FileSystem.hpp

```
#include "DiskDriver.h"
#include "JSONParser.hpp"
// many more includes

enum error_code { WRITE_FAIL, READ_FAIL, OK };

struct DataTag {
  Id id;
  size_t data_size;
};

class FileSystem : public DBInterface {
 //...
 error_code write( DataTag tag, const void* data);
 error_code read(DataTag tag, void* data) const;
 error_code scan(size_t offset);
 //...
};
```

STUBS/FileSystem.hpp

```
#include <map>
#include <vector>

enum error_code { WRITE_FAIL, READ_FAIL, OK };
using Id = int;
struct DataTag {
  Id id;
  size_t data_size;
};

class FileSystem {

 error_code write( DataTag tag, const void* data) {
  std::vector<byte> v(tag.data_size);
  memcpy(v.data(),data, tag.data_size);
  mStorage[tag.id] = v;
  return OK;
}

private:
 std::map<Id, std::vector<byte>> mStorage;
}
```

GARMIN.

Isolate a Global

RFC_pub.h

```
extern Window next_window;
```

rfc_dd.cpp

```
Window next_window = INV_WINDOW;

//8 uses in this file
```

Conditionally included if BUILD == DD_BUILD

rfc_hh.cpp

```
Window next_window = MY_WINDOW;

//93 uses in this file
```

Conditionally included if BUILD == HH_BUILD

rfc_common.cpp

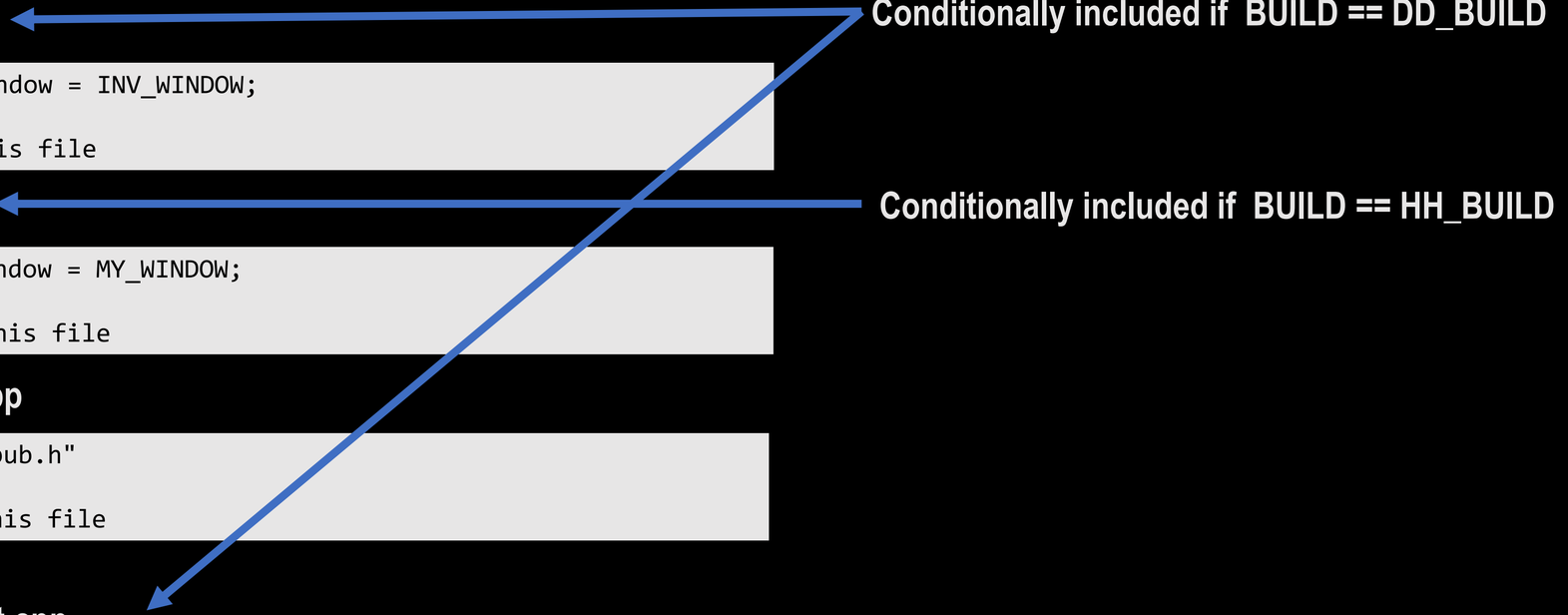
```
#include "RFC_pub.h"

//19 uses in this file
```

