Lessons learned from SDSS and how they apply to LSST

Robert Lupton

Princeton University

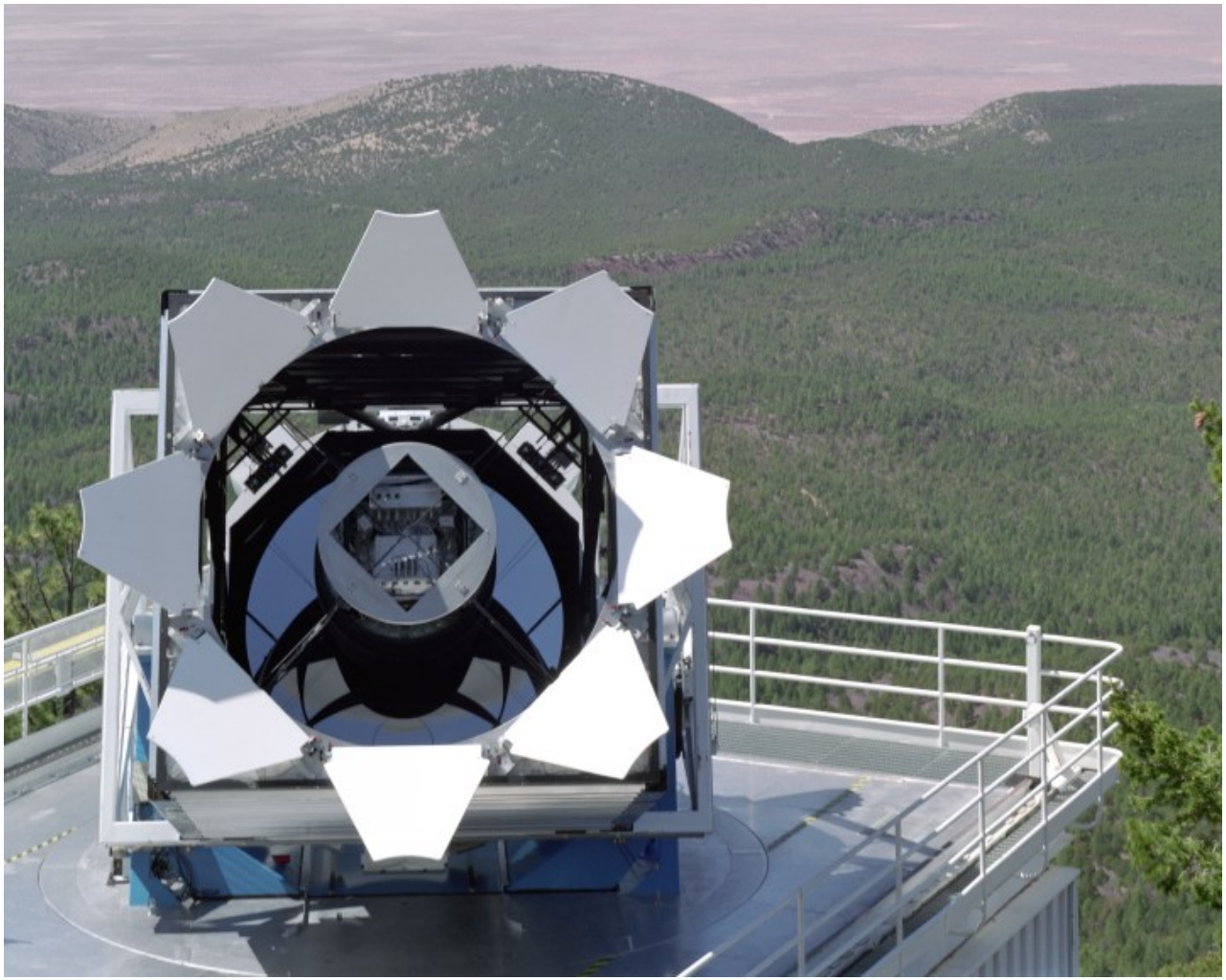
2014-05-28

Lessons I Learnt from the SDSS

