The software development process at the Chandra X-ray Center

Janet D. Evans[*], Ian N. Evans, Giuseppina Fabbiano
Smithsonian Astrophysical Observatory, 60 Garden Street, Cambridge, MA 02138

ABSTRACT

Software development for the Chandra X-ray Center Data System began in the mid 1990's, and the waterfall model of development was mandated by our documents. Although we initially tried this approach, we found that a process with elements of the spiral model worked better in our science-based environment. High-level science requirements are usually established by scientists, and provided to the software development group. We follow with review and refinement of those requirements prior to the design phase. Design reviews are conducted for substantial projects within the development team, and include scientists whenever appropriate. Development follows agreed upon schedules that include several internal releases of the task before completion. Feedback from science testing early in the process helps to identify and resolve misunderstandings present in the detailed requirements, and allows review of intangible requirements. The development process includes specific testing of requirements, developer and user documentation, and support after deployment to operations or to users.

We discuss the process we follow at the Chandra X-ray Center (CXC) to develop software and support operations. We review the role of the science and development staff from conception to release of software, and some lessons learned from managing CXC software development for over a decade.

Keywords: data processing, development processes, software development, system architecture

1. INTRODUCTION

The Chandra X-ray Observatory is the X-ray component of NASA's "Great Observatories" program. Launched in 1999, Chandra is in its tenth year of operations and continues contributing to many new astrophysical discoveries. The Chandra X-ray Center Data System[1] (CXCDS) provides end-to-end scientific software support for mission operations.

Early development of the CXCDS followed a formal and planned systems engineered approach based on the Chandra teams experience with previous spacecraft and X-ray missions. Science and engineering technical expertise joined to develop an operational system by launch that met the needs of the project. With a solid framework in place, the process of development in the post-launch years migrated to an approach that also borrowed from the spiral development model[2]. We have clearly defined phases of development, as well as scheduled iterative phases where risks are mitigated and requirements refined. Development includes a series of internal releases that are available for internal science test and feedback before the module is released in its final form.

To manage the process, the software group is divided into seven functional teams (Archive, Automated Processing/ Observation Cycle, Pipelines and Tools, Science Algorithms, Aspect Camera/Monitor and Trends Analysis, Telemetry/User Infrastructure, and Configuration Management) along the boundaries of various key areas. Each team develops software from the requirements phase through to the maintenance phase. To ensure end-to-end system functionality, our management structure includes a dedicated scientist who is responsible for interfaces, end-to-end integration, and system requirements, and a software development manager who is responsible for schedule, process, software, and release.

[*]janet@cfa.harvard.edu; phone +1 (617) 495-7160

2. CHANDRA DATA SYSTEM OVERVIEW

The CXCDS is a complex and large software system consisting of ~2.6 million lines of code. Table A summarizes software metrics over the various CXCDS groups. In the section below, we review the components of the Chandra data system and data flow (Fig. 1) to provide the reader with some insight on the software scope of the system.

Table A: CXCDS SLOC counts

Code	*SQL*	*Scripts*	*Java*	*Par*	*XML*	*TOTAL*
1,435,866	475,642	378,602	87,893	33,780	164,995	2,615,835

2.1 Software for Science Operations

The CXCDS team develops, maintains, and supports the software and hardware that drive most of the CXC-based *forward* (proposer to spacecraft) and *return* (spacecraft to observer) threads necessary to perform the Chandra observing program.

The forward thread begins with proposal submission and receipt software that manages development, validation, and submission of user science proposals, and their receipt by the CXC. Additional software applications support the organization of peer review panels, assignment of reviewers to panels, conflict checking proposed observations, and managing peer review statistics and reviews. Approved targets are promoted to the observing catalog (OCAT) database in preparation for science mission planning. Mission planning software extracts observations from the OCAT and supports assignment of them to weekly schedules in a way that satisfies scheduling constraints stated in the proposals. The resulting observation request (OR) list is submitted to the Chandra Operations Control Center (OCC) for detailed scheduling and command generation, and includes a detailed definition of each requested observation, including constraints and instrument configuration. The OCC returns the detailed observing plan, which is compared to the request to validate the schedule. Observations not completely scheduled are updated in the OCAT and made available for subsequent scheduling.

