Software development methods and processes for PHENIX on-line computing systems

Thomas Kozlowski

Physics Division, Los Alamos National Laboratory, Los Alamos, NM 87545

1 Benefits of Formal Methods and Processes

As is the case for most large detectors, PHENIX computing systems are large, complex, include many interacting subsystems, and they must be developed in an environment of limited resources, geographically dispersed human resources, and a tight schedule. In addition, high quality software is required that works as intended, when needed. One definition of what is meant by "quality" software is "conformance to requirements". Of course, it is necessary to first determine the correct requirements and their relative priorities.

The computer science community and the software development industry (aerospace in particular) have made progress in addressing these problems. The Software Engineering Institute has made many studies of and recommendations for development processes, including the definition of effective "key practices" (see http://www.sei.cmu.edu/). Most modern development processes and methods are based on the above definition of quality and promote conformance to requirements by explictly defining them and then verifying conformance throughout development. Addressing conformance early ("analysis") is seen as particularily important. Most modern methods recommend specific steps to measure and track conformance:

– review requirements for completeness and consistency
– review conformance to requirements at each stage
– review adherence to the plan and schedule
– review adherence to the process
– take timely and appropriate corrective actions to maintain conformance

The PHENIX on-line computing systems group (ONCS) in accordance with these practices proposed a development process that:

– was defined and documented
– was flexible

– had minimal bureaucratic overhead
– employed appropriate software engineering technologies
– built on “industry” practices

We chose a “spiral” life cycle of one or more iterations to accommodate the detector development environment. The stages within an iteration include planning, analysis, design, implementation and testing, and aach iteration ends in an evaluation stage that determines the next iteration. In most cases specific ”deliverables” are defined at each stage at which time they are reviewed. The formality should be tailored to the goals and risks to PHENIX for a particular project.

2 Analysis and Design Method

Because of the benefit of finding problems in understanding the system and the design early in the project, we thought it was important to choose an appropriate and effective analysis and design method. These more formal methods are not “magic bullets”, but provide a framework and discipline for doing, organizing and verifying the work of defining, designing, and implementing systems, and are rooted in the desire of the software “industry” to increase productivity based on lessons learned.

Studies and experience (See the introductions in the two Shlaer-Mellor books[1,2].) have shown that the root causes of software project failure center around incorrect or incomplete understanding of the problem or system. Therefore, an effective starting point in development is to carefully specify and document the system as a resource for further development and for review by all intereseted parties. Experience (and common sense) has shown the this specification and documentation is best done by focusing on the real and abstract entities (objects) in a system and their relations with each other. This is one value of an object-orientated approach.

The details of our reason for chosing Shlaer-Mellor OOA/RD and a more detailed description of the method may be found in a CHEP95 paper[3]. Some of the criteria we used to choose se a method in an informal survey of methods were:

– supports object-oriented analysis, design, and implementation
– explicitly supports modeling of real-time and concurrent applications
– appropriate for partitioning large systems for concurrent development
– accommodates imported software
– explicit criteria for completeness and correctness of a model
– compatible with incremental development and prototyping

– widely and successfully used
– good CASE tool support exists
– formal training available

The result was the choice of Shlaer-Mellor OOA/RD. It was the only method of those we looked at as appropriate candidates that met all our criteria. Its main advantages include:

– decomposition into orthogonal application, service, architectural, and implementation “subject matter” domains:
  · opportunities for relatively independent concurrent development
  · effective use of varying expertise in a group: physicists, hardware experts, design experts, system experts, etc.
  · service and implementation domains accommodate imported subsystems and software libraries
– relatively complete development process:
  · identification of specific deliverables
  · all steps are clearly defined
  · completion criteria
  · verifiable models (simulation)
  · possibility of code generation
– one of most widely used OOA methods

As a CASE tool, we originally chose ObjectTeam from Cadre Technologies, Inc. We later replaced this with another (and better) tool: Bridgepoint from Project Technology.

