Managing Defects in HPC Software Development

Presented to
OLCF Webinar Series

Tom Evans
ORNL, PI ExaSMR ECP Applications Project

November 1, 2017

ECP
EXASCALE COMPUTING PROJECT

NNSA
National Nuclear Security Administration

U.S. DEPARTMENT OF ENERGY | Office of Science

Before we start

- Since I cannot see anyone in this presentation format, feel free to totally vegout, use profane gestures, etc

Before we start

- Since I cannot see anyone in this presentation format, feel free to totally vegout, use profane gestures, etc
- I am not proselytizing; these are some techniques that have worked well for us over the last 20+ years; if you violently disagree see (1)

Before we start

- Since I cannot see anyone in this presentation format, feel free to totally vegout, use profane gestures, etc
- I am not proselytizing; these are some techniques that have worked well for us over the last 20+ years; if you violently disagree see (1)
- I will try to keep this short and sweet, in the end there is only 1 concept I would like you to take away from this—assuming item (1) does not apply

Before we start

- Since I cannot see anyone in this presentation format, feel free to totally vegout, use profane gestures, etc
- I am not proselytizing; these are some techniques that have worked well for us over the last 20+ years; if you violently disagree see (1)
- I will try to keep this short and sweet, in the end there is only 1 concept I would like you to take away from this—assuming item (1) does not apply
- I promise that there will be no distracting manager clip-art, sliding images, dissolution, etc.

Before we start

- Since I cannot see anyone in this presentation format, feel free to totally vegout, use profane gestures, etc
- I am not proselytizing; these are some techniques that have worked well for us over the last 20+ years; if you violently disagree see (1)
- I will try to keep this short and sweet, in the end there is only 1 concept I would like you to take away from this—assuming item (1) does not apply
- I promise that there will be no distracting manager clip-art, sliding images, dissolution, etc.
- If you require sparkly things in the presentation to keep you awake, please refer back to item (1).

Outline

❶ Research and Software Development

❷ The Complete Development Lifecycle

❸ Unit Testing

❹ Design-by-Contract^{TM}

❺ Summary

Research and HPC Code

Challenge

Manage SQE with discovery

Posit

Consider a new algorithm implemented in a multidimensional, parallel code.

- Theory predicts second-order convergence.
- Computational results are first-order instead of second-order.
- Is this a code bug or an error in analysis?

Research and HPC Code

Challenge

Manage SQE with discovery

Posit

Consider a new algorithm implemented in a multidimensional, parallel code.

- Theory predicts second-order convergence.
- Computational results are first-order instead of second-order.
- Is this a code bug or an error in analysis?

Research and HPC Code

Challenge

Manage SQE with discovery

Posit

Consider a new algorithm implemented in a multidimensional, parallel code.

- Theory predicts second-order convergence.
- Computational results are first-order instead of second-order.
- Is this a code bug or an error in analysis?

Research and HPC Code

Challenge

Manage SQE with discovery

Posit

Consider a new algorithm implemented in a multidimensional, parallel code.

- Theory predicts second-order convergence.
- Computational results are first-order instead of second-order.
- Is this a code bug or an error in analysis?

Research and HPC Code

- In other words, SQE and methods research are not only compatible, they are essential
- This is especially true for parallel scientific software, which is much more difficult to design, test, and analyze than serial software.
- We are interested in this case in performing software verification
- Software verification is a method for removing defects at code construction time

What is SQE

- SQE is the practice of managing the cost and quality of a software product
- **Guiding Principle**
  The cost of defect resolution increases with time from defect introduction*
- **Things fall apart**
  - Defects in model development
  - Defects in algorithmic selection
  - Defects in requirements
  - Defects in implementation

How to mitigate defects

- There are many methods for defect management
- Three techniques we use for software verification in an HPC environment
  - The complete development lifecycle
  - Unit-testing
  - Design-by-Contract$^{\text{TM}}$
- This list is by no means exhaustive (or a complete SQE process)
  - Notably missing, **reviews**
  - We do them, they work, but I’m not here to talk about them
- However, taken together these can help catch defects before they become an unbearable expense

Requirements Management in Scientific Software

- Requirements can be very difficult to pin down in scientific software development:
  - the vector keeps changing as new things are learned
  - as a community we often know what we want, but aren't necessarily good at saying it
- Software verification helps disambiguate language-based requirements into functional specifications
- As requirements change, software verification helps ensure that the software is keeping pace.
- Agility is key in scientific software development:
  - rapid prototyping
  - testing new methods, algorithms, and features

