5.2 Aspects of the ORPG Software Development Process

Michael Jain[1], Richard Adams[2], Hoyt Burcham[2], Eddie Forren[2], Gary Gookin[3], Zhongqi Jing[2],
Aamir Nawaz[3], David Priegnitz[2], Steve Smith[4], Dan Suppes[2], John Thompson[2]
([1] NSSL, [2] NSSL/CIMMS, [3] ROC/MATCOM, [4] ROC)

1.0 INTRODUCTION

The Open Systems Radar Product Generator (ORPG) Project was established in early FY96 with the National Severe Storms Laboratory (NSSL) responsible for software development. (Saffle and Johnson, 1997; Jain, 1997)

As of October 2001, System testing for the DOD and redundant (both FAA and NWS) ORPG configurations are nearing completion. The standard NWS ORPG configuration has completed System testing and Beta testing at three sites (Norman, Topeka, and Atlanta). Deployment of the standard NWS configuration has been underway with over two dozen operational sites being successfully deployed. Beta testing of the DOD and FAA configurations are scheduled to begin soon with deployment of these sites to follow the NWS installations. Deployment of all systems is currently scheduled to be completed by the end of FY02.

Reports from the NWS Beta test sites as well as recent feedback from the initial deployment sites have been extremely positive. Increased reliability and ease-of-use have been cited as two of the many positive attributes of the ORPG system. Based on these reports from the field, as well as test results from the formal System and Beta testing performed by the Radar Operations Center (ROC) for NEXRAD, it is quite apparent the ORPG software development effort has been successful in producing a high quality product that meets and exceeds the current operational requirements of the field. In addition, the architecture and design of the system is such that it will enable the ORPG to continually evolve to meet the ever changing requirements of it's users.

An analysis of other software development efforts have shown that a significant number of projects often end in failure, either through outright cancellation of the project or through delivery of defective or functionally deficient software (Johnson, 1999). The ORPG Project was certainly faced with numerous challenges that could have easily resulted in failure. However, the development was able to succeed in spite of these challenges. This paper will discuss some of the aspects of the software development approach of the ORPG that we believe contributed to the project's success.

2.0 PROJECT CHALLENGES

As with all projects, the ORPG development was faced with its own set of challenges. Some of these, in a general sense, are challenges common to many software development projects, such as: increasing scope, staffing, defining development processes, changing requirements, aggressive schedules, and program/project management structure.

Organizationally, neither the NSSL nor the ROC had ever attempted a development project of this type or magnitude. A significant learning curve was required of both organizations to accommodate the project management details of this type of development as well as how to operate within the program management structure established for this project. Development processes needed to be established that would work within the framework and constraints of the project.

Initially, the NSSL was tasked to produce a proof-of-concept and then, working with the ROC Software Engineering in the lead, aid in completing the ORPG software development. However, obligations to produce and support other NEXRAD legacy builds prevented the ROC from assuming this role. Because of this, the scope of NSSL's responsibility increased immensely when we were asked to complete the production development of the ORPG software. This significantly impacted staffing requirements, deliverable requirements, and level of performance.

Incomplete or ill-defined requirements are a common problem with many software development projects; ORPG was no exception. The documented requirements, available to the development team, did not fully describe what was actually expected of the final system. Processes to supplement the incomplete requirements documentation had to be developed.

Even when faced with these as well as other challenges, the project was still successful in terms of delivering a quality product that meets the operational needs of the end users as well as meeting the requirements to allow for the future growth of the system. We believe this success is attributed to the quality of the development team and its ability to effectively respond to external change. Resiliency and a willingness to adapt to the many and evolving challenges of the project proved to be an effective method of achieving success.

3.0 METHODOLOGY

In recent years, methodologies described as being "light-weight", "agile", or "adaptive" have emerged in the software development community (Fowler and Highsmith, 2001; Highsmith, 2000; Fowler, 2000; Berinato, 2001). These are in contrast to the more traditional "heavy" methodologies.

The ORPG software development team did not explicitly or consciously follow an "agile" methodology, since these methodologies have only recently been defined. However, an article in Software Development magazine (Fowler and Highsmith, 2001) describes some of the principles of Agile Software Development. A review of these principles showed many of these are congruent with the development principles adopted and practiced by the team in the development of the ORPG software. Independent affirmation, in a sense, that our development concepts and processes were consistent with some of current industry's software development leaders.

We will discuss how the ORPG development team applied these principles over the course of the development. Due to space limitations, only a subset of the principles will be discussed here. Quoted, italicized text are excerpts from the Fowler and Highsmith article.

