CORSIKA 8
Air-Shower Simulation and Development Workshop
A workshop to discuss the technology and science of air-shower simulations with CORSIKA 8, and meeting of CORSIKA 8 developers and developers-to-be.
Organization
Max Planck Institute for Nuclear Physics Heidelberg
12-15 July 2022

CORSIKA 8
Physics Validation (and Verification)

Lukas Nellen
ICN-UNAM
lukas@nucleares.unam.mx

CORSIKA 8 workshop, Heidelberg — 2022/07/13

Validation vs Verification

https://en.wikipedia.org/wiki/Software_verification_and_validation

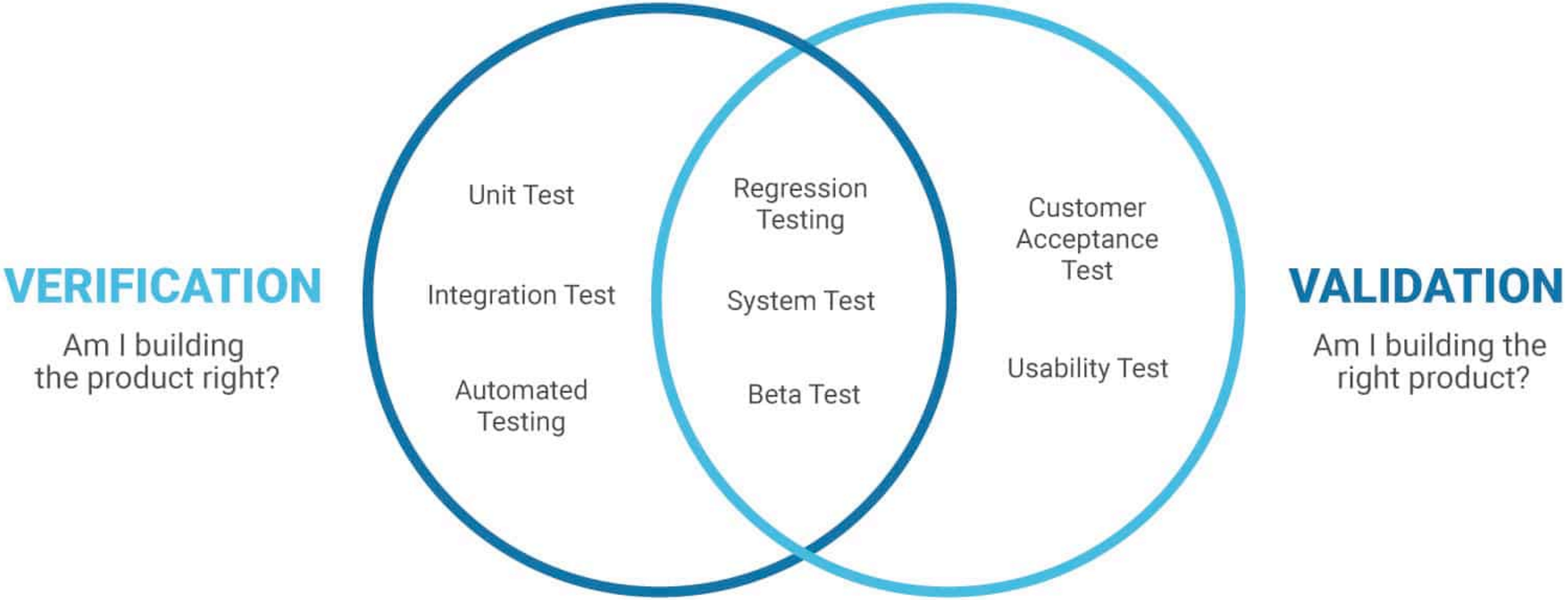
- Validation: Are we building the right product?
  - Pretty sure we do: specifications
    - See opening session
  - Validate physics:
    Code has to produce correct physics
    - Reasonable compared to existing codes
    - Reasonable compared to expectations
    - Handle the intended use cases

- Verification: Are we building the product right?
  - Unit tests
  - Do test applications run?

