Journey to a mature software process

by C. Billings
J. Clifton
B. Kolkhorst
E. Lee
W. B. Wingert

Development process maturity is strongly linked to the success or failure of software projects. As the word "maturity" implies, time and effort are necessary to gain it. The Space Shuttle Onboard Software project has been in existence for nearly 20 years. In 1989 the project was rated at the highest level of the Software Engineering Institute's Capability Maturity Model. The high-quality software produced by the project is directly linked to its maturity. This paper focuses on the experiences of the Space Shuttle Onboard Software project in the journey to process maturity and the factors that have made it successful.

There is currently much discussion in the software industry and in the literature about software process maturity and the correlation of process maturity to software quality. At its site in Houston, Texas, the IBM Federal Systems Company (FSC) develops highly reliable software for the federal government. One job, the Onboard Shuttle project, has been evaluated at the highest level on the Software Process Capability Maturity Model[1] of the Software Engineering Institute at Carnegie Mellon University. The FSC software development organization in Houston consistently produces high-quality software and receives accolades from auditing organizations across the nation. This organization was the first contractor to receive the prestigious NASA Excellence Award and is the only contractor to receive this award twice. The Houston site was twice named the IBM Best Software Lab and was twice awarded the Silver Level in an IBM internal assessment matched against the Malcolm Baldrige National Quality Award criteria. It might be assumed with such a consistent record of success, that Houston has discovered the "silver bullet" for software development. Of course, this is not so. Indeed, Houston was only enacting process principles that were known as early as 1960.[2] Sound program management techniques, software engineering principles, employee empowerment, and a culture dedicated to quality are the basis of this software development process. Houston's success is the result of following these processes with discipline and control. This discipline and control evolved over a period of 25 years of service to the federal government and prime contractors. Attention to customer requirements and extensive interaction with the customer are also crucial to the evolution of this mature software process.

Background of the Onboard Shuttle project

Focusing on project management, FSC developed a comprehensive set of software development standards in the early 1980s. Configuration management was rigorously practiced even before automated tools supported this activity. Configura-

tion management of requirements is absolutely essential for the development of large, complex software systems. In the 1970s Houston maintained detailed manual lists of software requirements changes and their impact on software development and testing.

A unique cultural heritage developed, fueled by focusing on "doing things right," accountability to the customer, and the determination of the National Aeronautics and Space Administration (NASA) to develop a space program that is safe for manned missions. Manned flight awareness was a part of every programmer's and engineer's training and a daily emphasis of the management team. Historically, employees were empowered to stop the "software assembly line," if quality issues arose. High visibility and national awareness of manned space flights contributed to this focus on quality.

Building on experience with the early space systems, Houston pioneered the development of the Space Shuttle Onboard Data Processing System. IBM and NASA, the customer, developed a strong relationship based on trust and a common mission. The IBM and customer team consistently produced software that was highly reliable and almost error-free. Houston was ultimately accountable for the operational performance of the entire system. A description of the Space Shuttle Onboard project is the key to understanding how this software development process matured.[3]

Houston's software development process produces highly reliable software for both the Shuttle Onboard project and support systems. Improvements to the development processes made during this project built on development practices already in place in Houston. Disciplined application of program management techniques, use of team reviews, audits, systematic data collection, and independent testing during the 1970s prepared the project for more process advancements in the 1980s.

The 1970s

The space shuttle program built the Primary Avionics Software System from the ground up in the 1970s. All efforts were directed toward developing an architecture and design for the Onboard Primary Avionics System. Major system deliveries were required for approach and landing tests and the orbital flight test software. Critical functions combined in the two systems provided the capability to fly the space shuttle ballistically into orbit, support the orbiter while in space, and fly it through re-entry to a safe landing on a dry lake bed at Edwards Air Force Base in California. Delivering major capabilities such as these while preparing for an operational flight environment presented serious challenges. According to plans at that time, when the space shuttle became operational, numerous software systems were to be managed simultaneously to meet the needs of the envisioned 30 flights a year. Strong software management[4] through program management and configuration control techniques was established early and designed with an eye to the future needs of the program.

The IBM and customer team consistently produced software that was highly reliable and almost error-free.

A special focus on requirements used engineers dedicated to requirements analysis. Engineers serving as requirements analysts interpreted the requirements. Each analyst worked closely with the NASA engineering community. Requirements analysts understood the intent of the requirements, helped to select the best implementation option, and made certain that the intent was communicated in the approved version of the requirements document. The analysts became the IBM experts on the requirements throughout the development of the software. Requirements analysis was recognized as an essential part of the software development life cycle.

With many parallel development activities underway, establishing and adhering to a system architecture was a fundamental problem. Houston formed a software architecture review board to address this issue. Chaired by a senior engineer, the board included representatives from each develop-

ment area of the project. The board established operating procedures, and the project followed these procedures whenever development affected the systems architecture. For example, when a new program module was created, its execution priority had to be approved by the board. This procedure assured a priority consistent with the critical nature of the function. The board published standards for coding certain operations to ensure correct synchronization of the multiple computer environment. Any deviation from these standards had to be approved by the board after analysis had verified that the change could be made without system degradation.

Manual processes that implemented the procedures of the board had no checks and balances to ensure that procedures were followed. Engineers and programmers understood the criticality of the software being produced and did not want to make a mistake in implementation. Review by the board helped share the serious responsibility that each engineer and programmer felt for the safe execution of the final product. The developers' acceptance of team review led to the use of reviews as the technique of choice for ensuring product quality. In addition, the enforcement of discipline in following the process set the stage for future success.

During the 1970s the project used measurements to track schedules and costs but had only begun to consider quality metrics. Houston monitored the total numbers of open problem reports but only as a program management indication of the progress toward delivery. Although quality measurement techniques were not advanced, a valuable activity was underway: data collection. Because of the necessity of total accountability on product problems to NASA, data were collected on all software problems for the project. Houston systematically collected and retained data about each problem. Every problem had to be explained to NASA. NASA asked probing questions, such as why was a mistake made, were there any other similar problems in the software, and what actions would be taken to prevent the same type of error in the future.