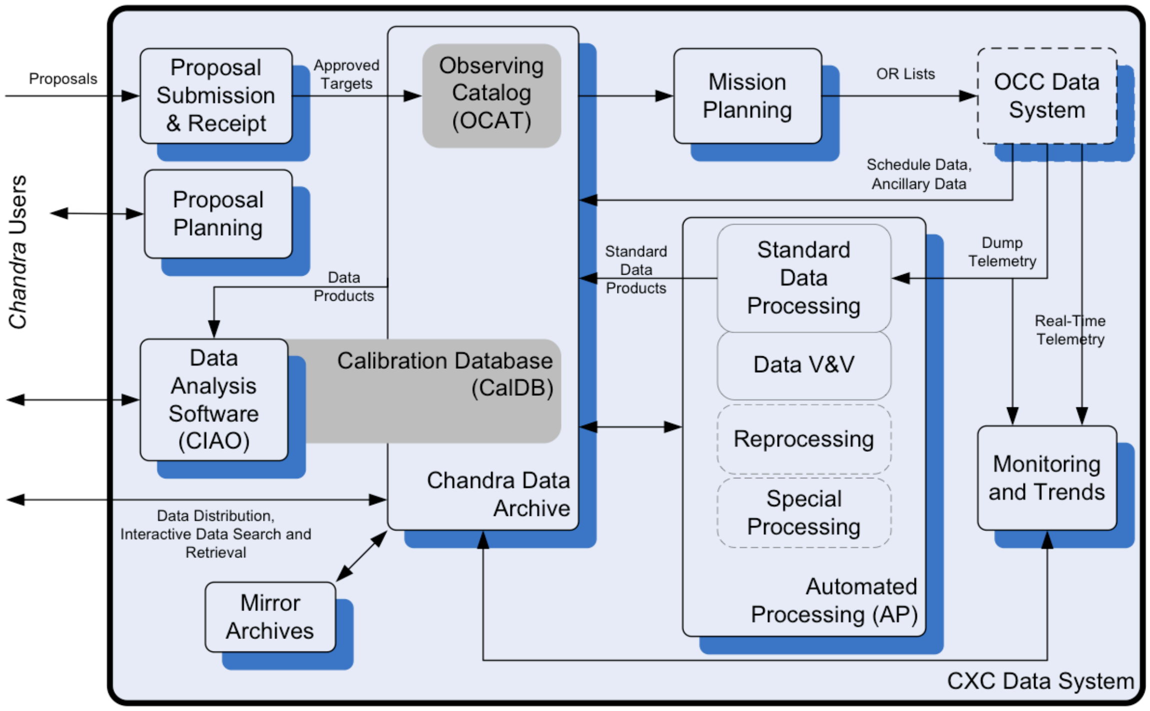

Fig 1. This figure depicts the CXC data system components and architecture[4]. The CXCDS automated processing facility is co-located with Flight Operations at the OCC. The Chandra data archive is hosted at the same location and mirrored off site. Proposal submission and receipt, CXC mission planning, and science user support are located at the SAO Garden Street location.

Receipt of dump telemetry data from the OCC begins the return thread. Standard data processing[3] (SDP) pipelines perform standard reductions to remove spacecraft and instrumental signatures and produce calibrated data products suitable for science-specific analysis by the end user. Each pipeline includes ~5–30 separate programs, or *tools*, and several dozen pipelines comprise the SDP thread. Processing is managed by an automated processing (AP) system that monitors data, instantiates pipelines, monitors pipeline status, and alerts operations staff when anomalies occur. Information extracted from the OCAT is used to manage observation processing, and to verify that the observation was obtained in accordance with the observer's specifications. Any data can be reprocessed as needed (because of prior errors, or improved calibrations), and custom manual workarounds can be applied to any observation that requires special handling. Public web-based tools provide current and historical information about observation processing status and issues.

