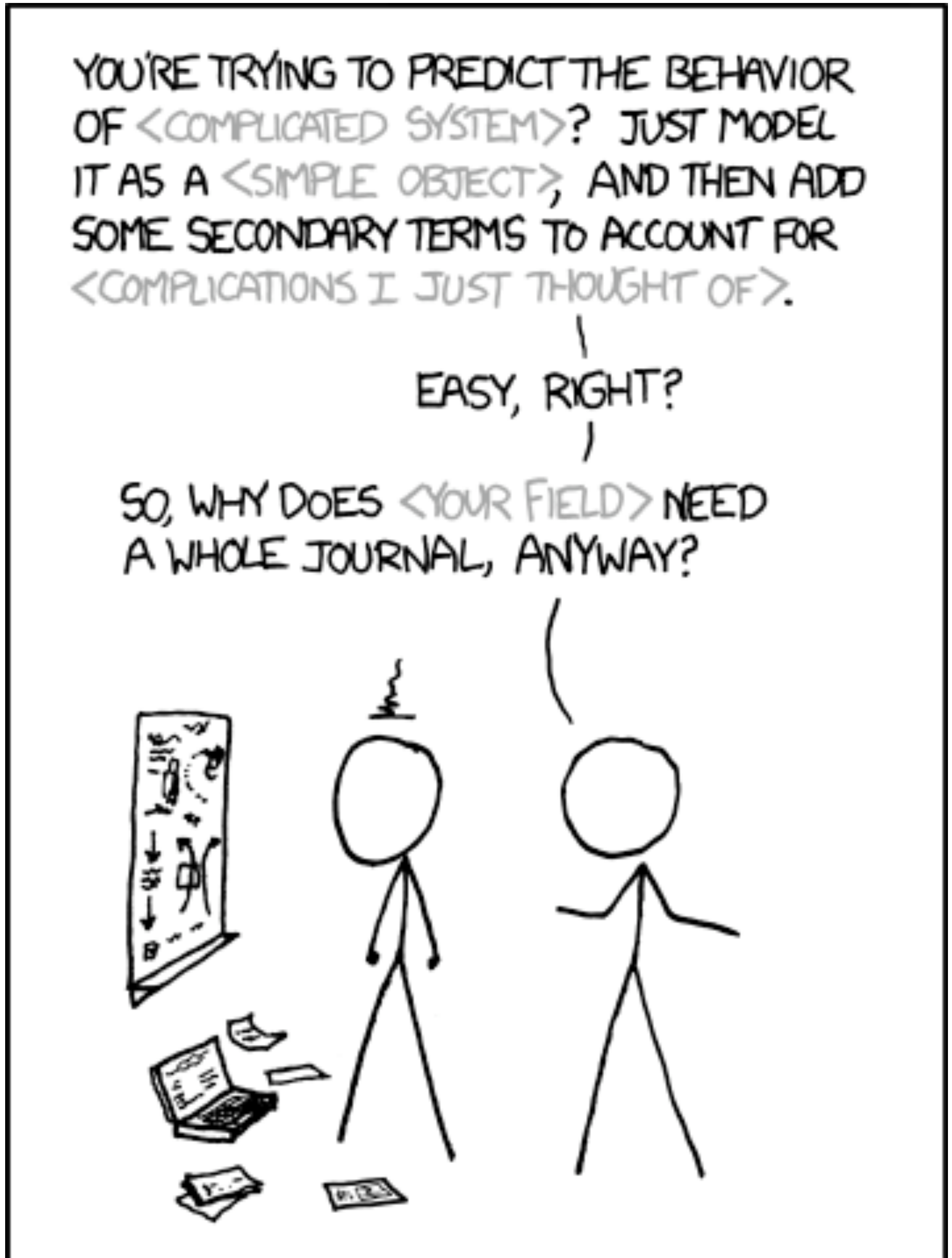


LIBERAL-ARTS MAJORS MAY BE ANNOYING SOMETIMES, BUT THERE'S *NOTHING* MORE OBNOXIOUS THAN A PHYSICIST FIRST ENCOUNTERING A NEW SUBJECT.

Disclaimer

The views here are only of the author and neither represent MICE nor Oxford. There are many ways to run a project. Please interrupt since so this is a discussion rather than preaching.

<talk>

Software Engineering in Particle Physics

Christopher Tunnell

Software Engineering in Particle Physics

Christopher Tunnell

Introduce some software engineering *stuff*

Software Engineering in Particle Physics

Christopher Tunnell

Introduce some software engineering *stuff* then describe the state of things in our field.

Software Engineering in Particle Physics

Christopher Tunnell

Introduce some software engineering *stuff* then describe the state of things in our field. Then next compare us to opensource projects

Software Engineering in Particle Physics

Christopher Tunnell

Introduce some software engineering *stuff* then describe the state of things in our field. Then next compare us to opensource projects before trying to explain useful lessons from industry.

Software Engineering in Particle Physics

Christopher Tunnell

Introduce some software engineering *stuff* then describe the state of things in our field. Then next compare us to opensource projects before trying to explain useful lessons from industry. Finally, I want to compare what we've learned to a HEP case study.

Software Engineering in Particle Physics

Christopher Tunnell

1. Introduce some software engineering *stuff*
2. describe the state of things in our field.
3. compare us to opensource projects
4. explain useful lessons from industry.
5. compare what we've learned to a HEP case study.

Software Engineering in Particle Physics

Christopher Tunnell

1. Introduce some software engineering *stuff*
2. describe the state of things in our field.
3. compare us to opensource projects
4. explain useful lessons from industry.
5. compare what we've learned to a HEP case study.

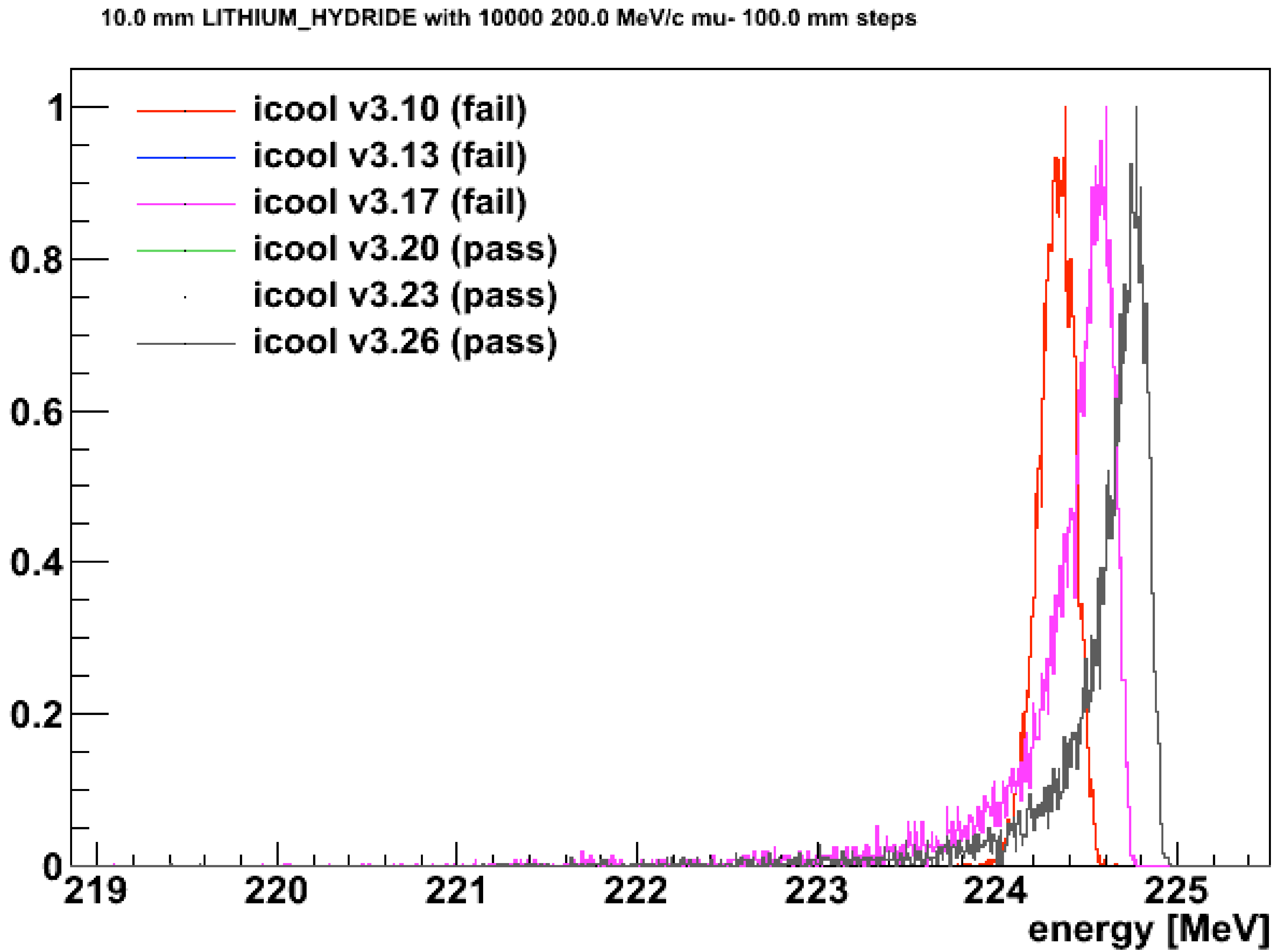
</>

Software Engineering

is a profession dedicated to designing, implementing, and modifying software so that it is of higher quality, more affordable, maintainable, and faster to build.

