Open Science Grid

Stabilizing and Sustaining our Software Through a New Approach to Software Life Cycle

The Open Science Grid Executive Board

V3, June 2009

The main value that the Open Science Grid (OSG) offers to more than 30 registered Science Communities (VOs) comes through the software tools that we test, package and support. These tools are deployed by more than 80 OSG sites to form a distributed horizontal layer on which end-to-end veridical capabilities can be built. The number of software layers required to build such a capability depends on the size of the science community they serve and the magnitude of the computational challenges the community faces. Given that the OSG supports some of the largest science endeavors undertaken by mankind, the software systems we have to deal with are large and complex. Bringing the power of the OSG to the fingertips of a domain scientist may involve more than a 100 software components that come from dozens of software providers. It is therefore not surprising that the OSG consortium has a vital interest in the quality and sustainability of the software tools that are used to fulfill our mission – scientific discovery through a shared distributed cyber infrastructure.

Software Hardening:

The problems we face in offering domain scientists a software stack that meets their needs and expectations are the result of fundamental flaws in our approach to the lifecycle of software. For example we assume that we can take any piece of software and stabilize/harden it - regardless of how it was designed or implemented. This assumes that it is just a matter of funding (money) to turn a demo-quality piece of software into a dependable and effective software tool that can be integrated into a complex software stack. Unfortunately, this is not the case. Production quality software that can last for many years and be interfaced to an evolving hardware and software infrastructure must be designed to do all this. It is not just a matter of hardening. The same is true to the way the software was implemented. In other words, high quality software starts at the design stage and continues at the implementation stage.

Designing and implementing high-quality software is not cheap – just ask any commercial software company. Since our community considers anything relate to software as "easy" when compared to the science mission and as "invisible" when placed next to hardware, it is impossible to establish software projects with verifiable deliverables and realistic budgets. We are missing the most basic tools required to evaluate proposals for software projects, monitor progress of such projects, adjust plans and budgets or evaluate the delivered software. For example, how can you evaluate a software tool with out a testing suite? We have developed detailed and elaborate processes for doing all this for "mortar and brick" infrastructure but failed so far to do the same for software. In other words, we do not treat software as infrastructure.

Since OSG is a software integrator, not a software developer or maintainer, that is driven by the science mission of our stakeholders we are in a unique position to guide the much needed transformation in the way we manage the lifecycle of the software tools that power our cyber infrastructure. Regardless of the direction we select for this transformation, it must be built on a comprehensive training and education program that covers the entire software lifecycle. The effectiveness of any effort to address our software problems will be determined by the skills and motivation of our workforce. We believe that given the diversity of the OSG stakeholders our experience is applicable to a broad range of software tools.

We recommend a three tier approach (each of which has training/education element) –

- Verify functionality of existing software - is it doing what it supposed to do? **Testing.**
- Evaluate functionality of existing software - is it doing what we want it to do? **Review of functionality.**
- Developing a new approach to software lifecycle. **Blue-print for doing software "right".**

Below we provide some examples of specific activities that fit this 3 tier approach to transforming the software lifecycle. The reflect the need for short term (less than 24 month) improvements in our deployed software that can not wait for the impact to any changes that we will implement in the way we design and implement software. It will take much longer (3-5 years) for such changes to have impact on the stability and sustainability of our deployed software.

Testing:
- Make unit, functional, and especially regression tests a required deliverable in any software project.
- Require a common build and test infrastructure such as Metronome (NMI build and test facility).
- Facilitate collaborations between successful open source and DOE development groups. (http://www.sqlite.org/testing.html)
- Sponsor testbed hardware and virtualization and/or overlay techniques that allow sharing of production systems.
- Established (independent) teams to evaluate software deliverables.

Review of Functionality:
- Make functional specification a required
- Make Security (external) auditing and training a requirement for development teams.
- Oversee architectural decisions as part of an overall model of software capabilities needed for the application domain and/or infrastructure domain.
- Establish (independent) teams to review and evaluate software specifications.

Doing software "right":
- Back the emphasis on hardening by being prepared to fund money for tools that help with hardening and will have wider impact in the market place. Negotiate DOE wide licenses?
- Educate throughout the educational life (high-school, undergraduate, graduate) level the underlying concepts and techniques for testing, auditing, validation, test-cases.
- Encourage (applicable) software engineering research
- Integrate hardening in a similar approach as security into software lifecycle – risk assessment, incident response (to faults, crashes, denial of service etc), include monitoring.
- Address release and patching procedures and policies as part of the software development lifecycle and deliverables.
- Communicate and disseminate software engineering concepts and practices to software development groups. Sponsor collaborations between software engineering and development practitioners.

Learn from successful commercial companies how to tests.