rfc_bark_detect.cpp

```
#include "RFC_pub.h"

//2 uses in this file
```



GARMIN

Isolate a Global

RFC_pub.h

```
extern Window next_window;
```

rfc_dd.cpp

```
Window next_window = INV_WINDOW;

//8 uses in this file
```

rfc_hh.cpp

```
Window next_window = MY_WINDOW;

//93 uses in this file
```

rfc_common.cpp

```
#include "RFC_pub.h"

//19 uses in this file
```

rfc_bark_detect.cpp

```
#include "RFC_pub.h"

//2 uses in this file
```

RFC_pub.h

```
void SetNextWindow(Window nextWindow);
Window GetNextWindow();
```

rfc_dd.cpp

```
#include "RFC_pub.h"

//All assignments use Set, all reads use Get
```

rfc_hh.cpp

```
#include "RFC_pub.h"
//All assignments use Set, all reads use Get
```

rfc_common.cpp

```
#include "RFC_pub.h"

namespace {
 Window next_window = RFC_INV_WINDOW;
}
void SetNextWindow(Window nextWindow) { next_window = nextWindow; }
Window GetNextWindow() { return next_window; }

//All assignments use Set, all reads use Get
```

rfc_bark_detect.cpp

```
#include "RFC_pub.h"

//All assignments use Set, all reads use Get
```

GARMIN®

Testing with Time

BackLightManager.hpp

```
class BackLightManager {
public:
 void SetTimeout(Milliseconds timeout);
 bool IsBacklightOn() const;
 void TurnBacklightOn();

private:
 StartBackgroundProcessingThread();
 Milliseconds mTimeout = 1_hr;
 bool mBackLightOn = false;
}
```

BackLightManager.cpp

```
#include "Timer/TimerAPI.h"

BackLightManager::TurnBacklightOn() {
 mBacklightOn = true;
 StartBackgroundProcessingThread();
}

void BackLightManager::StartBackgroundProcessingThread() {
  StartThread([this]() {
    const auto start_time = get_tick();
    const auto off_time = start_time + mTimeout;
    while(start_time + get_tick() < off_time) {
     //do some other stuff or sleep or whatever
    }
    mBackLightOn = false;
  });
}
```

modules/Timer/TimerAPI.c

```
#include "TimerAPI.h"
// many more includes

uint32_t get_tick() {
  //OS stuff to get the real time
  return tick;
};
```

GARMIN®

Testing with Time

PRJ/TESTS/TimerAPI_test.hpp

```
extern "C" {
 //force C linking for link time sub
 #include "../../modules/Timer/TimerAPI.h"
}

void TEST_set_tick(uint32_t newTime);
```

PRJ/TESTS/TimerAPI_test.cpp

```
#include "TimerAPI_test.hpp"

static uint32_t time = 0;

void TEST_set_tick(uint32_t newTime) {
 time = newTime;
}

uint32_t get_tick() { return newTime; }
```

GARMIN®

Testing with Time

PRJ/TESTS/TimerAPI_test.hpp

```
extern "C" {
 //force C linking for link time sub
 #include "../../modules/Timer/TimerAPI.h"
}

void TEST_set_tick(uint32_t newTime);
```

PRJ/TESTS/TimerAPI_test.cpp

```
#include "TimerAPI_test.hpp"

static uint32_t time = 0;

void TEST_set_tick(uint32_t newTime) {
 time = newTime;
}

uint32_t get_tick() { return newTime; }
```

ProjectManifest

```
<!-- All testable translation units go here -->
<!-- UNIT_TEST_BUILD == TRUE -->

 <!-- TEST SUBSTITUTIONS -->
 <ifeq name="UNIT_TEST_BUILD" vale="TRUE">
    <source path='PRJ/TESTS/TimerAPI_test.cpp'/>
    <else>
      <source path='modules/Timer/TimerAPI.c"/>
    </else>
 </ifeq>

<!-- All un-testable translation units go here -->
<ifeq name="UNIT_TEST_BUILD" value="FALSE">
 <!-- Everything else -->
</ifeq>
```

GARMIN.