wikipedia


F-WWOW AJ-DM
F-WWOW AJ-DM

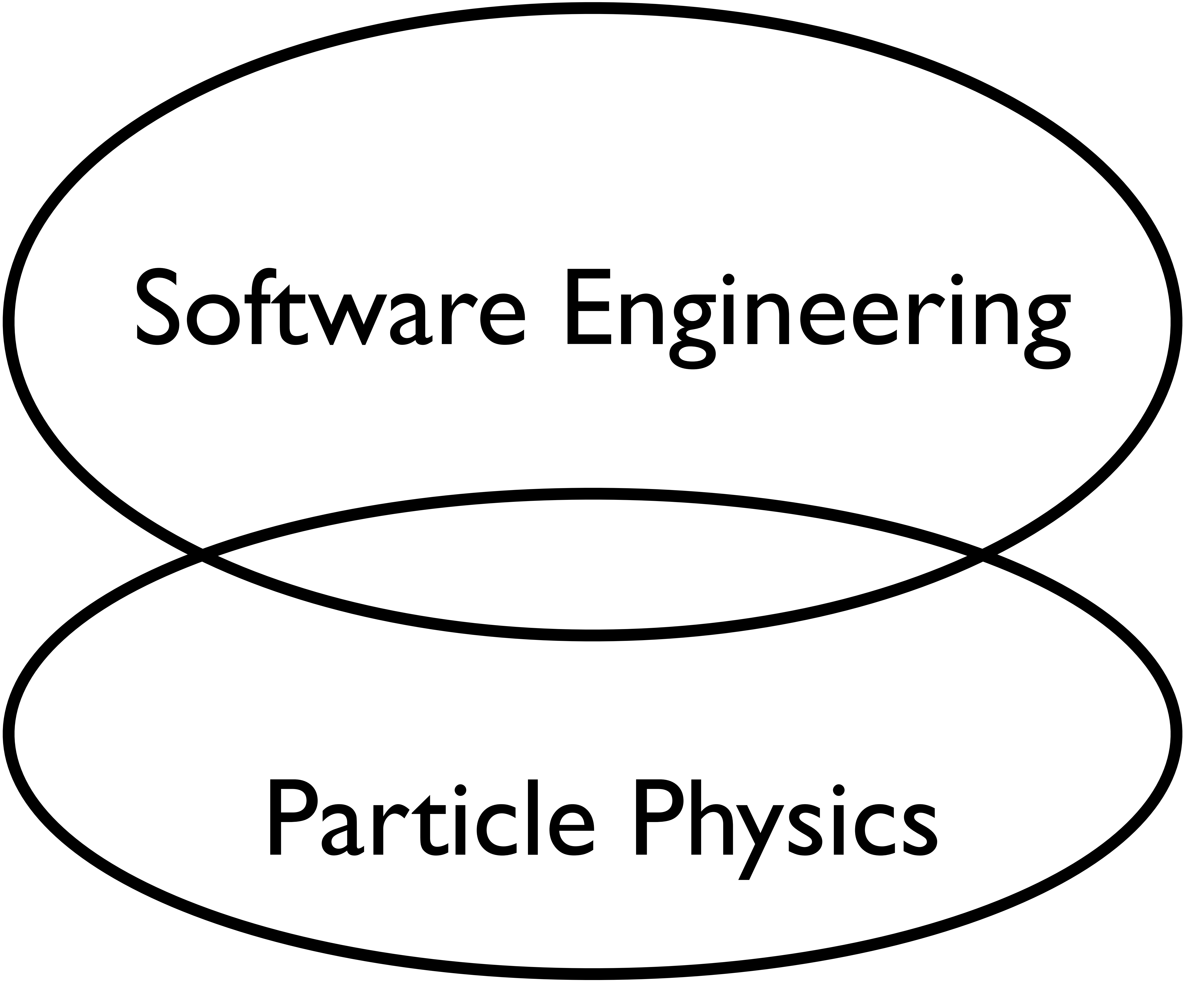


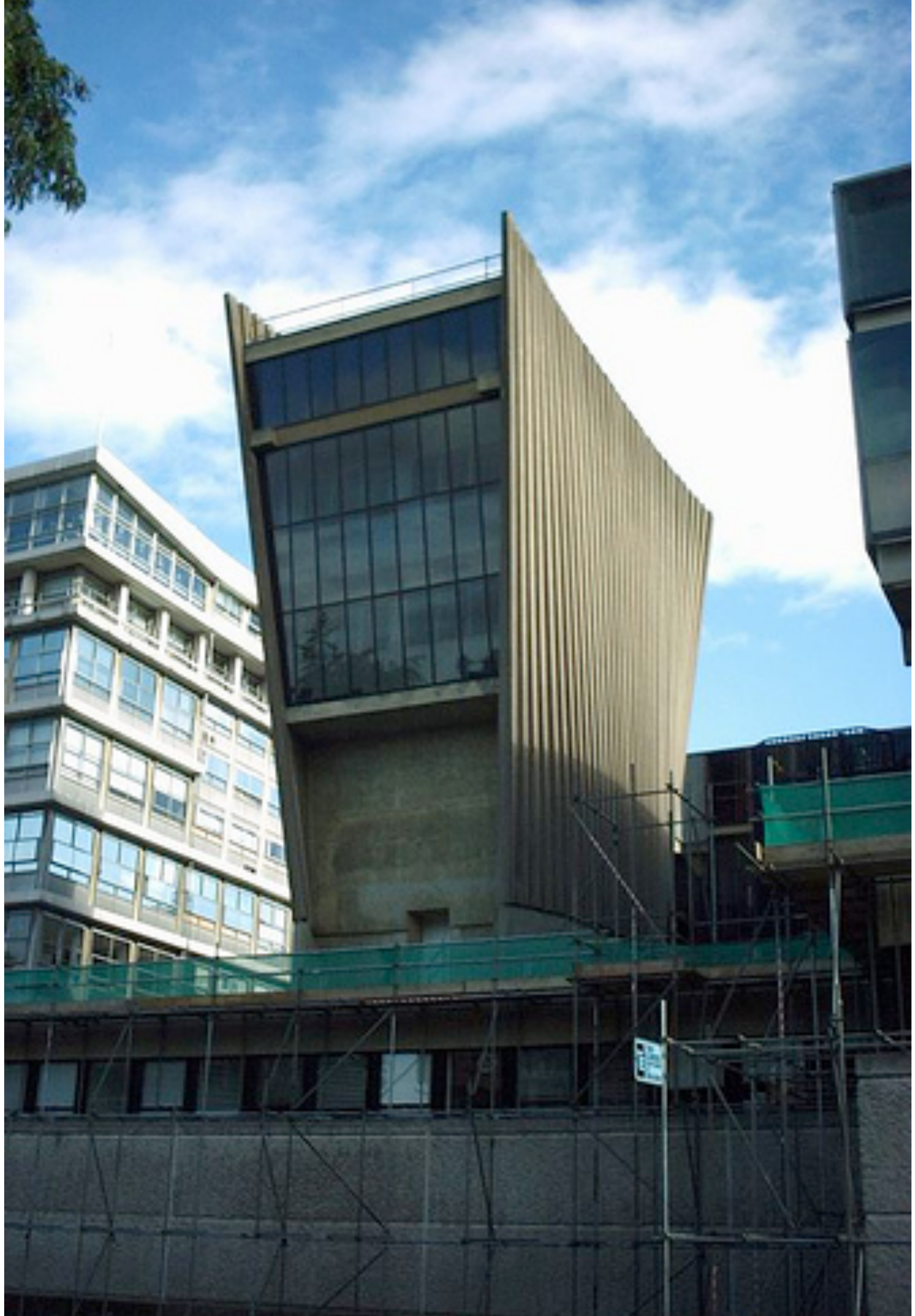
Work by Chris Rogers (STFC)