Monitoring and trends software performs limit monitoring of both dump and real-time telemetry for health-and-safety, and triggers real-time alerts of anomalies as configured by the CXC Science Operations Team. Databases support long-term trends analysis to predict future spacecraft and instrument function for forward planning.

The Chandra Data Archive[5] (CDA) securely stores all telemetry and data products created by SDP. Archive software restricts access to proprietary data, performs data distribution to observers, and manages public release of data once the proprietary period expires. Interactive data search and retrieval is supported by web-based and downloadable applications that are tailored to search the observing catalog, preview data, and download selected data. In addition to storage and retrieval of data, the archive supports internal functions including data packaging, compression, migration, backup, queue and license management, and security.

The Calibration Database[6] (CalDB) for Chandra is a configuration-managed, fully indexed, HEASARC-compliant, software-accessible data structure that is used in both standard SDP and the Chandra Interactive Analysis of Observations (CIAO) data analysis software package[7].

2.2 Science Software for Users

The CXCDS team develops and maintains applications to support users in their proposal preparation and data analysis. The proposal planning applications include a set of tools that predict the performance of Chandra and an interactive observation visualizer.

The CIAO software package is the distributable data analysis system for Chandra, and all users of the observatory (internal, guest, and archival) utilize the package to perform further calibrations and science-specific analysis, and extract publishable information from the pipeline data products. CIAO consists of over 85 analysis tools, and a suite of integrated applications for spectral and spatial modeling and fitting, visualization, and data manipulation. Some tools perform Chandra instrument-specific calibration and analysis functions, while others are designed to be compatible with a variety of missions while taking advantage of special knowledge of Chandra internals where appropriate. CIAO is designed to support both expert X-ray astronomers and newcomers to the X-ray domain. Python and S-Lang environments allow advanced users to develop sophisticated analysis scripts, and CIAO extends those languages by adding Chandra-specific modules and applications. A simple, interactive parameter interface meets the science needs of less computer-savvy astronomers. CIAO's unique analysis capabilities, (including a region library, the Chandra data model, the ChIPS plotting and Sherpa fitting applications) are layered on top of existing, well-tested community libraries, and are based on community standards to maximize reliability and compatibility. For each tool and application, on-line, searchable documentation is provided that explains the function and usage of the tool.

CIAO is ported to Sun/Solaris, several flavors of Linux, and Mac OS X. The CXC periodically polls the user community and continuously evaluates CIAO download trends to drive the choice of supported platforms.

2.3 The Chandra Data Archive

The CDA is a rich data repository and a significant scientific research and publishing resource. Several interfaces to the CDA are supported to meet the needs of the various user communities who wish to access Chandra data. WebChaSeR, the premier interface to the CDA, is a flexible and powerful search and retrieval tool that meets the needs of professional users and incorporates authorized access to proprietary data. Direct access by data visualization tools such as SAO's DS9 and CDS's Aladin is accommodated through integrated protocols, and existing IVOA-compliant interfaces support Virtual Observatory applications. In addition, the CDA provides an easy-to-use, image-oriented access tool for use by the general public.

CDA consists of a set of databases and a repository of data products. The primary archive is located at the OCC and access is largely restricted to operational entities. A secondary archive located at SAO contains all databases and a subset of the data products, accessible primarily to non-CXC users. The contents of this archive are such that it can take over the role of the primary archive in an emergency without impacting operations or the integrity of the mission. Data products from public observations are additionally replicated to an anonymous ftp site, which is the primary source for data downloads.