The method starts by breaking down a system at the highest level into a series of ”subject matter domains”. Each domain can than be implemented (relatively independently) using the OOA method or by ”traditional” development methods. For those domains that will be developed using the analysis method an Information Model is developed specifying in precise detail the objects, attributes and their relations in the domains. For each object that has a life-cycle, a State Model is created. For each state the processing that occurs at that state is specified by a Process Model, using either a restricted form of a data flow diagram or by a simple high level ”action specification language”.

3 Implementation by Model Translation

One of the domains, the Architectural Domain, has the very specific purpose of providing a recipe or mapping for translating other domain OOA elements into implementation domain elements. This is possible because of the complete and precise nature of the OOA models. Of course it follows that the elements of the

Architectural Domain must be equally complete and precise. This paradiam is suited for automated code generation, and promotes unformity and re-use of implementation elements.

4 Application to PHENIX

The ONCS group has just completed the first prototype system (a skeleton system with only essential capabilities). A primary goal was to gain experience in the method. We used automatic translation/code-generation via the BridgePoint Model Builder and Code Generator tools in conjunction with a purchased basic architecture. Group members received various levels of formal Shlaer-Mellor training (0-3 weeks), and we employed a Shlaer-Mellor OOA consultant. More details may be found in another paper in this conference[4].

4.1 Experiences

The most positive experiences were that automatic code generation really does work, and that (obviously) it is much easier to work at the level of the graphical models than C++ source code. The large analysis effort expended early in a project in getting the models right, results in better understood, documented, and therefore maintainable systems. The separation of the application and architectural domains should reduce the effort in subsequent development, especially with automated code generation. Education in the form of formal training and use of a consultant is valuable and important.

It has not been an entirely positive experience for various reasons. There is a definite concern about payback from the large investment in tools, training, and learning on the job. Developing analysis skills takes time (and talent). The long learning curve guarantees initial projects will take significantly longer than expected. There is a definite unease in developing software in a way that is quite different for many of us ("Where's the code?").

4.2 Some Observations

Development methods are definitely not "magic bullets". System development is difficult regardless of the method. Effort will be larger in initial projects and must be planned for, including formal and on-the-job training. It is important that the entire team be involved in learning and understanding the method at appropriate levels. We had problems because of this at integration time because of differences in understanding of the method across the team.

The ONCS Group is still considering costs and benefits and their effect on plans for the next iteration. The available resources and the schedule must be balanced against:

- the training and learning investment that is required
- the analysis investment that should result in systems that meet requirements and are more maintainable
- the architecture development investment that will result in more efficient code generation

The ONCS Group never adopted a "defined process" except for the set of defined deliverables and development steps that are a part of Shlaer-Mellor method. This is because of several reasons:

- an unsettled environment due to series of management and staffing changes
- disagreement on costs and benefits (doubt in applicability of more formal processes to our environment)
- pressures of schedules and resources
- difficulties in changing our ways of doing things (informal procedures and an inherent dislike of "bureaucracies")

Acknowledgements

This work has been funded by the U.S. Department of Energy.

References

[1] Sally Shlaer and Stephen J. Mellor, *Object Oriented Systems Analysis, Modeling the World in Data* Yourdon Press, 1988.

[2] Sally Shlaer and Stephen J. Mellor, *Object Oriented Systems Analysis, Modeling the World in States* Yourdon Press, 1992.

[3] T. Kozlowski et al., Shlaer-Mellor Object-Oriented Analyis and Recursive Design, an effective modern software development method for development of computing systems for a large physics detector, in: *Proceedings of the International Conference on Computing in High Energy Physics, Rio de Janeiro, Brazil, September, 1995* (World Scientific, 1996) 687-695.

[4] T, Kozlowski et al., Analysis, design, and implementation of PHENIX on-line computing system software using Shlaer-Mellor Object-Oriented Analysis and Recursive Design, in:*Proceedings of the Conference on Computing in High Energy Physics, Berlin, Germany, April 1997.*