Testing with Time

BacklightTesting.cpp

```
#include "TimerAPI_test.hpp"

TEST(BackLightTests, BacklightTurnsOffAfterTime){

  uint32_t testTime = 100;
  TEST_set_tick(testTime);

  BackLightManager blm;
  blm.SetTimeout(1_hr);

  blm.TurnBacklightOn();
  testTime += 3590000;
  TEST_set_tick(testTime);
  EXPECT_TRUE(blm.IsBacklightOn());
  testTime += 100010;
  TEST_set_tick(testTime);
  EXPECT_FALSE(blm.IsBacklightOn());
}
```

PRJ/TESTS/TimerAPI_test.hpp

```
extern "C" {
 //force C linking for link time sub
 #include "../../modules/Timer/TimerAPI.h"
}

void TEST_set_tick(uint32_t newTime);
```

PRJ/TESTS/TimerAPI_test.cpp

```
#include "TimerAPI_test.hpp"

static uint32_t time = 0;

void TEST_set_tick(uint32_t newTime) {
 time = newTime;
}

uint32_t get_tick() { return newTime; }
```

ProjectManifest

```
<!-- All testable translation units go here -->
<!-- UNIT_TEST_BUILD == TRUE -->

 <!-- TEST SUBSTITUTIONS -->
 <ifeq name="UNIT_TEST_BUILD" vale="TRUE">
    <source path='PRJ/TESTS/TimerAPI_test.cpp'/>
    <else>
      <source path='modules/Timer/TimerAPI.c"/>
    </else>
 </ifeq>

<!-- All un-testable translation units go here -->
<ifeq name="UNIT_TEST_BUILD" value="FALSE">
 <!-- Everything else -->
</ifeq>
```

GARMIN.

Part III: The Scout Rule

“Try and leave this world a little better than you found it”

–Robert Bayden-Powell

Final Letter to the Scouts

GARMIN.

A Note on Refactoring...

GARMIN®

Ways to maintain sanity

- Do NOT refactor AND add new functionality at the same time
- ALWAYS commit "working" code
- Make ONE change per commit
- Give yourself a TIME LIMIT

GARMIN.

New Code

- New code should be tested
- Make a new function or class, test in isolation, then integrate into existing code

GARMIN.

Data Accessors

GARMIN®

C.131: Avoid trivial getters and setters

Reason A trivial getter or setter adds no semantic value; the data item could just as well be `public`.

Note The key to this rule is whether the semantics of the getter/setter are trivial. While it is not a complete definition of "trivial", consider whether there would be any difference beyond syntax if the getter/setter was a public data member instead. Examples of non-trivial semantics would be: maintaining a class invariant or converting between an internal type and an interface type.

https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Rh-get

GARMIN.

Data Only Struct

```
struct Position {
 float lat;
 float lon;
};
```

GARMIN®

Data Only Struct

```
struct Position {
 float lat;
 float lon;
};
```

```
struct GeoPacket {
 GPS_fix_type fix_type;
 Position position;
 float course_true_degrees;
 float elevation_meters;
 float speed_mps;
};
```

GARMIN®

Data Only Struct

```
struct Position {
 float lat;
 float lon;
};
```

```
struct GeoPacket {
 GPS_fix_type fix_type;
 Position position;
 float course_true_degrees;
 float elevation_meters;
 float speed_mps;
};
```

```
struct WeatherPacket {
  struct {
    GeoPacket geo;
    size_t count;
    WeatherRequest s[8];
  } request;

  boolean has_geo;
  GeoPacket geo;
};
```