3. SYSTEM FUNDAMENTALS

The data system architecture was designed to meet the functional needs of the hardware components and processing flow of the observatory, balanced with design of system engineering components to support the development and maintenance of the project over the lifetime of the mission. Early attention was given to learn how the spacecraft sub-systems and science payload were built and how they would function during operations. At the same time, system architecture was developed with fundamental design characteristics that we believe ensured a stable and enduring framework for our project.

3.1 Spacecraft system

The Chandra X-ray Observatory consists of a spacecraft system and a telescope/science-instrument payload. The spacecraft system provides mechanical controls, thermal controls, electrical power, communication, command, data management, pointing, and aspect determination functions. The payload consists of a High Resolution Mirror Assembly (HRMA), high and low energy transmission gratings (HETG/LETG), two focal plane science instruments: the High Resolution Camera (HRC), and the Advanced CCD Imaging Spectrometer (ACIS), an Aspect Camera Assembly (ACA), and an EPHIN particle background monitoring instrument. The spacecraft also includes mechanisms for moving the gratings into and out of the HRMA X-ray beam and moving the science instruments into position in the focal plane and adjusting the mirror focus.

Instrument scientists worked with data system developers to detail the operation of spacecraft components and their role in the telescope system. At the same time, developers worked with scientists to provide insight on automated data reduction systems and archive data product design. We had a chance to demonstrate our early understanding of the system with the requirement to have an automated pipeline and archive system operational during end-to-end ground calibrations at the X-ray Calibration Facility (XRCF) in Huntsville, Alabama, 2 years prior to launch. The pre-launch data system was designed and built at the same time that the instruments and mirrors were being fabricated and assembled in the laboratory. The instruments were new technologies for an X-ray mission, and the only test data available prior to the start of calibration were stand-alone laboratory data from the instruments. The XRCF requirement was a programmatic trial-by-fire, and presented an immense challenge to the data system team. The team achieved the goal of supporting XRCF with a system that processed and archived the ground calibrations test data, but the software needed constant watching and attention. We learned many lessons about operations, data volume, data anomalies, error recovery, speed, and automation. As a principal result of the ground test, the core team formulated a clear vision of what had to be done to bring the system to launch readiness.

3.2 Internal characteristics

The high-level modular organization of the CXCDS software is shown in Figure 2. The architecture consists of layers of software components and modules separated by standardized application programming interfaces (APIs). Each layer hides the details of its internal structure and mechanisms from the other layers.

The *user interface* layer provides an interface to the 4 basic types of users identified by the project: local users — staff members accessing the CXCDS through a workstation physically connected to the CXC local area network (LAN); remote users — observers or researchers accessing the CXCDS LAN through external communication methods; anonymous users — scientific researchers accessing the CXCDS through a public interface; and autonomous users — scientific researchers who are executing portable CXCDS software installed and/or downloaded on their own workstation. The *data processing* layer contains software that performs the CXCDS-specific data processing or data transformation tasks. The majority of software developed for the system is contained within this layer, and the components determine application groups that tend to follow the functional capabilities of the software components. The *data management* layer contains the Sybase Relational Database Management System (RDBMS) software and CXCDS-specific server, client, and gateway components that support the Chandra Data Archive. This layer also contains software implemented as SQL-intensive tasks that are dedicated to the management or translation of particular

CXCDS databases or the assembly of subsets or views of the stored data. The *system* layer contains commercial off-the-shelf (COTS) system software, including the operating system, network software and distributed file system software, which implements a distributed processing and file system environment across networked hardware platforms and devices. This is the base layer of the software architecture, and also includes CXCDS-developed libraries which provide uniform application development and communication primitives.

