LSST Dark Energy Science Collaboration
Software Policy

Contents

May 2021 (v1.0). Effective on Nov 1, 2021

The initial version of this Policy was drafted by the 2019 Software Review Policy Committee (SRPC): Camille Avestruz, Matt Becker, Celine Combet, Catherine Heymans, Mike Jarvis, David Kirkby, and Joe Zuntz. The initial version can be found on the 2019 SRPC Confluence page (member access only). The 2020 Collaboration Council has modified the initial version based on Collaboration feedback to generate the current version.

DESC Software Policy

Executive Summary

In DESC the means we use to produce our science are principally writing and running software. This DESC policy provides a framework to encourage best practices in software development – in order to produce excellent science results, we need to first produce excellent software.

In this document, the words *shall* and *must* refer to a requirement, *should* and *are expected to* refer to objectives that promote the shared values of the DESC, and *recommend* refers to an encouraged best practice for implementation.

In summary:

- DESC members should perform enough testing and code review to be confident that their code is accurate and reliable. It is up to the authors to decide the appropriate level of review and testing to sufficiently validate the software used in the paper. This should be done during development, rather than waiting until working on the journal paper.
- The authors shall state what methods of code validation were employed by submitting a Software Summary Document (SSD) at the time of internal review of any associated papers or DESC research notes. This document also lists other information about the software that would be helpful to people trying to reproduce the results at a later date.
- All DESC software products (i.e., software that is developed by the DESC using DESC resources) for the creation of a DESC journal paper, including plotting scripts, must live in repositories with at least read-access for all DESC members no later than by the start of the internal review.
- This policy is designed to encourage all DESC software products used in a DESC journal paper to be made public under an open source license upon publication of that paper. While this is not a requirement, most DESC analyses should strive for this standard.
- Code development may be either open (i.e., on a public repository with an open source license) or closed (i.e., on a repository with restricted access or restrictive license). DESC Tools (e.g., CCL, see below) must have read-access for all DESC members during development.
- If code has been made public and licensed as open source, it can be used by any DESC member for any purpose, DESC or non-DESC, with the appropriate citations. Note that no expectation of maintainability or support is implied by releasing Analysis Software or Paper Support Scripts.
- The DESC Publication Policy provides additional information regarding the recognition of software contributions, for example, through authorship on DESC publications as well as citation of associated journal papers describing the software.
- When in doubt about how best to apply any aspect of this policy to a given situation, common sense should prevail, with an aim to further several guiding principles laid out below.
- The Collaboration Council should periodically reassess this policy, taking into consideration feedback from completed publication processes.

Guiding Principles

The following principles apply to most, if not all, DESC software, and the policies set forth in this document are intended to promote these principles. When ambiguities may arise in the application of the policy, the resolution should seek to promote these principles.

Reliability: We want our software to produce accurate and reliable results. All authors of a paper should have confidence in the reliability of code written by other members of the development team.

Reproducibility: It should be possible in principle for anyone to reproduce the analysis in any DESC paper if they have access to the underlying LSST data and/or any relevant upstream data products, sufficient computing power, etc.

Reusability: Many aspects of the analyses we will perform will be relevant to other future papers. Code should be designed to anticipate such reuse.

Agility: Development teams should be able to quickly respond to new or changing demands for software. The software policy should therefore encourage practices that speed up development (at least in the long term) by finding errors sooner rather than later and helping developers to produce high-quality code.

Flexibility: Software should be flexible to accommodate changes in analysis plans. It is rare that the original idea for an analysis works perfectly in the manner that it was originally designed.

Visibility: We want to ensure DESC code developers are visible and that their contributions are fully recognised in the wider astronomical community.

Trust: Any successful collaboration must be based on trust. We trust that all DESC scientists are interested in producing good, accurate scientific analyses. We trust that DESC developers strive to become better at developing good software. We trust that DESC members will be encouraging and non-judgmental in their communication, including those related to code reviews. And we trust that DESC development teams will make every effort to conform to best practices of software development.

Software Review and Publication Policy

The DESC Coding Guidelines (member access only) present the DESC-recommended approach to software development and code review. These are guidelines, not requirements. However, to encourage adherence to the guiding principles, we require a modest level of review for any software used in a DESC journal paper.

