# SYSTEMATIC APPROACH FOR THE TRANSITION OF RESEARCH ALGORITHMS TO OPERATION PRODUCTS FOR REAL-TIME PROCESSING AND END-USER APPLICATIONS

T Scott Zaccheo, Edward Kennelly, David Hogan, Hilary Snell, Alex Werbos
Atmospheric and Environmental Research
Lexington, Massachusetts, USA

## Abstract

The transition of algorithms and software from a research environment to operations is a complex process requiring a methodical series of steps to ensure success. As new systems are proposed and developed, new technology in the form of scientific and data processing algorithms is an integral part of system upgrades. The National Research Council Committee on NASA-NOAA Transition from Research to Operations recommends improved transitional processes for bridging technology from research to operations. The two key elements of this process are: A well-defined algorithm development processes that is shared by the science/engineering teams, and a two-tiered software framework comprised of a development/algorithm engineering and production environments, which share common infrastructure/interface elements. This shared data model interface provides common methods for integrating test and provides a seamless mechanism for transitioning. In addition, it also promotes algorithm “buy back” from the operational environment to the development environment that facilitates continued algorithm improvements and their transition to operations based on a shared baseline. In this paper we describe an overall framework for developing a research to operations transition plan, provide examples of the successful transition of diverse algorithms for multiple customers, and discuss recommendations for how the both the research community and operational centers could work together to ensure a more efficient transition.

## Introduction

Well defined processes and procedures are needed to effectively migrate evolving atmospheric and environmental remote sensing science for converting observations from existing, in development, and proposed remote sensing systems into operational products for use by a broad community from lay users to those examining the climatological record. At its core, this process needs sufficient structure to support the required oversight needed to guide national and international missions, while providing a cost effective solution in a fiscally constrained environment that is flexible enough to enable deployment of solutions that are able to evolve as science and technology progress. This work outlines a well-established process for the transition of algorithm software from research to operations, and explores some of its core elements.

The large-scale transition of science-based algorithms to real-time product production environments represents a unique software development challenge. Unlike many software development efforts that are solely customer driven, remote sensing ground processing systems are driven not only by end-user requirements but also the current state-of-the-art in sensor and algorithm development that have scientific and technological limitations. This leads to the need for a research to operation process that reflects the needs of three equal partners as illustrated in Figure 1.The three equal partners are: the customer, the algorithm developers and the software/system production team. The diagram is designed to illustrate roles and not necessarily an organizational structure. The teams involved in this process may be separate entities or contained within a single institution, but must be identified with distinctly separate roles and preferably filled by unique individuals in order to develop and maintain a balance that facilitates achieving the common goal of providing a state of the art ground processing system on-time, on-budget that can evolve into the future.

For government-funded programs the customer is usually the agency or agencies entrusted with the technical aspects of the procurement, and may include elements that provide post-delivery operations and oversight. Their charter is to define requirements that meet specific user needs, often a synthesis of inputs from a number of public and private user communities, and are charged with validating/"selling-off" these requirements upon delivery. In addition, they are often responsible for facilitating interactions and feedback between the algorithm development and production teams, and are the arbitrators of conflicts that impact cost and/or schedule. Commercial endeavors are similar in that there may be consumers including the end users themselves, which is often the case in custom or semi-custom development efforts, or individuals and groups that have assessed a market need via analysis or direct customer interactions. Often it is some combination of both.