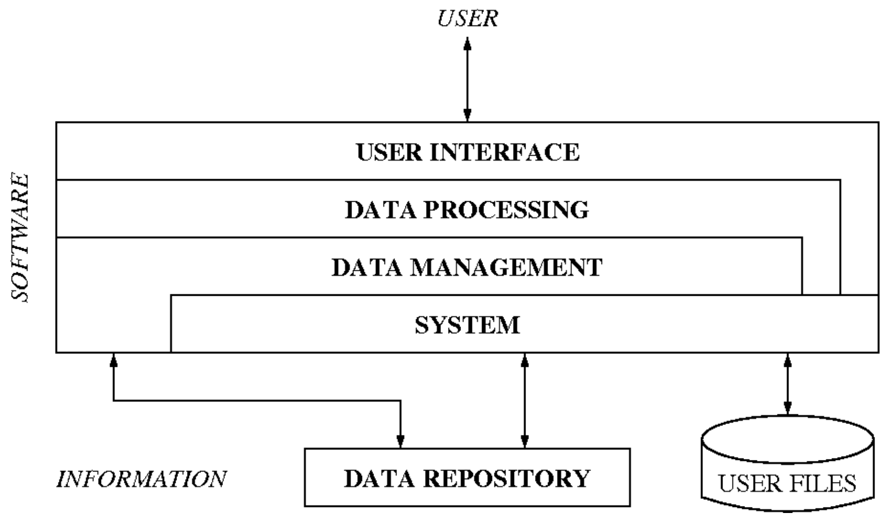

Fig 2. The CXCDS top-level architecture. Communication between layers is through well-defined APIs.

The fundamental design characteristics of the data system architecture were planned in detail to meet the needs of project that was fairly complex and had many variables. The characteristics that guided the design were modularity, flexibility, compatibility, extensibility, and reuse of off-the-shelf (OTS) software whenever possible. We also worked up front to address maintainability, performance, and for some parts of the system, portability. Many of these are mutual in that one tends to enhance others (*e.g.*, modularity and flexibility enhance extensibility). Others conflict in that they cannot be simultaneously maximized within a given component. Consequently, each characteristic was considered a design challenge in the face of others. The architectural solutions represented tradeoff decisions made between those characteristics in order to provide the maximum level of service through a design realizable within current technology and project resources.

As an example, several ways we met the modularity criteria was in designing software components and services that fit within the layered architecture. Communication between layers is through well-defined APIs. By isolating classes of functionality we allow software to specialize toward a single functional objective. APIs allow software components within a layer to change without affecting other components with which they communicate as long as relevant APIs stay the same. We have also developed libraries of common-use functions (*e.g.*, I/O and coordinate transforms), and provide them as a set of services available to all components of the system. Clearly, defined APIs and modular code may not lead to an open and independent architecture. For example, we inadvertently introduced circular dependencies in some libraries and as a result could not release parts of CIAO software independent of the whole system. With release of CIAO 4.0 (December 2007) we worked to identify and break those dependencies, and can now release parts of CIAO independently from the entire system.

Many of the tools included in CIAO are shared with SDP. A shared toolset has the advantage of guaranteeing consistent results, and means that tools developed specifically for data analysis can be directly incorporated into SDP if needs so dictate. Sharing results in a smaller code base for maintenance, but does create release challenges since a tool change can force a release of both SDP and CIAO.

We designed and developed the pre-launch CXCDS using the waterfall[8] method. With our first real system test at XRCF, and with the fundamental system components in place, we added elements of an iterative or spiral process. The stepwise approach to development includes tests at each increment to help ensure that we are modifying and enhancing the system to the requirements while maintaining the overall functional and architectural elements of the system.

4. DATA SYSTEM MAINTENANCE

CXCDS software maintenance efforts are directly driven from several directions. Changes to spacecraft and instrument operations, improvements to calibration algorithms, and revisions to permitted observations typically require updates to both the forward and return thread software, and rapid turnaround may be required. Enhancements to the data analysis system are typically driven by requests from users, and as a result of algorithmic enhancements that will increase the science return from the mission. Migration of operational hardware and software (operating systems, compilers, OTS) as components age and reach end-of-life typically require maintenance at the infrastructure level of the system, although in these cases rapid turnaround is usually not needed. Similar maintenance tasks are associated with the migration of user data analysis platforms, although these are less predictable than is the case for operation components.