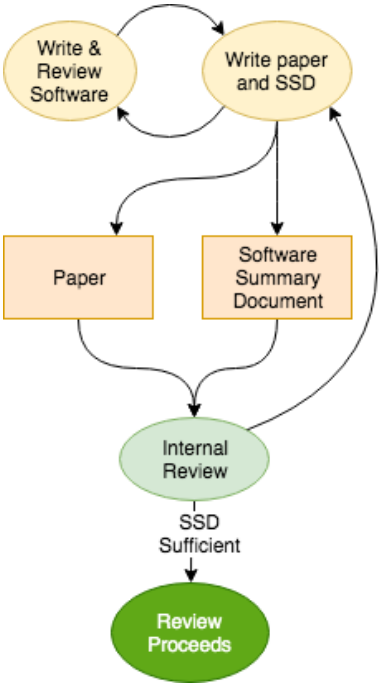
In this document, the words *shall* and *must* refer to a requirement, *should* and *are expected to* refer to objectives that promote the shared values of the DESC, and *recommend* refers to an encouraged best practice for implementation.

While this policy is applied at the time of internal review for a paper, it is relevant to the entire development history of a project. Following the principles from early in the development of the code base is likely to be more effective than adopting them at the end when writing the paper.

The Publication Board is in charge of implementing this Software Policy.

Review Process Overview

The embedded links in this section point to more information elsewhere in this policy document and the appendices.



- Software should be continuously reviewed, documented, and tested at an appropriate level during development.
- Authors are expected to perform enough testing and code review to be confident that the code is accurate and reliable.
- Authors shall certify their code development process by completing a Software Summary Document (SSD) describing all software used for their DESC journal paper. The SSD, along with all of the associated software, shall be made available with at least read-access to all DESC members as part of the internal review process.
- Internal reviewers shall verify that the Software Summary Document is complete, but are not normally expected to review the code.
- Further validation of the code may be requested by the internal reviewers if there are specific concerns about the results or their robustness. The DESC Publication Policy defines the process to resolve any disputes that arise during an internal review.
- Authors should strive to make all code used in the journal paper public with an open source license[1] upon publication[2].

The Software Summary Document

When submitting a DESC publication and its associated software for internal review, the authors must include a Software Summary Document listing all of the software used in the analysis and in the preparation of the paper. A template document is available here.

For the primary software developed for the paper, the authors shall provide the following information:

1. Link(s) to the software repositories that contain all the code that was developed for the paper. The repositories must be accessible to DESC members by the start of internal review.
2. Identification of each type of DESC software in the repositories.

[1] https://opensource.org/osd "Open-source software is a type of computer software in which source code is released under a license in which the copyright holder grants users the rights to use, study, change, and distribute the software to anyone and for any purpose."

[2] For example, the AAS also recommends that authors release code described in an article under an appropriate open source license; see https://journals.aas.org/policy-statement-on-software/

3. The kind of license of the software.
4. The tagged version number of the primary software used. If different versions were used for different aspects of the analysis, this must be noted.
5. Version numbers of any significant dependencies (e.g., numpy, scipy, astropy, fftw, etc.) or other software that was used but not developed for the paper. When using a standard DESC software environment, the tag for the environment will suffice to specify the versions of many dependencies at once.
6. A brief description of at least one system (e.g., NERSC, someone's laptop) where the code was installed and run successfully, including the operating system version.
7. The author(s) who contributed to the code development (as relevant to the paper) and, where appropriate, the additional names of DESC members who have successfully run the code (so years later, someone who may have trouble installing or running the code will have some idea who to ask for help).
8. Approaches taken to code validation. Depending on the code base and development team, this should include some combination of code review, unit tests, integration tests, or design consultation by someone not on the development team.
9. Where appropriate, links to a test suite and documentation should be provided.

See Appendix A for clarifications about these items for the different kinds of DESC software.

Paper support scripts shall also be listed in the SSD, but with less detail. For each repository containing paper support scripts, the authors only need to provide the following:

1. Link to the repository containing the scripts.
2. The version of the code used for the paper.
3. The author(s) who developed the code.
4. Whether at least one other person has looked at the code to ensure it is correct.

