Interview with Paul Price

The following is the dialog between myself and Paul following a request for “behind the scenes” activities that have been required to identify & diagnose particular issues that have arisen throughout the HSC-SSP processing and analysis that Paul would like to see made available/part of regular & automated QA-ing for LSST.

Paul:
Now that the pipeline works and produces good results generally, it's important to be able to identify the rare bad cases: QA at scale. That means we need a framework for calculating, recording and querying metrics. Perhaps the Gen3 Butler will help with this: I think we're certainly going to want to be able to select datasets for processing on the basis of metrics calculated from an earlier phase of processing.

Lauren:
Most definitely on the framework requirements! Might you be able to elaborate on that last sentence about being able to “select datasets”. Are you suggesting we could somehow define a dataset that is particularly suitable for/sensitive to revealing pathologies arising from a given (type of) metric? Is such a dataset something we could predefine (and would you have any examples of such for HSC data?), or do you see it as being possible/feasible /necessary to identify on a case-by-case basis? Also, we are really looking for some real world scenarios that you have faced and resolved, perhaps with difficulty/much manual labor because you didn’t already have appropriate tools in place. Do you have any examples of this type of procedure that has played out for HSC?

Paul:

- I want to be able to exclude from my coadd any exposures that have poor astrometry or photometry.
- I have some code that harvests some basic metrics from multiband results and generates survey statistics and plots.
- But integration with the Gen3 Butler would make things more efficient (no need for post-processing) and allow selection by the metrics.

For specific examples, see the plots in the HSC data release paper plotting metrics over the survey area.

> TODO: include some examples from http://adsabs.harvard.edu/abs/2018PASJ...70S...8A
> NOTE: I believe what Paul is requesting here as far as survey-wide plotting is precisely what Tim M.’s qa_explorer is providing capability for.
> NOTE: The request for dataset (sub-)selection based on metric values is something I don’t think we have yet considered (but is certainly a good one!)

- Only once you’ve identified bad areas on the survey do you need the usual drill-down tools (Topcat and/or Tim’s stuff, `--debug`, `pdb`, etc.) to diagnose and test.
- Without the whole-survey metrics, you don’t know where to direct the drill-down tools.

Lauren:
Ok, I think I get what you’re after here. Might you have any other wish-list capabilities coming from a different angle where you have a well-understood, high-quality, dataset for which all algorithms have been consistently providing “good” data products, but a change in the codebase results in widespread degradation of a certain metric? Do you see any value in the `lsstDebug` framework for this type of regression? I think most agree that, in principle, it could be very useful for diagnosing pathologies in a specific task (within the limitation that the different flavors the pathologies could take would have to have been premeditated to a deep enough extent that whatever numbers/plots are displayed in the debug mode would be revealing). However, in practice…things are not currently in a useful/user-friendly/coherent/consistent state, so a lot of work would need doing to get things up to a set of to-be-laid-out standards (which all subsequent code additions would also need to adhere to). So, the question there is a framework along the lines of `lsstDebug` worth putting (a lot of) effort into?

Paul:
I think the idea behind `lsstDebug` is useful, and that `lsstDebug` should be updated to be more `Config`-like, specifically that options can be activated/set on the Unix command-line. Using `Config` itself would also force more documentation (using the pex_config `Field`s) on what all the options mean.

A common problem with `lsstDebug` is that the code doesn't get tested for long periods of time. I don't think that can be avoided, since it's difficult to test interactive code in an automated manner.

I'm fine with `lsstDebug` code slowly bitrotting until it's needed, whereupon the user dusts it off as the first stage of debugging.

One might imagine calling the `lsstDebug` code in an automated manner just to make sure it doesn't break completely, but without someone looking at the plots that are produced, that has only limited value.

I would suggest that `lsstDebug` code gets factored out into its own function whenever possible.

Lauren:
Ooh… I like the config-like idea. Yes, ci-ing of the plots themselves is not really feasible without human eyes. We've been talking about this in the context of `pipe_analysis` and came to a similar “at least knowing it’s not completely broken” ci-ing would be better than nothing. We hope to get the repo cleaned up enough to move to `lsst` (from `lsst-dm`) and perhaps have it run as part of `ci_hsc` for this purpose.

Thanks for all your input. I'll pass it along to the QAWG tomorrow…and do let me know if you think of any other specific examples that had to be dealt with during HSC-SSP processing that may have passed me by.