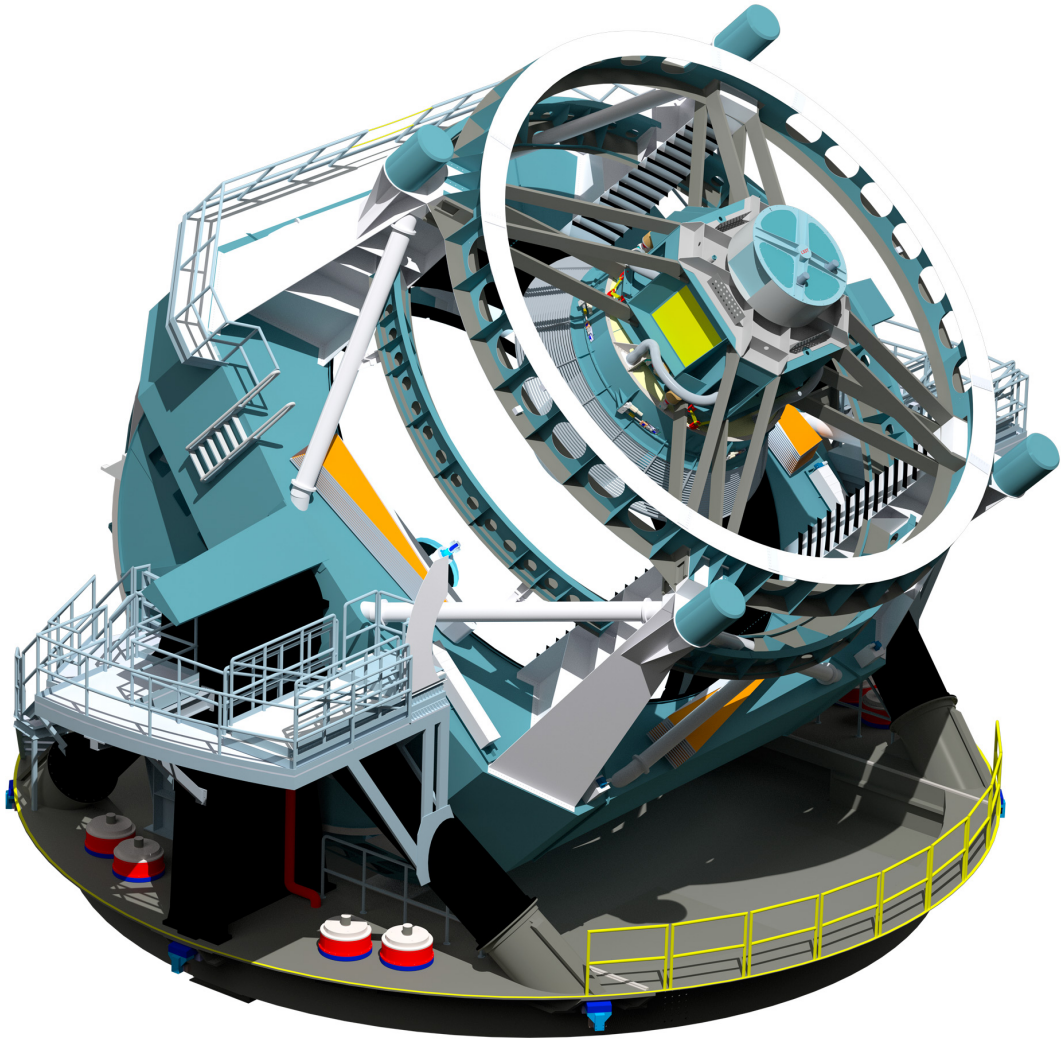
I first wrote these lessons down in 2002; what have we learned in the last 12 years?

Some of these are obvious, but were nevertheless ignored by the SDSS project.

