Experience Report

Can you develop a bug-free mission-critical system on time? This author's customized approach delivered a highly sensitive missile warning system to Cheyenne Mountain within budget and meeting all the customer requirements.

Process Control for Error-Free Software: A Software Success Story

Buford D. Tackett and Buddy Van Doren, ITT Industries

Software development projects typically sail through much of their life cycle and miss the early warnings of the high-risk issues that eventually break or damage them.[1] To reduce the risk of producing a poor-quality, late product, we created the State-Based Development Process, an approach based on systematic design and peer review.

A small team at ITT Industries used this development process to create a critical system under high schedule and operational pressure for the North American Aerospace Defense Command, a binational command involving the US and Canada that provides missile and air attack warnings and defense against air attacks. The approach was a powerful synthesis of people, process, and technology that yielded tangible improvements in software engineering.

0740-7459/99/$10.00 © 1999 IEEE

Deep Inside the Mountain

Completed in 1966, the Cheyenne Mountain Operations Center is a 15-building complex that houses NORAD, the US Space Command, and the Air Force Space Command. The complex is actually inside the mountain, 1,750 feet below its surface. In 1981, the Pentagon started the Cheyenne Mountain Upgrade, a $968 million, six-year project to update the facility's computer systems. By 1994, the project was a decade late and $1 billion over budget.

In the shadow of this quintessential "runaway" software project,[1] NORAD and the Airforce Space Command undertook another software project to develop an automated tracking and monitoring system (Atams) in April 1995, as highlighted in *Scientific American*.[2] The existing system forced a team of technicians to scan more than 20 monitors for a variety of complex alerts and warnings. Atams needed to control an "entire network with two monitors and a simple, consistent interface."[2] The first fully operational phase of Atams had to be deployed within one year.

The Air Force boldly decided to waive its standard approach and allow the development team to use a new process derived from the Harlan Mills Cleanroom methodology.[3] Implementing a system robust enough for this command and control environment was not easy, since it consisted of two "hot/shadow" configurations in Colorado and Nebraska, processing over 50 real-time messages and over 35 real-time, interactive status displays to the NORAD crews.

Amazingly, Atams was deployed on time and within budget, to the user's delight. The State-Based Process used to develop the Atams project was central to this success.

The State-Based Process

We designed the State-Based Process to improve productivity, quality, process, and overall risk management to produce a software-intensive, operational product. Our approach was evolutionary and incremental; each increment was fully executable, testable, and demonstrable to the user for observation and approval. This addressed a common problem in software projects: the user is rarely involved in product development.

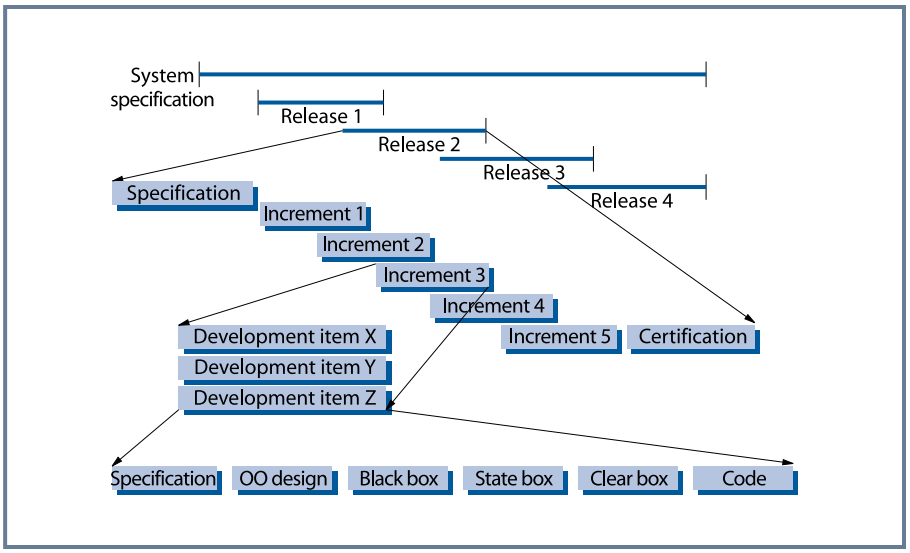


Figure 1. The evolutionary, incremental approach to developing software.

In our approach, each incremental release delivers functionality, and work on increments overlaps. A software item under development is treated as a development item, or DI, undergoing a series of state-to-state transitions with a prescribed set of operations acting on it within each state. Each DI within an increment proceeds through the process as a separately tracked entity (although dependencies that exist between DI's will influence the sequence and progression through the process). Figure 1 illustrates this approach.