Gaining information on each software problem required insight that could only be provided by the software development team. The systematic analysis of each problem did more than strengthen processes. Developers were key members of the analysis team and felt accountable for each error. The focus was always placed on the error and never on the individual who made the error. Nonetheless, professional pride made programmers feel responsible. From each problem the developers learned to avoid that error in the future. Each oversight reinforced the need to rely on process to remove errors.

Gaining information on each software problem required insight that could only be provided by the software development team.

Systematic data collection on problems became well-established with information retained in a database, both electronically and on paper. This repository made it possible to do trend analysis as soon as a formal measurement program was established in 1982. Database information became the basis for sophisticated reliability estimates and for research on software complexity metrics.

Data collection paid off quickly. Late additions to requirements were disruptive to development activities. Pressure to satisfy the customer's needs for software capabilities subverted project planning and development processes. Analysis of problem data from released software revealed that an out-of-control requirements management process was the primary cause. Use of the data convinced IBM management and the customer's requirements approval board to put more control on the requirements approval process, eliminating over-commitment as a cause of problems.

Finding answers to customer questions also required an audit of the software product. Once a problem was fully understood, the development organization designed an audit to find similar instances in the code, if they existed. Each instance identified was analyzed to determine whether it

was also in error. The audit results allowed customer questions to be answered with confidence. Any new problems found by the audit could be corrected before they caused the software to fail. The audit, in addition to the error causal analysis done for each problem, provided a basis for formal defect prevention initiatives.

Design and code reviews were conducted during those early days. Review participants included the development programmer, the requirements analyst for the area under review, and peer programmers. Reviews lacked a formal moderator, rigorous documentation and follow-up on issues identified, and a checklist of items to inspect. The developer was the one who decided whether a re-review was necessary. Configuration control was informally tied to the reviews, since the basis for the review was the approved requirements. Most software products were reviewed during this era. However, since a documented process was lacking, the teams did not have a consistent approach. Despite the lack of rigor, developers shared the responsibility for the product under review. This review method continued to reinforce the culture of team oversight and procedural discipline. Developers became accountable for each product error and were inspired to share responsibility for the quality of their software.

In time, project processes formalized the cultural acceptance of the need for procedural discipline. This was demonstrated dramatically during the flight of the Space Shuttle STS-49. Astronauts struggled to capture a malfunctioning satellite. NASA requested a change to the Remote Manipulator Arm (RMS) software to improve astronaut control of the RMS. This software update had to work the first time and had to be delivered in a matter of hours. With the astronauts waiting in orbit, programmers developed and tested the software change. They executed all required process steps, including inspections of requirements, design, and code before the update was released to NASA. The STS-49 astronauts successfully used the update with several other methods before the satellite was finally captured. In a crisis, a mature process is relied upon to produce the needed results, not ignored out of expediency.

During this era, an independent test organization verified the software before delivery. Testing was conducted with extensive use of simulation to model the operating environment of the software. Execution of the code was initially done on a simulated flight computer. When the Flight Electronic Interface Device became available, testing was conducted on actual flight hardware.

The combination of error causal analysis, an evolving focus on measurements, and a growing formalization of processes brought about a gradual decline in product errors. As shown in Figure 1, the software development process continued to mature throughout the 1980s and into the 1990s, based on the firm foundations that had been established in the 1970s. A culture of shared responsibility through team review, procedural discipline, data collection and analysis, product audits, independent testing, and controlled system architecture was in place.

The 1980s

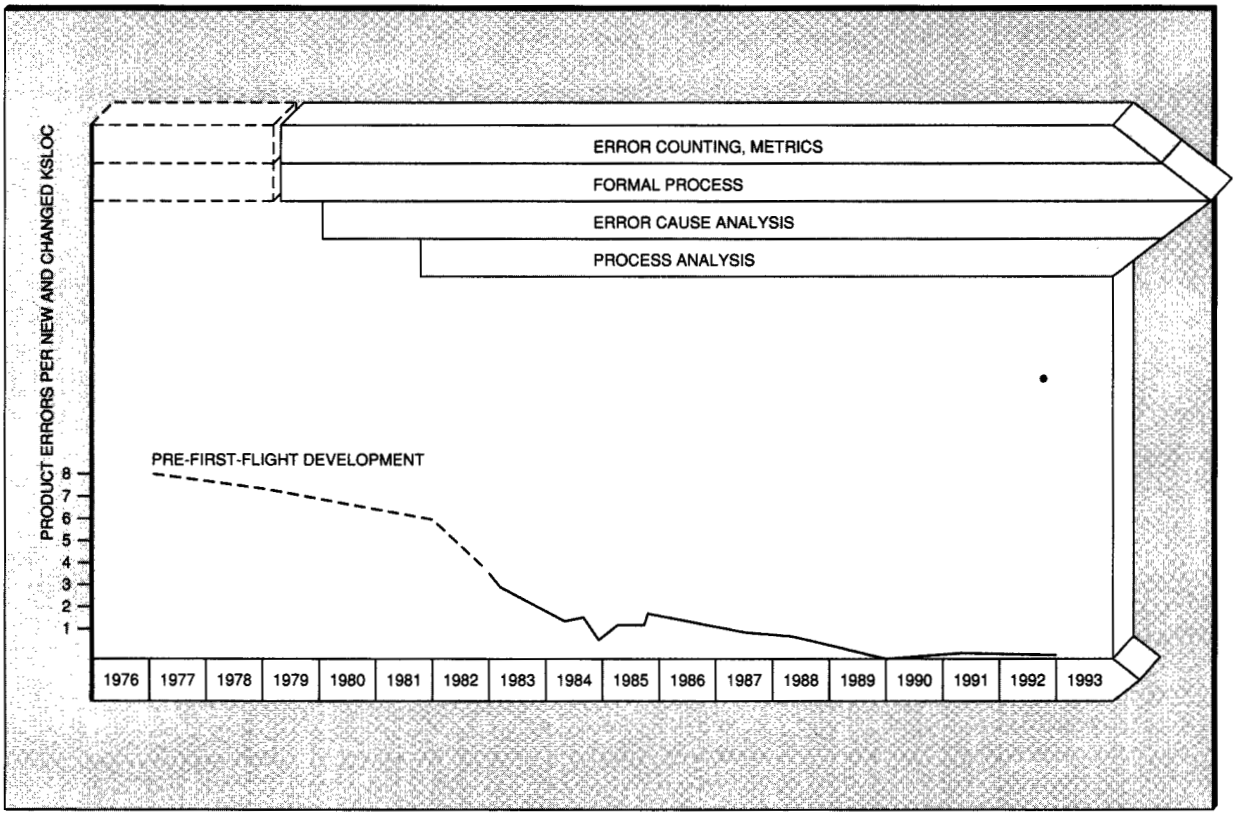
Project management. The shuttle program went through first flight and became operational in the 1980s. Moving to an operational environment with an increasing flight rate tested Houston's resolve to stick to its processes, but strong project management led to higher quality, reduced costs, and increased productivity.