Finally, the authors shall list other software used in making the paper, but which were not developed for it. For these, the authors should list:

1. A link to the code that was used.
2. The version of the code used for the paper.
3. How credit was given in the paper regarding the use of the code.

We expect common sense to prevail concerning the level of detail in the SSD document. For example, a directory of related scripts in a repository could be only a single entry in this document. Similarly, it is appropriate to summarize the usual way that the team has implemented code review without itemizing a list of exceptional cases.

The only requirement on the level of code review is that the authors believe that it is sufficient to ensure reliability of the code. We assume all DESC members are acting in good faith in this respect and are trying to follow best practices as much as possible given other constraints on the research project, such as personnel availability and time constraints. Note that time spent doing activities required to follow the best practices, such as code review, will often count towards builder status, reflecting the importance of high-quality code to the DESC.

Once the internal review is finalized, a static version of the SSD (e.g., a PDF) shall be rendered and attached to the Confluence page for the paper.

Types of DESC Software

The software produced by DESC will come in a number of different forms with different scopes, lifetimes, purposes, and user bases. We define five broad classes of DESC Software. The lead developers decide which classification is most appropriate. For software that could potentially become a DESC Tool, the authors should decide the classification in consultation with relevant working group conveners early in the development process. The appropriate level of information about each to include in the SSD varies somewhat among the different types, as detailed in Appendix A.

DESC Tools

DESC Tools are significant software packages conceived, designed, and developed within DESC for use in a wide variety of DESC analyses and potentially outside of DESC. DESC Tools must have read-access for all DESC members during development. DESC Tools are expected to be maintained with at least some amount of user support provided to DESC members. Examples of this include: CCL, imSim, fireCrown, ceci, and namaster.

Analysis Software

DESC Analysis Software is developed to perform a particular scientific analysis initially for a single DESC paper. This includes software that is developed to produce a data product – not technically analysis perhaps, but still developed for a specific DESC project or paper. The analysis software may evolve as the starting point for a similar analysis in the future. No expectation of maintainability or support is implied by releasing analysis code.

Paper Support Scripts

These are simple scripts used to present the analysis for a paper, for example plotting scripts or code to perform some simple calculations. There is often a gray area in terms of where analysis ends and visualization begins. Code in plotting scripts sometimes includes important final steps in a calculation. Thus reproducibility motivates that these scripts should be made available along with the primary analysis code. No expectation of maintainability or support is implied by releasing paper support scripts.

External Software

Sometimes there will be DESC-driven work on an existing (open- or closed-source) external code base, which originated and reached a reasonable level of maturity outside of DESC. When contributing to such code bases, DESC members are expected to follow all policies of the external packages regarding development, use, and citation. Examples include GalSim, Stile, Stomp, ngmix, and SNCosmo. Note that external software that is used for a paper but has no DESC development relevant for that paper is not covered by this policy, except that proper attribution must be given, and it must be listed in the SSD.

Software to Analyze non-LSST Data

For software developed to analyze data from non-LSST instruments for a DESC publication, there are two cases to consider. (1) If the software development is a collaborative effort based on an external collaborator agreement or inter-collaboration agreement, the details should be laid out in the agreement; code developed for the external data reduction does not fall necessarily under our software policy. (2) If there is no external collaborator agreement or inter-collaboration agreement in place, the DESC-developed software is covered by the DESC software policy following the same rules as Analysis Software.

Code Reuse Policy

Open Source Software

Software products that are available publicly under an open-source software license can be used by anyone for any purpose, consistent with the open-source software license. This applies to both DESC software that is under open-source development within DESC, DESC software products that have become open-source on publication of the associated DESC paper, and also external non-DESC software. The original paper(s) describing the software must be cited in any DESC paper where the software is used. If no such papers exist, a Zenodo citation, footnote or similar item pointing to the repository is acceptable. Software contributions can be recognized through authorship on DESC publications, following the DESC Publication Policy. In the spirit of collegiality, it is recommended that DESC members consult with the developers of DESC Analysis Software prior to re-use in a new analysis both within DESC and outside DESC.

The use of open-source paper support scripts can often be treated much like Stack Overflow examples. In most cases an inline comment in the code that uses sections of a previous support script is sufficient to give credit. In cases where paper support scripts are used extensively, however, formal attribution should be given with a citation or footnote in the associated paper.