We first identified what we believed were the major states through which a DI must transition. Then we identified the operations within each state needed to transform the DI into the form necessary to transition to the next state. We realized, however, that the State-Based process must be simple enough to be captured graphically on a single page, so that the team could reference it without a large, complex document. Most likely the team would not successfully use the process otherwise.

The resulting process thus uses a state transition diagram to describe the states through which each DI must pass to reach its final state. In this case, a DI transitions through 12 states, beginning with State 1, Specification Review, and ending with State 12, Certification. State 13 is the software's operational state. Figure 2 gives an overview of the State-Based Process.

We developed a process guide to detail the process and management activities that must occur within each state and each transition. As a DI enters a state, the process guidelines define what activities

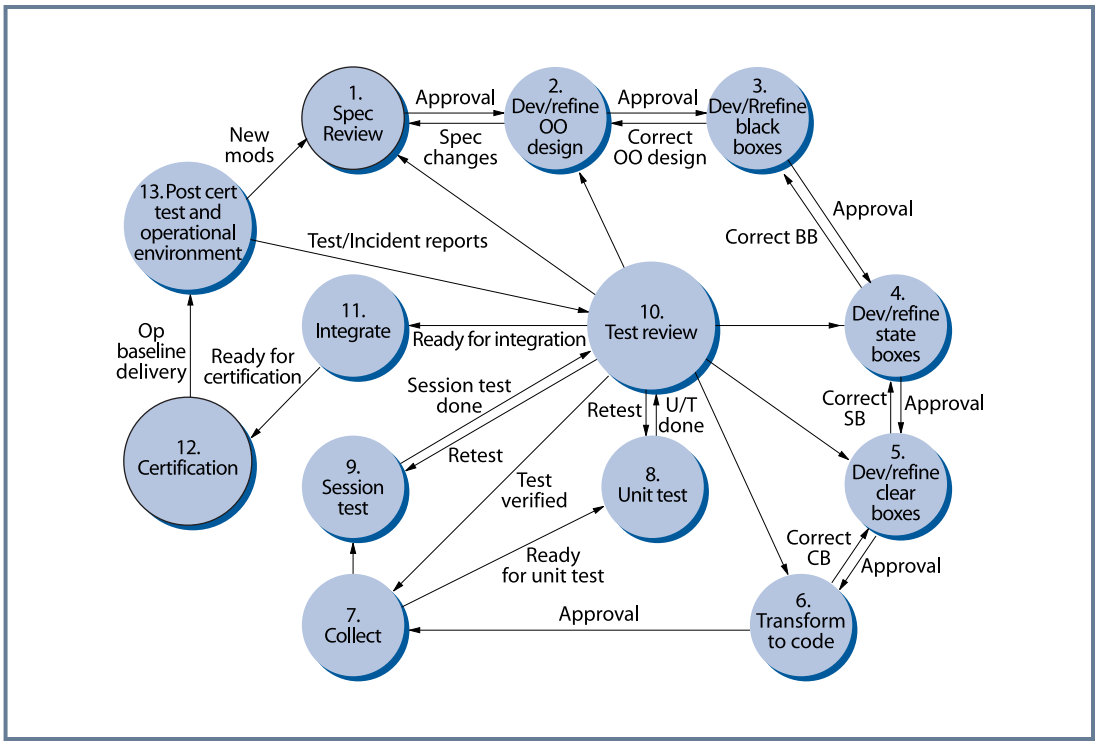


Figure 2. A summary view of the State-Based Process.

must be performed on it and what must be achieved to transition it to the next state.

Team walkthroughs constitute the major activity in which the DI's state is examined and verified, and transition decisions are made. Our objective was to "design a little" and then verify the correctness of the changes. When the lead engineer for a DI finished the design changes, he then delivered those changes to a team of reviewers and announced the walkthrough time and place. The reviewers individually inspected the changes and brought their findings to the walkthrough, which the lead developer always chaired. The process guide describes the roles of the lead engineer and the reviewers for each DI throughout its life cycle. The reviewers and the lead engineer formed a development team that had technical and emotional ownership of the DI. Defects were the responsibility of the entire team, not the lead developer's alone.