In the 1980s, project management strove to balance control and responsiveness. A system of boards evolved to ensure configuration control of the software, communication flow within the project, and a single interface to NASA. These boards dealt with all aspects of development and continue to be critical to the success and discipline of the project.

The original structure, illustrated in Figure 2, had two main boards: the Project Control Board (IBM) and the Customer Configuration Control Board. Additionally, Houston established three key subboards: the Discrepancy Report Board, the Requirements Review Board, and the Support Software Board. The Project Control Board, Discrepancy Report Board, and the Requirements Review Board controlled the three most important aspects of the process.

The Project Control Board had a key role as the clearinghouse for all project status and major decisions. All boards were subordinate to the Project Control Board. This board maintained the configuration of the software and established

Figure 1 Product quality improvement based on process focus



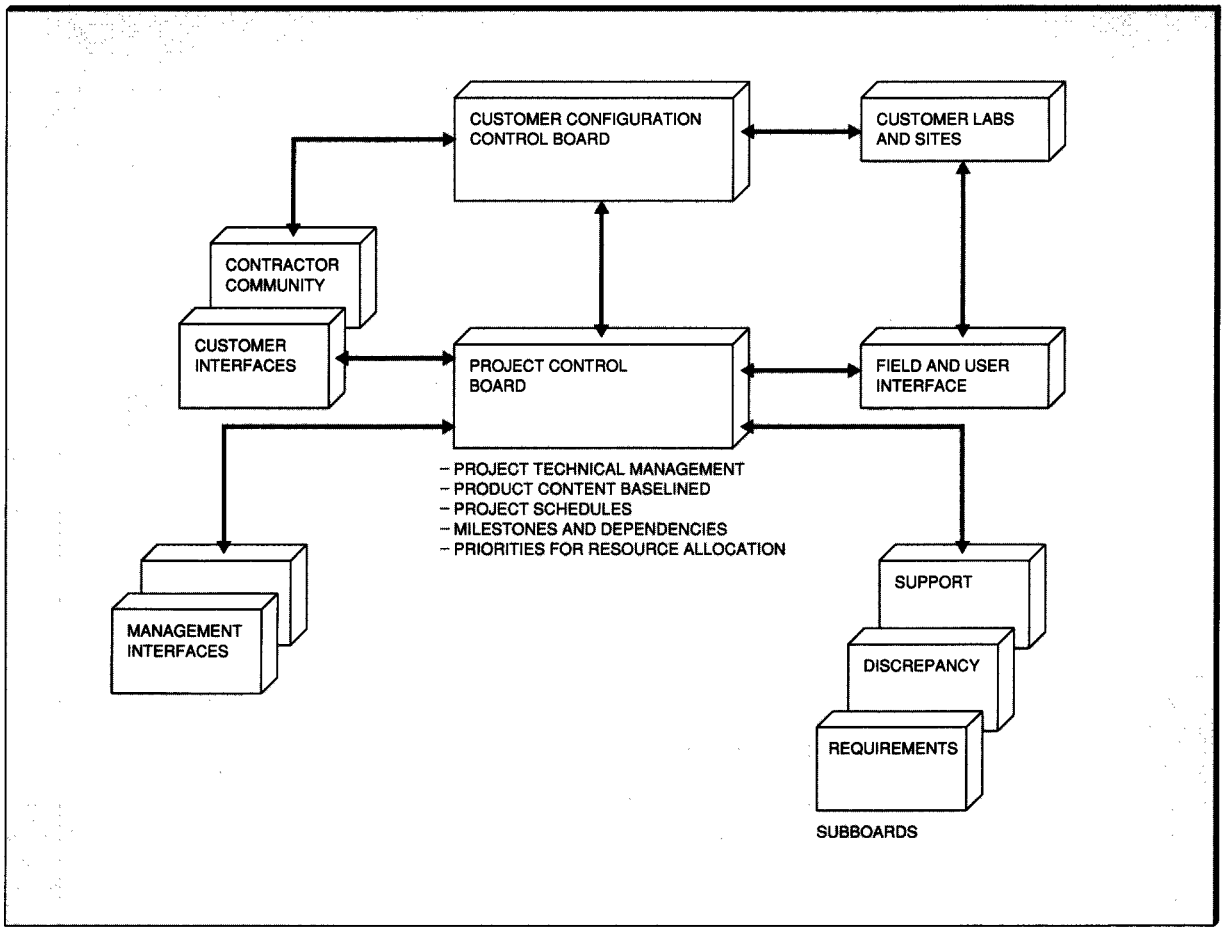
IBM's position on proposed changes. During the 1980s, this board became the primary point of contact with the customer. This board determined all schedules and milestones. Again, representatives of each major function had a voice in the decision.

The Discrepancy Report Board dealt with how Houston would respond to a problem found in the software. Problem reports came from internal sources or from one of over 200 external NASA user groups and external contractors. The board met weekly or on demand to discuss current discrepancies and their effect on the flight software. The Discrepancy Report Board determined Houston's recommendation to fix or not fix a software problem. A board representative presented this recommendation to the NASA Configuration Control Board where a final decision was made. A single point of customer contact was provided by the board. Additionally, the board provided an open forum for engineers to state their position on a software problem in a nonthreatening environment, fostering an atmosphere of trust.

In determining how and when to implement a fix, the Discrepancy Report Board coordinated inputs from many sources. A fix, for example, could be applied to an incremental release or to one or more specific flights. These decisions were based on fix criticality and current schedules. A representative of each stage of the development process had a voice prior to the final IBM position being determined.