The algorithm developers can either be a stand-alone entity with no affiliation or contractual ties to either the customer or the production team, or they can be associated with either the customer organization or production team via corporate affiliation or contractual arrangement, e.g. a sub-contract. In any event they should and need to possess some autonomy in order to implement an effective process. The algorithm developers are primarily responsible for the design and development of the science prototype, the documentation employed in the final implementation and the test data that are employed throughout the design and development life cycle. One of the often-overlooked aspects of the algorithm development process is the construction of configuration managed test data, which is not only vital to the development process itself, but also to the subsequent implementation and validation of the production/operational system. Optimally this suite of inputs and validated outputs not only define the expected system behavior under normal operating conditions, but also the expected values at internal intermediate point with the algorithm(s) and the behavior of the system under non-nominal/fault conditions. Validated intermediate data are invaluable in the development of production software from both a white/black box test perspective, and during debugging phases that naturally occur when the provided documentation and prototypes don't fully represent the desired production environment. These data are also critical in defining the acceptance criteria for sell off the program from the production team to the customer. In this role the algorithm developers are also charged with facilitating the design and development of the science acceptance criteria.

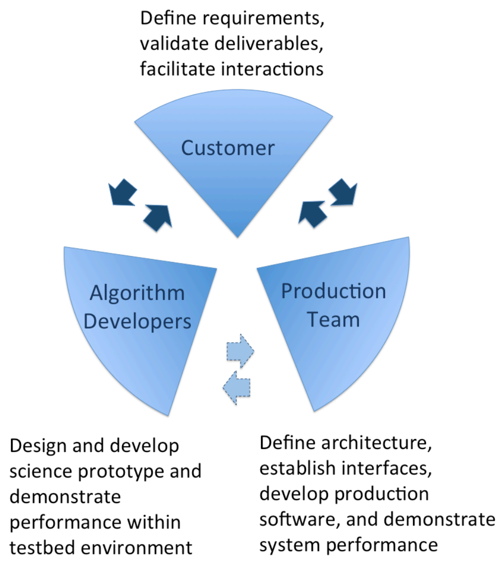
Define requirements, validate deliverables, facilitate interactions
Customer
Algorithm Developers
Production Team
Design and develop science prototype and demonstrate performance within testbed environment
Define architecture, establish interfaces, develop production software, and demonstrate system performance

*Figure 1 Interactions between three primary contributors to the research to operations process for the development of remote sensing ground processing production systems: the customer, algorithm developers and the production software team.*

Finally, the production team is responsible for defining the overall system architecture, establishing sensor, ancillary, auxiliary and meta- data interfaces for both for the algorithms input as well as its intermediate and output products. It converts algorithm documentation and possibly algorithm prototypes into operational software that normally resides in a larger execution infrastructure. This infrastructure is either co-developed as part of the research to operations program, derived from pre-existing infrastructure or previous developed environments that are being augmented as part of the effort. In cases where the infrastructure is designed, developed and deployed as part of the effort, the production team is often responsible for this effort and its development life cycle must be integrated into the transition process.  The production team is also responsible for integrating provided test data into the production environment to facilitate unit and integration testing and subsequent system validation. While the data for testing and validation maybe different, e.g. test cases providing both inputs and outputs verses those consisting of only inputs with blind outputs (or vice versa), the data ingest design/implementation should be agnostic to data format and the test cases themselves.

The two key players on the production team are the applied science or algorithm engineers and the software developers that work in concert to develop the final production system. Each has distinct roles in each phase. The primary focus of the algorithm-engineering team is to facilitate interactions with the algorithm developers throughout the development process to support issue resolution, to facilitate validate of operational performance against science prototypes, to assess results from comprehensive testing and to enable algorithm enhancements in order to address issues that arise during development and are critical to meet mission requirements. This science oriented team also audits algorithm information (often Government Furnished Information – GFI), resolves issues, maintains documentation, prepares test data, and supports software development for each stage of development. The other player is the software developers that are responsible for defining and designing the infrastructure and interfaces employed in the system, and the development and testing of the both the execution infrastructure as well as the algorithms themselves.