4.1 Requirements

The software group interacts with various CXC teams to establish priorities and requirements. The Science Data Systems (SDS) team works with instrument and spacecraft experts to update science instrument pipeline and data analysis algorithms to accommodate changes to calibrations, spacecraft operations, and observing strategies. CIAO science enhancements are identified and prioritized by SDS, who maintain a dialog with the user community through the Chandra Users' Committee and by individual interactions. Updated requirements for mission planning and proposal support software are provided by the CXC Mission Planning team and by the Chandra Director's Office. The Science Operations Team maintains requirements for aspect camera pipeline processing, telemetry limit monitoring (real-time and dump), and trends analysis software. The CXCDS operations groups provide requests for enhancements to archive and data processing operations interfaces needed to support their operations. The CXCDS end-to-end scientist maintains telemetry and non-science instrument software requirements.

4.2 Development

The CXCDS follows a model of development that includes frequent interactions between scientists and software engineers to refine and further develop initial requirements. Feedback from the science testing from several internal releases early in the process helps iron out misunderstandings in the detailed requirements, allows review of more intangible requirements like look and feel, and saves time overall. These interactions include e-mail exchanges, meetings, and requirements reviews. All of the steps are documented via e-mail archives, meeting notes kept in our internal documentation pages, and formal requirement documents whenever appropriate. Involvement of the various requirements leads in this process ensures scientific accuracy and utility.

Design follows when the requirements have reached sufficient maturity. Design reviews are held on substantial projects within the development team, and include the relevant scientists whenever appropriate. Design is documented by developers and managed by the software team leaders in unit development folders. Each task is assigned a directory where development and maintenance information are recorded. They include requirements, design, test plans, regression results, and notes. Pointers are provided from internal web pages for easy group access and review.

Software is developed on agreed upon schedules. An important component is to provide several internal releases (or drops) of the task before completion. The drop is available for science review and developers respond to test feedback as it is received. Any requirements clarifications or software problems are addressed within the next drop cycle. With the subsequent development iteration, we revisit the requirements and design phases to gather more details for the next functional set of scheduled enhancements.

4.3 Test

The CXCDS team is responsible for ensuring stability and performance of software releases through rigorous unit, regression, and integration testing to ensure that APIs between components are maintained and no unexpected regressions occur at the system level.

Unit tests are performed and documented by the developer and validated by the team leader. The requirement scientists verify these tests and are responsible for independent testing and reports for significant enhancements. These science unit tests are added to a regression test suite maintained by the science team.

Integration consists of an analysis of the release notes, and subsequently planning tests that need to be run to verify interfaces and correct results. We start with mini-tests to iron out the interface issues on short data sets, progress to more thorough standard tests of the system, and when required, run specialized functional tests. The integration lead verifies that software is operational at the system level. Test results are passed back to team leaders to verify at the sub-system level. Each team leader decides if a comparison test with previous results is appropriate, or an independent check of the data is required to verify the software changes. The requirements scientists are consulted if needed.

Portability tests are performed on the CIAO data analysis system that is distributed to users. Task performance and results are first verified on the development platform, followed by the portability platforms.

Packaged test scripts and data released with CIAO, known as *smoke tests*, allow users to verify that their copy of the software was properly built and installed. As part of our release package testing, we run the smoke tests on internal and external hardware with a variety of supported operating systems to verify the build and installation procedures. Each test in the test suite reports a "PASS" or "FAIL," and a log is provided with details of the execution. Smoke tests are also part of the Configuration Management (CM) weekly build and rollout process across platforms. CM makes new releases available for internal use and further testing when all of the smoke tests pass.

SDS and CXCDS scientists ensure that the pipelines, data products, and analysis software are scientifically accurate and meet the needs of their respective user communities. Acceptance testing is planned and executed by the operational teams for operational releases; unit, regression, and download testing is performed by SDS prior to a data analysis (CIAO) release. Reports are generated that become part of the software release documentation.