The Requirements Review Board ensured that project resources were fully utilized and schedules could be met. This board closely coordinated changes required with the resources available and kept the customer informed. Changes to the soft-

Figure 2 Onboard Shuttle control boards



ware could not be approved by the customer without inputs from this board. This was a key step in eliminating errors early in the process that were caused by poor requirements or over-commitment of resources. Inputs to the board included the readiness of the requirements for implementation, the interactions with other changes in process, and the development and verification costs associated with the change. This board provided project management with the costs and risks associated with each change to the flight software. This information allowed schedules to be developed, tracked, and coordinated with other project activities. Costs to the customer were reduced, since the earlier in a process that defects are removed, the less expensive they are to fix. For example, an error caught during the requirements phase is fixed once and cheaply, whereas the same error detected after delivery may have to be fixed on several released systems, escalating maintenance costs.

Key areas of the software project were brought together regularly by these boards. Each board provided an open forum to express concerns and provide status. The board chairpersons were part of an independent staff department, helping the boards to keep the customer's interests in focus.

Houston established other boards that dealt with specific aspects of the development process as shown in Figure 2. Parallel boards were estab-

lished for the ground support software. A board to track the development of user tools was established to foster communication and reuse of software among the various project elements.

Houston found that a strong customer-driven board structure was a key item for maintaining quality, delivering on schedule, and staying within budget. Houston's board structure mirrored the customer's organization. This structure provided each key customer process with a single point of contact with the analogous function in the development organization. All boards were staffed and chaired by experienced nonmanagement technical personnel who worked in the program for years, some since its inception. This method of project management has had quantifiable successful results:

- High-quality software
- An extremely satisfied customer (excellent customer evaluations)
- No significant budget overruns

The board structure is in place to ensure that this success continues.

Incremental release strategy. An incremental release strategy grew out of a need to isolate the development process from the day-to-day operations of the space shuttle program. For the first six flights of the shuttle, the flight software consisted of the software for the previous flight plus new capabilities. For example, the ability to abort a shuttle mission to Africa or Spain did not exist on the third flight but was added for the fourth flight. As the flight rate increased, this method would prove less manageable.

NASA and IBM decided to support multiple flights with one release of software called an operational increment, or OI. An OI would be reconfigured prior to each flight to account for the mission-specific parameters such as payload, orbit, time-of-year, etc. This approach allowed the development cycle to operate somewhat independently of the flight operations.[5]

Operational increment testing focused on new software capabilities rather than flight-specific testing. A new level of testing was created to address the flight-specific verification. The latter dealt with mission or flight operations and the former with the longer-term development items and maintenance. This strategy reduced the potential for resource conflicts and allowed optimization of each process to meet its particular customer's requirements. Today, a typical release of software is used for one year as shown in Figure 3.

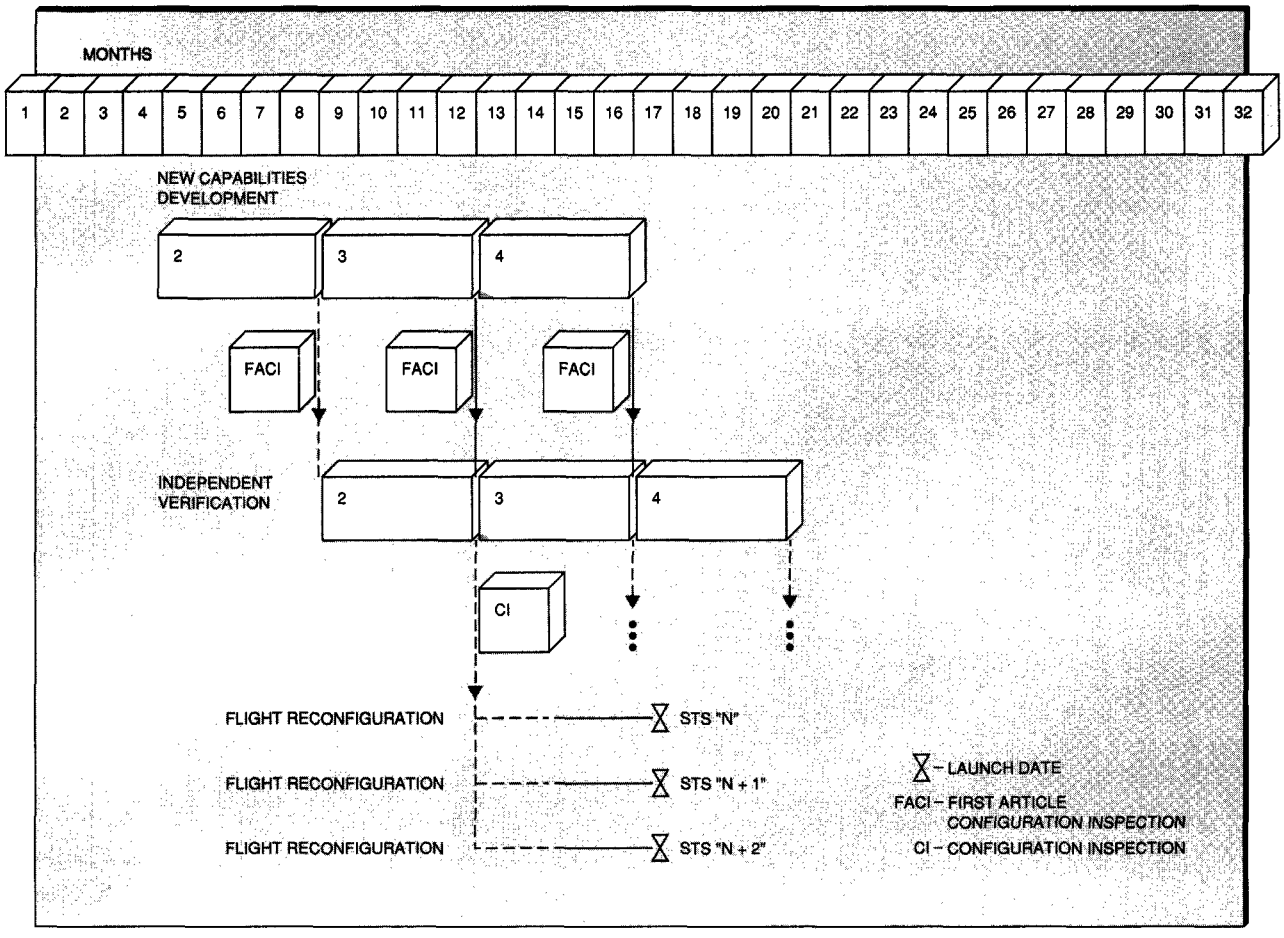
This approach is more flexible than a traditional waterfall process. It allows for new requirements and other changes to the software at multiple points along the way. This flexibility is demonstrated by the fact that the software that has flown shuttle missions has undergone more than 3000 requirements changes. Since the late 1970s, the software required more than 382 000 source lines of code to be added, modified, or deleted. These changes were implemented via more than 900 software builds and 175 patches. IBM has provided these evolutionary software versions to NASA through 260 separate software releases. Even though a typical development cycle is one year, Houston's incremental release process responds to the short-term needs of the customer as well.