Some of the challenges in developing a process that provides effective research to operation and accommodates all parties are centered on the management of these diverse groups and external sources, which drive their activities.  One such aspect is the management of different skill sets that often in conflict. The algorithm developers' emphasis is getting the science right and staying at the cutting edge, while the operational team is focused on deploying a solution that is correct, reliable, fast and maintainable.  In addition, different external forces tend to drive each organization in different directions. The customer is often driven by new requirements or cost/schedule constraints, which tend to drive the implementation or limit it scope.  The algorithm developers are naturally driven by advances in science that naturally occurs over the course of programs with substantial scale, and finally the production teams are driven by changes in technologies and the desire to incorporate new elements into the current applications. Managing these interests is a complex task in and of its self, and often impacts the cost/schedule. Optimally the overarching management needs to allow for the injection of emerging technologies, after assessment of risks. This is particularly true in those cases that may reduce overall cost and schedule.   In addition to a management approach, the overall system design should play a prominent role in mitigating the impact of emerging technologies by being sufficiently modular in nature to effectively accommodate advances in future releases, upgrade or refresh periods.

## Method

Two of the key elements that have been develop as part of a mature process for transitioning research to operations are a well defined production process that encourages participation of all parties throughout the entire development life cycle and beyond, and common set of software tools that facilitate the transition of algorithms from inception to 24/7 production.

A well-defined production process is key to implementing a successful transition of science research to operations. While there is not one exact formula for implementing such, there are several key elements that help facilitate it success. These elements are: 1) The incorporation of both science and software expertise into the production team, 2) Pre-defined mechanisms for resolving issues encountered during the development with both the algorithm developers and customer and 3) Standardized and configuration managed document that are peered review at formal steps along the process. A diagram of such a process is illustrated in Figure 2. This figure shows the basic elements of the production process that flows from left to right from requirements analysis to system sell-off at the software qualification test. The other two important aspect of this diagram are the inputs to the process, which consist of documentation/prototype information represented by the arrow labeled GPA (Ground Processing Algorithm) and test data both supplied by the customer and algorithm developers, and a common method for issue resolution that is consistent through the production development life cycle.

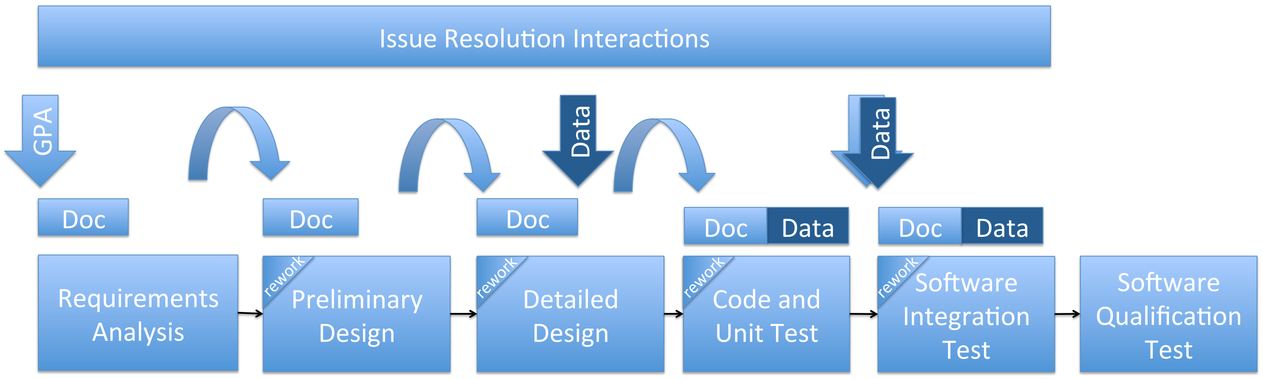

*Figure 2 Elements of an interactive product development process that promotes issue resolution via interactions at each defined stage of development between all parties (customer, algorithm developers and production team) to insure rapid and cost-effective implementation.*