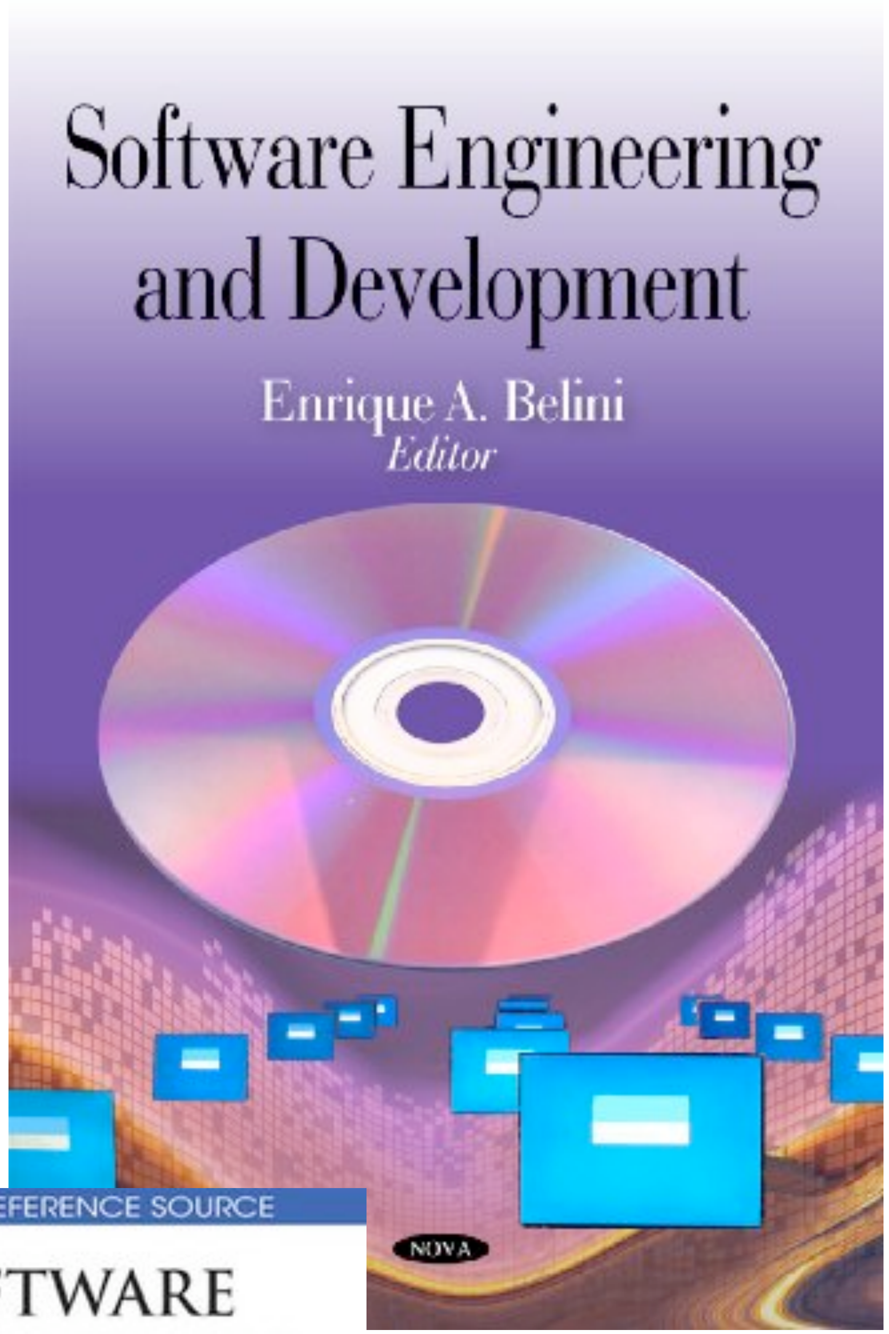
With respect to the recognition of the need for greater reliability of software, I expect no disagreement anymore. Only a few years ago this was different: to talk about a software crisis was blasphemy. The turning point was the Conference on Software Engineering in Garmisch, October 1968, a conference that created a sensation as there occurred the first open admission of the software crisis. And by now it is generally recognized that the design of any large sophisticated system is going to be a very difficult job, and whenever one meets people responsible for such undertakings, one finds them very much concerned about the reliability issue, and rightly so. In short, our first condition seems to be satisfied.

