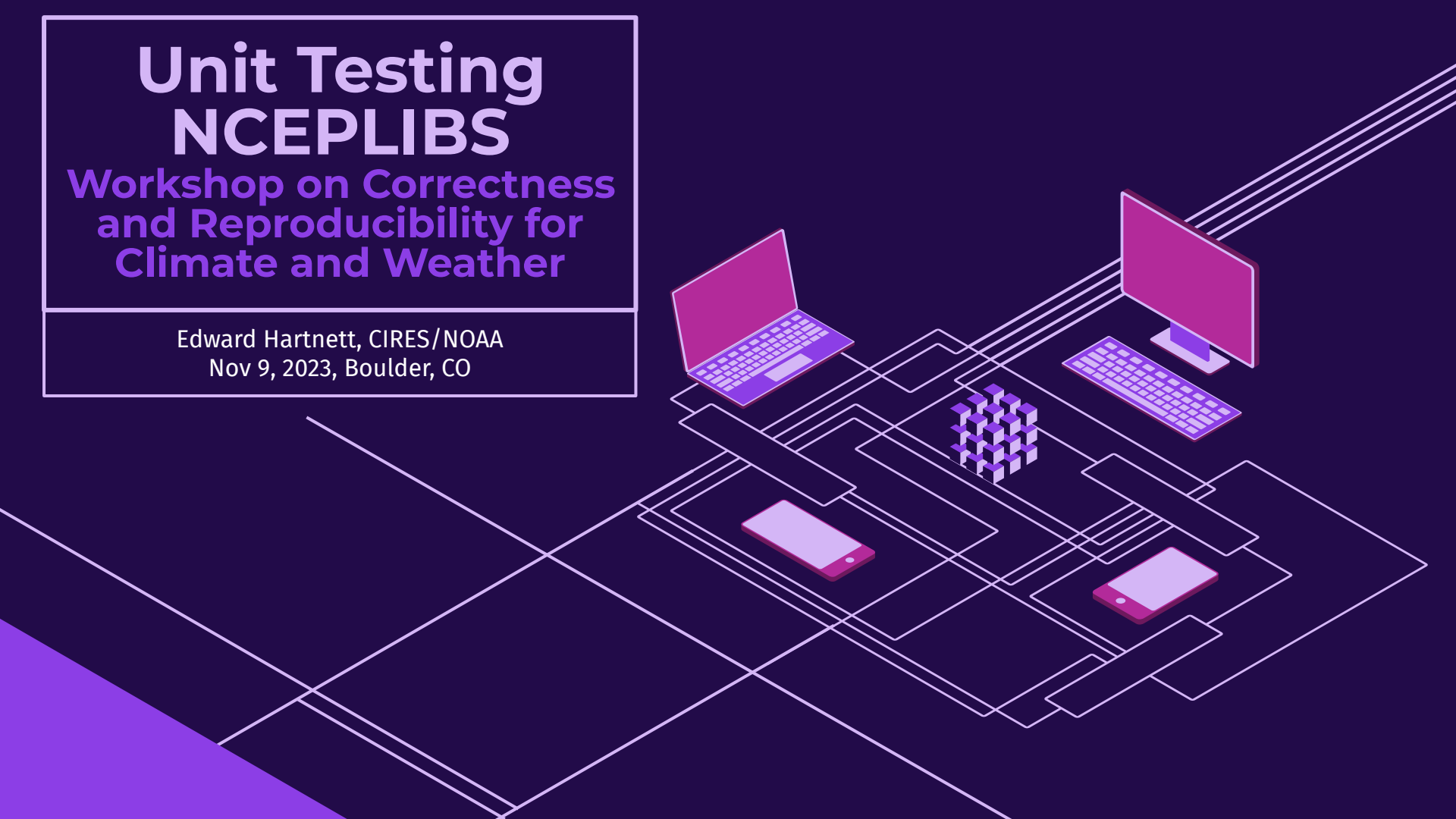
Unit Testing NCEPLIBS

Workshop on Correctness and Reproducibility for Climate and Weather

Edward Hartnett, CIRES/NOAA
Nov 9, 2023, Boulder, CO



Survey

How many routinely use unit testing when coding?