An incremental release strategy grew out of a need to isolate the development process from the day-to-day operations.

An incremental strategy was used as well to produce prototype versions of the flight software. Houston's prototypes involved setting up a "mock" incremental release and following the existing process in an expedited fashion. This technique dramatically improved final product quality. The development organization identified additional requirements errors (5 percent) and design or coding errors (23 percent). If these errors were discovered later, they would have been more costly to fix.

It is important to note that when NASA approved the final requirements, the prototype was retired, and the defined process was followed for the real implementation. This is a key step, since the pro-

Figure 3 Generic Onboard Shuttle operational increment



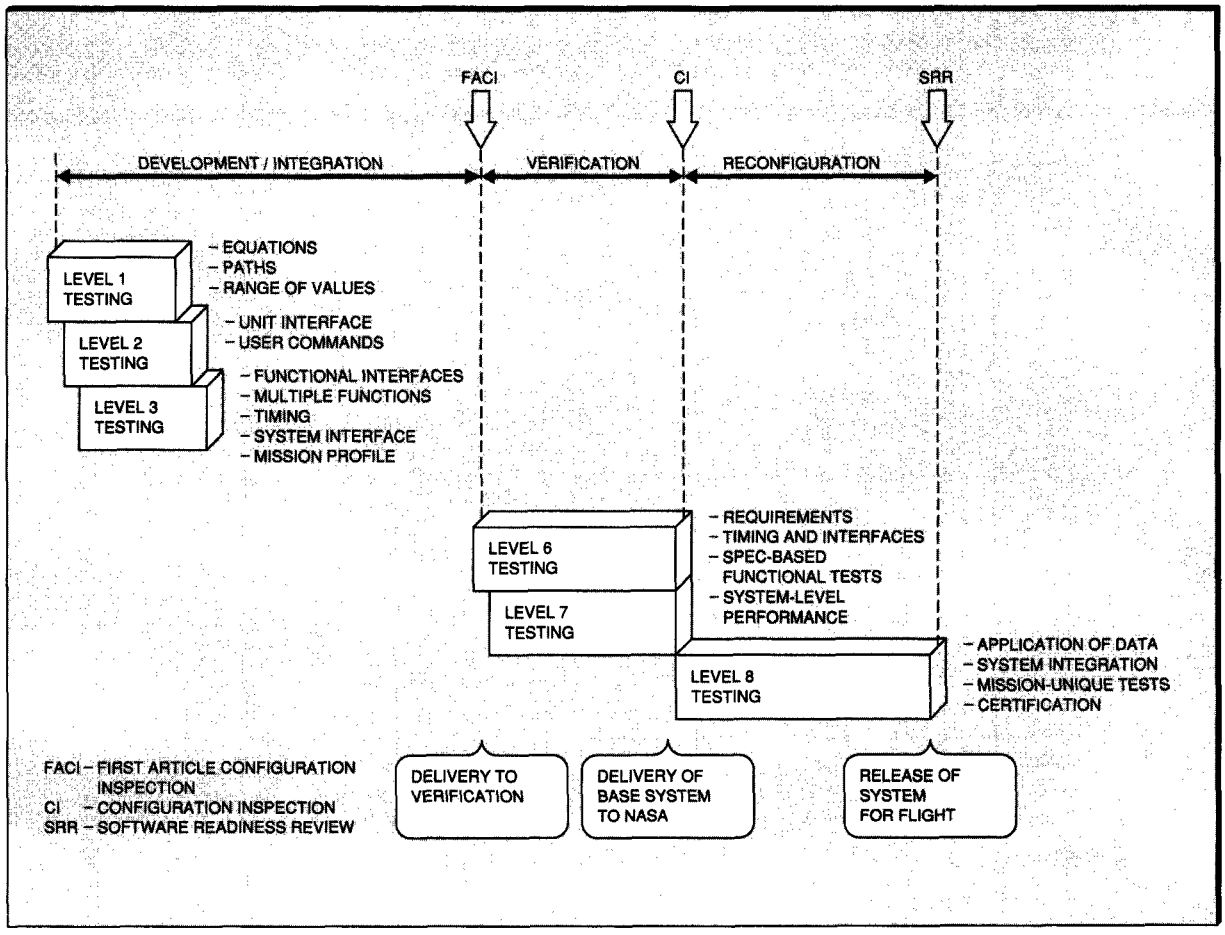
cess is sound and should not be bypassed. The advantages of prototyping are not lost, resulting in less rework and maintenance costs.

Requirements planning. With the introduction of the incremental release strategy, NASA began taking a longer-term look at software development. NASA focused on determining the strategic priority of candidate changes to the flight software. IBM provided guidance as to what could be accomplished with the skills available. Together, NASA and IBM planned the development of several releases to be done in the future. NASA approved all software changes affecting related areas together to simplify both the development and the verification process.

At any given time, NASA planned two to four years in advance. With this approach, all affected areas could plan their activities, such as training, with advance information. Of course, not all changes could be anticipated, but software was treated as a subsystem that could be improved and upgraded much like hardware components and often as a result of hardware upgrades.