The basic elements of a standard waterfall implementation of this process are: requirements analysis, preliminary and detailed design, code and unit test, software integration and testing, and software qualification testing leading to system sell-off. The production development process is initiated with inputs from the customer in the form of high-level mission/system requirements and ground processing algorithm information in the form of Algorithm Theoretical Basis Documents (ATBDs) developed by algorithm team. In this phase the production team audits algorithm information and incorporate this information into lower level system and software requirements. These efforts provide a first look at the desired algorithm(s) and the development of a mapping of customer requirements to algorithm capability. This important first look audit of ATBDs and comparison to requirements provide initial evaluation of production system. Upon conclusion of this analysis, the feedback, as shown in Figure 2, is provided to the customer and algorithm developers in order to proactively address gaps in requirements or proposed design, and allow for early intervention. The second phase is the design phase and may incorporate a number of steps. The diagram above shows distinct preliminary and detailed design efforts. The primary focus of the preliminary design is to detail the algorithm inputs and outputs from both a science and software perspective, and to develop the high-level software architecture that will accommodate the eventual algorithm implementation and deployment. The preliminary designs from a software perspective include the layout of required data interfaces, the general and specific data model that will be used to ingest sensor, ancillary, ancillary data and intermediate products from other algorithms, and those used to construct output data and the final data products themselves including accompanying static and dynamic meta data. In addition the general algorithm design principles are defined and the high-level companion execution infrastructure along with general methods for error handling, message passing and logging facilities are outlined during the primary design phase. The design artifacts associated with this phase are often provided in documentation form and may include functional decomposition of algorithm components in the form of activity, data model and sequence diagrams to describe the overall data flow. In the detailed design phase both the science and engineering analysis are revisited and expanded to include detailed description of the algorithms and infrastructure. These details may be captured in tailored set of documentation that meets specific program needs. These documents should include detailed algorithm description documents (ADD) that, unlike their companion ATBDs, focus on providing complete implementation details, and standard class diagrams that illustrate software module interfaces and interconnects and provide direct represent of the as build software. Finally, data from the customer/algorithm teams are assessed at the detailed design phase, and aid in the design of testing strategies. As illustrated in the diagram feedback to the algorithm developer and customer is encouraged at an appropriate frequency. At a minimum, the resulting design material, developed during any one phase should be reviewed by all parties at the conclusion of the phase.

The third phase is Code and Unit Test (CUT) where software elements of both the infrastructure and algorithm are implemented and tested. Integration of common test data provided by algorithm developers is critical in this development phase. Test data not only helps facilitated debugging and testing at both a method-based white box level and algorithm element black box level, but is also invaluable in validating the ATBD and ADD documentation and the expected results themselves. This early testing with externally provided data has proven invaluable, and helps facilitate early detection of algorithm implementation flaws both in algorithm design prototypes and the production implementation.

The final two phases of the process illustrated in Figure 2 are the software integration test (SWIT) and final software qualification/site acceptance testing. In SWIT the operational functionality of individual algorithms are validated along with the integration between the infrastructure and algorithms and any interdependencies between algorithms. Once again the early integration of test data that has been developed in collaboration with external partners is critical to the validation process. A highly integrated and open SWIT leads to almost anti-climatic qualification/acceptance testing since all parties are aware of the expected results and have in many case already addressed shortcomings.

While the process illustrated in Figure 2 depicts a waterfall development model, the same principles can and have been adapted to other software development paradigms. The basic tenants can be integrated into either a spiral or agile development process. The common addition is interjecting well-established customer/algorithm development team interactions into each spiral or sprint to facilitate feedback and issue resolution.

The successful transition of remote sensing algorithms to ground processing production not only requires an open production development process, but also the implementation of standard tools to help facilitate collaboration between the all parties. In particular, collaboration is essential between the algorithm developers and the production teams. These tools should also be extended beyond the use of standard collaboration tools for documentation and configuration management, documentation development and software development/testing, to mission specific tools that directly facilitate the migration algorithm science to production. One such set of tools is the development and implementation of a common two-tiered software development environment that not only aids in the development of production quality algorithm implementation, but in an optimal environment can also benefit the initial science algorithm development.