How many write software but have never written a unit test?

NCEPLIBS Libraries

- **Required libraries/utilities** for UFS and other NOAA applications
- 21 Repositories of Fortran 77 with some C/F90, not all active.
- Transitioned active repos to agile to **improve quality, reduce costs**
- Handles I/O, interpolation, **GRIB1/GRIB2, BUFR**
- Development roadmap:
  - 2020 - GitHub and CMake build systems
  - 2021 - Build system improvements
  - 2022 - CI, doxygen, unit testing
  - 2023 - Unit testing, refactoring
  - 2024 - Refactoring and modernization

NCEP library	Description
NCEPLIBS-bacio	binary I/O for the NCEP models
NCEPLIBS-bufr	working with the BUFR format
NCEPLIBS-g2	coding/decoding GRIB2 messages
NCEPLIBS-g2c	C decoder/encoder routines for GRIB edition 2
NCEPLIBS-g2tmpl	Utilities for GRIB2 templates
NCEPLIBS-gfsio	convert GFS Gaussian output into grib output
NCEPLIBS-ip	general interpolation library
NCEPLIBS-landsfcutil	initializing land-surface states
NCEPLIBS-ncio	read the GFS forecast files
NCEPLIBS-nemsio	I/O for the NCEP models using NEMS
NCEPLIBS-nemsiogfs	I/O for the NEMS-GFS model
NCEPLIBS-sfcio	surface files I/O
NCEPLIBS-sigio	I/O on the sigma restart file of the global spectral model
NCEPLIBS-sp	spectral transform library
NCEPLIBS-w3emc	decoder/encoder routines for GRIB edition 1
NCEPLIBS-w3nco	decoder/encoder routines for GRIB edition 1
NCEPLIBS-wrf_io	lightweight WRF-IO API library for Unified Post Processor (UPP)
NCEPLIBS-ip2	general interpolation library 2

Benefits of Unit Testing + CI

- Reduces risk of catastrophic software failure.
- Proven productivity gain (i.e. it's **much cheaper**).
- Testable code is modular, well-documented code with good APIs.
- Eliminates most **debugging**.
- Near-immediate feedback for programmer.
- Automatically run and results are automated.
- Packages can be individually tested (automatically) for **porting**.
- Many variants tested.
- Additional automatic checks like documentation, memory leak, compiler warning.

Unit testing dramatically increases programmer productivity.

Unit Testing Takes Discipline

- Testing needs to become central to development work.
- Tests must **check all output** carefully.
- Requires effort to fill in missing tests on legacy projects.
- Tests must be written before untested code is modified.
- Tests **must be provided with all new code**.
- Test code and CI system add on-going maintenance work.

Unit testing must be strict to yield full benefits.

Emotional Component

- For legacy programmers:

  more tests = more code = more pain.

- For modern programmers:

  more tests = more safety = faster coding.

Legacy programmers do not believe programming can be done in a better way.

Time is Not the Problem

- Programming without unit tests is very slow and inefficient.
- Tasks routinely take weeks or months, tests take hours.
- Once problem space is understood, very useful test code can be written in minutes/hours.

You always have time to program more efficiently.

System Tests are Not Enough

- System tests involve the whole codebase, so **debugging** is still required.
- Cannot test every code path.
- Output data changes are **hard to untangle**.
- Generally not immediate, generally manually run, sometimes manually interpreted.

System tests are necessary but not sufficient. Unit testing adds significant value and speeds development.

System Tests Cannot Detect All Bugs

- In UFS_UTILS test-writing, a bug was discovered in chgres_cube: “sets the SST values for small lakes that are far from other water bodies”
- This bug was not obvious and only found due to unit testing.
- Why degrade science results with undetected but avoidable software bugs?