The SDSS 2.5m telescope

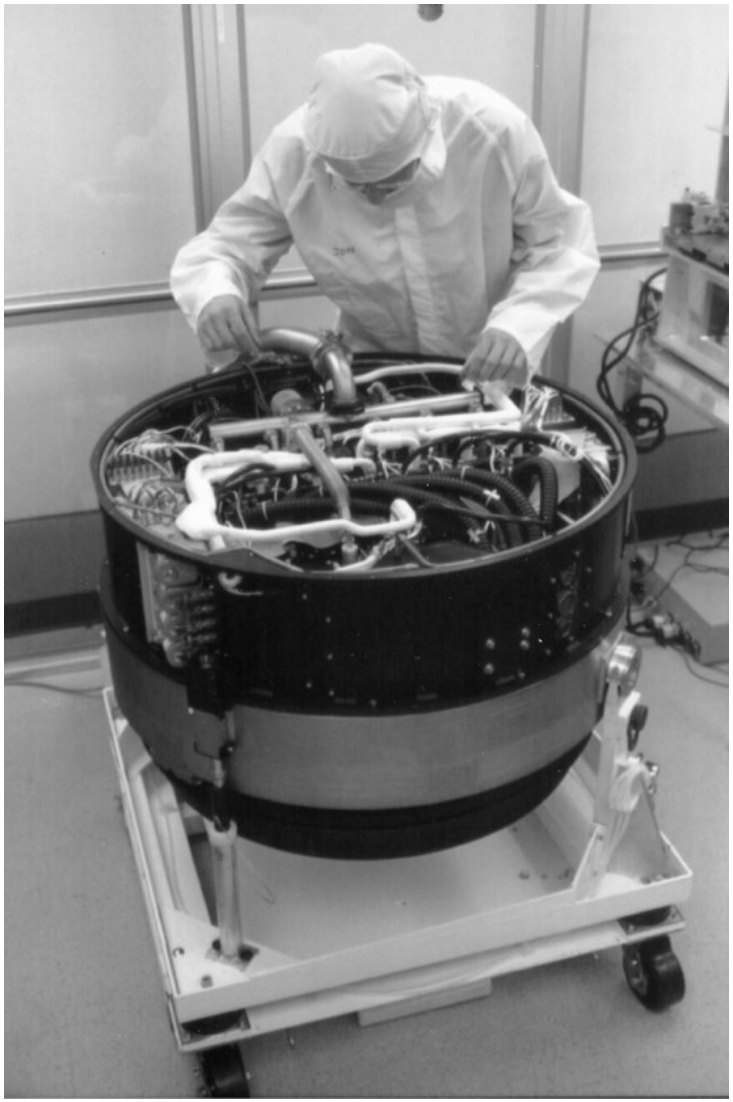


The LSSTelescope

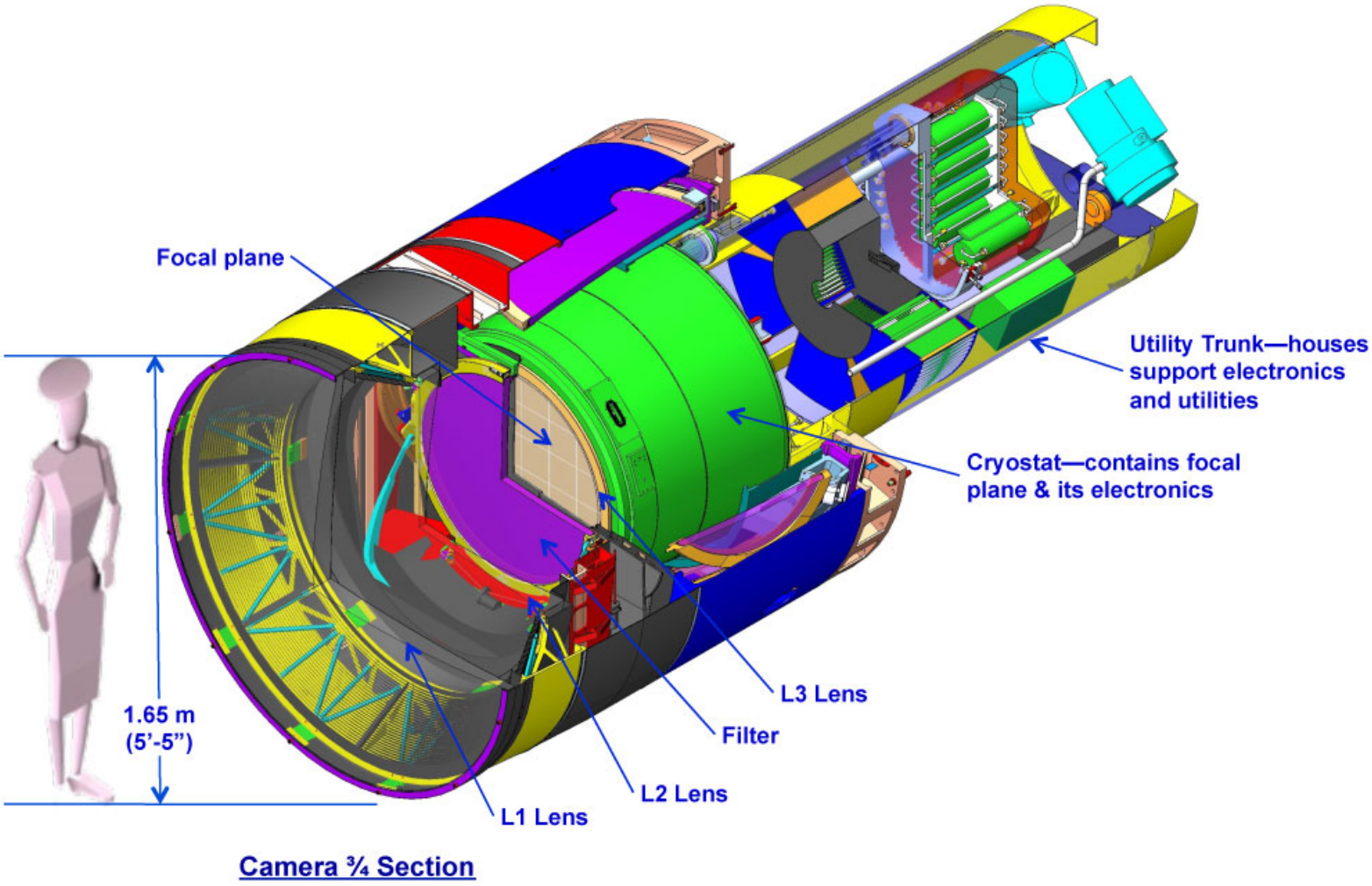


Three mirrors: an 8.4m primary, a 3.4m secondary, and a 5m tertiary.