Life-cycle changes—Independent verification. In the 1970s, Houston established an independent verification function as a separate line organization without managerial or personnel ties to the development organizations. Independent verification analysts maintained a healthy adversarial

Figure 4 Onboard Shuttle test levels



relationship with the software developers. Multiple test phases were defined (Figure 4), and the independent tests were based on an assumption that the software was untested by the development programmers. Verification analysts developed independent test plans for requirements-based testing, independent functional testing, and independent code desk checks and audits. Verification analysts were responsible for a system-level test phase that emphasized customer-oriented testing and shuttle community involvement in test planning and in analysis of test results.

Verification staffing nearly equaled the development levels, and verification personnel had requirements analysis, software development, and onboard systems experience. Configuration control of test products was a key element of test quality and test documentation and analysis. Results were controlled and archived with the incremental release software under test. This archive supplied reusable test components for regression testing and retest of changed software.

During the early 1980s, Houston changed the software process to improve early detection of software errors. Resource allocation was shifted to the front end of the software development life cycle to support formalized inspections. These inspections included mandatory involvement of independent verification personnel in software design and code inspections. This requirement

was initially resisted by verifiers, who felt it would compromise their independence. A modification in the inspection process to separate inspections of the software from inspections of unit and functional test plans and test cases satisfied this concern. The result of this process change was a dramatic increase in detection of software errors during inspections. The decrease in rework due to the early detection of errors more than paid for the shift of resources, thus increasing overall productivity.

Secondary benefits from the involvement of the verification group in software inspections include:

- Verification analysts were more knowledgeable of the implementation.
- Team fellowship and product ownership were fostered.

The independence of the verification group was not compromised for two reasons. First, verification analysts did not have knowledge of development testing and could continue to consider the software untested. Second, the verification analysts inspected the software from a different perspective than developers. Verification analysts considered inspections a "first test" and reviewed design and code for weak spots, constraints, data anomalies, and other characteristics that would typically be represented in test strategies. Verifiers uniquely find approximately 20 percent of inspection errors—errors that might otherwise have to be found through dynamic testing.[6]

When the Onboard Shuttle flight system became operational, the detail verification group adopted a delta test strategy to concentrate testing activity on only affected code statements and logic in new or changed code. This strategy required detailed tracking of software changes to test plans and cases developed to test that software. Configuration control and test documentation improved, resulting in even tighter control over test products.

A process team was formed to improve performance verification quality and effectiveness. The system performance verification process began to take shape in the early 1980s when performance testing was designed and a review process was established to improve the quality of the test products. A system performance testing board reviewed and approved (1) preliminary verification assessments (considerations on whether or not to explicitly test a changed requirement, how much to test, etc.); (2) test specifications (how to test, what conditions will be tested, etc.); and (3) test reports (actual results vs expected results, analysis of discrepancies, etc.). This board resolved most technical disagreements and was the forerunner of the process teams, discussed in the next section.

Process assessments. Throughout the 1980s, FSC Houston sought out independent evaluations of its processes. Applying for the NASA Excellence Awards, Malcolm Baldrige Award, and internal IBM quality awards, as well as internal IBM assessments, provided measurements against widely accepted (and consistent) criteria. Houston used these evaluations as a means to identify process "weak spots."

For example, in early 1984, a team working under Watts Humphrey rated the two largest projects in FSC Houston against a set of criteria that were to become the Process Capability Maturity Model of the Software Engineering Institute at Carnegie Mellon University. A one-week independent review of each project was conducted, concentrating on software development processes for each life-cycle phase, as well as processes spanning the life cycle: performance, information development, quality assurance, and change control. Process attributes were evaluated against 5 levels (with 5 as the highest rating). The Onboard Shuttle project average across the 11 areas was 3.15, and the system test phase scored 4.

The following areas were suggested for improvement:

- Data collection, analysis, and feedback were insufficient at the process level.
- Proven methodologies were not being consistently used in inspections.
- Test process consistency, configuration management, and coverage measurement could be improved.
- Documentation preparation was largely done manually.

On the basis of the assessment recommendations, a continuous focus on these items brought about process and product improvement. In 1989, the Onboard Shuttle project was evaluated by a NASA

team using the Software Engineering Institute (SEI) Capability Maturity Model. The shuttle project scored a "5," the highest possible rating. A review of these results by Humphrey confirmed the NASA findings.[7]

Process improvements. Measurements and inspections were important in improving the process.

Process and product measurements. To understand processes and the effects of change, an organization must be able to measure its processes. During the 1980s the Onboard Shuttle project went from the "primitive" project measurements of the 1970s to precise measurements of software quality and the development process. Major project measurements are:

- Software quality measurements monitored as a group

  Early detection percent

  $$= \frac{\text{major inspection errors} \times 100}{\text{total errors}}$$

  Process error rate

  $$= \frac{\text{valid errors pre-delivery}}{\text{thousand source lines of code (KSLOC)}}$$

  Product error rate

  $$= \frac{\text{valid errors post-delivery}}{\text{thousand source lines of code (KSLOC)}}$$

- Process measurement

  Total inserted error rate

  $$= \frac{\text{major inspection errors + all valid errors}}{\text{thousand source lines of code (KSLOC)}}$$

Collecting measurements is not enough. It is necessary to properly analyze and understand the information. Trends in process errors, for example, must be examined in conjunction with the trends in early detection and product errors. If early detection trends increase and both process and product error trends decrease, the trends are favorable. If, in contrast, process errors decrease and product errors increase, the software development process must be examined to find the weaknesses that allow errors to be delivered to the customer.

Measurements of the subprocesses gauge their effectiveness and the effect of process changes. For example, concentration on the early detection measurement resulted in improvements to the design and code inspection process discussed in the next subsection. Measurements demonstrated the increased effectiveness of the process. As seen in Figure 5, process analysis through measurement has demonstrated improved processes that have resulted in improved software products.

