UNIT TESTING IN HEP

MARIAN HEIL

IPPP, DURHAM
EXAMPLE SHERPA

- **Years of development**
  
  First release 2003 (Herwig and Pythia even older)
  
  9600 git commits since then (1.5 commits/day)
  
  Oldest unmodified file from 2005

- **Many developers**

  15 different authors on release notes, and at least 31 different code contributors

- **Large code bases**

  1100 source and 700 header files for a total of >280k LOC (+18k comments)
  
  At a rate of 2 LOC/h this makes **16 years of non-stop writing** for a single person

No one can know all of the Code!
1. Implementation was always broken
2. Introduce bugs in old (unrelated) code
3. Breaking interfaces and different builds
1. Implementation was always broken
   No proper validation

2. Introduce bugs in old (unrelated) code
   No proper regression testing

3. Breaking interfaces and different builds
   No proper testing of different setups

4. Automate testing
Example: `t_main` from HEJ

```plaintext
add_test(
  NAME t_main
  COMMAND HEJ ${tst_dir}/jet_config.yml ${tst_dir}/2j.lhe.gz
)
```
NAIVE EXECUTION

+ Simple to write
+ Closest to reality
- Unspecific
- Only catches most severe bugs

Relies on self-checking code
e.g. through `throw` or `assert`
Example: `shtest-yodacnv` from YODA

```
#!/bin/bash
set -e
# Run program
yodacnv ${YODA_TESTS_SRC}/test1.yoda yodacnv.dat
# compare to reference
diff -u yodacnv.dat ${YODA_TESTS_SRC}/yoda2flat-ref.dat
rm -f yodacnv.dat
```
Example: Martix element for multiple PSP

test_ME_generic in HEJ

```c++
for(auto const & event: events){
    const double our_ME = ME.tree(event).central;
    if(!std::getline(wgt_file,line)) break;
    const double ref_ME = std::stod(line);
    if( std::abs(our_ME/ref_ME-1.) > EP )
        return EXIT_FAILURE;
}
```
COMPARE TO REFERENCE DATA

+ Quick to write
+ Easy to understand
- External validation
- Easy to miss edge-cases

Use if result is otherwise unknown
  e.g. cross section, matrix element
Example: Symmetries in matrix element

```c++
for(auto const & event: events){
    const double our_ME = ME.tree(event).central;
    if(!std::isfinite(our_ME))
        return EXIT_FAILURE;
    // invariant under p->-p
    auto reverse = reverse_event(event);
    const double reverse_ME = ME.tree(reverse).central;
    if(std::abs(our_ME/reverse_ME-1.)>EP_REVERSE)
        return EXIT_FAILURE;
    ...
}
```
Example: `testMatVec.cc` from rivet

```cpp
const Vector3 v3_9090i = rot90*rot90.inverse()*v3;
const Vector3 v3_90i90 = rot90.inverse()*rot90*v3;
assert(fuzzyEquals(v3, v3_9090i));
assert(fuzzyEquals(v3, v3_90i90));
```
AUTOMATE VALIDATION

+ Can find new bugs
+ Done anyway for validation
  ○ Tests exactly/only what you want
    Check coverage with e.g. gcovr

- Needs a priori knowledge
- Hard to write & understand

Standard, use whenever possible
e.g. interfaces, symmetries, bug fixes
RUNNING THE TESTS

Register each test in build framework, e.g. in CMake

```cmake
# CMakeLists.txt
add_test(
  NAME test_name
  COMMAND my_test_exe
)
```
and execute all of them

$ ctest # or make test
Test project /path/to/HEJ/build
Start 1: classify
1/45 Test #1: classify ......................... Passed 0.47 sec
...
Start 45: reconstruct_W
45/45 Test #45: reconstruct_W ................. Passed 0.28 sec

100% tests passed, 0 tests failed out of 45

Total Test time (real) = 14.86 sec
# .gitlab-ci.yml

build:
  image: DOCKER_IMAGE  # vary image to test different builds
  stage: build
  script:
    - mkdir build && cd build
    - cmake ..
    - make -j $(nproc)
  artifacts: # save build folder
    paths:
      - build

test:
  image: DOCKER_IMAGE  # vary image to test different builds
  stage: test
  script:
    - ctest -j $(nproc)
  needs: # download build
  - job: build
  artifacts: true

Requires Docker container with all necessary software

e.g. hepstore/hepbase-xyz

dockers for HEJ based on gitlab.dur.scotgrid.ac.uk/hepsw
AUTOMATED TESTING ON GITLAB

Streamline HepMC interface
- removed `add_variation` function (now in init_event)
- renamed `init_kinematics` -> `init_event` & protected it

16 jobs for 102 with rivet3 in 8 minutes and 9 seconds (queued for 7 minutes and 55 seconds)
AVOIDING BUGS

1. Check initial implementation
   - Documentation (Doxygen, Spinx, link to paper, ...)
   - Static code analysis (valgrind, clang-tidy, flake8, ...)
   - Validation

2. Don't break existing code
   - Rerun validation
   - Compare to previous results
   - Keep track of bug (fixes)

3. Test different builds
   - Multiple docker images
   - Get user feedback (bug reports or merge requests)
THANK YOU!