The SDSS Camera



The LSST Camera



3.2 GPixels every 17s; c. 400 MB/s

20 TB per night; 60 PB over 10 years for the raw data and 15 PB for the catalog

SDSS's Photo

```
/*
 * copy the symmetrised image back to the original data region, where
 * it will become the deblending template (the original pixel values
 * are preserved in the parent's atlas image).
 *
 * We must of course only do this within the master_mask
 */
     copy_region_within_mask((REGION *)data, sym, mmask,
                                                  aimage_drow, aimage_dcol);
/*
 * we next want to run the object finder on that symmetrised image; the image
 * is smoothed, and extra peaks rejected --- see improve_template() for details
 */
     obj1 = objc->color[c];
     if(obj1->flags & OBJECT1_DEBLENDED_AS_PSF) {
/*
 * no need to check template, as we created it as a multiple of PSF
 */
     } else {
        float threshold = fiparams->frame[c].ffo_threshold;

        shAssert(obj1->mask == objc->aimage->mask[c]);
        phObjmaskDel(obj1->mask); objc->aimage->mask[c] = NULL;
        obj1->mask =
          improve_template(mmask, c, rowc,colc, data, aimage_drow,aimage_dcol,
                           scra, scrb, rsize + filtsize, csize + filtsize,
                           fiparams->frame[c].smooth_sigma, filtsize,
                           npeak_max, smoothed_ai, threshold, ngrow);

        if(obj1->mask == NULL) {
           objc->flags &= ~OBJECT1_DETECTED;
           obj1->flags &= ~OBJECT1_DETECTED;
        }
     }
  }
/*
 * we've found the templates in all colours. They are represented by the
 * pixels in the original data region, within the OBJECT1->mask
 *
 * Now go through them looking for objects which we didn't detect
 * in any band; in this case, the object wouldn't have been found at all
 * if it wasn't part of a blend, so dump it.
 *
 * Actually we cannot just dump it here as we've got an array with all the
 * children in it, and we'd have to move the others down. Instead, mark
 * the entire OBJC as not DETECTED, and we'll dump it when we get a chance.
 */
  for(c = 0;c < ncolor;c++) {
     objc->flags |= (objc->color[c]->flags & OBJECT1_DETECTED);
  }
  if(!(objc->flags & OBJECT1_DETECTED)) { /* not detected in any band */
     phAtlasImageDel(*smoothed_ai, 0); *smoothed_ai = NULL;
--   deblend.c   38% L1365 CVS-1.128   (C Abbrev)----2:55PM 0.39----------------
```

LSSTApplications

0988eee230226a4295d29193cba03c822aa32f72+447,1488d58525a00af02894486a48494c5db9e3c94e+505,158f99052810cdf801b7ab6560141fa317948

LSSTDataManagementBasePackage

Main Page | Related Pages | Modules | Namespaces | Classes | Files | Examples | Search

LSST Documentation

Introduction

The LSST source documentation is generated using doxygen; this wrap-up of various packages dependent on datarel fa860280f9bb1088ca88ec79c7e4566e11c38167+567 was generated by using a script from `lsstDoxygen`.

Packages for which documentation was gathered are: lsst::afw, lsst::ap, lsst::base, lsst::coadd::chisquared, lsst::coadd::utils, lsst::ctrl::events, lsst::daf::base, lsst::daf::butlerUtils, lsst::daf::persistence, lsst::datarel, lsst::geom, lsst::ip::diffim, lsst::ip::isr, lsst::meas::algorithms, lsst::meas::astrom, lsst::meas::deblender, lsst::meas::extensions::multiShapelet, lsst::ndarray, lsst::pex::config, lsst::pex::exceptions, lsst::pex::harness, lsst::pex::logging, lsst::pex::policy, lsst::pipe::base, lsst::pipe::tasks, lsst::sconsUtils, lsst::shapelet, lsst::skymap, lsst::skypix, lsst::utils

Mainpages in sub-packages:

- lsst::afw; the LSST Application Framework
- lsst::coadd::chisquared; Support for chi-squared coadds
- lsst::coadd::utils; Support for coadds
- lsst::meas::algorithms; support for measurement algorithms
- ndarray; Multidimensional Arrays in C++
- lsst::pex::config: configuration data management
- lsst::pex::exceptions; LSST Exceptions
- lsst::pex::harness; the LSST Pipeline Harness
- lsst::pex::logging: writing messages to logs
- lsst::pex::policy: configuration data management
- lsst::pipe::base; Base package for pipeline tasks
- lsst::pipe::tasks: Pipeline tasks
- lsst::skymap; sky pixelization

Generated on Sun Sep 29 2013 18:51:36 for LSSTApplications by doxygen 1.8.2

A Small Team Building the SDSS camera

A Larger Team of Astronomers using the SDSS data

... Two or Three Years Before Construction

Lesson 1: Project Managers

You need a strong and impartial project manager. SDSS is a collaboration of a large number of institutions and we have never managed to take technical decisions unimpeded by politics.

Lesson 2: Funding

Don't go into a project that isn't fully funded. Our problems (see 1) were compounded by a continuing need to placate universities that might put in money.

I'm not sure that I followed The Rule when I started working on LSST; but failing to follow the rule makes a good project manager even more important.

The view from 2014

H. H. Munro (Reginald at the Theatre. 1904)

``When I was younger, boys of your age used to be nice and innocent.”
``Now we are only nice. One must specialise these days''

I no longer innocently believe that all we need is project managers, as they come from quite another world. We *do* need them, of course, but we need to manage them…

At least I don't have to placate nation states that might put in money.