DESC Software under private development

DESC teams may choose to develop their software in a private repository. Repositories for DESC tools must allow read access to all DESC members during development. Other types of DESC software must be made accessible to all DESC members by the time of internal review.

DESC software that is under private development can be made available to non-DESC members in one of three ways.

1. The software development team can agree to move to an open-source software development model.
2. The non-DESC members can apply for External Collaborator status, in which case any resulting paper would be a DESC paper.
3. The team can use the following process:
   (a) The developers contact the relevant WG conveners to outline the purpose of their software development and why it is in DESC's interest. The relevant Coordinator may be consulted to help resolve any uncertainty.
   (b) With their conveners' concurrence, the developers write a brief proposal. This must describe i) which DESC Project(s) it relates to, ii) what would be shared outside the DESC, iii) with whom or with which collaboration, iv) to what end, v) the anticipated ending date (or date at which the sharing arrangement would need to be renewed by DESC), vi) what development model would be used (e.g., open or private repositories), and vii) any anticipated papers, including why they would not be DESC papers.
   (c) The proposal must be shared with the relevant WG and the Pub Board, with a one-week period for comments.
   (d) If after the comment period the WG conveners (or Coordinator if a convener is conflicted) still concur, then the proposed sharing is approved. The proposal and WG concurrence must be posted in Confluence for the record.

If non-DESC papers result from a code sharing arrangement described above, questions of authorship are to be decided between the DESC members who contributed to the code and the parties with whom the code has been shared. However, DESC will require that the publications include an acknowledgement statement for DESC.

Appendices

These appendices provide some clarifications and implementation guidance about the above policy The suggestions given here are usually highly recommended, but they are not required by policy.

A. Clarifications about what to include in the SSD for different types of DESC software

Appendix E demonstrates that while software development techniques like code review and testing can be effective when used together, they come at a significant cost in short-term development time[3]. Thorough testing of short-lived software is relatively more costly than long-lived software and it is therefore appropriate to have different standards applied to different types of software which we clarify here. The DESC Coding Guidelines (member access only) have a more detailed discussion about best practices that developers should try to follow (to the extent practical) on each of these topics.

Licenses

- All code should have some kind of license to tell other people whether they are allowed to use your code and what kind of restrictions or requirements you have put on its use. Not licensing your code can have serious unintended consequences[4].
- We recommend either BSD-like or MIT licenses rather than a copyleft license like GPL. See the DESC Coding Guidelines (member access only) for more discussion about the trade-offs and why we make this recommendation.
- You may use one license for an entire repository. This will generally be the way Analysis Software and Support Scripts are licensed. We recommend that all or most of these pieces of software live in a single DESC repository connected to the paper where they are used to simplify the licencing and versioning of them.
- A DESC Tool would normally live in its own repository with its own license.

Versioning

- DESC Tools: We highly recommend using semantic versioning. These look like 2.4.3 for instance, where 2 is the major version, 4 is the minor version and 3 is the bugfix revision number. See the DESC Coding Guidelines (member access only) for details about how this should work.
- Analysis Software: Semantic versioning is still recommended, but it doesn't need to be as strict. Mostly, we just want some kind of tagged version number for any run of the code that gets used in the paper. If you forget to tag before running, it is acceptable to go back to the commit you used and tag after the fact.

[3] For example, research has found that effective code reviews should cover less than 200 lines of code per hour (Kemerer & Paulk, 2009, online).

[4] https://choosealicense.com/no-permission/

- Paper Support Scripts: For these, the final version in the repo at the time the paper is ready is probably the right one. To make it easy to reference this version, one should normally tag the repo at that point, e.g., as IRv1 or something similar, and list this version in the Software Summary Document.
- External Software:We have little control over the versioning in this case. Hopefully, one can still get official release versions for the code with any contributions from DESC developers. If for some reason it is necessary to run from the master (or other) branch rather than an official release, at least specify the commit hash that was used.
- For versions of dependencies, the point is to note at least one version of each that is known to work. You do not need to scrupulously keep to one and only one version of each for the duration of the research. But if a year or so later, the code is found to be broken when run with a different set of dependency versions, someone can try to roll each one back to the listed version number to try to figure out which change is at fault.