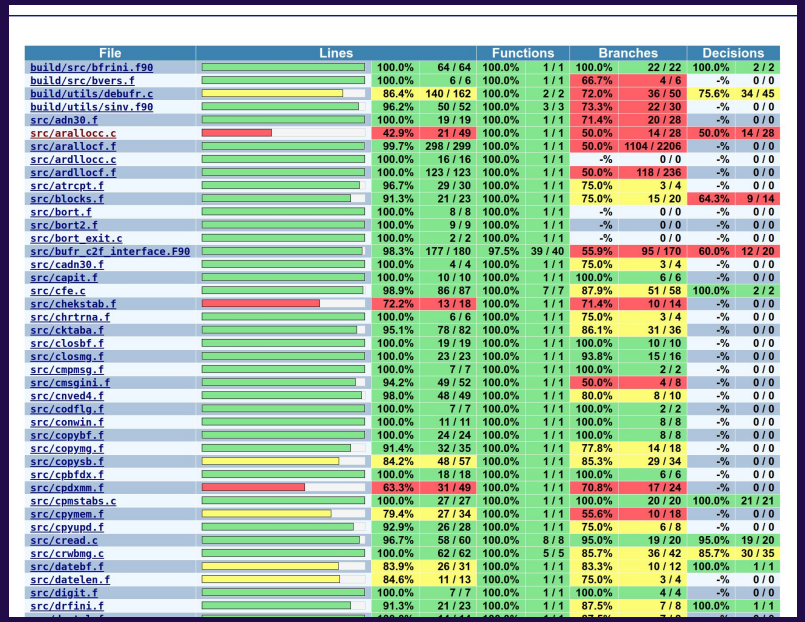
```
TEST FAILED

 SST SHOULD BE:   273.160000000000

   SST FROM TEST:   268.684500000000
```

https://github.com/NOAA-EMC/UFS_UTILS/issues/441

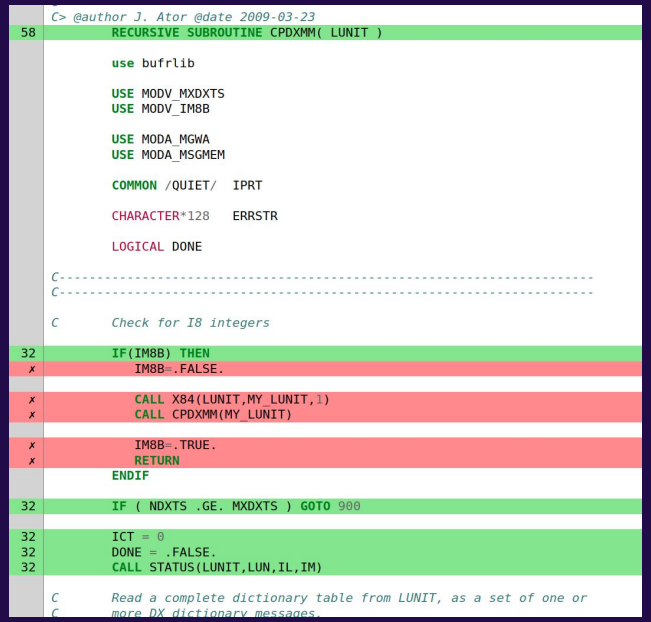
These avoidable bugs can degrade science results without being visible in system test results.

Unit Testing and Test Coverage

- Unit tests are small programs which call our code and check output for correctness.
- Unit tests are run by programmer/CI with “make test”. **All** tests must pass.
- All tests are automated - no human is required to decide if tests pass/fail.
- Test code coverage is measured to ensure we are testing all our code.