Roberts' Paradox

Unfortunately I'm naming it not for me, but for Eric Roberts at Stanford who in 2000 wrote a report for the US National Academy with the blessing of the ACM. * The paradox is that:

- There are unemployed software engineers
- There is a shortage of software engineers

The resolution is that the shortage is of the best engineers, not the median:

> *If the best software developer can do the work of 10, 20, or even 100 run-of-the-mill employees, a single-person company that attracts such a superstar can compete effectively against a much larger enterprise*
> *[…]*
> *In some cases, software developers who fall at the low end of the productivity curve may be essentially nonproductive or even counterproductive*

*the *A*ssociation for *C*omputing *M*achinery

How should you manage projects?

Hardware and Software are *different*:

- Hardware
  You design everything including every detail ; then you build it. Building the system is slow and hard.
- Software
  You design everything including every detail ; then you're done. Designing the system is slow and hard.

How should we manage them?

- Hardware
  Little of our hardware pushes the state of the art, and once the design is done an external expert can see if it makes sense. Requirements + reviews work well.
- Software
  Software is different. At the time you need to pass your FDR/CD-3 the codes are not written. And if you ignore Roberts' Paradox they may never be.

Software holds its contingency in people.

Lesson 3: Don't generate an inverted management structure

Work to ensure that everyone in a position of authority has a clear view of their own strengths and weaknesses, and try to ensure that decisions are taken for technical reasons wherever possible.

If you are forced into a situation where the software effort is large, divide and conquer --- manage the project as a tree with a branching ratio of less than 10.

The view from 2014

This is still a challenge. LSST has just set up a ``Project Science Team'' of the scientific leaders of the three parts of the project (as opposed to the people who have the proper titles) and maybe that'll help?

These days all software efforts are probably too large to be run as monolithic projects so we're faced with dividing and conquering (and probably administering a far-flung empire). An Agile methodology seems to be a good fit to astronomical software efforts, but you have to convince your team and your managers.

Astronomy hasn't figured out to manage distributed responsibility --- what do we do if institution A needs help from institution B, but B is busy with things that they are contractually required to deliver?

While tools (*e.g.* video conferencing) help, travel is still essential to building and maintaining trust. It was a struggle to maintain holding one LSST all-hands meeting a year; I believe BaBar had four.

Lesson 4: When, What, and How to Review

- If some component is failing, admit it even if this involves embarrassing people and institutions. You should not, of course, make this any more public than necessary.
- Put resources where they're needed, not where it's politically convenient to put them.
- Saying that broken code is good enough isn't a conservative approach, even if it saves resources in the short term.
- Only hold reviews if you really want to learn from the review board.

The view from 2014

I couldn't have said it better! Nothing's changed.

The first two sub-lessons are about problems caused by institutional loyalties, especially when money is distributed along institutional (or national) lines. There's a danger that structural units (such as LSST's Telescope/Camera/DM split) will result in another set of divided loyalties.

Lesson 5: Software Engineering

Standard software practices are necessary; for example:

- Source code control at the level of files (*e.g.* cvs)
- Code control at the level of releases that can be reconstructed, along with their dependent products.
- Enforced rules about tests associated with each new feature
- Tools, preferably integrated with the code/release control system, to track problems and feature requests (*e.g.* Gnats/Gnatsweb).
- An insistence on adhering to standards; *e.g.* coding to ISO C89 and Posix 1003.1.
- An insistence that all code should compile with no warnings on all platforms, with all warning turned on. [This required us to write an error-warning filter that can be used to remove certain warnings considered unimportant, *e.g.* `constant in conditional constant` due to `do { ... } while(0);` in header files]

The view from 2014

There are some obvious updates:

- These days, the default's `git` rather than cvs
- LSST used trac and is switching to Jira

but I think that we've basically won these battles (with the possible exception of the testing). What we haven't done is to change graduate education to teach these topics at the same time as PDEs. Courses are beginning to appear (*e.g.* my APC524 is the second largest graduate course at Princeton), but they're not yet considered part of the core curriculum. Last time I taught it, only one physics graduate student took the class.

AST	CBE	CEE	CHEM	EE	FIN	MAE	MAT	PACM	PHY	PPL	UNK
7	8	8	1	4	1	10	1	2	1	1	6

Testing

Some aspects of testing are easy:

- When you get a problem report capture the bug as a unit test before you fix it.
- If you want to add a feature, write the test before (or while) you write the code

Some take more work, but can be achieved; for example

- continuous integration/regression testing, maybe with some sort of fuzzy logic for when the results are still correct.

Some are really hard.

- How do I write a test that I can correctly disentangle an arbitrary number of overlapping stars and galaxies?

LSST is adopting an approach from HEP: write a comprehensive simulator. We're even getting a particle physicist to write it.

One difference is that, while our simulator should evolve to be as complicated as **some** reality, it may never evolve to represent **the** reality that is the LSST project.

Lesson 5b: Don't Write Your Main Program in C

I should have included this Lesson in the original talk.

Compiled languages such as C are the wrong tools for the job. SDSS used TCL and object-oriented C glued together using a home-brewed interface, and with enough introspection to allow things like:

```
set minval [expr [exprGet $calib.calib<$index>->sky] - 2*[exprGet $calib.calib<$index>->skysig]]
```

Most data-debugging involved repeatedly sourcing TCL scripts while keeping the offending data in memory.