Walkthroughs

Walkthroughs were the heart of the process; they provided clear and organized project management. We recognized from the beginning, however, that it was important to maximize the effectiveness of a walkthrough and minimize its overhead. We faced a constant battle to get the developers to schedule and conduct enough walkthroughs to keep the size of the review material reasonable. They tended to go too long between reviews and include too much in each review. In other words, there was a constant creep from "design a little" to "design a lot." Management needed to mitigate this and ensure the process itself didn't create obstacles that encourage developers to delay the walkthrough process. For this reason, we kept the roles and requirements for the walkthrough simple.

Each walkthrough had a well-defined list of entry and exit requirements. In most cases, the exit requirements consisted of walkthrough minutes, an approval to proceed, and the actual software development item products. Developers produced the minutes using a template from the Process Guide, and included metrics collection, decisions, and action items for every defect noted during the walkthrough. The approval to proceed determined whether or not a transition was made out of the current state and to which state the object proceeded. The process allowed customization so that certain states could be skipped or, if significant problems were encountered, the approval to proceed could direct the process to return to an earlier state.

Within each state, the guidelines outlined an initial, intermediate, and final walkthrough. Multiple walkthroughs were held within a state before the approval to proceed directed a state transition. Figure 3 shows this internal state walkthrough process.

The lead developer was responsible for producing the announcement and minutes for each walkthrough. Templates for both were provided online. The lead distributed the announcement along with

the documents required for the review. After the minutes were produced, team members received an e-mail message that included a file name and location of the minutes and the names of everyone who had action items.

The minutes consisted of three major areas: metrics, general comments, and action items. The action items were the main reason for the walkthrough, since its objective was to uncover every defect and ensure they are corrected. When problems were uncovered, each defect had an action item with a named individual given responsibility for completing it. Action items were tracked to closure via the walkthrough itself. The lead developer was responsible for reviewing all previous action items at the beginning of the next walkthrough. Outstanding action items could prevent approval to proceed or, by explicit decision, be carried forward to the next walkthrough, with annotation in the minutes.

Inspections

Each walkthrough included a dialogue focused on the results of each reviewer's individual inspection of the DI. Inspection was essential preparation for the walkthrough: the success of the walkthrough depended on the reviewers' professional and conscientious examination of the DI. If a reviewer had not had sufficient time to inspect the materials, we postponed the walkthrough. The objective was not to simply check a box on a form, but to uncover every defect. Reviewers were assigned at the start of the process and remained the same throughout; this developed a team ownership that was evident when we uncovered defects—the attitude was, "how did *we* miss that?" We never had a problem with pride of authorship or damaged egos.

The process also yielded an exciting technical benefit: it raised junior developers' expertise level. When an experienced developer reviewed a junior developer's work, after several walkthroughs the junior developer began to pick up on the expert advice. He began to ask the same kinds of questions in other walkthroughs in which he was the reviewer.

Metrics

Metrics were collected throughout the process to assess status; they included walkthrough duration, number of participants, external inspection time, pre- and post-walkthrough overhead, and major and minor defects or issues uncovered. We defined major defects as those that could negatively affect the operational system. Minor defects included every other problem. At the end of each walkthrough, both schedule status and technical status of the DI were also assessed. This status was a major input for management briefings.

Additionally, our metrics captured the return on investment in terms of the resources required to conduct walkthroughs and the number of defects uncovered. At the beginning of each walkthrough, every reviewer was polled on the amount of time they spent on inspections. The lead developer had the authority to cancel any walkthrough if insufficient review time had been spent.

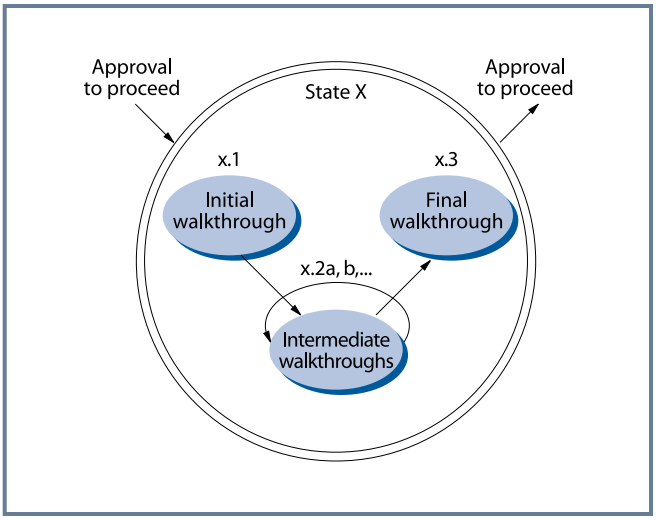


Figure 3. Internal state walkthrough process.