- Formal validation/verification is expensive

  …formal methods can prove costly, however, representing as much as 80 percent of total software design cost

Validation vs Verification



VERIFICATION

Am I building
the product right?

Unit Test

Integration Test

Automated Testing

Regression Testing

System Test

Beta Test

Customer Acceptance Test

Usability Test

VALIDATION

Am I building the
right product?

Verification and Validation topics

- How
  - Test technology
  - Which process to test:
    Desirable: ALL – Feasible? Always?
  - Applications for testing
    Cover a large part of configuration space
- When (and what): from fast (frequent) to time consuming
  - On each commit (push)
  - Before/with each merge
  - Periodically: weekly / monthly
  - **Before release**

Reasonable execution times (discuss!)

- Provide times as wall-time
  - Consider release builds for realistic testing

- Push: < 30 min
  - Build, unit tests; gcc, clang, production distro?
- Merge: <180 min
  - Build, unit tests, core validation (define!) (decide if clang needed)
- Weekly: <day
  - More complete tests
  - Larger set of architectures and compilers
- Release: week to month

Verification

Verification and Basic Validation

Full Validation

Extended Validation

- More complex or demanding tests require more computing power for testing

What to validate

- Numbers
- Distributions
  - histograms
  - graphs
- Avoid numbers with large, relative changes

Validation tools

- Numbers
  - Floating point: size of difference, never test for equality
  - Absolute and relative comparisons

- Graphs, histograms
  - internal: vectors or matrices (treat like vector?)
  - Use standard metrics
    - All bins (both needed?)

$$||u-v||_1 = \sum_{\text{bins}} |u_i - v_i| \qquad ||u-v|| = \sum_{\text{bins}} (u_i - v_i)^2$$

    - Biggest difference

$$||u-v||_\infty = \max_{\text{bins}} |u_i - v_i|$$

  - Could include bin specific weights, e.g., to reduce importance of tails

Validation applications

- Standard CORSIKA 8 configuration
  - compiled for release

- Need to collect typical scenarios
  - Energies
  - Primaries
  - Geometries
  - Environments

- How to implemente validation
  - Integrate validation into CORSIKA framework and applications
  - Alternative: analyze output in separate program

Be generous with tolerances: lesson from Auger

- Expect the unexpected
  - System upgrade can cause changes
  - Seen changes from glibc
    Expected compiler dependencies
- Need to work across range of distributions

I finally ran enough tests to get the clear impression that the numerical changes causing more validations to fail are a result of changes in glibc, not in the compiler.

glibc 2.27 and older
  G4StationSim (expected)
glibc 2.28
  G4StationSim (expected)
  RdSdCheck
  RdSdSimCheck
glibc 2.29 and newer
  HdValidatrix
  G4StationSim (expected)
  RdSdCheck
  RdSdSimCheck

There are two things that worry me (slightly):

- *glibc changes affect numerical results. This is across distributions*
- *We develop on recent glibc, but our grid production runs on a distribution with an oldish glibc (2.17). Containers can help here*

To Do (1)

- Prepare tools for validation
  - Strategy how to build validating applications

- Develop standard application(s) for C8
- Configuration for reasonable use cases
  - Look at types of experiments
  - Highest Energy (Auger, JEM/EUSO, GCOS)
  - HAWC/SWGO/LHAASO
  - IACT (CTA, …)
  - Solar physics etc (single particle, detector BG, …)
  - More … (feedback needed, maybe consult C7 users)

- Implement in CI

To Do (2)

- Plan what to run when
  - Evaluate CPU time and wall time
  - Determine resources needed to support regular development

- Different distributions
  - Identify main distributions used on production sites

- Special plan for verification / validation before releases
  - How much statistics is needed?
  - Computing resources?