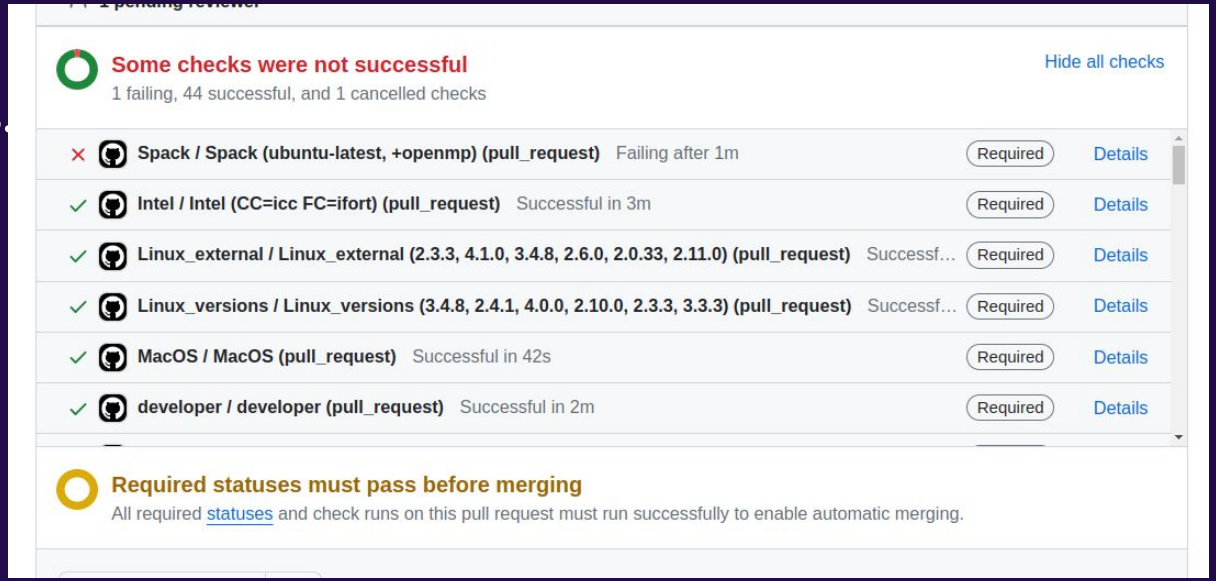
File	Lines			Functions		Branches		Decisions	
build/src/bfrini.f90		100.0%	64 / 64	100.0%	1 / 1	100.0%	22 / 22	100.0%	2 / 2
build/src/bvers.f		100.0%	6 / 6	100.0%	1 / 1	66.7%	4 / 6	-%	0 / 0
build/utils/debufr.c		86.4%	140 / 162	100.0%	2 / 2	72.0%	36 / 50	75.6%	34 / 45
build/utils/sinv.f90		96.2%	50 / 52	100.0%	3 / 3	73.3%	22 / 30	-%	0 / 0
src/adn30.f		100.0%	19 / 19	100.0%	1 / 1	71.4%	20 / 28	-%	0 / 0
src/arallocc.c		42.9%	21 / 49	100.0%	1 / 1	50.0%	14 / 28	50.0%	14 / 28
src/arallocf.f		99.7%	298 / 299	100.0%	1 / 1	50.0%	1104 / 2206	-%	0 / 0
src/ardllocc.c		100.0%	16 / 16	100.0%	1 / 1	-%	0 / 0	-%	0 / 0
src/ardllocf.f		100.0%	123 / 123	100.0%	1 / 1	50.0%	118 / 236	-%	0 / 0
src/atrcpt.f		96.7%	29 / 30	100.0%	1 / 1	75.0%	3 / 4	-%	0 / 0
src/blocks.f		91.3%	21 / 23	100.0%	1 / 1	75.0%	15 / 20	64.3%	9 / 14
src/bort.f		100.0%	8 / 8	100.0%	1 / 1	-%	0 / 0	-%	0 / 0
src/bort2.f		100.0%	9 / 9	100.0%	1 / 1	-%	0 / 0	-%	0 / 0
src/bort_exit.c		100.0%	2 / 2	100.0%	1 / 1	-%	0 / 0	-%	0 / 0
src/bufr_c2f_interface.F90		98.3%	177 / 180	97.5%	39 / 40	55.9%	95 / 170	60.0%	12 / 20
src/cadn30.f		100.0%	4 / 4	100.0%	1 / 1	75.0%	3 / 4	-%	0 / 0
src/capit.f		100.0%	10 / 10	100.0%	1 / 1	100.0%	6 / 6	-%	0 / 0
src/cfe.c		98.9%	86 / 87	100.0%	7 / 7	87.9%	51 / 58	100.0%	2 / 2
src/chekstab.f		72.2%	13 / 18	100.0%	1 / 1	71.4%	10 / 14	-%	0 / 0
src/chrtrna.f		100.0%	6 / 6	100.0%	1 / 1	75.0%	3 / 4	-%	0 / 0
src/cktaba.f		95.1%	78 / 82	100.0%	1 / 1	86.1%	31 / 36	-%	0 / 0
src/closbf.f		100.0%	19 / 19	100.0%	1 / 1	100.0%	10 / 10	-%	0 / 0
src/closmg.f		100.0%	23 / 23	100.0%	1 / 1	93.8%	15 / 16	-%	0 / 0
src/cmpmsg.f		100.0%	7 / 7	100.0%	1 / 1	100.0%	2 / 2	-%	0 / 0
src/cmsgini.f		94.2%	49 / 52	100.0%	1 / 1	50.0%	4 / 8	-%	0 / 0
src/cnved4.f		98.0%	48 / 49	100.0%	1 / 1	80.0%	8 / 10	-%	0 / 0
src/codflg.f		100.0%	7 / 7	100.0%	1 / 1	100.0%	2 / 2	-%	0 / 0
src/conwin.f		100.0%	11 / 11	100.0%	1 / 1	100.0%	8 / 8	-%	0 / 0