State notes

A set of state notes associated with each state assisted the lead developer and team as they proceeded through the process. The notes were continually revised. Referencing the state notes as they entered a state gave the developers valuable insight into past lessons learned, suggestions, questions to ask, hints, critical items to check, and any other information collected during previous activity in this state. The team was responsible for identifying items to be added to the state notes, with the lead developer taking the update action.

Keys to Success

The Atams development team used the State-Based Process with great success, and it continues to undergo refinement and improvement. The State-Based Process was designed to allow management and users the ability to see inside the product during development. Both management and the development team would be reluctant to develop software without it. Management was especially

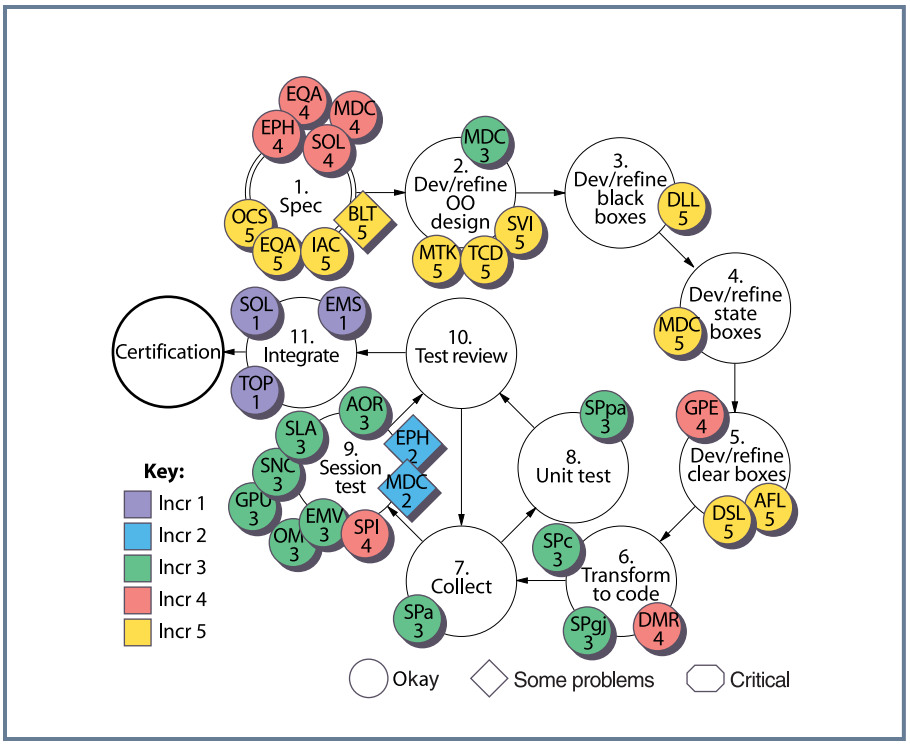


Figure 4. Management status view of a development item.

pleased that they could query any developer about the status of his work and receive a clear answer on the state he was in, the state he was moving toward, and the schedule status.

User involvement

The users were heavily involved throughout the life-cycle development. Eliciting requirements is one of the most difficult tasks in developing systems, and arriving at a true requirement set is challenging because it is difficult to express in human language the behavior of a computer. Codification of human thought into software, even using modern tools such as use cases, is fraught with difficulty and prone to error.

We depended on continuous, face-to-face user involvement from the specifications state through the design state. The incremental approach allowed us to gain user feedback early and often. We frequently asked the users' opinions on interface and functionality to ensure we fulfilled the system requirements. In the project's first few weeks, we developed user interfaces using a display builder tool. In an early walkthrough, 12 operational crew members voted on screen display colors. They participated in our walkthroughs daily until the screens were exactly what they wanted. We were not afraid of their input throughout the project—we asked for it repeatedly. No one saw this as a late requirements change, because it allowed us to produce a high-quality product. When we delivered the software, the users were not surprised by anything because they had been an essential part of the development team.

Management benefits

Project management was greatly improved because the managers could track the developers' progress and see "inside" the software product. A major objective driving our process definition approach was to reduce overall risk by wedding the management process directly to the technical process.

Figure 4 shows a typical management briefing slide from the Atams project, showing the status and progress of the development items within the increments for Release 3.0. Each DI is represented by a bubble. The bubble's color indicates the DI's increment number and the shape indicates its development status—a circle is "okay," a diamond is "some problems," and an octagon is "critical." This overview permits assessing and tracking the entire project status.