4.4 Release

Releases are planned and scheduled as part of the high level data center schedule, which is approved by upper management and by the NASA project. Releases usually have "drivers" and often carry along other work at lower priority completed in the time period since the last release. Depending on the time criticality of the driver, a risk assessment is performed, and items are included in the release (or not) based on their risk against the schedule.

The schedule is set based on estimates of completion of the driver tasks. When implementation of a project involves cross group dependencies, we also develop a "lien" list that identifies who needs to do what in order for the task to be completed. The lien list is briefed at high-level management meetings so that it is clear at the project level what the dependencies are, and their status, leading to the completion of the project.

We support major releases and patch releases. Major releases usually include changes to library software and require a full build of the system. They are often planned well in advance and are not a response to an immediate observatory issue. Patch releases only affect specific tools or applications in the system, and require a partial rebuild of the system.

4.5 Configuration Management

CM builds our system on a daily basis as a check on the integrity of recent code changes, runs automated build tests at the subsystem level, and supports software releases for operations and data analysis on a regular basis. Software maintenance includes management of a large number of OTS (~95 total) components used throughout the system.

CM is essential for a complex software development environment like ours. We use a commercial system (ClearCase), configured for our environment internally, and augmented by procedures for the software developers and teams. ClearCase uses a labeling system to identify software available for builds. We support nightly, weekly, and release builds of the operations and data analysis systems.

Each software team is free to include their code in the nightly builds at any time. The goal of these builds is to check that interfaces to other software are functional. Code that is labeled and built for the weekly builds is usually software that has been targeted for the next release. Release code is labeled with a specific identifier prior to a code freeze, and a dedicated build is completed and made available to integration.

5. INFORMATION MANAGEMENT

5.1 Documentation

Internal web pages are maintained with procedures, schedules, reports, project and other information helpful to development and management.

User documentation includes help files associated with each tool or application in the system. Help on specific tools or topics are available at the command line and also from public web pages. The developers provide an initial version of the documentation to the science team, who then migrate the information into language that scientific users can more easily digest, and also provide further examples targeted at those users. The science team also documents recipes, or "threads," which users can follow step-by-step to understand and apply a data analysis function by example.

User workshops and demos are sponsored periodically by the science team and supported by the development staff.

5.2 Bug Tracking

All bugs and enhancement requests are logged in a tracking system that was developed internally. The internal system is targeted at our needs, and includes pointers to relevant parts of the email archive where the issues were reported and discussed. Bugs and enhancements are prioritized and reviewed at several points during the planning and development process.

5.3 Helpdesk

A *helpdesk* team, operated by SDS staff scientists, with backup from the software team, responds to user queries and issues. The Helpdesk team maintains a database and "frequently-asked-questions" list to support their efforts, allowing most user queries to be addressed quickly. The science and software teams support the resolution of difficult problems usually via email. Bugs or enhancement requests resulting from helpdesk interactions are forwarded to the software team for entry in the software bug database.

6. CHALLENGES

Schedule, resources, and development methodologies are not always symbiotic. We challenge ourselves to be true to our model of development while meeting the project needs within the team of developers, scientists, and computer specialists upon which the software system relies. Our process does not follow the definition of one model but takes from several models, along with our historical experience, to define a framework that works effectively for us. From the waterfall model we have defined project phases (requirements, design, development, and test) and follow them in that order. We diverge from the waterfall model in that we do not complete each phase before moving to the next. We spend up-front time understanding the entire specification so that the early design considers the final goal. We accept written specifications, prototypes, email messages, or a series face-to-face meetings to convey science requirements. We choose the programming or scripting language, understand platform and runtime needs, and plan our data structures and library use during the initial design phase. We borrow from the spiral model by assessing risks (or in our case liens) on the project and develop our software as a series of design efforts and releases (or drops) of functional units internally. The test and feedback loop and functional definition of each drop are defined and scheduled up front. Our historic contribution borrows from years of science and development experience. Certain standards are defined within our project and community (*e.g*., HEASARC-compliant FITS file headers) and we assure that those standards are met. Some of the documentation and process descriptions that we follow are designed to support the fact that our group is distributed across 3 geographically distinct locations in Cambridge, MA. We meet every few weeks but do not interact daily. Well defined development processes help manage the distance.