GARMIN®

Data Only Struct

```
struct Position {
 float lat;
 float lon;
};
```

```
struct GeoPacket {
 GPS_fix_type fix_type;
 Position position;
 float course_true_degrees;
 float elevation_meters;
 float speed_mps;
};
```

```
struct WeatherPacket {
  struct {
    GeoPacket geo;
    size_t count;
    WeatherRequest s[8];
  } request;

  boolean has_geo;
  GeoPacket geo;
};
```

```
struct SatellitePayload {
  SatPayloadT type;

  union {
    ActivationPayload activation;
    GenericBinaryPayload binary;
    DBAck database_ack;

    LocateRespPld locate_response;

    MailCheckPayload mail_check;

    PresetMessage preset;
    ReferencePointPayload reference_point;
    MapShare shared_map;

    TextMessageText text_message;

    TrackingPacket tracking;

    WeatherPacket weather;
  } data;
};
```

GARMIN®

Data Only Struct

```
struct Position {
 float lat;
 float lon;
};
```

```
struct GeoPacket {
 GPS_fix_type fix_type;
 Position position;
 float course_true_degrees;
 float elevation_meters;
 float speed_mps;
};
```

```
struct WeatherPacket {
  struct {
    GeoPacket geo;
    size_t count;
    WeatherRequest s[8];
  } request;

  boolean has_geo;
  GeoPacket geo;
};
```

```
struct SatellitePayload {
  SatPayloadT type;

  union {
    ActivationPayload activation;
    GenericBinaryPayload binary;
    DBAck database_ack;

    LocateRespPld locate_response;

    MailCheckPayload mail_check;

    PresetMessage preset;
    ReferencePointPayload reference_point;
    MapShare shared_map;

    TextMessageText text_message;

    TrackingPacket tracking;

    WeatherPacket weather;
  } data;
};
```

```
void AddGeoWeatherReqPayload(StallitePayload& payload) {
  payload.type = SPT_WEATHER;
  payload.weather.has_geo = true;
  payload.weather.request.geo.position.lat = currentPos.lat;
  payload.weather.request.geo.position.lon = currentPos.lon;
}
```

GARMIN®

Data Only Struct

```
struct Position {
 float lat;
 float lon;
};
```

```
struct GeoPacket {
 GPS_fix_type fix_type;
 Position position;
 float course_true_degrees;
 float elevation_meters;
 float speed_mps;
};
```

```
struct WeatherPacket {
  struct {
    GeoPacket geo;
    size_t count;
    WeatherRequest s[8];
  } request;

  boolean has_geo;
  GeoPacket geo;
};
```

```
struct SatellitePayload {
  SatPayloadT type;

  union {

    TrackingPacket tracking;

    WeatherPacket weather;
  } data;
};
```

```
void AddGeoWeatherReqPayload(StallitePayload& payload) {
  payload.type = SPT_WEATHER;
  payload.weather.has_geo = true;
  payload.weather.request.geo.position.lat = currentPos.lat;
                                             currentPos.lon;
```

```
void AddGeoWeatherReqPayload(StallitePayload& payload) {
  GeoRequest geo;
  geo.SetPosition(currentPos);
  WeatherPayload weather;
  weather.SetRequest(geo);
  payload.Add(weather);
}
```

GARMIN.

Remember our extern isolation?

RFC_pub.h

```
void SetNextWindow(Window nextWindow);
Window GetNextWindow();
```

rfc_dd.cpp

```
#include "RFC_pub.h"

//All assignments use Set, all reads use Get
```

rfc_hh.cpp

```
#include "RFC_pub.h"
//All assignments use Set, all reads use Get
```

rfc_common.cpp

```
#include "RFC_pub.h"

namespace {
 Window next_window = RFC_INV_WINDOW;
}
void SetNextWindow(Window nextWindow) { next_window = nextWindow; }
Window GetNextWindow() { return next_window; }

//All assignments use Set, all reads use Get
```

rfc_bark_detect.cpp

```
#include "RFC_pub.h"

//All assignments use Set, all reads use Get
```

- Single definition
- Prevent accidental change when reading
- Ability to add diagnostics
- Reduce locations where variable is changed
- Ability to mock or override reads/writes
- Prevent proliferation to other files

GARMIN®

Tip: Adding Call Site Diagnostics

GARMIN.