Corresponding author address: Michael Jain, National Severe Storms Laboratory, 1313 Halley Circle, Norman, OK 73069-8480; e-mail <Michael.Jain@noaa.gov>

3.1 *"Welcome changing requirements, even late in development" ... "Rather than resist change, the agile approach strives to accommodate it as easily and efficiently as possible, while maintaining an awareness of its consequences."*

Practically all projects are faced with some degree of changing requirements over the course of development. Not only did the ORPG Project need to deal with this issue but also with the issue of there being a substantial number of undocumented and/or poorly documented software requirements.

Legacy RPG requirements documentation was provided and used as guidance early in the project and utilized through most of the development. These requirements documents, in themselves, were incomplete, lacking important detailed requirements and procedural information of the legacy system as well as containing some inaccuracies. Clearly, the legacy documentation were insufficient in describing the RPG for the purpose of building a new system that would possess much of the same legacy behaviors. A draft System Specification containing high-level requirements, including some of the new "open systems" requirements, was available mid-way through the project. In addition, requirements imposed on the software by the system design and specific hardware selections were not available until the latest stages of the software development.

The software development team was able to supplement the missing requirements through direct and indirect methods. Direct access of information was obtained by tapping "legacy expert" knowledge. Since this was a legacy enhancement-type project, the software development team included a "legacy expert". This person had been involved in the original contractor development of the RPG and later became a member of the ROC software maintenance staff. This individual served the team as the primary source of expert knowledge regarding the design, implementation, and behavior of the legacy RPG. This proved invaluable in filling in many of the requirements gaps. Other legacy experts were consulted for selected functional areas.

Indirect requirement supplementation came in the form of discrepancy reports or what were termed as "issues". Over the course of development, frequent software releases to the ROC development and test teams were provided. They in turn tested and exercised the software providing feedback to the software development team in the form of "issues" which documented software coding defects or when certain requirements were not implemented or implemented improperly. This became the primary mechanism for which the undocumented requirements were communicated and ambiguous or incomplete requirements were clarified. As these requirements issues were submitted, the development team would assess and incorporate them into the system.

3.2 *"Deliver working software frequently, from a couple of weeks to a couple of months, with preference for the shorter timescale"*

A very effective process the team implemented was an automated, nightly integration build of the ORPG application software and supporting infrastructure. Each night the software would be built incorporating software updates introduced during the previous day. Limited regression testing would occur the next day to determine the viability of the build. Integration errors were detected quickly and developers were able to address problems promptly.

Although software deliveries were not provided to the end user, they were provided to the ROC for testing. It was very beneficial, particularly later in the project, to make the software available to the ROC on a frequent basis. The nightly builds allowed the ROC to pull software builds as needed, with updates as short as one week during intense phases of testing later in the development. These software builds provided ever increasing functionality as well as addressing previously submitted "issues", correcting software defects and incorporating the implementation of previously undocumented requirements.

Providing frequent software deliveries allowed the ROC to test and evaluate the system and hardware designs and implementation. It also expanded the overall scope of software testing from development testing to somewhat "independent" testers who had specific knowledge and expertise on the ORPG system/hardware design/implementation as well as testers who had extensive legacy RPG knowledge, and provided an opportunity to incrementally refine user interface components and general ORPG behaviors through continual feedback from ROC staff.

3.3 *"Build projects around motivated individuals, give them the environment and support they need and trust them to get the job done" ... "in the end, it's the people who make the difference between success and failure" ... "For many people, trust is the hardest thing to give. ... managers must trust their staff to make the decisions about things they're paid to know about."*

It was well understood when this project began, and certainly as the scope expanded, the NSSL did not have the staff with the proper background and experience level required to make this project a success. For this reason, NSSL immediately began recruiting and constructing the team which eventually became an integrated team of ten NSSL, NSSL/CIMMS, ROC, and ROC/MATCOM staff. Care was taken in assembling the team which resulted in a successful, synergistic mix of individuals having backgrounds in commercial software development (both private and military contracting), academic and research development, meteorology and meteorological radar development, distributed processing, graphical user interface (GUI) development, software documentation, configuration management, software maintenance, software testing, and knowledge of the legacy RPG. And, as important as the technical expertise, the team was composed of highly motivated people whose attitudes and personalities allowed them to work very well together and enjoy the challenges of this project. This team, and subsequently the project, benefitted from an extremely high retention rate, losing only two team members in six years.

Program management provided the development resources (computers, software development tools, etc). The NSSL provided a positive and creative development environment where innovation and creativity were encouraged and supported. Development emphasis was placed on producing an improved product of high quality that met the needs of the end users, which not only included the operational field but also those who would also contribute to future development and maintenance of the system.

3.4 *"The most efficient and effective method of conveying information with and within a development team is face-to-face conversation" ... "the distinction between agile and document-centric methodologies is not one of extensive documentation versus no documentation; rather a differing concept of the*

blend of documentation and conversation required to elicit understanding."