Code Review

- Code review is an opportunity for developers and users to ensure correctness and reproducibility of functioning code, increase shared knowledge of the code base, and hone team-building communication skills that emphasize building a healthy software development culture. Much of the thrust of good code review is to verify that the code has been adequately tested for the kinds of use cases that are relevant to a given project.
- DESC Tools: We expect a fairly structured code review process as described in the DESC Coding Guidelines (member access only), at least once the code has matured to the point of being ready for a paper. (Early development is often less structured in this regard, which is perfectly acceptable.) While there will likely be a fair amount of variation in the specifics of how different development groups perform code review, the SSD should document the process whereby multiple people have looked at the code. In addition, since DESC Tools are ideally going to be used by many people after their release, multiple people should have successfully installed and used the code.
- Analysis Software: Ideally, most such software will have semi-formal reviews, similar to DESC Tools. If this is not possible the minimum level of code review is that at least some other author (or other DESC member) has looked at the code and signed off on it.
- Paper Support Scripts: These do not need any formal review process. However, we do recommend that at least one other author review any code where any kind of non-trivial calculation is performed.
- External Software: Ideally, this will involve code review similar to that for DESC Tools. DESC collaborators should follow whatever code review workflow is expected for the software being contributed.
- While good code review can be time consuming, it is an important contribution to building the DESC collaboration. We therefore note that time spent reviewing code for collaboration software can count toward DESC builder status.

Test Suite

- DESC Tools should strive to have a relatively robust test suite, as described in detail in the DESC Coding Guidelines (member access only). A detailed description of the test suite is not required in the SSD, but you should say broadly what kinds of tests you ran and where the tests live. Note that there are no specific requirements as to the comprehensiveness of the test suite; the authors should just make sure they are comfortable with the level of the testing being done.
- Analysis Software should usually also have a significant test suite. However, the tests would normally be less robust to usage modes that are not being used for the paper. Rather, the focus should be on ensuring to the authors of the paper that the code is correct for the way it is used in the paper.
- Paper Support Scripts do not need tests. Typically the "test" is that the plot looks nice.

- External Software should typically include tests of any contributions made by DESC members. They should conform to whatever testing paradigm is used by that software development team.

Documentation

- DESC Tools should have a reasonable level of documentation so that the tool is useful to others in the collaboration. It is hard to develop comprehensive documentation, and the documentation may need to grow over time, possibly taking input from user feedback. However, at the time of a paper, there should be enough documentation for someone else in DESC to be able to install the software and run some fairly typical use case(s). As usual, we do not have specific requirements on the documentation, but we expect authors of DESC Tools to take responsibility for making their tool usable, which includes documenting it well.
- Analysis Software does not need to have particularly good outward-facing documentation. Often the documentation may just be doc strings and inline comments explaining to code reviewers and other authors how various parts of the code work. Additionally, a short explanation (in a README for instance) explaining how to install and run the code is useful.
- Paper Support Scripts typically would not have much documentation beyond the usual level of inline commenting and possibly a longer comment at the top of the script describing what it is intended to do.
- External Software contributions should conform to whatever level of documentation is expected by that software development team.

Code Style

- We do not place any requirements on code style for any of the classes of DESC Software. However, as usual, there are recommendations in the DESC Coding Guidelines (member access only) if you want suggestions.

B. Detailed Instructions to the Internal Reviewers

The internal reviewers for a paper should verify that the authors have included the Software Summary Document with the paper and that it includes all the required items for each type of software.

In most cases, they are not expected to look directly at any code or do any code review themselves of any software. Rather they should merely verify that the authors are asserting that a reasonable level of code review, testing, documentation, etc. has been done. The authors are ultimately the arbiters of when there has been sufficient code review and testing to produce accurate results for the paper. We assume DESC developers will always be acting in good faith in this respect, and the internal reviewers should trust their judgement about this.

The internal reviewers should only request more code review of any software, or any additions to the test suite, where there is a specific concern involving the results.