If you can use C++20

BTPairingViewModel.h

```
void UpdatePairingState(PairingState newState);
```

BTPairingViewModel.cpp

```
void UpdatePairingState(PairingState newState) {
  ...
}
```

BTPairingViewModel.h

```
#include <source_location>

void UpdatePairingState(PairingState newState,
  std::source_location loc = std::source_location::current());
```

BTPairingViewModel.cpp

```
#include <source_location>

void UpdatePairingState(PairingState newState, std::source_location loc) {
  DBG_PRINT("call %s from %s:%d", __func__, loc.file_name(), loc.line());
  ...
}
```

GARMIN®

C or pre-C++20

BTPairingViewModel.h

```
void UpdatePairingState(PairingState newState);
```

BTPairingViewModel.h

```
void UpdatePairingStateTmp(PairingState newState);

static void UpdatePairingStateLog(PairingState newState, const char* file, int line){
 DBG_PRINT("call %s from %s:%d", __func__, file, line);
 UpdatePairingStateTmp(newState);
}

#define UpdatePairingState(state) UpdatePairingStateLog(state, __FILE__, __LINE__);
```

BTPairingViewModel.cpp

```
void UpdatePairingState(PairingState newState) {
...
}
```

BTPairingViewModel.cpp

```
void UpdatePairingStateTmp(PairingState newState) {
  ...
}
```

GARMIN®

Use Public Functions, Even In Private

GARMIN®

Following member variables

VHFDevice.hpp

```
class VHFDevice {
public:
 UpdateRate GetUpdateRate() const;
 void SetUpdateRate(UpdateRate rate);
 bool IsDeviceAsleep() const;

private:
 void NewPacketCallback(Packet pkt);
 UpdateRate mCurrentUpdateRate;
};
```

VHFDevice.cpp

```
void VHFDevice::NewPacketCallback ( Packet pkt) {
...
  if(pkt.updateRate == mCurrentUpdateRate) {
    return;
  }

  if(mCurrentUpdateRate == UpdateRate::Asleep) {
   WakeUp();
  }

  mCurrentUpdateRate = pkt.updateRate;
  if(pkt.updateRate == UpdateRate::Asleep) {
   VHFRadio_Set_Sleep_Enabled(1);
  }

  ...
}
```

GARMIN.

Following member variables

VHFDevice.hpp

```
class VHFDevice {
public:
 UpdateRate GetUpdateRate() const;
 void SetUpdateRate(UpdateRate rate);
 bool IsDeviceAsleep() const;

private:
 void NewPacketCallback(Packet pkt);
 UpdateRate mCurrentUpdateRate;
};
```

VHFDevice.cpp

```
void VHFDevice::NewPacketCallback ( Packet pkt) {
...
  if(pkt.updateRate == mCurrentUpdateRate) {
    return;
  }

  if(mCurrentUpdateRate == UpdateRate::Asleep) {
   WakeUp();
  }

  mCurrentUpdateRate = pkt.updateRate;
  if(pkt.updateRate == UpdateRate::Asleep) {
   VHFRadio_Set_Sleep_Enabled(1);
  }

  ...
}
```

VHFDevice.cpp

```
void VHFDevice::NewPacketCallback ( Packet pkt) {
...
  if(pkt.updateRate == GetUpdateRate()) {
    return;
  }

  if(IsDeviceAsleep()) {
   WakeUp();
  }

  SetUpdateRate(pkt.updateRate);
  ...
}
```

GARMIN.

And what does this gain us?

VHFDevice.cpp

```
void VHFDevice::NewPacketCallback ( Packet pkt) {
...
  if(pkt.updateRate == GetUpdateRate()) {
    return;
  }

  if(IsDeviceAsleep()) {
   WakeUp();
  }

  SetUpdateRate(pkt.updateRate);

  ...
}
```

- Single path for external and internal callers
- Dogfooding public API
- Changes to logic occur in one location
- Ability to override for testing

GARMIN®

Ease cognitive burden

GARMIN.

Is my bug in here somewhere?

```
void process_barks(uint8 dir_idx, packet& pkt) {
 ...
 validate_pkt(pkt);
 ...
 calculate_bark_rate(&pkt);
 ...
 send_notification(pkt);
 ...
}
```

GARMIN.