The basic elements of such a design are illustrated in Figure 3. This figure shows the two product development environments. At the top of the figure is the CUT implementation environment that facilitates rapid production development cycles and the development of unit test environments that employs standard data interfaces that provide identical functional through a common data model interface (DMI), and the use of a common algorithm framework that can employed seamlessly in their deployment in the operational environment. This architecture/ engineering development environment provide an environment that facilitates ingest of both internally developed and externally provided test data, and supports the analysis of resulting diagnostic, intermediate and output data/products while minimizing the implementation overhead often associated with production environments. The lower left hand side of the diagram illustrates the production environment. While this implementation is far more complicated and addresses a number of real-time processing issues not implemented in the algorithm development environment, it employs the same mechanism for instantiating an algorithm and provide the DMIs with identical interface specifications as those deployed in the development environment illustrated at the top of the figure. In adopting this process the production team inherits the ability to transition an algorithm or suite of core science software back and forth between its development and operational environments without altering the core algorithm code base. This "buy back" process enables a single code base to be tested both environments as augmented where appropriate without software re-work.

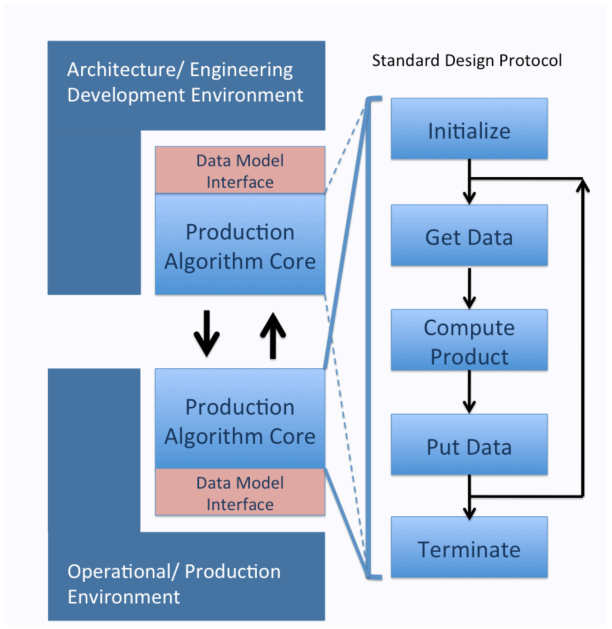
Architecture/ Engineering Development Environment
Data Model Interface
Production Algorithm Core
Production Algorithm Core
Data Model Interface
Operational/ Production Environment
Standard Design Protocol
Initialize
Get Data
Compute Product
Put Data
Terminate

*Figure 3 Illustration of two tiered development model that enable seamless transition between algorithm development and operation/production environment based on common abstract data model interface (DMI) and standardized algorithm framework.*

The right-hand side of the diagram illustrates an example algorithm "shell" that provides implementation examples using the provided DMIs and in many case standard mechanisms for constructing an algorithm instance and associated interfaces to the data and other services, e.g. error handling and logging facilities. This use of a data model interface promotes software re-use/re-engineering, and its modular design practices ensure efficient algorithm migration from the start. In the optimal environment these tools are deployed at the algorithm development facility and are employed during the science exploration phase of the process making it the transition and production testing as seamless as possible.