Complete Development Lifecycle

- The developer is responsible for the **complete** implementation of a feature including:
  - Requirements
  - Derivation
  - Construction
  - Deployment
- Documentation and verification is implicit in each phase
- Reviews and team collaboration are essential

Developers are responsible for all phases of code development

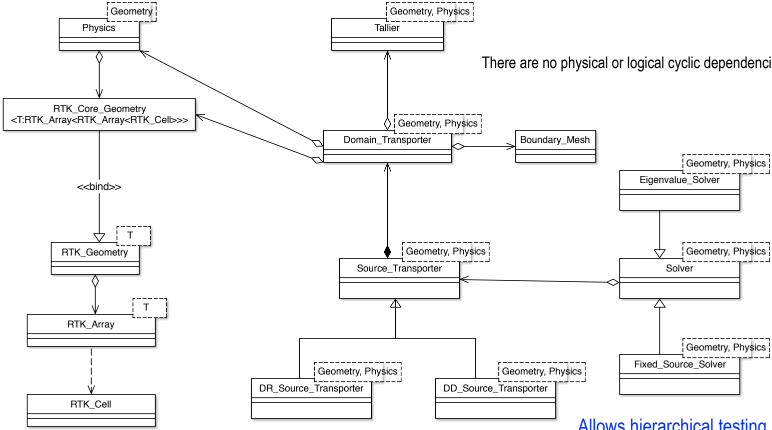
Unit Testing

Unit testing is a form of software verification

- It ensures that each part of the software performs its contracted task
- The effectiveness of unit-testing is greatly enhanced by the following two code design practices:
  - Acyclic code design
  - Design-by-Contract$^{\mathrm{TM}}$(see later)

We practice a method of unit testing in which the unit test is written either before, or concurrently with, the executable code.

Acyclic Code Design



There are no physical or logical cyclic dependencies

Allows hierarchical testing

EXASCALE COMPUTING PROJECT

An Example—Reactor Geometry

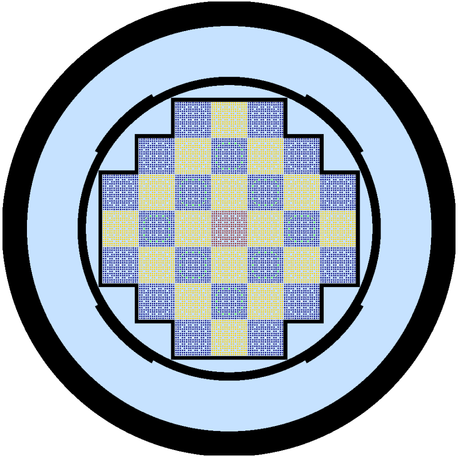
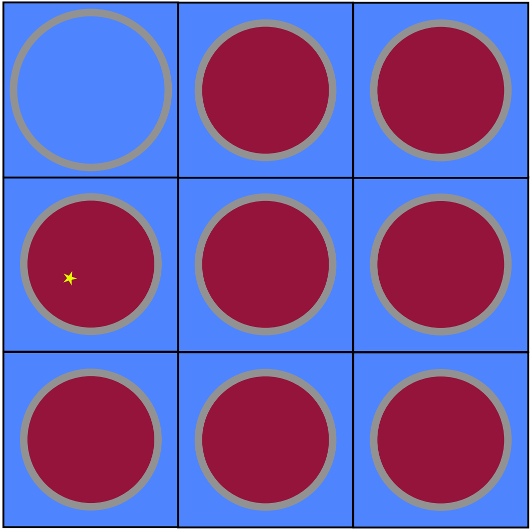


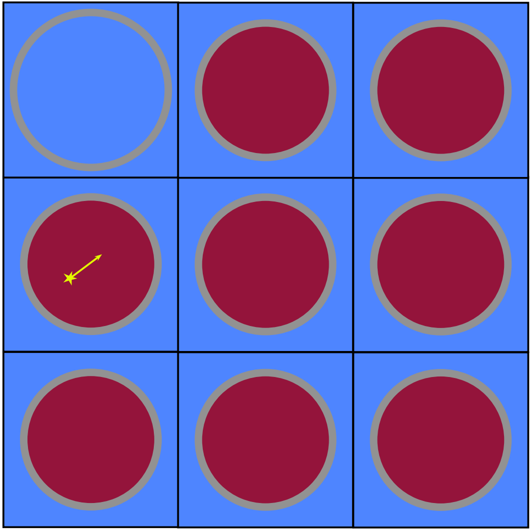
Figure: Small modular reactor core model.