Does pkt get modified here?

```
void process_barks(uint8 dir_idx, const packet& pkt) {
 ...
 validate_pkt(pkt);
 ...
 calculate_bark_rate(&pkt);
 ...
 send_notification(pkt);
 ...
}
```

error: binding reference of type 'packet&' to 'const packet' discards qualifiers

error: invalid conversion from 'const packet*' to 'packet*'

Compiles, we can ignore this function

GARMIN.

Does pkt get modified here?

```
void process_barks(uint8 dir_idx, const packet& pkt) {
 ...
 validate_pkt(pkt);
 ...
 calculate_bark_rate(&pkt);
 ...
 send_notification(pkt);
 ...
}
```

error: binding reference of type 'packet&' to 'const packet' discards qualifiers

error: invalid conversion from 'const packet*' to 'packet*'

Compiles, we can ignore this function

```
void validate_pkt(const packet& pkt){
 ...
}

void calculate_bark_rate(const packet* pkt) {
...
}
```

error: assignment of member 'packet::is_valid' in read-only object

GARMIN.

Does pkt get modified here?

```
void process_barks(uint8 dir_idx, const packet& pkt) {
 ...
 validate_pkt(pkt);
 ...
 calculate_bark_rate(&pkt);
 ...
 send_notification(pkt);
 ...
}
```

error: binding reference of type 'packet&' to 'const packet' discards qualifiers

Compiles, we can ignore this function

Compiles, we can ignore this function

```
void validate_pkt(const packet& pkt){
 ...
}
```

error: assignment of member 'packet::is_valid' in read-only object

GARMIN

Reduce possible bug locations

```
void process_barks(uint8 dir_idx, packet& pkt) {
 ...
 validate_pkt(pkt);
 ...
 calculate_bark_rate(&pkt);
 ...
 send_notification(pkt);
 ...
}
```

Need to look into this function since it does modify pkt

Compiles, we can ignore this function

Compiles, we can ignore this function

GARMIN

Reduce scope and const

```
void validate_pkt(packet& pkt) {
  bark_cnt* bark_cnt;
  bark_cnt last_bark_cnt;
  uint32 now;
  bool bark_cnt_is_delta_based;
```

GARMIN®

Reduce scope and const

```
void validate_pkt(packet& pkt) {
  bark_cnt* bark_cnt;
  bark_cnt last_bark_cnt;
  uint32 now;
  bool bark_cnt_is_delta_based;
```

Reduce scope to first usage, then add const

```
const auto now = timer_function();

if (INVALID_INDEX!= pkt.dir_idx) {
  const auto last_bark_cnt = pkt.bark_cnt_last_bark_hd_rate_update;
  const auto* bark_cnt = &pkt.bark_info;
  const auto bark_cnt_is_delta_based = pkt.bark_hd_info_is_delta;
```

GARMIN®

Reduce scope and const

```
const auto now = timer_function();

if (INVALID_INDEX!= pkt.dir_idx) {
  const auto last_bark_cnt = pkt.bark_cnt_last_bark_hd_rate_update;
  const auto* bark_cnt = &pkt.bark_info;
  const auto bark_cnt_is_delta_based = pkt.bark_hd_info_is_delta;
```

104 lines later...

```
// add "potential" bark back
*bark_cnt += asset_item->potential_bark_to_ignore;
```

Cannot assign to variable 'bark_cnt' with const-qualified type 'const bark_cnt' (aka 'const unsigned short')

GARMIN®

Reduce scope and const

```
const auto now = timer_function();

if (INVALID_INDEX!= pkt.dir_idx) {
  const auto last_bark_cnt = pkt.bark_cnt_last_bark_hd_rate_update;
  auto* bark_cnt = &pkt.bark_info;
  const auto bark_cnt_is_delta_based = pkt.bark_hd_info_is_delta;
```