We are currently nearing completion of the development of the Chandra Source Catalog[9] (CSC) processing system and archive. Production start is planned for fall 2008. Our development process was applied at the system level as well as the module level to define the goals that the system would meet, as a series of internal releases and system tests. These releases allowed us to gain confidence in our new system as we approach completion of the catalog production release by running tests with increasing science and system functionality with each internal drop. Problems that were identified in a drop test were rolled into the subsequent release, along with the increased functionality we had scheduled. We traded off driver tasks with lesser priority tasks to make the decision as to whether a schedule slip was needed or whether the functionality could be moved to a later release.

The data system team has met its operations challenges with much effort and refinement of our software system both prior to launch and during the mission. We (scientists and programmers) first had to learn about the instrument and spacecraft operations. We had to learn about processing systems and what it meant to run continuously day and night. We had to produce data high-quality products that would provide sensible input to a data analysis system and endure in an archive over a long period time. We had to learn about data volume, hardware, networks and the capacity to move and store data efficiently. We had to learn about data anomalies that come from how observations are planned that are out of the ordinary or glitches that occur in passing data back to the observatory from the spacecraft communication systems. We had to address speed and decide if the solution was in new hardware or better programming approaches, and we had to learn that processing management had to be robust in order to run 24/7 while at the same time provide flexibility for easy configuration of special cases or extensions to the original system. We had to re-design some components of our system because despite our efforts the long evolution of some specifications and software made them unwieldy. We re-designed several components to change choices of OTS software or to keep current with technology.

The team is dedicated to our process and methods of development for the CXCDS. As a result of these efforts, data from completed observations are provided to the observer ~1 day after their completion on the satellite. CIAO has won positive feedback from the Chandra user community for the unprecedented flexibility and generality of the software and the thoroughness of the documentation. We continue to meet the needs of an Observatory and are doing our part to contribute to the astrophysics discoveries of Chandra.

ACKNOWLEDGEMENTS

Support for development of the CXCDS is provided by the National Aeronautics and Space Administration through the Chandra X-ray Center, which is operated by the Smithsonian Astrophysical Observatory for and on behalf of the National Aeronautics and Space Administration under contract NAS 8-03060.

REFERENCES

[1] Evans, J. D., et al., “The Chandra X-ray Center data system: supporting the mission of the Chandra X-ray Observatory,” Proc. SPIE 6270, pp. 62700N (2006).
[2] Boehm, B., “A Spiral Model of Software Development and Enhancement,” Computer 21, 61–72 (1988).
[3] Evans, I. N., et al., “The Chandra X-ray Observatory data processing system,” Proc. SPIE 6270, pp. 62701U (2006).
[4] Smithsonian Astrophysical Observatory, “Data System Software Design,” AMO-2410 Rev. B, (1996).
[5] Rots, A. H., “The Chandra Data Archive,” ASP Conf. Ser. 216, 184–186 (2000).
[6] Graessle, D. E., et al., “The Chandra X-ray Observatory calibration database (CalDB): building, planning, and improving,” Proc. SPIE 6270, pp. 62791X (2006).
[7] Fruscione, A., et al., “CIAO: Chandra’s data analysis system,” Proc. SPIE 6270, pp. 62701V (2006).
[8] Royce, W. W., “Managing the Development of Large Software Systems: Concepts and Techniques,” Proc. IEEE WESCON 26, 1–9 (1970).
[9] Evans, I. N., et al., “Planning and developing the Chandra Source Catalog,” (these proceedings).