An Example—Reactor Geometry



❶ Sample starting neutron

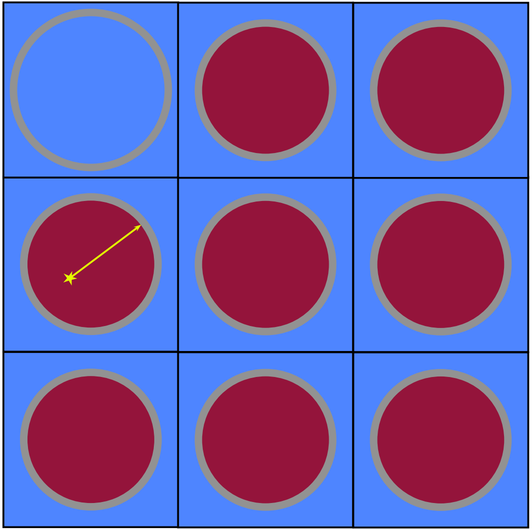
An Example—Reactor Geometry



1. Sample starting neutron
2. Sample distance to collision

$$d_{\text{col}} = \frac{\log(\xi)}{\sigma(\mathbf{r}, E)}$$

An Example—Reactor Geometry



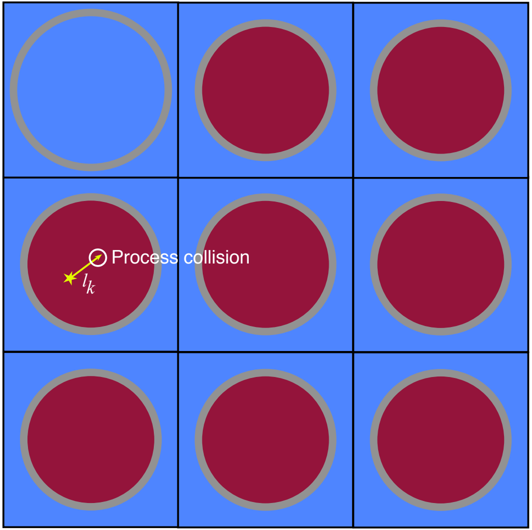
❶ Sample starting neutron

❷ Sample distance to collision

$$d_{\mathrm{col}} = \frac{\log(\xi)}{\sigma(\mathbf{r}, E)}$$

❸ Calculate distance to boundary

An Example—Reactor Geometry



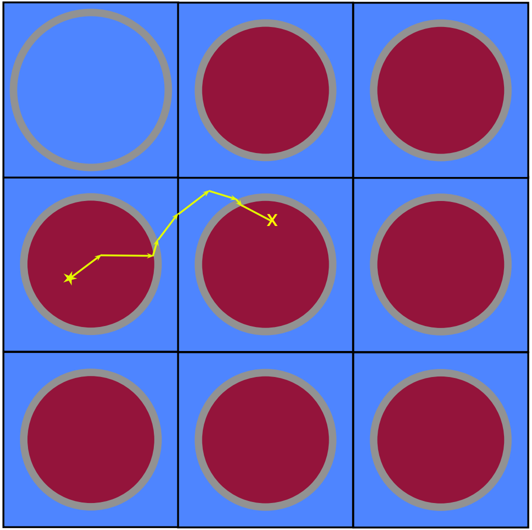
❶ Sample starting neutron
❷ Sample distance to collision