GARMIN®

Document understanding

```
/**
*Adds the specified track point to the temporary course
*/
void add_tpt_to_temp_course(HANDLE file_hndl, uint16 trkpt_cnt,
                                    const position* posn, uint16_t alt) {
  Record rcrd_mesg;
  if (trkpt_cnt == 0) {
    rcrd_mesg.start_position_lat = (int32_t)posn->lat;
    rcrd_mesg.start_position_long = (int32_t)posn->lon;
    total_distance = 0.0f;
  }

  /*--------------------------------------
  Keep track of cumulative
  --------------------------------------*/
  if (trkpt_cnt > 0) {
    int32_t alpha_scale = 0;
    const int32_t lat = (last_posn.lat + posn->lat) / 2;
    lon_scale(lat, &alpha_scale, NULL);

    const uint32_t delta_distance_sc =
                compute_dist(posn, &last_posn, &alpha_scale, NULL);
    const float delta_distance = (float)(SEMI_TO_MT * delta_distance_sc);
    total_distance += delta_distance;
  }
...
}
```

GARMIN®

Document understanding

```
/**
*Adds the specified track point to the temporary course
*/
void add_tpt_to_temp_course(HANDLE file_hndl, uint16 trkpt_cnt,
                                    const position* posn, uint16_t alt) {
  Record rcrd_mesg;
  if (trkpt_cnt == 0) {
    rcrd_mesg.start_position_lat = (int32_t)posn->lat;
    rcrd_mesg.start_position_long = (int32_t)posn->lon;
    total_distance = 0.0f;
  }

  /*--------------------------------------
  Keep track of cumulative
  --------------------------------------*/
  if (trkpt_cnt > 0) {
    int32_t alpha_scale = 0;
    const int32_t lat = (last_posn.lat + posn->lat) / 2;
    lon_scale(lat, &alpha_scale, NULL);

    const uint32_t delta_distance_sc =
                compute_dist(posn, &last_posn, &alpha_scale, NULL);
    const float delta_distance = (float)(SEMI_TO_MT * delta_distance_sc);
    total_distance += delta_distance;
  }
...
}
```

GARMIN®

Document understanding

```
/**
*Adds the specified track point to the temporary course
*/
void add_tpt_to_temp_course(HANDLE file_hndl, uint16 trkpt_cnt,
                                  const position* posn, uint16_t alt) {
  Record rcrd_mesg;
  if (trkpt_cnt == 0) {
    rcrd_mesg.start_position_lat = (int32_t)posn->lat;
    rcrd_mesg.start_position_long = (int32_t)posn->lon;
    total_distance = 0.0f;
  }

  /*--------------------------------------
  Keep track of cumulative
  --------------------------------------*/
  if (trkpt_cnt > 0) {
    int32_t alpha_scale = 0;
    const int32_t lat = (last_posn.lat + posn->lat) / 2;
    lon_scale(lat, &alpha_scale, NULL);

    const uint32_t delta_distance_sc =
                 compute_dist(posn, &last_posn, &alpha_scale, NULL);
    const float delta_distance = (float)(SEMI_TO_MT * delta_distance_sc);
    total_distance += delta_distance;
  }
...
}
```

```
enum distance_calc_type { DISTANCE_RESET, DISTANCE_ACCUMULATE };

/**
*Adds the specified track point to the temporary course
*and either reset or add to the total course distance
*/
static void add_tpt_to_temp_course(FFS_fhndl_type file_hndl,
                                   distance_calc_type distance_calc,
                                   const position *posn, uint16_t alt) {
  Record rcrd_mesg;
   if (distance_calc == DISTANCE_RESET) {
     rcrd_mesg.start_position_lat = (int32_t)posn->lat;
     rcrd_mesg.start_position_long = (int32_t)posn->lon;
     total_distance = 0.0f;
   } else if (distance_calc == DISTANCE_ACCUMULATE) {
    int32_t alpha_scale = 0;
    const int32_t lat = (last_posn.lat + posn->lat) / 2;
    lon_scale(lat, &alpha_scale, NULL);

    const uint32_t delta_distance_sc =
                 compute_dist(posn, &last_posn, &alpha_scale, NULL);
    const float delta_distance = (float)(SEMI_TO_MT * delta_distance_sc);
    total_distance += delta_distance;
  }

...
}
```

GARMIN.