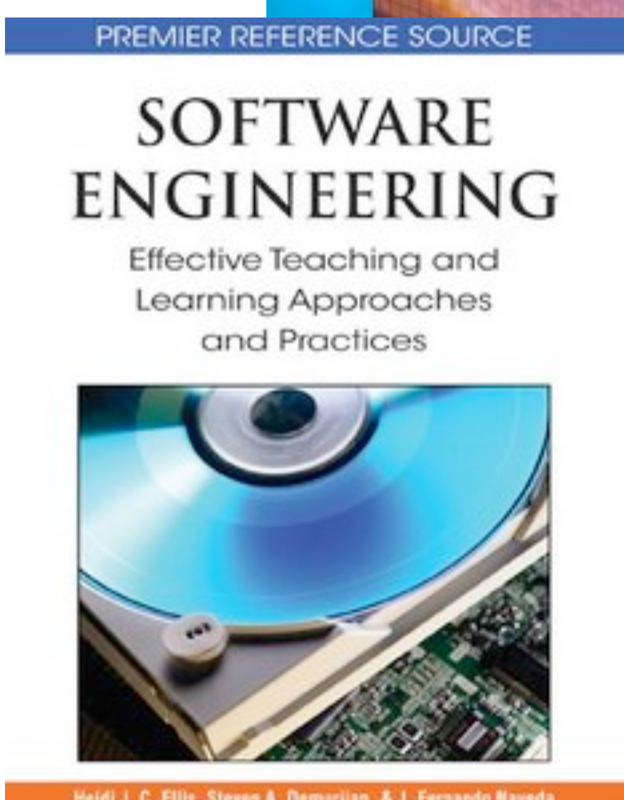
-Dijkstra

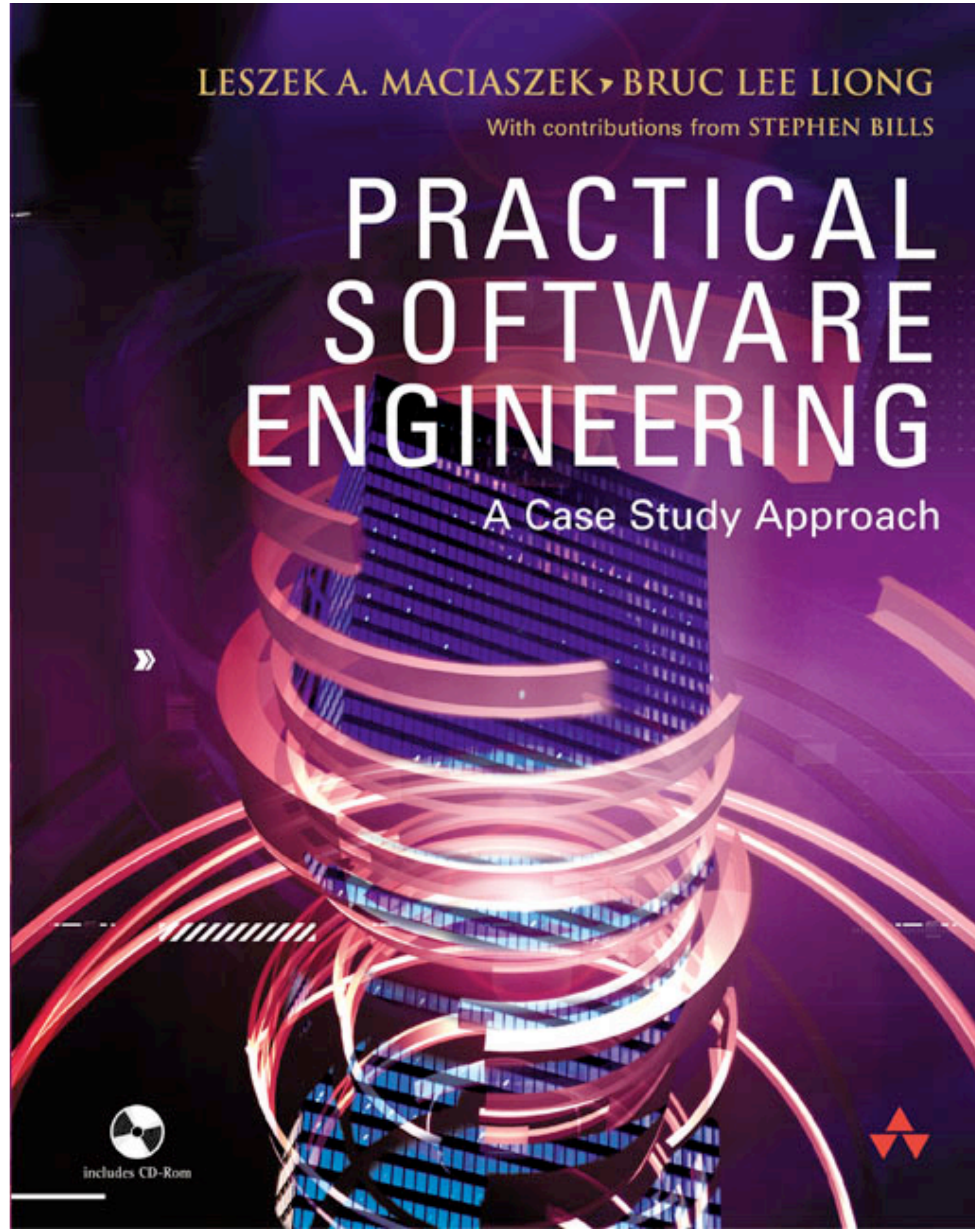
Software Engineering


Software Engineering
Particle Physics




Software Engineering and Development
Enrique A. Belini
Editor
NOVA


PREMIER REFERENCE SOURCE
SOFTWARE ENGINEERING
Effective Teaching and Learning Approaches and Practices
Heidi J. C. Ellis, Steven A. Demurjian, & J. Fernando Naveda


LESZEK A. MACIASZEK BRUC LEE LIONG
With contributions from STEPHEN BILLS
PRACTICAL SOFTWARE ENGINEERING
A Case Study Approach
includes CD-Rom

Lingo

Particle Physics	Software Engineering
Renormalization Offshell Weak currents Data quality Luminosity QED	Refactoring Sprints Agile Regression tests Continuos Integration RTFM

</I>

<2>

State of our field: Three Points

1. Physicists write software and
2. initially, there is a training problem for basic software skills but also
3. our software culture makes on the job training impossible

State of our field: Three Points

1. Physicists write software

Survey

- All computational sciences
- Published in English so mainly USA, Canada, UK, and Northern Europe
- ~50% researchers, ~25% grad. students, ~25% technicians/managers
- ~10% physicists of some sort and 1 "theological engineer" (who was removed)

Jo Erskine Hannay, Hans Petter Langtangen, Carolyn MacLeod, Dietmar Pfahl, Janice Singer, and Greg Wilson: "How Do Scientists Develop and Use Scientific Software?" *Proc. Second International Workshop on Software Engineering for Computational Science and Engineering* (SECSE'09), May 2009.

Survey Results

- ~48 hour work week
- 30% of time developing software
- 40% of time using software
- 75.2% never use a supercomputer

Jo Erskine Hannay, Hans Petter Langtangen, Carolyn MacLeod, Dietmar Pfahl, Janice Singer, and Greg Wilson: "How Do Scientists Develop and Use Scientific Software?" *Proc. Second International Workshop on Software Engineering for Computational Science and Engineering* (SECSE'09), May 2009.

Opinion #1

- Physicists start with plots, then reductionism starts and they dig into code
- Physics models require physicists
- Junior people told to write code for their institution's 'collaboration committment'
- Physicists write code because funding agencies do not hire programmers; somebody must fill the gaps

HEP Code Size

- geant4: 1.3M lines C++; 1/3 comments
- root: 3.2M lines C++; 1/5 comments
- CMSSW: 8.5M, 30 languages; 1/8 comments
- Linux Kernel: 5M C++
- CPython: 1M C++; PyPy 1M Python

Language	files	blank	comment	code
C++	17455	674643	486282	3227042
XML	3360	21050	18010	2274171
Python	14709	177730	148786	1137938
C/C++ Header	15175	257820	230921	900909
Fortran 77	137	10477	29707	205099
Javascript	277	47279	120895	192594
Bourne Shell	854	19554	18159	115838
Perl	406	16353	12860	73280
C	123	11950	12908	55933
Java	288	9409	7695	44732
HTML	279	4030	1591	37189
SQL	255	3634	2702	21497
C Shell	324	3897	2611	14098
CSS	149	2321	1779	11154
Visual Basic	9	1013	0	11140
m4	13	835	242	8195
JSP	27	1190	631	5831
make	80	1319	679	3998
PHP	42	748	238	3849
Bourne Again Shell	56	482	427	2255
XSLT	20	269	36	1500
XSD	3	191	154	1361
ASP.Net	28	148	0	1170
VHDL	9	117	214	1121
Lisp	2	90	81	549
sed	2	0	0	160
awk	2	13	4	118
Teamcenter def	4	4	0	97
DTD	3	0	2	59
Expect	2	1	2	25
DOS Batch	2	13	10	22
SUM:	54095	1266580	1097626	8352924

Today's Three Points

1. Physicists write software and
2. initially, there is a training problem for basic software skills but also

Survey Results (again)

- Nearly all self-learned. Followed by peer mentored. Lastly: courses.
- Self-assessment of knowledge gaps
  - Software construction
  - Verification and testing
- Respondents think testing is important

Jo Erskine Hannay, Hans Petter Langtangen, Carolyn MacLeod, Dietmar Pfahl, Janice Singer, and Greg Wilson: "How Do Scientists Develop and Use Scientific Software?" *Proc. Second International Workshop on Software Engineering for Computational Science and Engineering* (SECSE'09), May 2009.

Opinion #2

- C++ FQA: "picking up a new language is easier for a C++ programmer than working in C++"
- Teach initial course (software-carpentry.org). Then wait 6 months. Then code review with students.
- Code review their first commits

State of our field: Three Points

1. Physicists write software and
2. initially, there is a training problem for basic software skills but also
3. our software culture makes on the job training impossible

Opinion #3

- Poor documentation and testing
  - Large ramp-up time
  - Rarely automated tests of physics or functionality (think of plane analogy)
- Long code retention: MINUIT from 70s
- Well-defined specifications impossible

Opinion #3 (cont.)

- one "software guy" effect
- few year contracts of serial development
- hiring decisions are physics-based; nobody reviews your code (even to publish)
- learn to code like preexisting code

</2>

<3>

Open source projects

- Linux, Python, etc..
- More than the source code being available
- Community driven and managed work
- People develop for 'fun' and 'love'
- Collaboration puts out work to the world for others to improve upon

Comparison: Open Source v. Physics

- Can't fire people; required to collaborate
- Global development through email
- Documentation is not fun and requires flogging developers
- Have open code (ideally): open source to security arguments are like open source to physics arguments

Lessons from open source projects

(This will be a list of unrelated 'lessons' that help demonstrate the things that programmers think about)

Lessons from open source projects

"bikeshedding"

Lessons from open source projects

"bus factor"

Lessons from open source projects

"Mission statements and specifications prevent feature bloat"

Lessons from open source projects

"Mission statements and specifications prevent feature bloat"

Think ROOT: plotting program, file structure, fitting program, distributed computing, C++ interpreter (eek!), QT and GUI creator, STLplus, GSL wrappers, etc.

Zawinski's Law

"Every program attempts to expand until it can read mail. Those programs which cannot so expand are replaced by ones which can."

Lessons from open source projects

Code is read more than written

But which of these adages can be proven with data?

</3>

<4>

Software Engineering Concepts

- Code Review
- Tests (unit, functional, integration)
- Software effort analysis
- Distributed Version Control (git, bzr, etc.)
- Refactoring

What makes better programmers?

- Of interest to employers...
- Personality not a good indicator based on 'personality models' [Saleh et. al 2010]: extraversion, agreeableness, conscientiousness, neuroticism, openness to experience
- Collaboration abilities: too many collaboration models, people recorded, bad predictor [Hannay et al. 2007]

What makes better programmers?

- Intelligence?
- IQ ~ learning ability
- IQs != planing or prioritisation abilities
- Intelligence models: creative, practical, analytical
- Consistent and inconsistent types of work; skill + intelligence matter [Schmidt/Hunter 1998]

What makes better programmers?