$$d_{\text{col}} = \frac{\log(\xi)}{\sigma(\mathbf{r}, E)}$$

❸ Calculate distance to boundary
❹ Move particle
❺ Tally state data

$$\phi = \frac{1}{V}\sum_k l_k$$

An Example—Reactor Geometry



❶ Sample starting neutron

❷ Sample distance to collision

$$d_{\text{col}} = \frac{\log(\xi)}{\sigma(\mathbf{r}, E)}$$

❸ Calculate distance to boundary

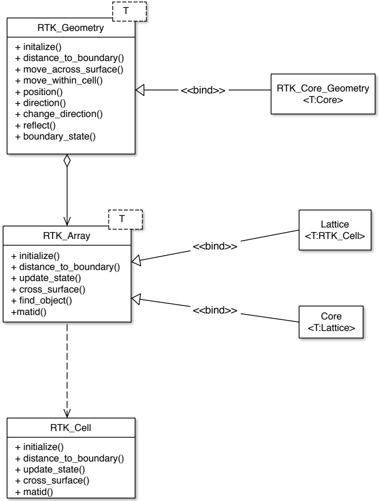
❹ Move particle

❺ Tally state data

$$\phi = \frac{1}{V} \sum_k l_k$$

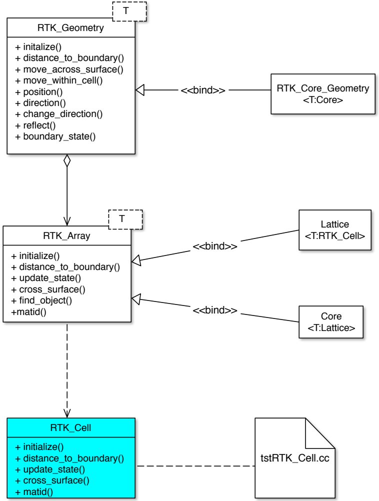
❻ Repeat 2–5

First Level—`RTK_Cell`



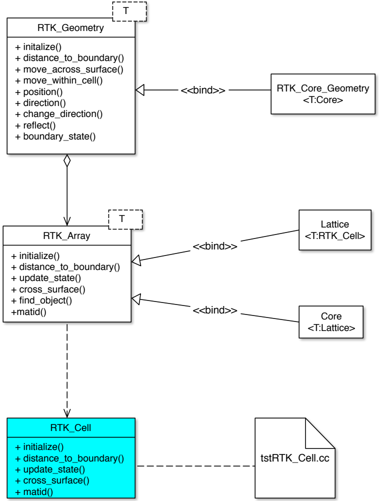
- Here is the class diagram for the `RTK_Geometry` part of the code

First Level—`RTK_Cell`



- Here is the class diagram for the `RTK_Geometry` part of the code
- Starting at the lowest level of the class hierarchy, we can write a unit test that unambiguously tests `RTK_Cell`

First Level—`RTK_Cell`



- Here is the class diagram for the `RTK_Geometry` part of the code
- Starting at the lowest level of the class hierarchy, we can write a unit test that unambiguously tests `RTK_Cell`
- There are many frameworks that support this—**GoogleTest**, TeuchosTest (Trilinos)
- Some extra details are required to support advanced architectures

`tstRTK_Cell.cc`—The old way

```
#include "Nemesis/gtest/nemesis_gtest.hh"

TEST(SingleShell, track)
{
    RTK_Cell pin1(1, 0.54, 10, 1.26, 14.28);

    pin1.initialize(Vector(0.0, 0.55, 0.0), state);
    EXPECT_EQ(1, state.region);
    EXPECT_EQ(0, state.segment);
    EXPECT_EQ(1, pin1.cell(state.region, state.segment));

    Vector r     = Vector(0.0, 0.59, 0.0);
    Vector omega = Vector(1.0, 0.0, 0.0);
    pin1.initialize(r, state);
    pin1.distance_to_boundary(r, omega, state);
    EXPECT_SOFTEQ(state.dist_to_next_region, 0.63, 1.e-12);
    EXPECT_EQ(Geo_State::PLUS_X, state.exiting_face);
    EXPECT_EQ(1, state.region);

    // ...
}
```

- In MP/multithreaded codes this way straitforward
- Instantiate the object and test its state and behavior
- “garbage-in/garbage-out”
- “Hand” calculations stored in repository using Jupyter Notebook
- On heterogeneous computing environments extra work is required

`tstRTK_Cell.cc`—The “new” way

```
#include "Nemesis/gtest/nemesis_gtest.hh"
#include "RTK_Cell_Tester.hh"

TEST_F(Single_Shell, construction)
{
    construct();
}

TEST_F(Single_Shell, tracking)
{
    track();
}
```