If there is some red flag in the paper that points to a potential problem in the code, internal reviewers may look at the code themselves and potentially request additional tests or adjustments to the software as part of a collaborative enterprise to improve the code, and therefore the paper. By looking directly at the code if they are so inclined, they may be able to provide better feedback to the authors about what might be wrong.

C. Detailed Instructions to Authors

When starting a new project, it is recommended that authors create a repository within the LSSTDESC GitHub workspace where the related analysis software (if any) and paper support scripts can live. This will

simplify some aspects of the Summary Document, since most of your entries will have the same repo and license. It will also help organize the work by having a single place for collaborators to upload the various scripts they contribute to the analysis. If you are building a DESC Tool, the repository must be readable to all DESC members.

During development of the more significant pieces of software (those that will be classified as either a DESC Tool or Analysis Software), they should try to adopt the recommended best practices for code review, testing, and documentation as early as possible. Good practices will help find bugs and design errors earlier, which will tend to save you significant time in the end.

If part of the analysis code uses other software not written by the authors for this project, then the authors are responsible for following the Code Reuse Policy. This should be considered as early as possible, since it can be quite difficult to extract a dependency from your code if you discover later that you are not allowed to use some piece of software.

Once the paper is ready, the authors should prepare a Software Summary Document. They are required to submit this document along with the paper at the time of internal review. If the developers have been following the recommended best practices described herein, it should be quite straightforward to write the Software Summary Document and to have it accepted by the internal reviewers. A template document is available here. One of the authors should copy this to a new document and fill in the appropriate items.

The authors should then provide a link to this document along with the paper when submitting the paper to the working group conveners for internal review. A link to this document should also be put somewhere appropriate in your main repository for the analysis code. The README file is usually a good place to put this link. Also, verify that the repository is at least readable to all DESC members.

We have created a mock up of what the Software Summary Document for CCL might have looked like had this policy been in effect at the time. Authors might find it helpful to look at this worked example when preparing their own document. *Note: some of the specifics are guesses, so this shouldn't be taken as literally correct.*

D. Questions Considered by the Framers of this Policy

In the course of developing this policy document, the Software Review Policy Committee (SRPC) considered many issues regarding code review, testing, reproducibility, etc. Summaries of these issues in the form of questions and their answers are reproduced here for posterity.

Should we require formal code review for all DESC work? No. As documented in Appendix E, code review is a human-labor intensive activity. We felt that the degree of code review should reflect the degree of long-term investment the DESC would like to make in the code.

Should we require 100% test coverage or have specific numerical goals for test coverage? No. While test coverage is a decent metric for advanced projects, early-stage development can be hindered by excessive test coverage because test suites tend to cement APIs and hinder rapid iteration. Further, a numerical goal of 100% test coverage, even for mature tools, may have a relatively small return on investment.[5]

How easily reproducible should DESC analyses be? This issue generated a fair amount of internal debate, with very divergent opinions within the SRPC. On the one hand, the absolute reproducibility of any given analysis by any person, DESC member or not, is clearly a fundamentally good thing. On the other hand, a requirement to make that task easily achievable could impose serious burdens on DESC analysts. By design, the main types of DESC code that perform an analysis, namely Analysis Software and Paper Support Scripts, are expected to have less intensive forms of code review so that teams can iterate on an analysis quickly. Rapid iteration on an analysis is more important than absolutely perfect code, at least

[5] C. Prause, J. Werner, K. Hornig, S. Bosecker, and M. Kuhrmann, "Is 100% Test Coverage a Reasonable Requirement? Lessons Learned from a Space Software Project." 10.1007/978-3-319-69926-4_25. (2017, online)

at first. The end product of this rapid iteration is generally code that is somewhat difficult to run and understand by people outside of the core analysis team. Cleaning up the final analysis scripts so that they can be easily run by an outside person could easily be non-trivial at the end of this process.

In order to impose a limited requirement on reproducibility while not overly burdening DESC analysts, at least two people should know how to install and run any significant part of the analysis. This guidance, coupled with other important data in the Software Summary Document (i.e., the versions of key libraries/dependencies, the tagged versions of the analysis codes, and which machine/system was used), would be sufficient to ensure a reasonable degree of reproducibility.

