CHEP 2015
13-17 April 2015, Okinawa, Japan

Testable physics by design

Maria Grazia Pia
INFN Genova, Italy

C. Choi, M. C. Han, S. Hauf, G. Hoff, C. H. Kim, H. S. Kim, S. H. Kim, M. Kuster, P. Saracco, G. Weidenspointner

Hanyang University, Seoul, Korea
EU XFEL GmbH, Hamburg, Germany
CAPES, Brasilia, Brazil
MPE, Garching, Germany

Foreword
Due to limited time allocation, there is room to highlight concepts only. Details will be documented and discussed in a dedicated journal publication.
I have a dream...

Maria Grazia Pia, INFN Genova

The simulation is not validated
In the literature...

- Limited documentation of **simulation validation**
  - Mostly in the form of specific use cases compared to measurements in the same experimental scenario
    - Do they apply to similar/different use cases?
    - How to extrapolate the results to different scenarios?

- Hardly any **validation of basic physics models** implemented in Monte Carlo codes
  - Why?

- Ongoing projects on **uncertainty quantification**
  - See CHEP 2013, P. Saracco et al.
  - Methods to predict the uncertainty of simulation observables based on knowledge of the uncertainties of simulation “ingredients”
You need an experiment to test a cross section

Testing total cross sections calculated by *G4PEEffectFluoModel*

You can find the photoelectric cross section `G4PEEffectFluoModel` class in `$G4INSTALL/source/processes/electromagnetic/standard/include (G4PEEffectFluoModel.hh header file )` and `$G4INSTALL/source/processes/electromagnetic/standard/src/ (G4PEEffectFluoModel.cc implementation)`. `G4PEEffectFluoModel` has a `ComputeCrossSectionPerAtom` public member function, which returns the total photoelectric cross section for a given element corresponding to a given photon energy:

```cpp
G4double ComputeCrossSectionPerAtom(const G4ParticleDefinition*,
        G4double kinEnergy,
        G4double Z,
        G4double A,
        G4double, G4double)
```

This is what we need indeed!

We create a simple unit test `G4PEEffectFluoModelTest.cc`, which instantiates a `G4PEEffectFluoModel` object and invokes `ComputeCrossSectionPerAtom` in pre-defined configurations of photon energy and target element. We place the unit test in `$APCDIR/test`.

We build the test:

```bash
cd $APCDIR/test
setenv TESTTARGET G4PEEffectFluoModelTest
gmake
```

Then we run the test:

```bash
$G4WORKDIR/bin/Linux-g++/G4PEEffectFluoModelTest
```

Maria Grazia Pia, *INFN Genova*
Post-RD44 electromagnetic software design

Hidden dependencies on other parts of the software

One needs a geometry (and a full scale application) to test any photon cross section

Difficult to test $\Rightarrow$ no testing often

Reverse engineered

Do UML diagrams exist? Are they maintained?
Peer reviews?

Maria Grazia Pia, INFN Genova
Photoionisation cross section

Cross sections in Geant4 “standard” photoelectric model are based on “improved” Biggs-Lighthill parameterisation

Detangling

Testable
Open - closed

Can be validated in a unit test

New models

Photoionisation

Cross section models can be compared with statistical categorical tests

Handles any tabulated cross section

New models

G4CsTabula

+ CrossSection(2, int, e: double): double

G4CsComptonPenelope

+ CrossSection(2, int, e: double): double

G4CsComptonStandard

+ CrossSection(2, int, e: double): double

G4CsPhotoelectricBiggs

+ CrossSection(int, double): double
+ CrossSection(int, int, double): double

G4CsPhotoelectricEbel

+ CrossSection(int, double): double
+ CrossSection(int, int, double): double

G4CsPhotoelectricVerner

+ CrossSection(int, double): double
+ CrossSection(int, int, double): double

G4CsTabula

+ CrossSection(int, double): double
+ CrossSection(int, int, double): double
1. Continuing Change

- A program that is used and that as an implementation of its specification reflects some other reality, **undergoes continual change** or becomes progressively less useful. The change or decay process continues until it is judged more cost effective to replace the system with a recreated version.

2. Increasing Complexity

- As an evolving program is continually changed, **its complexity**, reflecting deteriorating structure, **increases** unless work is done to maintain or reduce it.
Refactoring

is a disciplined technique for improving the design of an existing code

“Refactoring is the process of **changing** a software system in such a way that it does not alter the **external behavior** of the code yet improves its internal structure.”

Refactoring begins by designing a **solid set of tests** for the portion of code under analysis

Is this all what we need?
Sweeping under the carpet?

Was the original code verified?

Was the original code validated?

What was the test coverage?

Were the test process and the test results documented?

By improving the design, **refactoring can be a technique to make software testable**

IEEE Standard 1012
Software Verification & Validation
ISO 12207

Maria Grazia Pia, INFN Genova
Testing à la Feather

Legacy code often lacks tests

Techniques to make existing code testable

1. Identify change points
2. Find an inflection point
3. Cover the inflection point
   a. Break external dependencies
   b. Break internal dependencies
   c. Write tests
4. Make changes
5. Refactor the covered code

A narrow interface to a set of classes
If anyone changes any of the classes behind an inflection point, the change is either detectable at the inflection point, or inconsequential in the application

If the class we want to cover creates its own objects internally

Techniques to deal with irritating parameters, hidden dependencies etc.

Feasible, but painful...
Most of these problems can be easily solved if we simply write tests as we develop our code
- …and we *maintain* the tests
- …and we *regularly execute* them
- …and we *investigate* the reasons for failure

If a test is hard to write, that means that we have to find a different design which is testable

- It is always possible
- **Software design reviews**: care about testability
// G4HadronElastic
// 29 June 2009 (redesign old elastic model)

G4double dd = 10.;
G4Pow* g4pow = G4Pow::GetInstance();
if (A <= 62) {
    bb = 14.5*g4pow->Z23(A);
    aa = g4pow->powZ(A, 1.63)/bb;
    cc = 1.4*g4pow->Z13(A)/dd;
} else {
    bb = 60.*g4pow->Z13(A);
    aa = g4pow->powZ(A, 1.33)/bb;
    cc = 0.4*g4pow->powZ(A, 0.4)/dd;
}

Testable?  
Calibrated?  
Epistemic uncertainties?

G4ChipsAntiBaryonElasticXS
lastPAR[43]=920.+03*a8*a3;
lastPAR[44]=93.+.0023*a12;

G4GoudsmitSaundersonMscModel
if(i>=19)ws=cos(sqrtA);

G4EmCorrections
if(15 == iz) {
    if(3 > j) { tet = 0.25*Z2*(1.0 + 5*Z2*alpha2/16.); } 
    else { tet = 0.25*Z2*(1.0 + Z2*alpha2/16.); } 
}

Epistemology!

G4UrbanMscModel
Detector design, experimental strategies, physics results depend critically on software

…which is often untested because it is untestable

– Or became untestable in the course of its evolution

Making software testable

– Improving software design (refactoring)
– Breaking dependencies (techniques à la Feathers)
– Embedding testability in the software design

Testability must be maintained

Epistemological issues: domain knowledge and implementation details

Ongoing effort to make Geant4 physics testable and to test it

http://www.ge.infn.it/geant4/papers