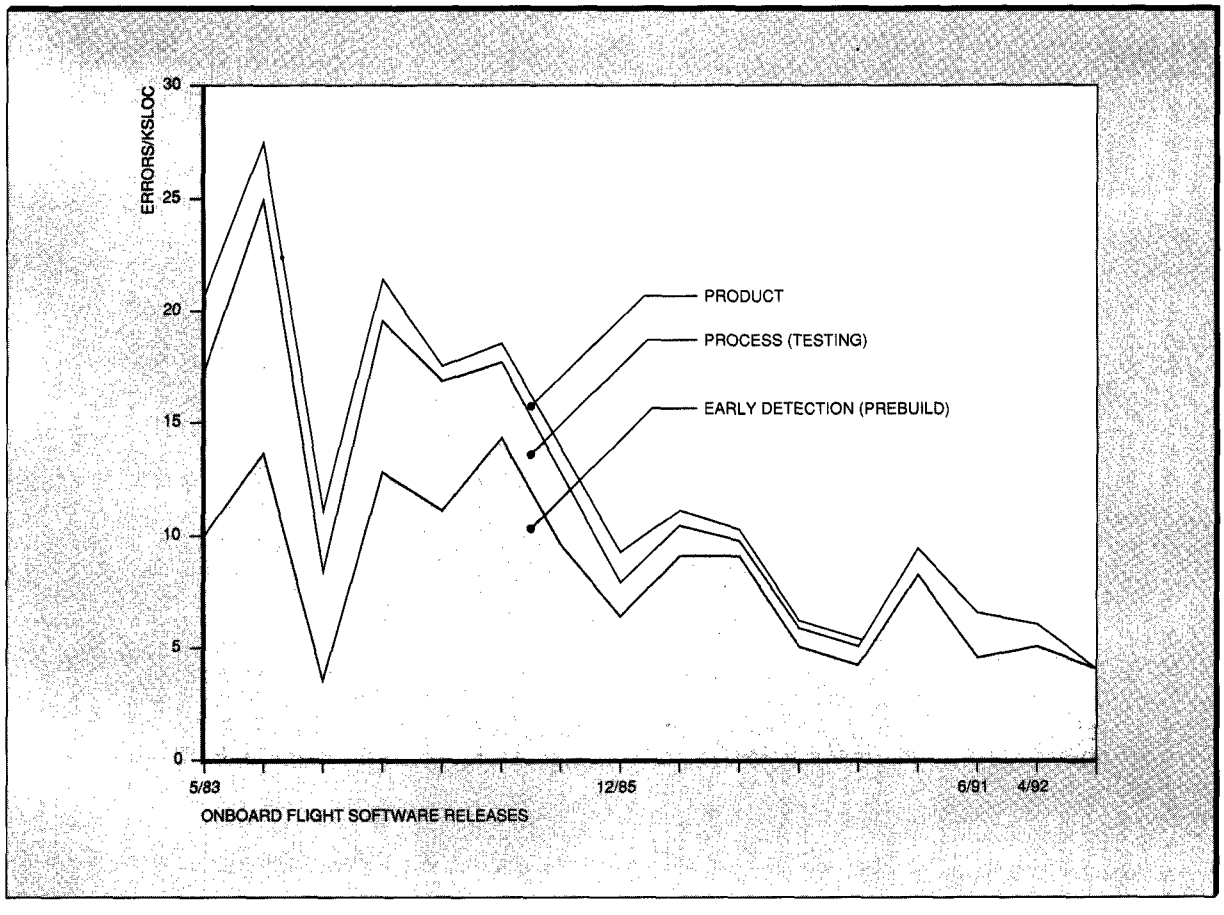
To understand processes and the effects of change, an organization must be able to measure its processes.

Inspections. In 1981, a mandatory inspection process was formalized. This led to a significant increase in early error detection. The formal inspection process required checklists for design and code inspections, a formally trained moderator team, and participation by the requirements analysts and verification analysts. Later improvements included assignment of specific responsibilities for each inspection participant and further refined procedures.

Checklist item responsibilities were assigned to individual team members, but the team goal was to detect all errors in the design and code. The primary reason for conducting a meeting in addition to individual inspection activity is the synergy created through face-to-face interaction. The moderator is a formally trained chairperson of the inspection meeting and has overall responsibility for the inspection activities.

Procedures were refined to define the contents of the inspection packages; scheduling algorithms

Figure 5 Quality improvement in the 1980s



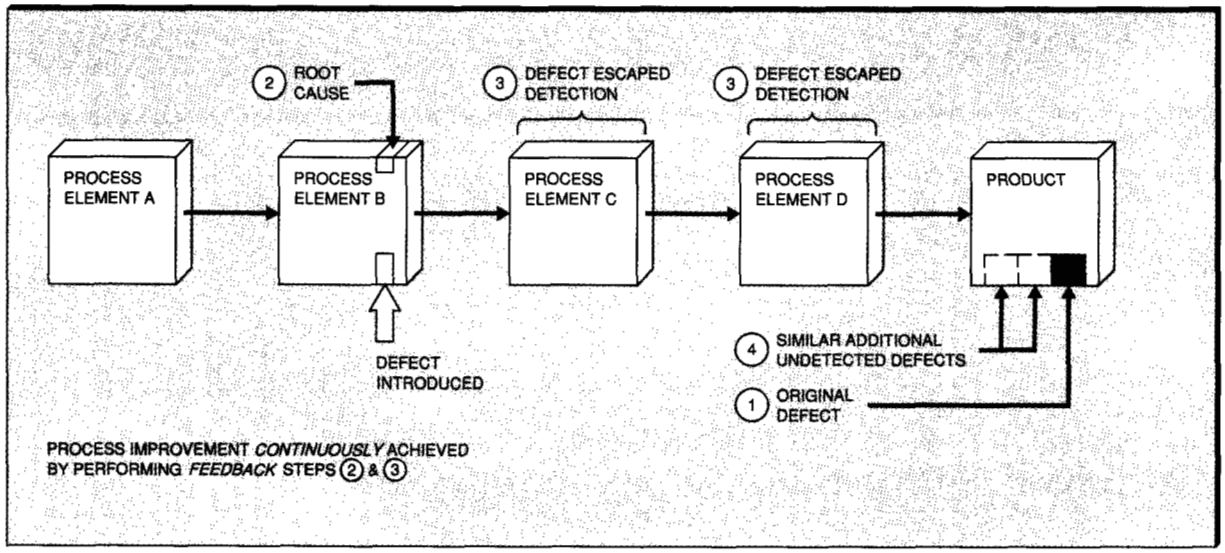
were designed to allow adequate preparation time for all participants; formal documentation of action items were required; and waiver mechanisms for process deviations were initiated.