src/copybf.f		100.0%	24 / 24	100.0%	1 / 1	100.0%	8 / 8	-%	0 / 0
src/copymg.f		91.4%	32 / 35	100.0%	1 / 1	77.8%	14 / 18	-%	0 / 0
src/copysb.f		84.2%	48 / 57	100.0%	1 / 1	85.3%	29 / 34	-%	0 / 0
src/cpbfdx.f		100.0%	18 / 18	100.0%	1 / 1	100.0%	6 / 6	-%	0 / 0
src/cpdxmm.f		63.3%	31 / 49	100.0%	1 / 1	70.8%	17 / 24	-%	0 / 0
src/cpmstabs.c		100.0%	27 / 27	100.0%	1 / 1	100.0%	20 / 20	100.0%	21 / 21
src/cpymem.f		79.4%	27 / 34	100.0%	1 / 1	55.6%	10 / 18	-%	0 / 0
src/cpyupd.f		92.9%	26 / 28	100.0%	1 / 1	75.0%	6 / 8	-%	0 / 0
src/cread.c		96.7%	58 / 60	100.0%	8 / 8	95.0%	19 / 20	95.0%	19 / 20
src/crwbmg.c		100.0%	62 / 62	100.0%	5 / 5	85.7%	36 / 42	85.7%	30 / 35
src/datebf.f		83.9%	26 / 31	100.0%	1 / 1	83.3%	10 / 12	100.0%	1 / 1
src/datelen.f		84.6%	11 / 13	100.0%	1 / 1	75.0%	3 / 4	-%	0 / 0
src/digit.f		100.0%	7 / 7	100.0%	1 / 1	100.0%	4 / 4	-%	0 / 0
src/drfini.f		91.3%	21 / 23	100.0%	1 / 1	87.5%	7 / 8	100.0%	1 / 1



```
C> @author J. Ator @date 2009-03-23
58          RECURSIVE SUBROUTINE CPDXMM( LUNIT )

            use bufrlib

            USE MODV_MXDXTS
            USE MODV_IM8B

            USE MODA_MGWA
            USE MODA_MSGMEM

            COMMON /QUIET/  IPRT

            CHARACTER*128   ERRSTR

            LOGICAL DONE

C-----------------------------------------------------------------------
C-----------------------------------------------------------------------

C           Check for I8 integers

32          IF(IM8B) THEX
 ✗            IM8B=.FALSE.

 ✗            CALL X84(LUNIT,MY_LUNIT,1)
 ✗            CALL CPDXMM(MY_LUNIT)

 ✗            IM8B=.TRUE.
 ✗            RETURN
            ENDIF

32          IF ( NDXTS .GE. MXDXTS ) GOTO 900

32          ICT = 0
32          DONE = .FALSE.
32          CALL STATUS(LUNIT,LUN,IL,IM)

C           Read a complete dictionary table from LUNIT, as a set of one or
C           more DX dictionary messages.
```