RAII

GARMIN®

Prevent Maintenance Resource bugs

```
void updateColor(ID itemId, Color newColor) {
  ReserveMutex(&itemMutex);
  item* itemToChange = get_item_from_id(itemId);
  if(itemToChange == NULL) {
    ReleaseMutex(&itemMutex);
    return;
  }

  itemToChange->SetColor(newColor);
  ReleaseMutex(&itemMutex);
}
```

GARMIN®

Prevent Maintenance Resource bugs

```
void updateColor(ID itemId, Color newColor) {
  ReserveMutex(&itemMutex);
  item* itemToChange = get_item_from_id(itemId);
  if(itemToChange == NULL) {
     ReleaseMutex(&itemMutex);
    return;
  }

  itemToChange->SetColor(newColor);
  ReleaseMutex(&itemMutex);
}
```

GARMIN.

Prevent Maintenance Resource bugs

```
void updateColor(ID itemId, Color newColor) {
  ReserveMutex(&itemMutex);
  item* itemToChange = get_item_from_id(itemId);
  if(itemToChange == NULL) {
    ReleaseMutex(&itemMutex);
    return;
  }

  itemToChange->SetColor(newColor);
  ReleaseMutex(&itemMutex);
}
```

```
struct MutexLocker {
 MutexLocker(Mutex& mutex) : mMutex(mutex) {
   ReserveMutex(&mutex);
 }
  ~MutexLocker() { ReleaseMutex(&mMutex); }
 private:
   Mutex& mMutex;
}


void updateColor(ID itemId, Color newColor) {
  MutexLocker(&itemMutex);
  item* itemToChange = get_item_from_id(itemId);
  if(itemToChange == NULL) {
    return;
  }

  itemToChange->SetColor(newColor);
}
```

GARMIN.

Recognize Algorithms

GARMIN®

Recognize Algorithms

```
bool IsFileInFilesToKeep(const char* aPath) {
  bool found = false;
  for (const auto& file : filesToKeep) {
    if (strcmp(file, aPath) == 0) {
      found = true;
      break;
    }
  }
  return found;
}
```

GARMIN®

Recognize Algorithms

```
bool IsFileInFilesToKeep(const char* aPath) {
  bool found = false;
  for (const auto& file : filesToKeep) {
    if (strcmp(file, aPath) == 0) {
      found = true;
      break;
    }
  }
  return found;
}
```

```
bool IsFileInFilesToKeep(const char* aPath) {
  const auto found = std::find_if(begin(filesToKeep),end(filesToKeep),
                [aPath](const char* file) {
                  return strcmp(aPath, file) == 0;
                });
 return found != end(filesToKeep);
}
```

GARMIN.

Boolean arguments

```
class CourseMap {
 public:
  // Constructor/Destructor.
  CourseReviewMap(RecordId aId, bool aIsActivity = false);
  ...
 private:
  bool mActivity;
};
```

GARMIN®

Take out the guesswork

```
class CourseMap {
 public:
  // Constructor/Destructor.
  CourseReviewMap(RecordId aId, bool aIsActivity = false);
  ...
 private:
  bool mActivity;
};
```

```
enum struct NavType { Course, Activity };

class CourseMap {
 public:
  // Constructor/Destructor.
  CourseReviewMap(RecordId aId, NavType aNavType = NavType::Course);
  ...
 private:
  const NavType mNavType;
};
```

GARMIN.

Take out the guesswork, and prepare for the future

```
class CourseMap {
 public:
  // Constructor/Destructor.
  CourseReviewMap(RecordId aId, bool aIsActivity = false);
  ...
 private:
  bool mActivity;
};
```

```
enum struct NavType { Course, Activity, Track };

class CourseMap {
 public:
  // Constructor/Destructor.
  CourseReviewMap(RecordId aId, NavType aNavType = NavType::Course);
  ...
 private:
  const NavType mNavType;
};
```

GARMIN.

“Any fool can write code that a computer can understand. Good programmers write code that humans can understand.”

–Martin Fowler

Refactoring: Improving the Design of Existing Code

GARMIN.

Take the time to write code for maintainability

GARMIN®

Questions?

GARMIN®