## Conclusions

The process outlined above has been developed, employed and matured from both the algorithm development perspective as well as the production point of view on a number of programs over the last 15 years at Atmospheric and Environmental Research, Inc [1]. This includes the development and/or co-development of several the algorithm frameworks and data model interfaces to meet program needs and constraints. Some examples of these efforts are as the algorithm developer of key components of United States Air Force Weather Agency's Cloud Depiction and Forecast System (CDFS-II) [2], as both the algorithm development and production team for the Air Force space weather Communication/ Navigation Outage Forecast System (C/NOFS), as key member of the GOES-R Level 1b and 2 product production team [3, 4] and as customer, algorithm and production developers for in-house commercial product development. In the CDFS-II program, AER has played a continuing role as the cloud mask and cloud properties algorithm developers and have been an integral part of the transition from science development to implementation and assessment in a 24/7 operational environment designed to prove integrated remote sensing cloud produces from multiple LEO and GEO mission it to a single analysis/forecast view. The production-oriented algorithm was successfully transferred to a separate production team via standard documentation, prototypes and regular interactions between both parties. In the development of the C/NOFS data center, AER played a dual role of both algorithm developers as well as the production team. In this case, rules were established in order to insure the equal partnership between the algorithm development effort and the production team was maintained. The development of the production GOES-R L1b calibration and L2+ product software follow a full implementation of the process described above. In this case, AER was responsible for developing the ground processing algorithm software and co-developed the required data model interface/algorithm framework that has successfully been transition into integrated end-to-end production facility. As part of this effort a full suite of algorithm test tools were developed that provided unified test harness for thoroughly testing the algorithm with both test data developed in house as well as that provided by the Government GOEs-R science team.

A number of lessons learned can be drawn from successfully implementing the process and tools outlined above on these and other programs. The overarching lessons fall into three general categories: science, software and management. From a science perspective the general themes that the authors believe need to be adhered to are as follows:

1) The algorithm developers should strive to produce clearly documented algorithms that represent a point design and not a "work in progress" that may have unanswered questions and possible alternative untested solutions

2) As part of the science definition process the production acceptance criteria should be clearly defined and appropriate test data constructed to demonstrate that these criteria are unambiguously met

3) The science community needs to buy into the concept of "buy back" to facilitate post-production updates and analysis

The lessons learned from a software perspective are:

1) Employ a standardized data model interface that abstracts low level access to data and services to ensure that the core algorithms are insulated from both the development and production environments in order to maintain interoperability. In addition, algorithm developers need to adopt these abstract interfaces in order to facilitate the "buy-back" process from algorithm inception to production
2) Early integration of continual regression testing based on common data and software framework is essential to achieving cost and schedule goals by reducing debug efforts and allowing for the identification of potential problems early in the production cycle

Finally from a management perspective the lessons taken from these experiences are:

1) The transition process must account for both the development and production schedules
2) Promoting and facilitating interactions between developers and the production team is critical to prevent the natural tendencies to transition elements of the design in an "over the wall" fashion that leads to design disconnects and cost/schedule impacts
3) The process should adequately account for 'rework" to accommodate resolution of unanticipated science and/or software issues that are part of the normal development process

The process and design approach describe above, which has been employed and refined over a number real-world programs, is highly effective in transiting research science software to operations in a cost-effect and schedule driven environment. The method outline above can be applied to a wide ranging set of development effort from large systems of systems designed to meet the combined objectives of multiple space-based mission to smaller scale efforts the may only address the production of tailored products for select customers or spatial regions. In addition the process can be applied irrespective of the desired software development process paradigm, e.g. waterfall, spiral or agile, and end user computation platform, e.g. standalone to virtual and cloud environments.

## References

[1] T S Zaccheo, J Galantowicz, D B Hogan, E J Kennelly, and H E Snell, "Testbed Architecture for Rapid Prototyping and Assessment of Environmental Remote Sensing Algorithms," in *Proceedings of SPIE Conference #5548*, Denver, Colorado, August 2004.

[2] F D Bieker, R W Evans, and G B Gustafson, "Cloud Depiction and Forecast System," in *Proceedings, The Battlespace Atmospheric and Cloud Impacts on Military Operations (BACIMO)*, Monterey, CA, 2003.

[3] A E Werbos, A Copland, E Steinfelt, P A Van Rompay, and T S Zaccheo, "End-to-End Design, Development and Testing of GOES-R Level 1 and 2 Algorithms," in *AMS 29th Conference on Environmental Information Processing Technologies*, Austin, TX, 2013.

[4] A Werbos et al., "GOES-R Software Implementation Design: Design and Implementation of GOES-R Level 2+ Product Generation Algorithms," in *92nd American Meteorological Society Annual Meeting*, New Orleans, LA, Jan, 2012.