Run on Each Commit on Each Pull Request

- Ubuntu/MacOS/Spack
- GNU/Intel compilers
- Versions of dependencies.
- CMake build options.
- Shared/static builds.
- Documentation check.
- Memory checking.
- Compiler warning check.
- Test code coverage analysis.



CI doesn't let anyone merge broken code, saving team time.

HOW WE WROTE NCEPLIBS TESTS

1. Set up testing in CMake, add first test, set up CI to run tests, test coverage analysis, memory checking.
   - No extra test “framework” is needed.
   - CMake provides nice default output.
2. Documentation upgrade to doxygen.
3. Convert existing manual tests to automatic testing.
4. Use code coverage analysis to fill in gaps (aim for 85% coverage).
5. Enforce test code coverage to ensure new code is tested.

Test writing is impactful, easy and fun.

NEW PRACTICES - CODING

When coding:

- Write tests first/concurrently with code.
- Update/improve documentation.
- Writing tests encourages modular code with good API.
- Make changes **quickly**, with **confidence that nothing will break**.

Unit tests allow programmers to improve the code with confidence and speed.

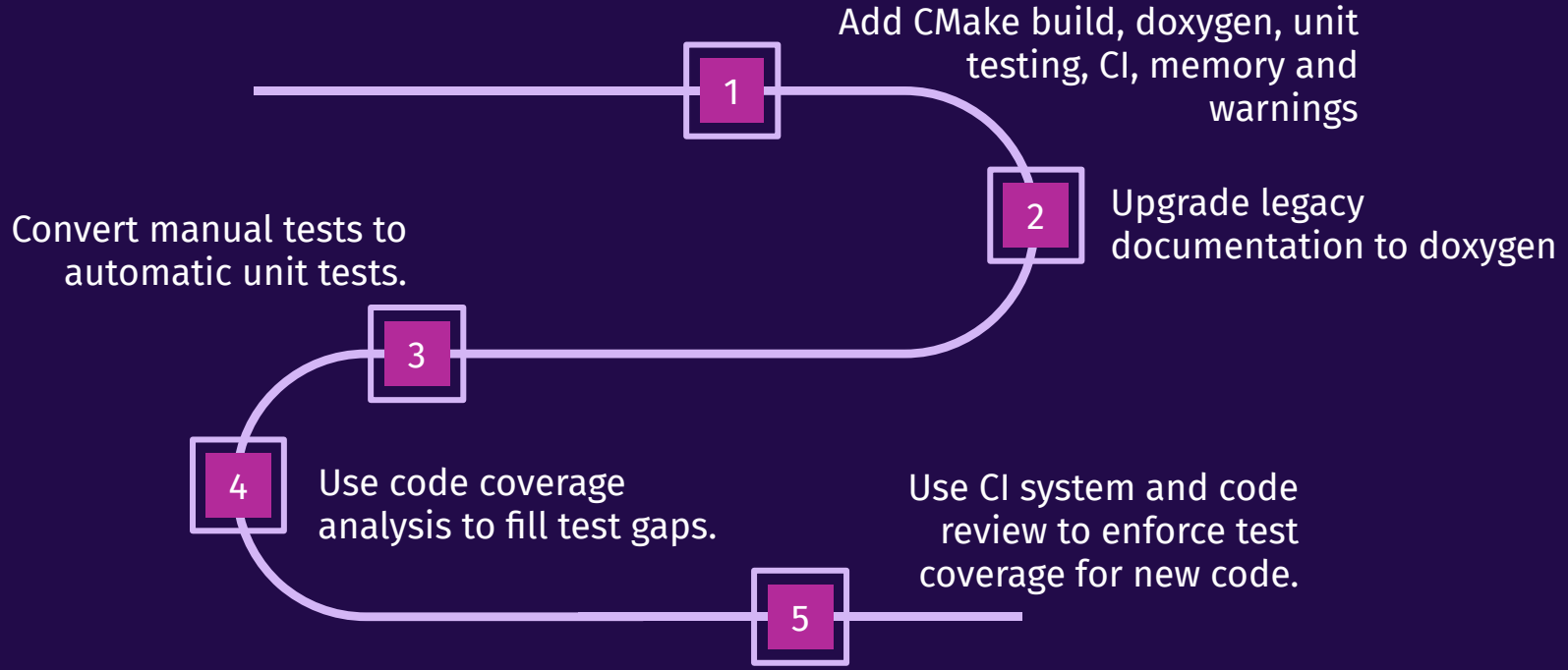
NEW PRACTICES - DEBUGGING

All NCEPLIBS bug investigations result in more unit tests:

- Start with a “**reproducer**” test at the top level.
- Continue to work down the call stack until problem is resolved.
- Tests remain part of the code.
- A final effort is made **generalize** the tests.

All NCEPLIBS Debugging Efforts Result in Tests which prevent recurrence.

ROADMAP to NCEPLIBS Unit Testing



NCEPLIBS Unit Testing Team

- Team size is small - part time effort from 3-4 FTEs.
- We had some help from student interns.
- We've been engaged in documentation/unit testing (part-time) for ~ 3 years.
- Total time devoted to testing: ~1 FTE for 1 year.

This allows us to maintain and release these codes more quickly and efficiently.

NCEPLIBS Unit Testing Coverage 11/23

Repository	Purpose	LOC	Test Code Coverage
NCEPLIBS-bacio	File I/O Fortran/C Library	1.4K	88%
NCEPLIBS-bufr	BUFR Fortran/C Library	32K	86%
NCEPLIBS-g2tmpl	GRIB2 Templates	3K	85%
NCEPLIBS-g2	GRIB2 Fortran Library	14K	75%