- This is a new maturing field
- Progress being made measuring abilities
- Software abilities != effort estimation

x10 Productivity

- Original study by Sackman et. al in 1960s:
  - 20 to 1 coding time
  - 25 to 1 debugging time
  - 5 to 1 program size
  - 10 to 1 execution speed
- Experience uncorrelated to productivity

x10 Productivity

- 166 programmers, 18 organizations [Demarco and Lister 1999]
- Good programmers vary within groups
- Groups vary between one another (3.4 to 1 [Boehm et. al 1984]

x10 Productivity

- Lotus 123: 260 staff years for 400k LOC
- Excel: 50 staff years for 649k LOC
- Lotus famously late, Excel Microsofts 'best product'

People's First Job

- Peer mentoring helps
- Classes of people: movers and stoppers
- Biggest difference is management structure since 'small picture of whole' damaging [Microsoft self-measurements]

Conway's Law

"...organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations."

</4>

<5>

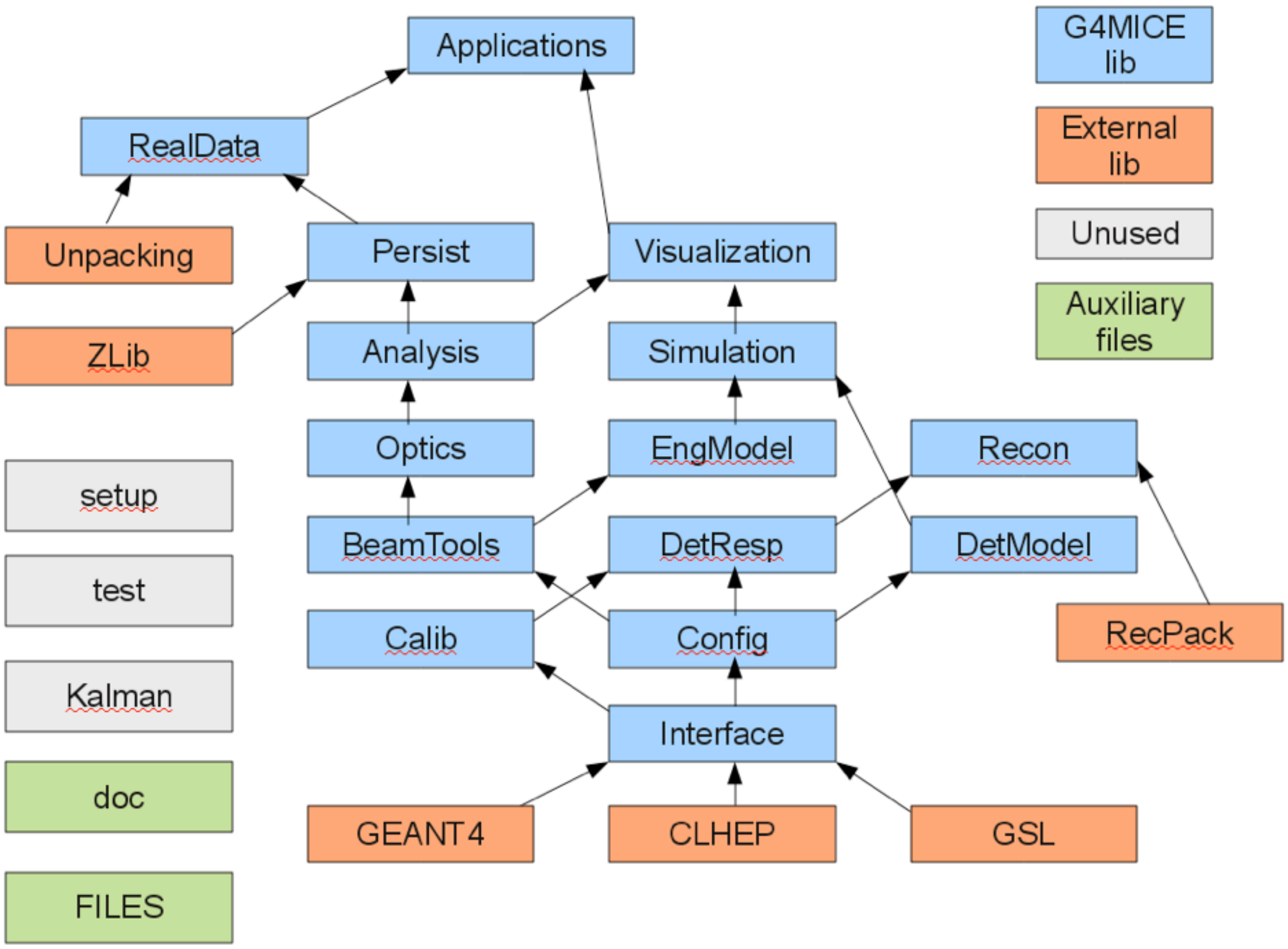
Case Study: MICE

- As many detectors as ATLAS but as many people as a liquid sphere neutrino detector
- Accelerator and particle physics code
- Long code life and large bus factors

Case Study: MICE

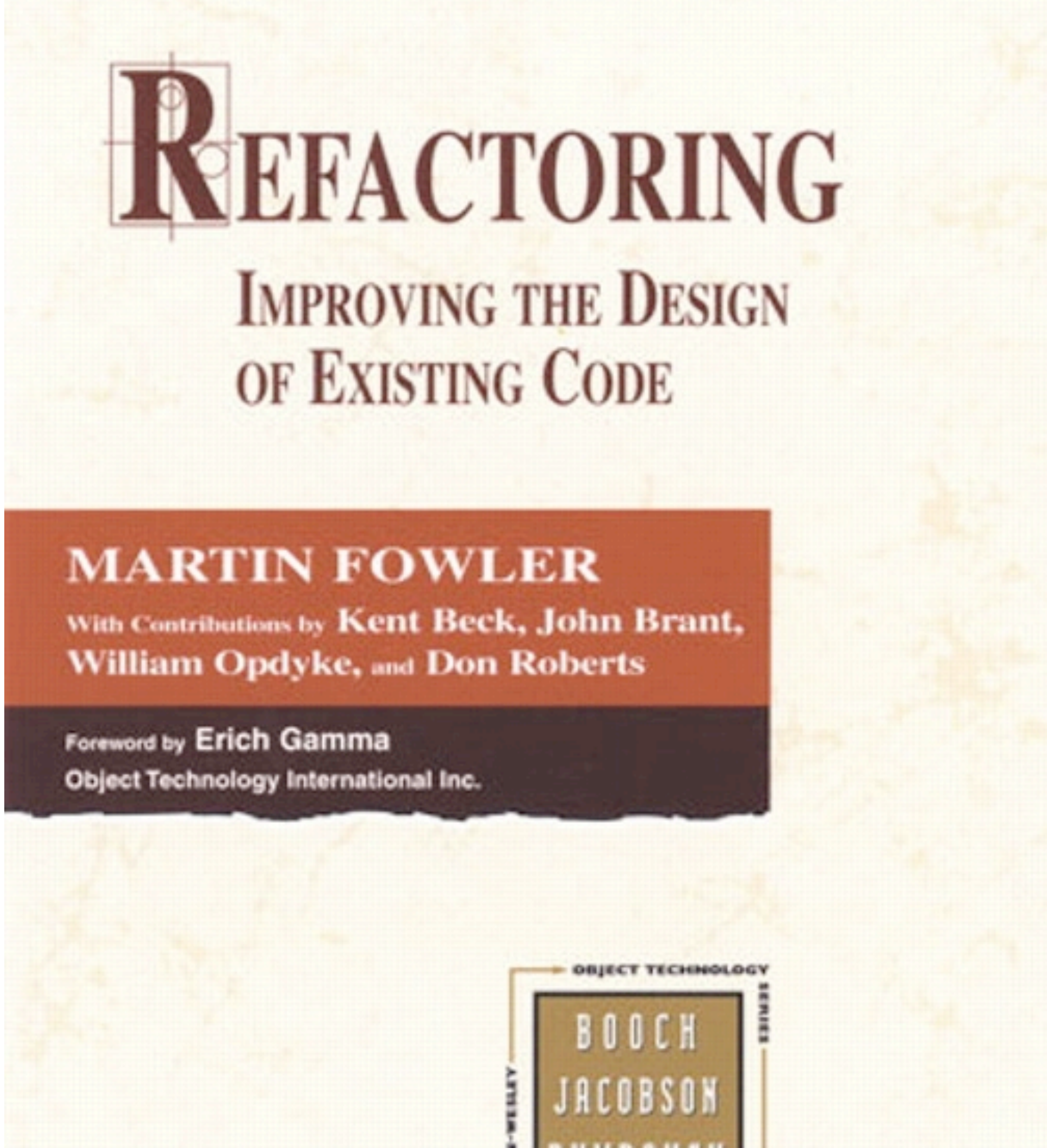
- G4MICE since 2002
- C++
- Major project managers left
- Much of the code 'legacy' due to age/ experience loss

