You're trying to predict the behavior of <complicated system>? Just model it as a <simple object>, and then add some secondary terms to account for <complications I just thought of>.

Easy, right?

So, why does <your field> need a whole journal, anyway?

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.
Software Engineering in Particle Physics

Christopher Tunnell
Introduce some software engineering stuff
Introduce some software engineering stuff then describe the state of things in our field.
Introduce some software engineering stuff then describe the state of things in our field. Then next compare us to opensource projects
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.
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.
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.
10.0 mm LITHIUM_HYDRIDE with 10000 200.0 MeV/c mu- 100.0 mm steps

- icool v3.10 (fail)
- icool v3.13 (fail)
- icool v3.17 (fail)
- icool v3.20 (pass)
- icool v3.23 (pass)
- icool v3.26 (pass)

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
<table>
<thead>
<tr>
<th>Lingo</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Particle Physics</strong></td>
</tr>
<tr>
<td>Renormalization</td>
</tr>
<tr>
<td>Offshell</td>
</tr>
<tr>
<td>Weak currents</td>
</tr>
<tr>
<td>Data quality</td>
</tr>
<tr>
<td>Luminosity</td>
</tr>
<tr>
<td>QED</td>
</tr>
<tr>
<td><strong>Software Engineering</strong></td>
</tr>
<tr>
<td>Refactoring</td>
</tr>
<tr>
<td>Sprints</td>
</tr>
<tr>
<td>Agile</td>
</tr>
<tr>
<td>Regression tests</td>
</tr>
<tr>
<td>Continuous Integration</td>
</tr>
<tr>
<td>RTFM</td>
</tr>
</tbody>
</table>
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)
Survey Results

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

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

**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)

- Self-assessment of knowledge gaps
  - Software construction
  - Verification and testing
- Respondents think testing is important
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
\textlangle/2\rangle
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?
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 ≠ planning 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
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."
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

- RealData
  - Unpacking
  - ZLib
- Applications
  - Persist
    - Analysis
    - Optics
  - Visualization
    - Simulation
    - EngModel
    - DetResp
    - DetModel
- Visualization
- Interface
  - Config
    - Interface
- RecPack
- 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
Data Structure

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

```json
{
  "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 History

No builds in the queue.

## Build Executor Status

<table>
<thead>
<tr>
<th>#</th>
<th>Master</th>
<th>S</th>
<th>W</th>
<th>Job</th>
<th>Last Success</th>
<th>Last Failure</th>
<th>Last Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Idle</td>
<td>![sun]</td>
<td>![cloud]</td>
<td>MAUS_aslalinejad</td>
<td>11 days (#1)</td>
<td>N/A</td>
<td>2 hr 38 min</td>
</tr>
<tr>
<td>1</td>
<td>fedora14_32</td>
<td>![sun]</td>
<td>![cloud]</td>
<td>MAUS_carlisle</td>
<td>11 days (#29)</td>
<td>13 days (#25)</td>
<td>1 hr 24 min</td>
</tr>
<tr>
<td>1</td>
<td>fedora14_64</td>
<td>![sun]</td>
<td>![cloud]</td>
<td>MAUS_fayer</td>
<td>11 days (#7)</td>
<td>13 days (#3)</td>
<td>1 hr 46 min</td>
</tr>
<tr>
<td>2</td>
<td>Idle</td>
<td>![sun]</td>
<td>![cloud]</td>
<td>MAUS_nonVms_nightly</td>
<td>1 mo 0 days (#10)</td>
<td>13 days (#38)</td>
<td>4 hr 6 min</td>
</tr>
<tr>
<td>1</td>
<td>Idle</td>
<td>![sun]</td>
<td>![cloud]</td>
<td>MAUS_per_commit_gcc</td>
<td>10 hr (#176)</td>
<td>N/A</td>
<td>8 min 21 sec</td>
</tr>
<tr>
<td>1</td>
<td>Idle</td>
<td>![sun]</td>
<td>![cloud]</td>
<td>MAUS_robinson</td>
<td>11 days (#1)</td>
<td>N/A</td>
<td>3 hr 7 min</td>
</tr>
<tr>
<td>2</td>
<td>Idle</td>
<td>![sun]</td>
<td>![cloud]</td>
<td>MAUS_rogers</td>
<td>13 days (#47)</td>
<td>6 days 9 hr (#50)</td>
<td>1 hr 24 min</td>
</tr>
<tr>
<td>1</td>
<td>Idle</td>
<td>![sun]</td>
<td>![cloud]</td>
<td>MAUS_trunk</td>
<td>1 day 18 hr (#35)</td>
<td>N/A</td>
<td>1 hr 25 min</td>
</tr>
<tr>
<td>1</td>
<td>Idle</td>
<td>![sun]</td>
<td>![cloud]</td>
<td>MAUS_tunnell</td>
<td>10 hr (#39)</td>
<td>N/A</td>
<td>1 hr 24 min</td>
</tr>
<tr>
<td>1</td>
<td>Idle</td>
<td>![sun]</td>
<td>![cloud]</td>
<td>MAUS_verguilov</td>
<td>17 hr (#1)</td>
<td>N/A</td>
<td>1 hr 22 min</td>
</tr>
<tr>
<td>1</td>
<td>Idle</td>
<td>![sun]</td>
<td>![cloud]</td>
<td>MAUS_VMs_nightly</td>
<td>N/A</td>
<td>3 days 9 hr (#23)</td>
<td>1 day 1 hr</td>
</tr>
</tbody>
</table>

**Legend**
- ![S] for all
- ![M] for failures
- ![L] for just latest builds
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
restoring ostream format

MEvent::Dump(...) : Not restoring the stream format state of an ostream.

**Impact:** The next output operation not expect the stream format state being set, resulting in incorrectly formatted output. More information...

unpacking

Defects contributing to defect:
- metric_changed (MEvent.cpp:79)
- of_path (MEvent.cpp:126)

```
/* Log: MEvent.cpp,v 1.2 2008/04/29 07:36:39 dag */
Revision 1.2 2008/04/29 07:36:39 dag
Add () for switch cases for better portability.
Revision 1.1 2008/04/14 11:40:45 dag
Initial revision
Revision 1.5 2008/01/29 16:38:35 dag
Introduce private vectors preserving the references at.

Originally created by J.S. Graulich June 2007
```

```cpp
#include "MEvent.h"

MEvent::MEvent(void *d):MDdataContainer(d),nFragments(0),
MEvent::SetDataPtr( d );

void MEvent::Init()
{
    fragment.clear();
nFragments = Nequipment();
    subEvent.clear();
nSubEvents = NsubEvent;
    if ( PayloadSize() )
    {
        unsigned char *ptr(PayloadPtr());
        if ( nSubEvents>0 ) // init the vector of subEvent
            MEvent subEvt;
        while (ptr < data + *EventSizePtr())
            subEvt.SetDataPtr(ptr);
```
</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: Making Software

Funding issues (maybe grids ate it)