Should there be a standing committee for reviewing software? No, for many reasons. This would be a lot of work for members of the committee. Members of the same analysis team are usually the most qualified to review their own software. Code review and testing are intimately tied to the results of a given analysis, so an external committee would often not have the appropriate context to perform useful reviews. Finally, such a committee would increase the iteration time for analyses and slow progress significantly. Having the internal review committee for the final analysis results verify the Software Summary Document is a simpler and more manageable policy, while still encouraging good code review practices for the majority of DESC software development.

How strongly should good coding practices be enforced? The SRPC did not feel that heavy handed enforcement of the software policy is either necessary or appropriate. We recognize that many developers in DESC are relatively new to collaborative coding and would find strict enforcement of good development practices to be quite onerous and hard to comply with. Therefore, as much as possible we tried to let the policy encourage a software culture in DESC that will nurture the development of such skills over time. We mostly chose to rely on the honor system and the assumption of good faith by all parties. The oversight that does exist is intended to serve primarily as a reminder for developers to follow good practices.

Can software be reviewed retroactively? While technically possible, a retroactive code review would not generate most of the benefits of continuous code review during development. Thus the SRPC does not recommend retroactive reviews. On the other hand, if the internal review committee identifies a specific issue they reasonably believe is related to a code bug, of course the analysis team may retroactively review the relevant section of code and write additional tests.

What can the DESC do to encourage the development of skilled coders in the collaboration? The SRPC felt that mentorship by more experienced developers and more accessible summaries of the DESC Coding Guidelines (member access only) were the main routes to achieving this goal. The SSD essentially provides a checklist of the major aspects of good code development, which we expect will inspire newer developers to look into the Coding Guidelines to learn more about these various aspects. More efforts at advertising the help and advice available through #desc-software-help could be useful here. Depending on the code base, a member of the DESC Pipeline Scientists and Computing Infrastructure personnel might be the best resource.

How can the DESC maintain and encourage a healthy culture around software development, testing, and review? The SRPC felt that explicitly mandated rules for code review and testing would not achieve this goal. They could also have significant potential downsides in terms of wasted time and resources. Instead, this policy uses the honor system and peer review to maintain reasonable levels of code review and testing within the collaboration. Ultimately, the SRPC felt that such a culture needs to develop organically through mentorship, collaboration, and encouragement among development teams, rather than strict policies. The formal guidance that DESC code should generally be made public, which is important for both reproducibility and visibility, provides incentive to encourage developers to produce high-quality software.

How can the DESC ensure that junior members of the collaboration get credit for their software development and review activities? The SRPC came up with several ideas to address this concern.

First, code review should count towards builder status within the DESC. Second, major analyses can and should publish DESC notes, with junior members as the lead authors, describing the supporting software, its implementation, validation etc. Third, there are several journals devoted specifically to software where junior members could in fact publish these notes. Some other ideas proposed later include leading software development teams and DESC projects, and of course presenting at meetings.

What are the software review policies of other collaborations? We did not take a complete census, but we did reach out to a few collaborations.

- **Dark Energy Survey (DES) and Kilo-Degree Survey (KiDS)**
  DES and KiDS do not have a formal software review policy. They both however have a requirement that key analyses be validated on simulations in as far as this is possible. This requirement is a form of an integration test.
- **The Dark Energy Spectroscopic Instrument** (DESI)
  The DESI pipeline development team has code review standards and CI testing for the instrument and operations code bases. Their practices follow those listed for DESC Tools. However, analysis code by the collaboration is not subject to this standard. At least one science working group has adopted code review practices on its own. Finally, the DESI operations team informally allows pull requests to be merged without review as long as the tests pass. This practice allows critical pull requests that block progress to be merged since the team is small. This practice may be appropriate for small teams building Analysis Software.
- **A Toroidal LHC ApparatuS** (ATLAS)
  The ATLAS experiment employs a similar code review and testing process to that described for DESC Tools. One interesting feature specific to ATLAS is that they have two people on shift to review contributions by the collaboration. One person, the level 1 code reviewer, handles all incoming pull requests. If they need additional technical help on verifying the changes, the pull request gets sent to the level 2 code reviewer. After that, pull requests can be escalated to designated experts on the particular section of code. The practice of having code reviewers on shift seems to be related to the size of the collaboration, their geographic extent, and the size of their code bases. For example, the level 1 and level 2 code reviewers are ideally in very different time zones. This practice is probably not applicable to the DESC. Finally, as of writing this document in 2019, the use of continuous integration and code review in ATLAS is only a few years old.