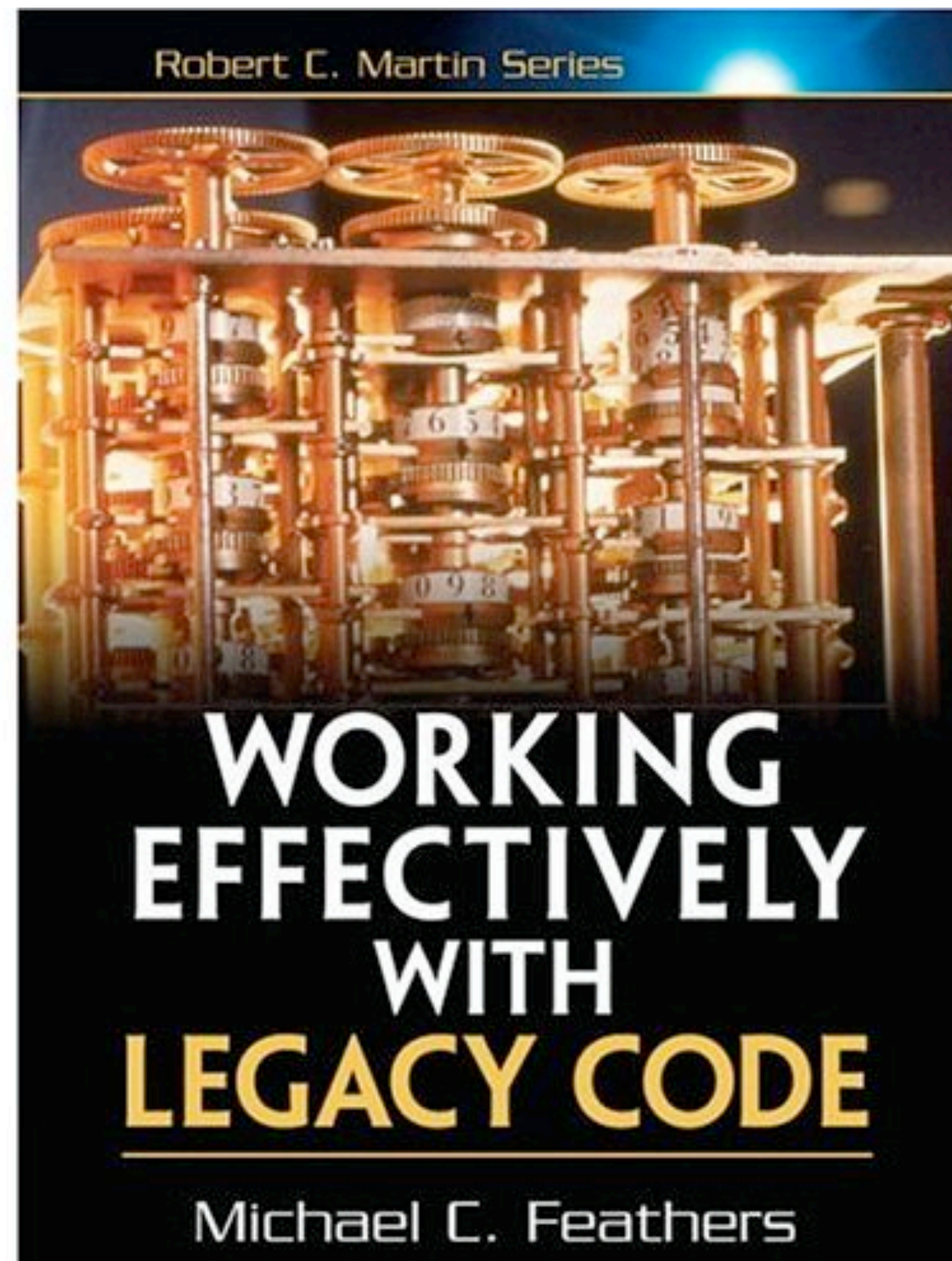
G4MICE



Applications
RealData
Unpacking
Persist
Visualization
ZLib
Analysis
Simulation
Optics
EngModel
Recon
BeamTools
DetResp
DetModel
Calib
Config
RecPack
Interface
GEANT4
CLHEP
GSL
setup
test
Kalman
doc
FILES
G4MICE lib
External lib
Unused
Auxiliary files

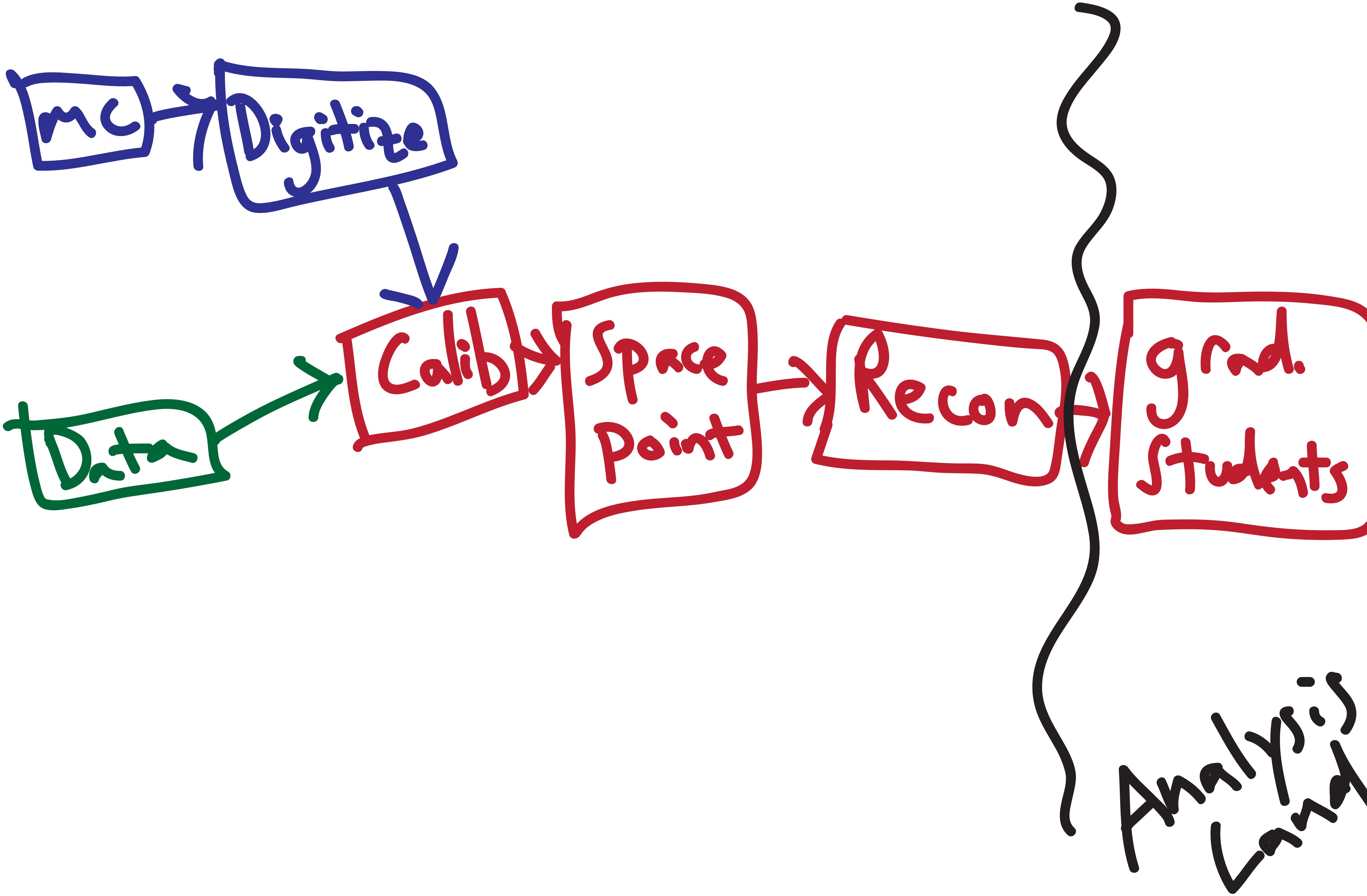
Refactoring is a "disciplined way to restructure code". Legacy code is code you can't change and verify it still works. Similarly code nobody understands.