Host-side driver—host-only test code and defined tests

RTK_Cell_Tester.hh

```
#include "Nemesis/gtest/Gtest_Functions.hh"
#include "Geometria/rtk/RTK_Cell.hh"

class Single_Shell : public Base
{
  protected:

    void SetUp()
    {
        SP_Cell pin1 = std::make_shared<RTK_Cell>(1, 0.54, 10, 1.26, 14.28);
        SP_Cell pin2 = std::make_shared<RTK_Cell>(1, 0.45, 2, 1.2, 14.28);
        pins         = {pin1, pin2};
    }

    void construct();
    void track();

    Vec_Cell pins;
};
```

Bridge code—connects host-side driver with kernel implementation

RTK_Cell_Tester.cu

```
void Single_Shell::track()
{
    geometria_cuda::RTK_Cell_DMM dmm(*pins[1]);
    auto pin = dmm.device_instance();

    thrust::device_vector<int>    ints(50, -1);
    thrust::device_vector<double> dbls(50, -1);

    single_shell_kernel2<<<1,1>>>(
        pin, ints.data().get(), dbls.data().get());

    thrust::host_vector<int>    rints(ints.begin(),
                                      ints.end());
    thrust::host_vector<double> rdbls(dbls.begin(),
                                      dbls.end());


    int    n   = 0, m = 0;
    double eps = 1.0e-6;

    EXPECT_EQ(1, rints[n++]);
    EXPECT_SOFTEQ(rdbls[m++], 1.2334036420, eps);
    EXPECT_EQ(State::INTERNAL, rints[n++]);
    EXPECT_EQ(0, rints[n++]);

    // ...
```

```
__global__
void single_shell_kernel2(
    geometria_cuda::RTK_Cell  pin,
    int                       *ints,
    double                    *dbls)
{
    State  state;
    Vector r, omega;
    int    n = 0, m = 0;

    // Pin intersection tests
    {
        r     = {  0.43,    0.51,    1.20};
        omega = { -0.07450781,  -0.17272265,   0.98214840};
        pin.initialize(r, state);
        ints[n++] = state.region;
        pin.distance_to_boundary(r, omega, state);
        ints[n++] = state.exiting_face;
        ints[n++] = state.next_region;
        dbls[m++] = state.dist_to_next_region;
    }

    // ...
```

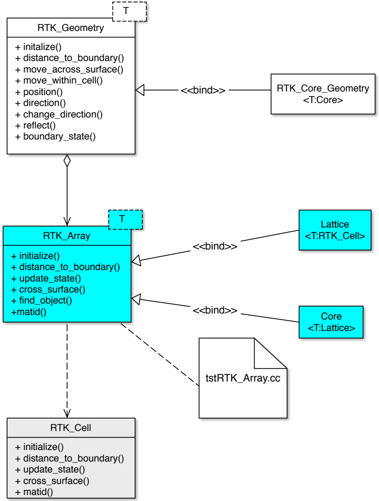
Test Output

```
Testing on 1 processors
Exnihilo 6.2 (branch ’omnibus_cuda’ #20e8c851 on 2017JUL10) [debug] [DBC=7]
SCALE 6.3 (r23123: #c743536b on 2017JUL06) [debug] [DBC=7]
[==========] Running 2 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 2 tests from Single_Shell
[ RUN      ] Single_Shell.construction
[       OK ] Single_Shell.construction (381 ms)
[ RUN      ] Single_Shell.tracking
[       OK ] Single_Shell.tracking (2 ms)
[----------] 2 tests from Single_Shell (383 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test case ran. (384 ms total)
[  PASSED  ] 2 tests.
In ./GeometriaCUDA_tstRTK_Cell.exe, overall test result: PASSED
```

```
PACKAGE_ADD_CUDA_LIBRARY(
  Geometria_cuda_test_cuda
  SOURCES RTK_Array_Tester.cu
  DEPLIBS Geometria_cuda
  TESTONLY)

ADD_NEMESIS_TEST(tstRTK_Cell.cc  NP 1
  DEPLIBS Geometria_cuda_test_cuda)
```

- Integrated into CMake build system
- Compile-Edit-Debug development cycle
- Continuous integration