Should we encourage open source development? The members of the SRPC believe that open source development has many advantages: other users may find critical bugs that were overlooked by DESC members, and they may contribute ideas for improvements. In this mode, work by DESC scientists becomes more visible and will be used widely with the proper attribution and citations.

However, we recognize that this opinion is not universally shared within the DESC. In particular, some members believe that visibility and recognition are improved when development is closed until an official release of the software in conjunction with a paper. Therefore, the policy allows for either open- or closed-source development, with different implications for code reuse in each case.

Should we allow DESC code to be used for projects outside of DESC before being presented in a corresponding DESC paper? This depends on how the code is being developed. If the code is being developed in a public repository and licensed under an open source license, then all non-DESC members are explicitly allowed to use the software according to the license. It would thus be unfair for DESC members to be restricted from doing so by DESC policy. The SRPC discussed this issue extensively, and we felt that the benefits of increased visibility to DESC software (and accordingly the DESC developers of this software) outweighed the potential concerns about software being used before it was "official", especially since non-members are allowed to do so.

The Code Reuse policy thus explicitly allows such use for projects that are being developed publicly as an open-source project. It also describes how code being developed privately may be shared under specific

circumstances.

E. Bug/Defect Rates and the Effectiveness of Testing and Code Review

The rate of bugs in software and the effectiveness of techniques to remove them has been under active study since the 1970s. The consensus is that software defects have a rate of about 25-50 per 1000 lines of code[6]. Further, most techniques to remove those bugs, when used by themselves, are largely ineffective. For example, unit testing, integration testing, and regression testing have been found to catch approximately 25-35% of bugs[7]. Code review is similar, finding approximately 25% of bugs for the informal practice used by most scientists. Highly structured code inspections have better defect finding rates, approaching approximately 60%. Importantly, code review tends to identify code design and maintainability defects at a much higher rate than functional defects[8]. The fractions of each type that are detected in open source projects are about 75% design/maintainability defects and 25% functional defects. The difference between the kinds of defects found among various forms of testing and code review indicate that using both together, which is a common practice in big projects, should identify more defects than either alone.

Interestingly, very extreme measures must be taken to remove all bugs from your code. For example, the team that wrote the software for the NASA Space Shuttle achieved defect rates of only one per 100,000 lines of code[9]. Their methodology includes extensive design documentation, code reviews, unit testing, and an opposing internal team whose only job is to test the software. As an organization, they also focused extensively on the process they used to write software. When bugs were found, they attempted to correct their software-writing process to prevent future bugs.

Code review itself can be quite expensive in human terms. For example, research has found that effective code reviews should cover less than 200 lines of code per hour[10]. At this rate, reviewing version 2.0.1 of CCL with 13000 lines of code would require 65 hours. If we follow the recommendation that no person review code for more than approximately an hour in a single sitting, then it would take months for a single person to review the full CCL code base. Finally, note that it is the total number of lines written that determines the total time spent, not the total number of lines in the code base at present. Thus the actual amount of time spent by the CCL team reviewing code is significantly higher because changes usually include deletion of old code as well as additions of new code.

[6] W. S. Humphery, “The Software Quality Challenge”, The Journal of Defense Software Engineering, Vol 21, (2008; online)

[7] S. McConnell, “Code Complete: A Practical Handbook of Software Construction”, Second Edition, Microsoft Press, Redmond, WA (2004)

[8] Moritz Beller, Alberto Bacchelli, Andy Zaidman, and Elmar Juergens. 2014. Modern code reviews in open-source projects: which problems do they fix?. In Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014). ACM, New York, NY, USA, 202-211. (online)

[9] C. Fishman, “They Write the Right Stuff”, Fast Company & Inc (1996, online)

[10] C. F. Kemerer and M. C. Paulk, "The Impact of Design and Code Reviews on Software Quality: An Empirical Study Based on PSP Data," in IEEE Transactions on Software Engineering, vol. 35, no. 4, pp. 534-550, July-Aug. 2009. (online)