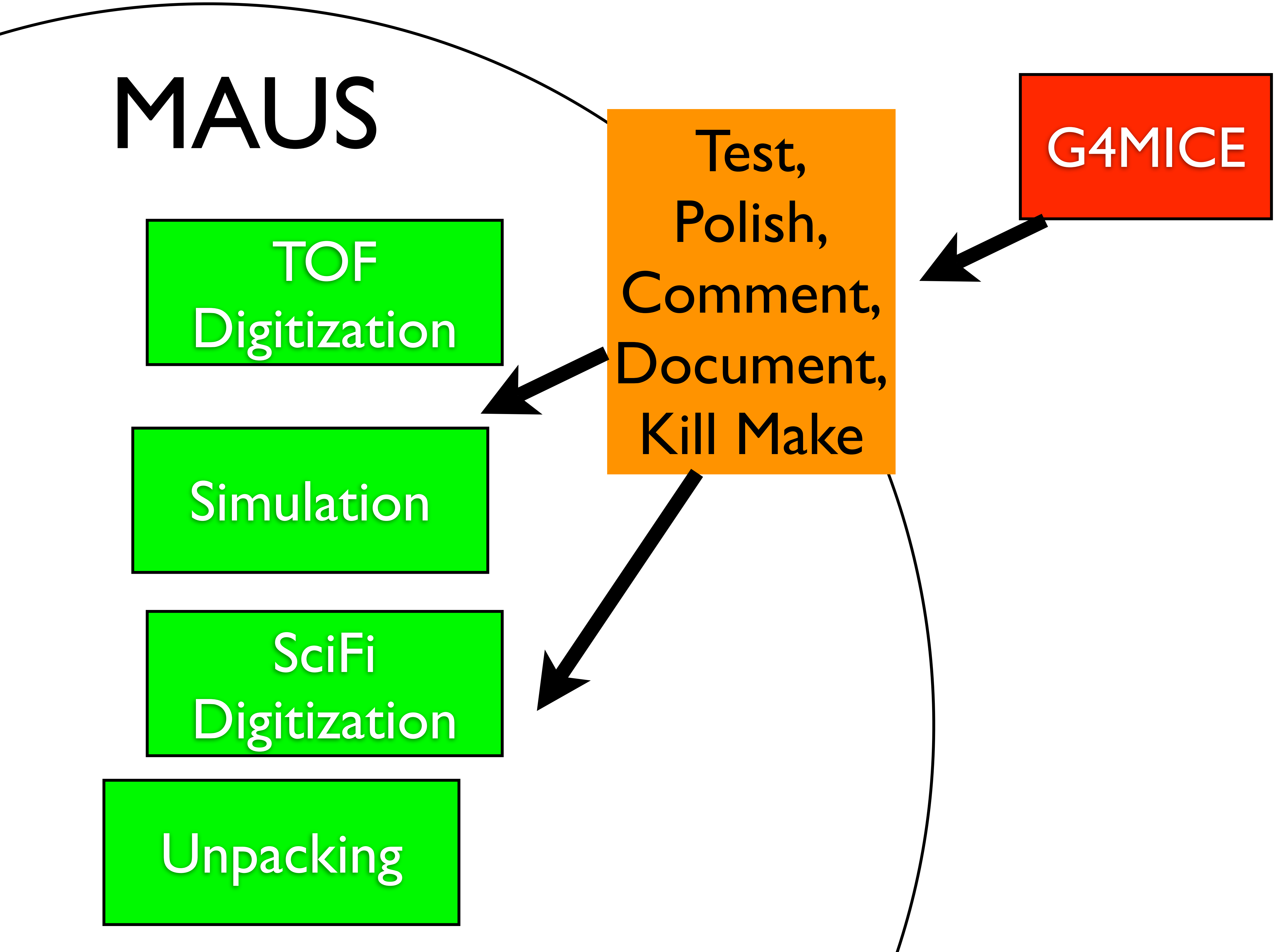


Case Study: MICE

- MAUS in 2010
- C++ and Python (using SWIG) since Python fills gaps
- Triage code (dead? expired? fix? keep?)
- Introduce testing requirements, code branches, style guides, documentation requirements, automated testing
- Well received: people want to do things correctly



MC
Digitize
Data
Calib
Space Point
Recon
grad. Students
Analysis Land


MAUS
TOF Digitization
Simulation
SciFi Digitization
Unpacking
Test, Polish, Comment, Document, Kill Make
G4MICE

Data Structure

- No ROOT (TBaskets)
- JSON format
- Extendable
- spill['mc_particle'][0]['energy'] = 210


```
{
    "mc_particle": [
        {
            "energy": 210,
            "particle_id": 13,
            "position": {
                "x": 0.0,
                "y": -0.0,
                "z": -5000
            },
            "random_seed": 10,
            "unit_momentum": {
                "x": 0,
                "y": 0,
                "z": 1
            }
        }
    ]
}
```

Build Queue
No builds in the queue.

Build Executor Status

#	Master
1	Idle
	fedora14_32
1	Idle
	fedora14_64
1	Idle
	heplnm071 (offline)
	heplnx101
1	Idle
2	Idle
	heplnx102
1	Idle
2	Idle
	opensuse113_32
1	Idle
	opensuse113_64
1	Idle
	pplxint5 (offline)
	pplxint6 (offline)
	sl48_32
1	Idle
	sl48_64
1	Idle
	sl55_32
1	Idle
	sl55_64 (offline)

All

S	W	Job ↓	Last Success	Last Failure	Last Duration
		MAUS_aslaninejad	11 days (#1)	N/A	2 hr 38 min
		MAUS_carlisle	11 days (#29)	13 days (#25)	1 hr 24 min
		MAUS_fayer	11 days (#7)	13 days (#3)	1 hr 46 min
		MAUS_nonVMs_nightly	1 mo 0 days (#10)	13 days (#38)	4 hr 6 min
		MAUS_per_commit_gcc	10 hr (#176)	N/A	8 min 21 sec
		MAUS_robinson	11 days (#1)	N/A	3 hr 7 min
		MAUS_rogers	13 days (#47)	6 days 9 hr (#50)	1 hr 24 min
		MAUS_trunk	1 day 18 hr (#35)	N/A	1 hr 25 min
		MAUS_tunnell	10 hr (#39)	N/A	1 hr 24 min
		MAUS_verguilov	17 hr (#1)	N/A	1 hr 22 min
		MAUS_VMs_nightly	N/A	3 days 9 hr (#23)	1 day 1 hr