The view from 2014

Experience has heartily endorsed this, with obvious updates:

- TCL $\rightarrow$ `python`
- C $\rightarrow$ C++
- Homebrewed interfaces $\rightarrow$ `swig`

The choice of `swig` or `boost::python` or `cython` or `ctypes` or `CPython` is a religious matter. To quote Scott Meyers (in *Effective C++*):

> *Developers enjoy arguing about style issues almost as much as they enjoy arguing about which is the One True Editor. (As if there's any doubt. It's Emacs.)*

Class Libraries and Frameworks

Astronomical software is a good map onto the OO paradigm; we manipulate

- Images
- Pixel Masks
- Point Spread Functions (PSFs)
- Peaks
- Object's `Footprints'
- Sources and Objects
- …

and each of these maps nicely onto a class. Sometimes you can go further; for example some measurements consists of applying some functor to every pixel in a `Footprint`; but some algorithms are naturally written as old-fashioned functions looping over pixels.

We are also making extensive use of run-time-configurable plugins.

Being able to `import` a new C++ algorithm or reconfigure an old one from the command line or a configuration file (or both) is **very** nice, but it does make the overall system harder to understand.

Lesson 6: Distribute Data and Information Freely

- Be as open as possible, and make all information and discussion open to the entire collaboration as soon as practical.

  We wrote a mailing lists manager that archives on the web, and to which anyone in the collaboration may subscribe. Our software problem report system is also available on the web, as is a listing of all papers that are being prepared for publication.

- Make data available to the collaboration (or the world) as soon as possible

  It should in a form as close to the final format as possible. In SDSS it was very hard to find recent data, and it was months (years?) before it started to appear in the science database.

The view from 2014

The first item's a bit dated, isn't it? But it turns out that just making mailing lists doesn't mean that people use them, and wikis are often write-only, so this is still good advice. Web bug-trackers took off, though.

LSST's learned the second lesson (our design has an early delivery of the database, and a careful layering of the user interface and the backend); but not all astronomical projects are so enlightened.

I should have entitled this lesson:

- Lesson 6: Distribute Data, Code, and Information Freely

The LSST code is distributed under the GPL; it's at `https://dev.lsstcorp.org/cgit`, (but is moving to `https://stash.lsstcorp.org/projects/DM`) and we plan to mirror it to github.

Data Distribution

LSST's data distribution's a bit more complicated:

- Changes in the sky generate alerts which are available world-wide within 60s.
- You have immediate access to all data if you're working in the US or Chile, or at IN2P3 in France.
- All the data will be world-available with a delay of a couple of years.
- We are negotiating extending instant-access to other groups around the world.

Why this complexity? Because we're not fully funded (*cf.* Lesson 2).

But we *do* mean all the data! Every bit read off the camera, as well as the final science-grade catalogues, will be available, the latter via a data base (not dissimilar to that provided by the SDSS).

Lesson 7: Strive to ensure that the software takes full advantage of the hardware, even at the beginning of a project

This is partly a matter of principle, and partly because if you don't push to the instrumental limit you don't really know if things are working as well as they should. There is some tension in achieving this, and you need someone to keep the schedule.

The view from 2014

This is still true, but it's especially difficult for LSST. The project's being funded by the US Department of Energy as a *Major Item of Equipment* and by the National Science Foundation as a *Major Research Equipment Facilities Construction* project; in other words they are building the facility, not doing the science.

The LSST project is responsible for:

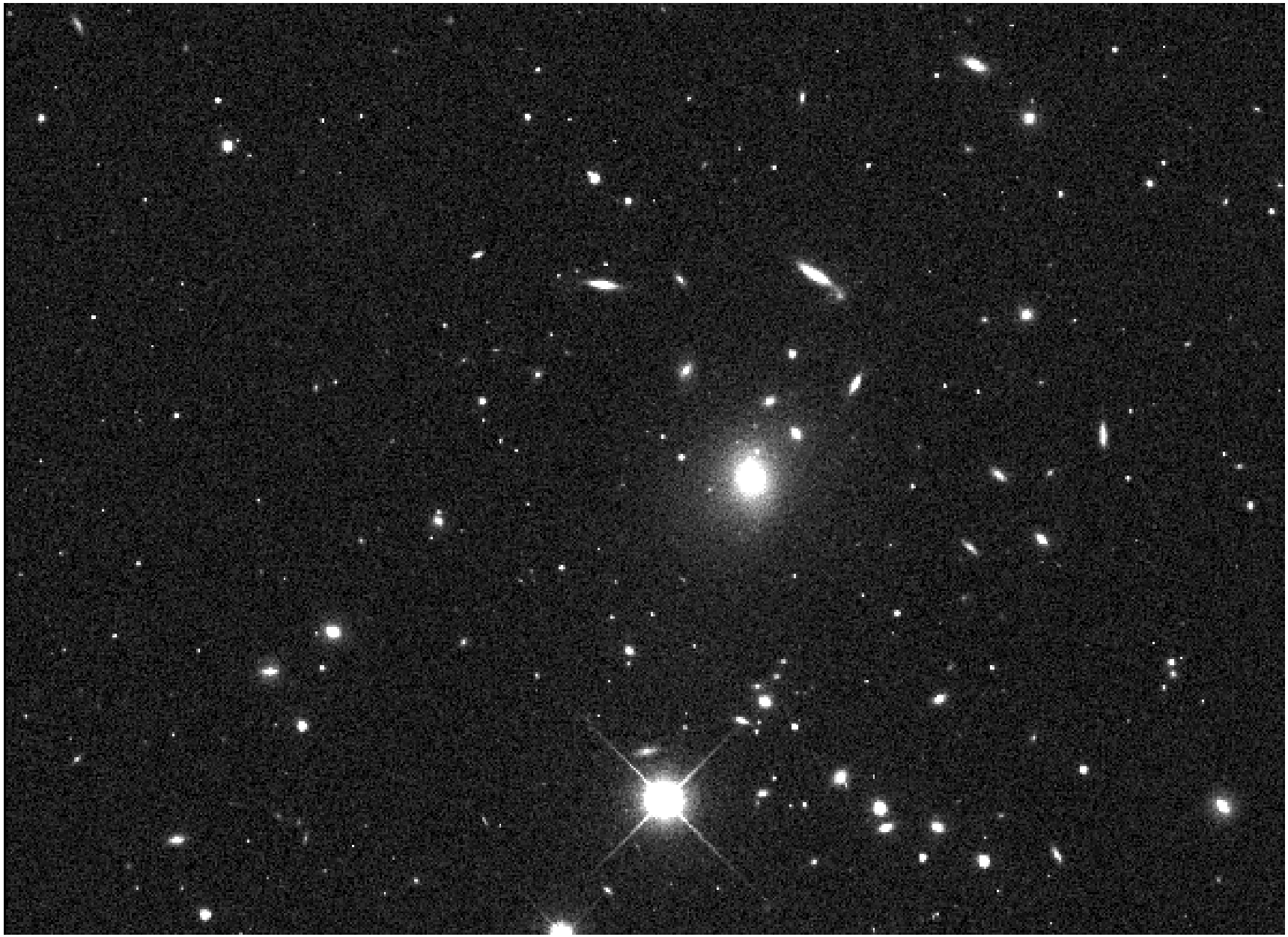
- Level 1: Real-time processing
- Level 2: Yearly reprocessing of all data taken to date producing catalogues satisfying the SRD†

but it is not responsible for:

- Level 3: Anything else

†Science Requirements Document

An SDSS frame



A Level 2 Catalogue (Dustin Lang/David Hogg)

Catalogues

At least for extra-Galactic fields to Sloan depths, the catalogues appear to be close to sufficient statistics for the sky; position, brightness, shape(?), size. The catalogues are the entry point for much of the astronomical community.

Some of the measurements (e.g. the `shape' of galaxies --- think of the 2^{nd} moments) are primarily interesting only statistically. We need to analyse the correlations between the shapes of *many* faint galaxies to measure the interesting quantities --- and each measurement is difficult and beset by systematic errors.

Do you trust me?

Level 3: extra large-scale (especially pixel-level) computing

You may not like the way that my team measures the shapes, especially when you realise that on any single exposure the objects are only detected at c. 1σ. For example, the original SDSS pipelines produced essentially useless shapes; a group at Michigan went through our catalogues of objects and made a far better measurement. Within a year or so we had co-opted their algorithm into the official pipeline.

There will be also scientifically interesting parameters that are not measured in Level 2, either because they are not of general interest, or because they are only applicable to a subset of objects, or simply because we didn't think of them; the astronomical community may nonetheless be interested in measuring them.

How can we make this easier?

Enabling Level 3

There are various aspects to enabling and encouraging Level 3:

- Support the use of the LSST software framework
- Build on multiple OSs and platforms (laptops through supercomputers)
- Provide the robust production tools that we're using internally
- Give access to a modest amount of computing and storage (10% of LSST total)

I.e. give the community the tools that we're using ourselves, *e.g.* use the plugin architecture in Level 2 so it'll work for Level 3.

What do I mean by, ``Support the use of the LSST computing framework''? At the very least:

- Make it good enough that people want to use it
- Provide binary installers on common platforms
- Operate a help desk
- Write good documentation

Documentation

C. L. Dodgson (The Hunting of the Snark. 1876)

"The method employed I would clearly explain,
While I have it so clear in my head,
If I had but the time and you had but the brain ---
But much yet remains to be said."

The Butcher

The minimum standard for C++ documentation seems to be doxygen, but it's a pretty low standard (using `rst` and `sphinx` for `python` seems to work a little better).

I haven't solved the problem of getting scientists to write good documentation, although there's a consumer for almost anything:

A review of Don Knuth's TEXBook:

One of my favorite tech books ever. The book is like Felini's 8 1/2, a self-referential and multi-readable journey through elegance in typography.

Documentation

My current theory is that we should give up on scientists writing introductory and how-to documents and instead employ professionals working in close collaboration with the development team.

One thing we're just starting to play with is a `stackoverflow` clone to provide the help desk and simultaneously the top-level documentation that we need.