Second Level—`RTK_Array`

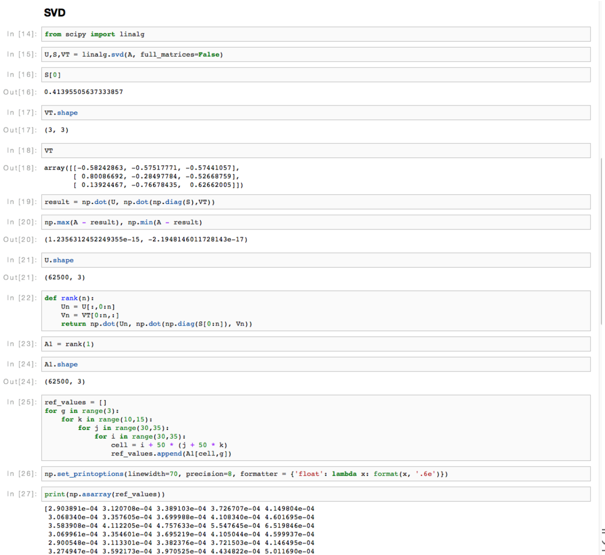


- Having verified `RTK_Cell` we proceed to the next level
- Individual unit-tests work their way up dependency chain
- After completion of a feature, unit tests remain in the code base for both regression and continuous integration testing

Testing tools

- Python and Jupyter notebook are useful for generating "by-hand" results
- Easily stored with code so that tests can be modified and examined

```
CMakeLists.txt
SVDTestBase.hh
SVDTestBase.cc
nb/SVDTestBase.ipynb
nb/tstHybrid_Data_Field.ipynb
tstAdjoint_Builder.cc
tstHybrid_Data_Field.cc
tstSVD_Operator.cc
tstSVD_Solver.cc
```



SVD

```
In [14]: from scipy import linalg

In [15]: U,S,VT = linalg.svd(A, full_matrices=False)

In [16]: S[0]

Out[16]: 0.41395505637333857

In [17]: VT.shape

Out[17]: (3, 3)

In [18]: VT

Out[18]: array([[-0.58242863, -0.57517771, -0.57441057],
                [ 0.80086692, -0.28497784, -0.52668759],
                [ 0.13924467, -0.76678435,  0.62662005]])

In [19]: result = np.dot(U, np.dot(np.diag(S),VT))

In [20]: np.max(A - result), np.min(A - result)

Out[20]: (1.2356312452249355e-15, -2.1948146011728143e-17)

In [21]: U.shape

Out[21]: (62500, 3)

In [22]: def rank(n):
             Un = U[:,0:n]
             Vn = VT[0:n,:]
             return np.dot(Un, np.dot(np.diag(S[0:n]), Vn))

In [23]: A1 = rank(1)

In [24]: A1.shape

Out[24]: (62500, 3)

In [25]: ref_values = []
         for g in range(3):
             for k in range(10,15):
                 for j in range(30,35):
                     for i in range(30,35):
                         cell = i + 50 * (j + 50 * k)
                         ref_values.append(A1[cell,g])

In [26]: np.set_printoptions(linewidth=70, precision=8, formatter = {'float': lambda x: format(x, '.6e')})

In [27]: print(np.asarray(ref_values))

         [2.903891e-04 3.120708e-04 3.389103e-04 3.726707e-04 4.149804e-04
          3.068340e-04 3.357605e-04 3.699988e-04 4.108340e-04 4.601695e-04
          3.583908e-04 4.112205e-04 4.757633e-04 5.547645e-04 6.519846e-04
          3.069961e-04 3.354601e-04 3.695219e-04 4.105044e-04 4.599937e-04
          2.900548e-04 3.113301e-04 3.382376e-04 3.721503e-04 4.146495e-04
          3.274947e-04 3.592173e-04 3.970525e-04 4.434822e-04 5.011690e-04
```

Design-by-Contract™

- DBC enforces a function “contract” by testing the input, execution, and output of a function.
- In other words, DBC provides a software mechanism for enforcing a design contract on a function.
- DBC is also known as Programming by Contract and Contract First Development.
- See Meyer, Bertrand: Design by Contract, in *Advances in Object-Oriented Software Engineering*, eds. D. Mandrioli and B. Meyer, Prentice Hall, 1991, pp. 1-50 for more details.