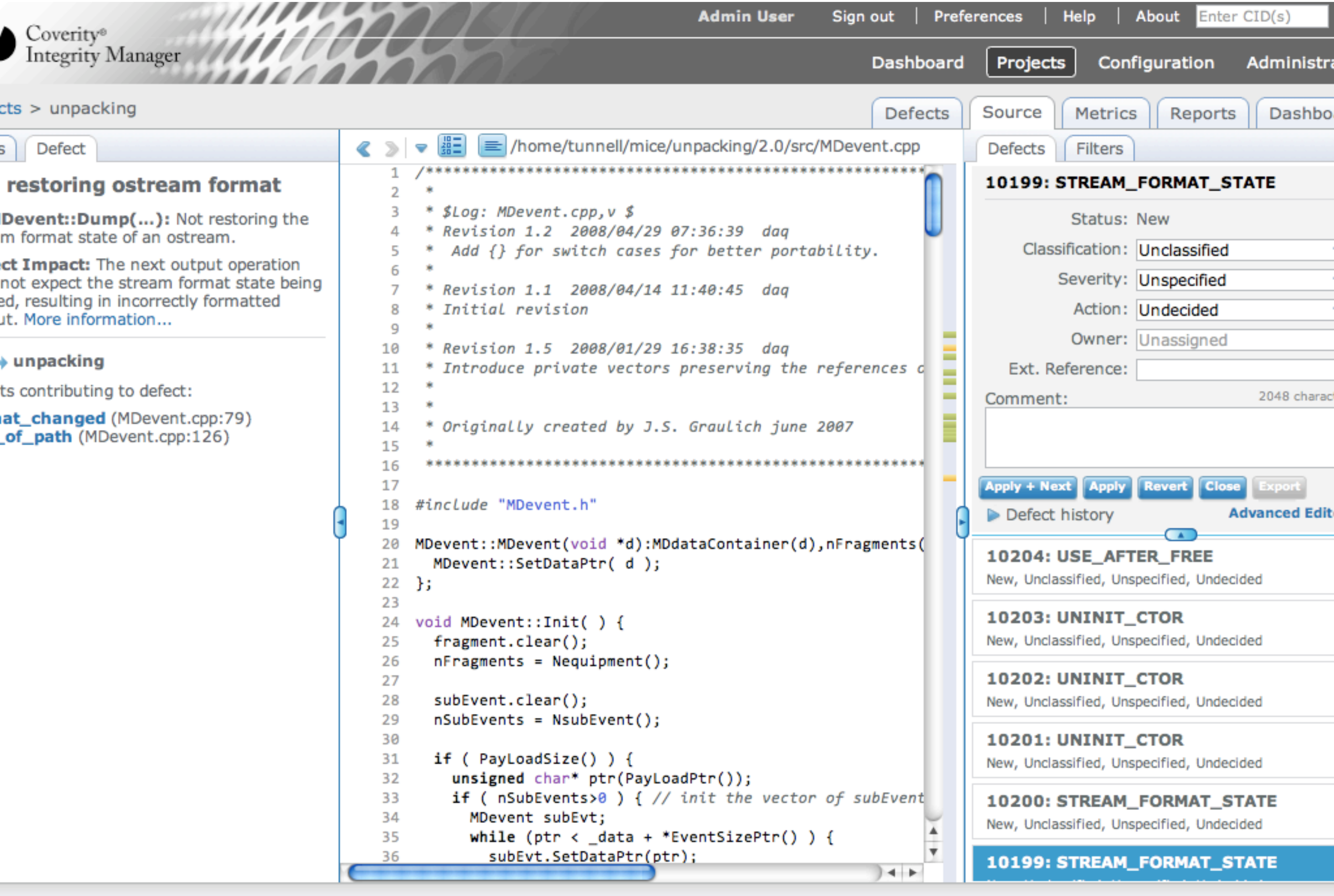
Icon: S M L

Case Study: MICE

- Trying to use these lessons from industry
- Trying to answer questions:
  - 'how do we know some functionality works?'
  - 'how do we know the physics is correct?'
- Long way to go, but we'll get there

Coverity

- Static code analyzer
- Used by industry (defense, telecom, finance, etc.)
- Finds bugs, memory leaks, seg. faults, etc.
- Used for ROOT
- Generously provided by Coverity for MICE


Coverity Integrity Manager
Admin User
Sign out
Preferences
Help
About
Enter CID(s)
Dashboard
Projects
Configuration
Administra
cts > unpacking
Defects
Source
Metrics
Reports
Dashbo
Defect
restoring ostream format
Devent::Dump(...): Not restoring the m format state of an ostream.
ct Impact: The next output operation not expect the stream format state being ed, resulting in incorrectly formatted ut. More information...
unpacking
ts contributing to defect:
at_changed (MDevent.cpp:79)
_of_path (MDevent.cpp:126)
/home/tunnell/mice/unpacking/2.0/src/MDevent.cpp
/***
 *
 * $Log: MDevent.cpp,v $
 * Revision 1.2 2008/04/29 07:36:39 daq
 * Add {} for switch cases for better portability.
 *
 * Revision 1.1 2008/04/14 11:40:45 daq
 * Initial revision
 *
 * Revision 1.5 2008/01/29 16:38:35 daq
 * Introduce private vectors preserving the references o
 *
 *
 * Originally created by J.S. Graulich june 2007
 *
 ****
#include "MDevent.h"
MDevent::MDevent(void *d):MDdataContainer(d),nFragments(
  MDevent::SetDataPtr(d);
};
void MDevent::Init( ) {
  fragment.clear();
  nFragments = Nequipment();
  subEvent.clear();
  nSubEvents = NsubEvent();
  if (PayLoadSize()) {
    unsigned char* ptr(PayLoadPtr());
    if (nSubEvents>0) { // init the vector of subEvent
      MDevent subEvt;
      while (ptr < _data + *EventSizePtr()) {
        subEvt.SetDataPtr(ptr);
Defects
Filters
10199: STREAM_FORMAT_STATE
Status: New
Classification: Unclassified
Severity: Unspecified
Action: Undecided
Owner: Unassigned
Ext. Reference:
Comment:
2048 charac
Apply + Next
Apply
Revert
Close
Export
Defect history
Advanced Edit
10204: USE_AFTER_FREE
New, Unclassified, Unspecified, Undecided
10203: UNINIT_CTOR
New, Unclassified, Unspecified, Undecided
10202: UNINIT_CTOR
New, Unclassified, Unspecified, Undecided
10201: UNINIT_CTOR
New, Unclassified, Unspecified, Undecided
10200: STREAM_FORMAT_STATE
New, Unclassified, Unspecified, Undecided
10199: STREAM_FORMAT_STATE

</5>

Software Engineering in Particle Physics

Christopher Tunnell

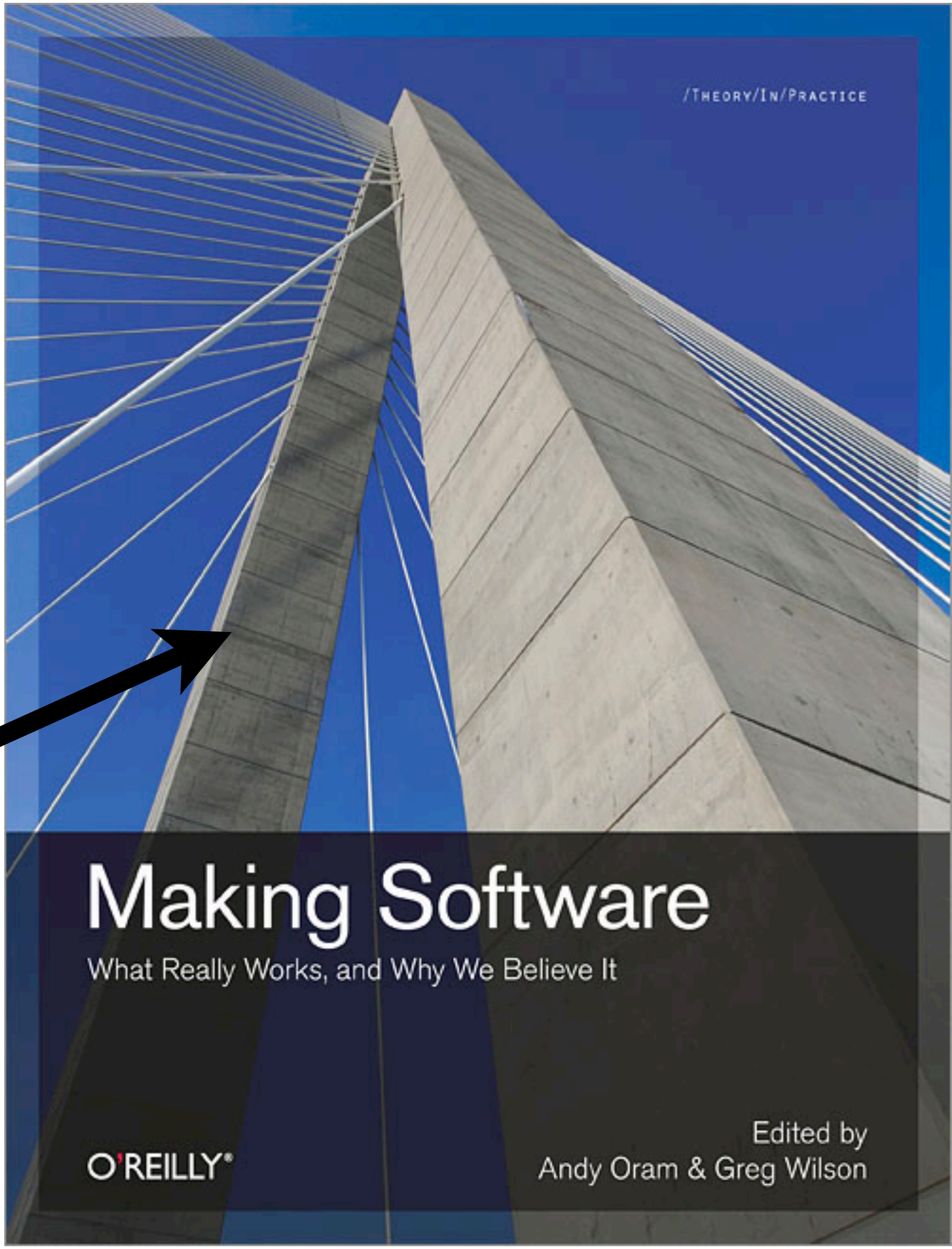
1. Introduce some software engineering *stuff*
2. describe the state of things in our field.
3. compare us to opensource projects
4. explain useful lessons from industry.
5. compare what we've learned to a HEP case study.

</talk>

Recommended Reading

To learn: software-carpentry.org

To study:



Funding issues
(maybe grids ate it)