The entire software development team resided within the same work area, increasing the opportunities of face-to-face transfer of information. Developer-to-developer communication was encouraged and many informal design discussions and peer reviews occurred. Team meetings, especially during the development phase proved useful in keeping the entire team attuned to team-wide activities. One result of this level of communication was the team was self correcting; if a design or implementation was determined to be deviating too far from the established architecture, the team nudged the design/implementation back within accepted norms.

During development, an emphasis was placed on documenting application programming interfaces (API) or common libraries designed to be shared between developers. The concept behind producing documentation during development was to communicate crucial information required between developers for using shared software or interfacing with other software components of the system. This documentation primarily took the form of Unix man pages. There was a tendency to not perform certain forms of documentation due to the ongoing evolution of the system and having to continually update and maintain the documentation as well as the software.

One-on-one discussions between legacy staff and ORPG software developers as well as technical interchange meetings involving several staff members were very effective in gathering the undocumented requirements and aided in gaining a better understanding of the expected behaviors of the system.

3.5 *"Working software is the primary measure of progress" … "Working software is the measure of progress because there's no other way of capturing the subtleties of the req's: Documents and diagrams are too abstract to let the user 'kick the tires'".*

An initial goal of the development team was to establish a working thread as soon as possible. Targeted first, were what were believed to be the more critical and high risk aspects of the software to be developed. Among these included the porting of the legacy FORTRAN algorithms and product generators from the monolithic, tightly coupled architecture into the open, loosely coupled architecture of the ORPG, mastering the legacy communication protocols and requirements to the RDA and product users, the generation and distribution of the NEXRAD products, control of the RDA, beginning the development and refinement of the graphical user interface for the new ORPG, and the development of an infrastructure that would support an open, expandable, and distributed system. These major milestones were accomplished very early on in the project, thus eliminating the higher risk aspects of the software development. As indicated earlier, having working software and knowledgeable people exercise it uncovered many undefined requirements that would have otherwise been missed and possibly left out of the final product.

3.6 *"Continuos attention to technical excellence and good design enhances agility" … "Agile approaches emphasize quality of design, because design quality is essential to maintaining agility … agile processes assume and encourage the alteration of requirements while the code is being written. As such, design cannot be a purely up-front activity to be completed before construction. Instead, design is a continuous activity that's performed throughout the project."*

The initial year of the project was spent on design and verifying certain design concepts through pathfinding. The software development team concentrated on discussing the requirements and decomposing a complex system into smaller, simpler components. Iterative discussions, primarily around a whiteboard, resulted in the initial high level design concepts of the ORPG. Essential design concepts were captured in text and with simple diagraming tools. These concepts were reported in various conference papers as well as at routine ORPG software build reviews held for ORPG stakeholders.

Over the course of the project, the software design of specific components of the system were allowed to be refined as new requirements were introduced or better designs were discovered. Instead of continuing down a path of a poor design, the team was allowed to discard the poor design and replace it with a higher quality design.

The GUI design underwent an extended, iterative refinement. At an early stage, a human factors team was involved in providing design input for the GUI. Later, the ROC organized several end user reviews that contributed to the design. In the final stages, refinement of the GUI occurred through the submission of issues and/or discussions sparked by these issues.

3.7 *"The best architectures, requirements and designs emerge from self-organizing teams" … "the best designs (architectures, requirements) emerge from iterative development and use rather than from early plans."*

A highly compatible and technically competent team was assembled for the ORPG Project. They were allowed to be self-organizing and constrained by few process rules. This resulted in a highly creative, innovative, and productive team. An iterative development of the ORPG was conducted where working software was provided to expert test teams, who in turn provided valuable feedback to the software development team. The ORPG software architecture and design were allowed to evolve as needed, resulting in very robust, resilient, and user effective system.

4. CONCLUSION

We acknowledge, as many others have, there are no "silver bullets", no specific techniques, no specific sets of procedures or processes that will assure success of a software development project. Each project, each situation, is unique. We found for the ORPG Project, the principles of an "agile" or "adaptive"-type development worked well and believe this approach can be successfully applied to many other projects. To enhance the probability of success, managers responsible for development projects should continually recognize the various challenges of their particular project and be willing to adapt their processes and procedures to meet these challenges.

5. ACKNOWLEDGMENTS

The authors would like to acknowledge the Office of Science and Technology (OS&T)/NEXRAD Product Improvement (NPI) for funding this project. We would like to thank ROC staff for their contributions towards supporting the software development. And finally we would like to thank NSSL management for their support and encouragement.

6. REFERENCES

Available upon request from the authors.