DBC Implementation

- Some languages (Eiffel, GNU C^2) have built in support for DBC.
- DBC is implemented in our codes using M4 (FORTRAN) or CPP (C/C++).
- Types in C++ or FORTRAN modules are automatically checked by the compiler:
  - `Require`: input conditions
  - `Check`: execution conditions
  - `Ensure`: output conditions
- DBC macros can be toggled at compile time to avoid performance costs associated with in-code tests.
- We also support device implementations

A DBC Example

- You are asked to provide a routine to calculate square roots—ok this is a manufactured example
- Being a clever person you realize you can solve this as a nonlinear problem using Newton's method:

$$x_{n+1} = x_n + \frac{f(x_n)}{f'(x_n)} \, ,$$

where $f(x_n) = x_n^2 - S$

A DBC Example

- You are asked to provide a routine to calculate square roots—ok this is a manufactured example
- Being a clever person you realize you can solve this as a nonlinear problem using Newton's method:

$$x_{n+1} = x_n + \frac{f(x_n)}{f'(x_n)},$$

where $f(x_n) = x_n^2 - S$

- You deliver your unit-tested, verified solution:

```
double my_sqrt(double S)
{
    double xn = 1.0;

    for (int n = 0; n < 10; ++n)
    {
        xn = 0.5 * (xn + S / xn);
    }

    return xn;
}
```

But there’s trouble brewing in science

- Some indeterminate time later—after you’ve moved onto much more exciting things—you start getting complaints or bug reports

But there's trouble brewing in science

- Some indeterminate time later—after you've moved onto much more exciting things—you start getting complaints or bug reports
- John has spent 2 weeks tracking spurious results down to your routine that returned a value of 200.514691 ($\varepsilon > 10^{-6}$) for 40 200.25

But there’s trouble brewing in science

- Some indeterminate time later—after you’ve moved onto much more exciting things—you start getting complaints or bug reports
- John has spent 2 weeks tracking spurious results down to your routine that returned a value of 200.514691 ($\varepsilon > 10^{-6}$) for 40 200.25
- Tara also has a problem with you because she is doing Spherical Harmonics in complex space and tried to take the square root of -4 and got -4.8017607

But there’s trouble brewing in science

- Some indeterminate time later—after you’ve moved onto much more exciting things—you start getting complaints or bug reports
- John has spent 2 weeks tracking spurious results down to your routine that returned a value of 200.514691 ($\varepsilon > 10^{-6}$) for 40 200.25
- Tara also has a problem with you because she is doing Spherical Harmonics in complex space and tried to take the square root of -4 and got -4.8017607
- You reply that the routine was thoroughly tested and is performing as designed, so what gives

But there’s trouble brewing in science

- Some indeterminate time later—after you’ve moved onto much more exciting things—you start getting complaints or bug reports
- John has spent 2 weeks tracking spurious results down to your routine that returned a value of 200.514691 ($\varepsilon > 10^{-6}$) for 40 200.25
- Tara also has a problem with you because she is doing Spherical Harmonics in complex space and tried to take the square root of -4 and got -4.8017607
- You reply that the routine was thoroughly tested and is performing as designed, so what gives
- Pandemonium ensues

This is a defect resulting from ambigous requirements

- Nothing is more common in scientific programming
- How could DBC have helped?
- Lets look at how adding DBC may have aided things

This is a defect resulting from ambigous requirements

- Nothing is more common in scientific programming
- How could DBC have helped?
- Lets look at how adding DBC may have aided things
- First, we decide we will not handle complex math
- Second, we check for a tolerance at the end

```
double my_sqrt(double S)
{
    Require(S > 0.0);

    double xn = 1.0;

    for (int n = 0; n < 10; ++n)
    {
        xn = 0.5 * (xn + S / xn);
    }

    Ensure(std::fabs(xn*xn - S) > 1.0e-6 * S)
    return xn;
}
```

Moral of the story

- This still won’t win any programmer-of-the-year awards, but you get the point
- Adding DBC “contracts” allows both developers and clients to codify potentially ambiguous requirements
- In particular, at review time DBC can help a reviewer determine if the requested service is doing what is required
- Downstream, if the function is used in manner that is outside of design parameters, at least we know