NCEPLIBS-g2c	GRIB2 C library	19K	74%
NCEPLIBS-ip	Fortran Interpolation Library	13K	65%
NCEPLIBS-sp	Fortran Spectral Interpolation	8K	53%
NCEPLIBS-grib_util	GRIB1/2 Utilities	22K	24%
NCEPLIBS-w3emc	GRIB1 Fortran Library	63K	2%
NCEPLIBS-prod_util	UFS Utilities (mostly GRIB1)	0.5K	0%

NCEPLIBS Community Contributions

- We've already had significant outside contributions to the GRIB2 code.
- We are happy to work with outside developers (from trusted partners, with detailed review).
- The CI systems impersonally enforces many community contribution requirements.
- Unit testing gives team and contributor confidence.
- We can help with unit testing if that's needed.
- **Team time is saved** when contributions can be checked before human intervention.

Unit tests and CI lower the bar for community contributions.

Unit testing is an engineering best practice universal in the software industry.

Unit testing is a discipline which allows us to program faster and spend less time on maintenance.

AGU Poster

IN51B-0416: NCEPLIBS GRIB and BUFR Libraries: Maintaining and Modernizing NOAA's Libraries for WMO Data Formats

Edward Hartnett, Jeff Ator, Hang Lei, Alexander Richert, Jack Woolen, A. King, A. Hartnett

ABSTRACT

The software libraries used at NOAA to read and write data files for WMO formats GRIB and BUFR have been under continuous development for at least the last 30 years.These libraries play a vital role in many operational systems at NOAA, and are also used by teams and scientists outside NOAA. In this paper we describe the libraries and their capabilities, and document recent modernization efforts. We trust that these libraries will continue to provide value to NOAA and the science community for decades to come.

Session Number and Title: IN51B: Advances in Earth and Space Science Informatics
Session Date and Time: Friday, December 15th; 8:30 AM – 12:50 PM PST

Location: Moscone Center, South, Poster Hall A-C

CONCLUSION

There has been much improvement in NCEPLIBS testing and automation.

The NCEPLIBS libraries and utilities are agile, portable, and benefit from community contributions. They are ready to provide value for NOAA for decades to come.