The process offered a number of benefits to the project manager:

- It gives managers and developers an easily understood one-page graphic representation of project and item status.
- It provides concise, definable states with clear entry and exit criteria, activity steps, procedures, and methodologies.
- It provides natural points and clear guidance for valuable capture of metrics.
- It promotes team ownership.
- It facilitates continuous process improvement.
- It increases product quality.

Quality assurance and process improvement

We designed the State-Based Process to engage every team member in quality assurance and process improvement activities and to make them a natural part of their development paradigm. We believe that the only way to ensure quality in the end product is to build QA activities into the process from beginning to end, and we achieved this through the discipline of inspections and walkthroughs.

Additionally, process improvement becomes the responsibility of every member of the development team. The Atams team owned the process and were quick to change it when they discovered a weakness. The process guidelines helped formalize the process improvement activities in several ways.

♦ When defects were discovered, team members considered whether the defect resulted from a process weakness. If so, they discussed process improvement inputs and annotated them as action items in the walkthrough minutes.

♦ The team updated state notes from inputs gathered in the walkthroughs and annotated them as an action item in the minutes.

♦ The weekly team meetings permitted discussions of how to improve the process. This input was also captured as an action item in the meeting minutes.

Results

The State-Based Process was used throughout the Atams project with great success, in terms of both productivity and quality. Our typical walkthrough involved five to six people, lasted almost one hour, required over two hours of external inspection, and took almost one hour of pre-walkthrough overhead and 48 minutes of post-walkthrough overhead. The average return on investment per walkthrough was 2.5 major defects found and more than 6 minor defects uncovered. This equates to just over one hour of staff time per defect when considering all the costs for both major and minor defects—a small price to pay for the return. Another way to look at it is that 748 defects did not live to traumatize the test or operational environment. Since only one defect survived into the first operational release, technically speaking, we uncovered 748 of the 749 defects before compilation.

These numbers are actually rather conservative, since the defect count should be much higher—we estimate approximately three times higher, because in many cases multiple defects were counted as one. For example, problems detected in a particular scenario diagram were usually tagged as one defect. Also, an undetected error in the design or specifications could have resulted in multiple defects later. Of the four operational incident reports written for Atams, only one applied to the over 37,000 hand-developed lines of code.

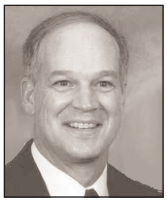
Many other factors beyond the scope of this article contributed to the eventual success of Atams. For example, the tools we used were a major force multiplier. But the people factor really deserves one final comment. The Atams project was a challenge from the beginning, mostly due to schedule constraints. We are convinced that we would not have succeeded without the process, but it also took dedication by an enthusiastic staff. We believe that team ownership of the process was key to their enthusiasm, since a significant part of their commitment went to improving the process.

The State-Based Process does not require "superprogrammers." Our experience showed that the process brought out the best in each programmer. They felt empowered by the process, not hampered by it. We didn't have to fight the process or the team to get the indicators and insight we needed. In fact, we empowered them to do it for us. ❖

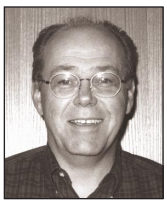
References

1. B.W. Boehm, "Software Risk Management: Principles and Practices," *IEEE Software*, Jan. 1991, pp. 32-41.
2. W.W. Gibbs, "Command and Control: Inside a Hollowed-Out Mountain, Software Fiascoes—and a Signal Success," *Scientific American*, Aug. 1997, pp. 33-34.
3. H.D. Mills et al., "Cleanroom Software Engineering," *IEEE Software*, Sept. 1987, pp. 19-24.

About the Authors



Buford D. Tackett is a senior systems engineer at ITT Industries, System Division, and was program director for Atams. He is a retired Air Force Lt. Colonel and the former White House Director of Technical Plans for the National Security Council during the Bush Administration. He holds a BS in computer science from Kansas State University, an MS in software engineering from Auburn University, and is a doctoral candidate in management.



Buddy Van Doren is a senior systems engineer for ITT Industries, System Division, where he is in charge of implementing knowledge management practices. He has spent more than 32 years in the development and maintenance of information systems for commercial and defense applications. He holds an MS in electrical engineering from the University of California at Berkeley.

Readers may contact Tackett at ITT Industries, System Division, 4450 E. Fountain Blvd, Colorado Springs, CO 80935-5012; e-mail bdtackett@csprings.com.