Real DBC Example—`distance_to_boundary`

```
__device__
void RTK_Cell::distance_to_boundary(
    const Space_Vector &r,
    const Space_Vector &omega,
    Geo_State_t        &state) const
{
    DEVICE_REQUIRE(soft_equiv(vector_magnitude(omega), 1., 1.e-6));
    DEVICE_REQUIRE(omega[X]<0.0           ?
                   r[X] >= d_extent[X][LO] :
                   r[X] <= d_extent[X][HI]);
    DEVICE_REQUIRE(omega[Y]<0.0           ?
                   r[Y] >= d_extent[Y][LO] :
                   r[Y] <= d_extent[Y][HI]);
    DEVICE_REQUIRE(omega[Z]<0.0 ?
                   r[Z] >= 0.0  :
                   r[Z] <= d_z);
    // ...
    DEVICE_CHECK(db >= 0.0);
    // ...

    DEVICE_ENSURE(state.dist_to_next_region >= 0.0);
    DEVICE_ENSURE(state.exiting_face == Geo_State_t::INTERNAL ?
                  state.next_region >= 0 : true);
    DEVICE_ENSURE(state.next_segment >= 0 && state.next_segment < d_segments);
```

- Valid argument types are checked by the compiler
- `DEVICE_REQUIRE` checks that input arguments are and object is in a valid state
- `DEVICE_CHECK` in-function checks
- `DEVICE_ENSURE` object and arguments are in a valid state at output

Software Verification Advantages

The purpose of unit-testing is to provide software verification as close to code construction time as possible.

- finds code defects at construction time
- provides an automated, explicit review of the code and enables Continuous Integration
  - a mechanism for review is to have one developer write the test and the primary developer writes the code
  - when the test passes, the software component is automatically reviewed
  - provides a testing basis for Continuous Integration

Software Verification Advantages

- makes porting to new platforms easier
- easier to find esoteric compile/link-time errors
- DBC can be used to verify interfaces to client code
- DBC incurs no cost in production code
- easier to run profiling, memory, and development tools on unit tests than on a full executable
- unambiguous statement of code design requirements

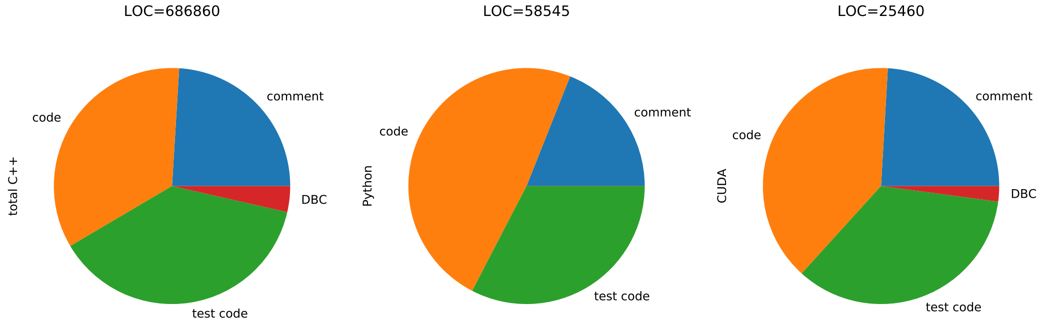
Software Verification Advantages

- provides a sanity check on code refactors
- incorporating timing data allows a time-history profile of code performance to be compiled:
  - run automated unit-tests nightly
  - as new code is developed compare timing histories to catch inefficient or costly implementations
- provides simplified “usage” documentation for a piece of code
  - in our example, a new developer could easily learn the mechanics of the `RTK_Geometry` component by studying the unit tests

Disadvantages and Costs

- The most significant disadvantage is the perceived cost associated with unit tests
- Our experience shows a cost of between 4-8 to 1 in writing code with unit tests
- This cost is minimal compared to the debugging cost incurred throughout a product lifecycle
- In other words, the disadvantages are few unless you have developers who unfailingly write “Bug-Free Code”
- Codes that are not structured according to acyclic design concepts may have prohibitive unit-test costs
- Finding and abiding the 80/20 rule takes developer experience

Yes, we actually do this



Final Thoughts

- Review one takeaway: The cost of defect resolution increases with time from defect introduction
- Use this as a guiding principle to improve productivity and tailor it to fit your needs—you don't need to do what we or others do!
- Applying this principle will sometimes add up-front costs, but it has the advantage of catching defects when they are introduced; this will result in significant savings downstream

Acknowledgments

- This manuscript has been authored by UT-Battelle, LLC, under Contract No. DE-AC0500OR22725 with the U.S. Department of Energy.
- This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nations exascale computing imperative.
- This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.