Lesson 8: Democracy

Neither Science nor Software can be run as a democracy. Not all participants are equal, and it's folly to pretend that they are. This is not to say that the most senior (or smartest) individual should simply lay down the law.

The view from 2014

I stand by this Lesson. Scientists' dynamic range is at least as large as that of software engineers. But brilliance doesn't always bring the ability to run a research group.

Fritz Zwicky was the discoverer of Dark Matter and (with Baade) of Supernovae, which they explained as neutron stars forming (before Robert Oppenheimer).

Jesse Greenstein on Fritz Zwicky:

``Fritz was a self-proclaimed genius, and in many ways he was one.

He was not popular with the establishment, and he was often very wrong. He was an extraordinarily original thinker, but he refused to work either with modern technology or with any elaborate theoretical or measuring apparatus.

So he was a problem child; since I was nominally running things, I had my problems with him.

He was a very interesting person.''
Interview with Rachel Prud'homme, 1982

Lesson 9: Avoid single points of failure

OK, so this is totally obvious, but there are subtler aspects.

- If one person is allowed to become essential it implies that it's proved impossible to find someone else who could fill their rôle.

  In consequence, if they are on the critical path, and problems arise, it's hard to add resources to solve the problem.
- If someone with an essential job isn't very good, then an essential component of your system isn't going to work very well.

Avoid single points of failure

Let me update this lesson. Hire as many people as you can who have with the ability to become single points of failure; then try to manage the project so that they don't.

Lesson 10: Find some way to reward people working on the project

In SDSS we did this by promising them early access to the data via a proprietary period. Not only is this impossible for publicly funded projects, but it doesn't really work very well. One problem is that the promise of data in the distant future doesn't help a post-doc much; another is that the community (at least in the US) doesn't value work on the technical aspects of a large project. I don't think that the solution `Hire Professional Programmers' is viable (although hiring a significant number of *competent* software professionals is a good idea. My experience has been that we cannot afford to hire good programmers).

<hobbyhorse> My personal belief is that the only long term way out of this is to integrate instrumentation (hardware and software) into the astronomy career path, much the way that the high-energy physicists appear to have done (at least from the outside). </hobbyhorse>

The view from 2014

I don't think that much has changed, except that I'm less pessimistic about hiring good programmers.

Janel Garvin, Dr. Dobbs Journal (2013-10-01)

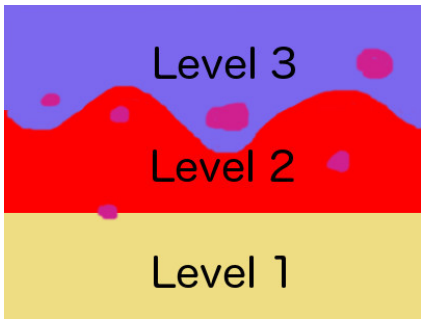
So, all told, developers are not the lonely, antisocial nerds that they are portrayed to be, nor are they free-wheeling socialites.

However, the problem of luring brilliant scientists to work on building the software remains, although it isn't hard to get them to join the project to do science. To be honest, there's a lot of enthusiasm for LSST in the community and they are thinking about how they can help; but they're not very interested in building the project.

Level 3

One problem with the LSST is that the funding agencies are only paying for Levels 1 & 2, but much (although not all) of the fun is in creating Level 3. Furthermore, astro culture makes it essentially impossible to use graduate students to work on Levels [23] so if students want to learn about data they, too, are forced up to level 3. However, the skills to make Level 3 work are in the project.

In reality, I hope to see something like:



where the levels overlap and then mix (with parts of Level 3 settling into Level 2). More accurately, that the people working on the different levels will overlap and mix, working on the things that are needed to get out the science. The challenge is to make this work in the world of EVM.‡

‡Earned Value Management.

Training Our Successors

EU Initial Training Networks (ITN)

> *Embarking on a research career is not always easy. And yet today's young researchers are vital to Europe's future. At Marie Curie Actions, we are well aware of that [...]*
>
> *Our [ITNs] offer early-stage researchers the opportunity to improve their research skills, join established research teams and enhance their career prospects.*

I've been involved in a couple of ITNs (including GAIA), and I think that we can learn from the idea if not the details.

- Teach a series of {Summer/Winter/Spring/Autumn/Candlemas/May Day/Lammas/All Hallows} schools to an evolving set of students and post-docs
- Concentrate on techniques (instruments, software, statistics) tied to the science
- Get the experts from on and off project involved

After a couple of years, we should have a knowledgeable younger generation, ready and itching to do science with the next generation of telescopes. Now I just need to find the cash.

The Emperor's New Clothes

I've spent most of my ``career'', and written and re-written hundreds of thousands of lines of code, processing imaging data.

One of the many things I worry about when giving talks like this is the thought that actually it's an easy problem, and that my concerns are really just a way of making myself feel important.

Maybe the real question is, ``What should we expect from our university faculty?''

Is it of intrinsically higher worth to be able do QCD calculations in your head and invent new massless fields than to love CRTP and see your way to code a subtle new statistical algorithm? And which is a more valuable skill to teach our students?

The End