The results were dramatic. Early detection improved from the 50 percent levels of the 1970s to above 80 percent in the 1980s.

This success caused the inspection process to be propagated to requirements and test products. The inspection process was modified for each development stage. For example, the customer was included in the requirements and test inspections but excluded from the design and code inspections. Formal inspections dramatically improved the quality of requirements, significantly reduced verification test resource usage, and increased test effectiveness.

Defect prevention process. The Onboard Shuttle project defect prevention process is based on audits and analyses. In the 1970s errors were classified and, if a particular error was severe, audits were performed to detect other instances of that error class. In one case, a critical error was detected in multipass data usage. The symptoms and characteristics of this error class were identified, and an intensive analysis effort was conducted by the development and verification organizations to find other instances. Global variables were another area of concern due to the complexity of global data usage and computer synchronization.

Figure 6 Four-step defect prevention process



Tools were developed and a process established to prevent insertion of global data errors.

The causal analysis and defect prevention process consists of identifying classes of errors, searching for their causes, and modifying the processes to prevent the occurrence of those errors in the future. Special teams investigate every error. These teams are composed of members from all software development phases and are responsible for determining how an error escaped detection and for finding any similar errors.[8]

The teams use the following rigorous four-step approach shown in Figure 6:

1. Find the error and fix it.
2. Find and eliminate the cause of the error.
3. Fix other faults in the process that allowed the error to go undetected through the process.
4. Look for similar, as-yet-undetected, errors and eliminate them too.

In addition to the four-step process, a periodic analysis of error trends is conducted. If as a result of this analysis it is concluded that the process needs to be changed, changes are designed and implemented. In this disciplined approach, both improving the quality of the product through a systematic error search and improving the quality of the processes to prevent future occurrences of such errors have combined to produce the near-zero product error rates in the software.

Applying the shuttle process. The Houston process has been successfully tailored for use in other projects. Tailoring reduces or modifies activities that are not appropriate for the new application. For example, without the requirement to have a system able to support human life, the large investment in independent testing can be reduced. Although most Houston processes relate well to other projects, requirements analysis is particularly applicable. In any development project, understanding the customer's requirements and documenting them correctly will reduce errors more than any other single step. Houston applied the requirements process to a small engineering lab upgrade. The resulting set of requirements provided a clear picture of what was needed to satisfy the users. Although the customers and developers were frustrated at first by slow progress, the actual implementation proceeded smoothly, and the end result clearly benefited from the requirements effort.

Additionally, Houston transferred the code inspection and requirements analysis process to its

ground support software project during the 1980s. The result has been a drop in product error rates from 0.72 errors per 1000 lines of code in 1986 to 0.30 in 1993.[9]

Even on small software teams, most or all of the process can be used effectively. A team can decide the level of rigor they will impose on themselves based on the criticality of their application. Aspects of the shuttle process are used today on the space station program, the air traffic control system, and others.

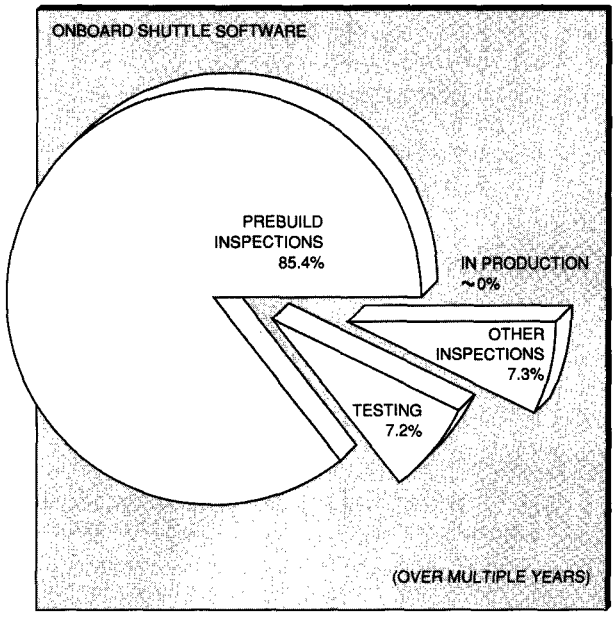
Reliability. During the 1980s, the shuttle project began to use a large historical database to predict the reliability of software. This gave Houston the ability to redirect resources to reduce errors before the software was released. Using models, Houston is able to predict when and how many errors are likely to be found.[10] The analysis of the data found several key points. First (and central to Houston's approach to defect elimination) is that all software errors cannot be found by testing. In fact, less than 10 percent are found in shuttle testing today (Figure 7). This outcome is due to the emphasis placed on removing errors prior to the test phase. Second, about half the errors that escape the inspection process are found by testing. The remaining errors are found by static analysis (code audits, desk checks, etc.). Lastly, reliability model data are used to decide when to stop testing as opposed to waiting for time or money to run out. Reliability measurements require access to historical data to be accurate. These historical data, combined with the knowledge of the planned software changes, allow reliability to be estimated.

The 1990s

The 1990s brought the need to be more competitive through increased productivity, while maintaining the software quality that the customer has come to expect. Houston pursued a strategy of selective insertion of new technology into the software development process in combination with actions to optimize the existing process. Commercial off-the-shelf hardware and software are used to enact the process. Process ownership teams work to optimize the development processes.

Enactment of the software development process. Houston learned from experience and through

Figure 7 **Where errors are found in the space shuttle software**



benchmarking with other companies that computer-assisted software engineering (CASE) tools are, at best, only a partial answer to productivity improvement.[11] Since the shuttle software is written in unique programming languages, a lack of commercial products increases the challenge of technology insertion. Rather than focus on the use of technology to design and code software, the Onboard Shuttle project has focused on enabling the formal processes that govern the production of the software. The objective is to free the software developers from administering the process and to focus their technical skills on producing software.

Process ownership teams applied. In 1990 it was recognized that technical management of the software development process needed to reside with people who performed the processes. They knew better than anyone what worked and what could be improved. If the process could be optimized, the development teams had the insight to make the changes.

Following the model of the successful design and code inspection teams, ownership teams were assigned to each of the processes: requirements

evaluation, development design and code, development test, and independent test. Each team monitored and controlled its process. The team's responsibilities were to:

- Document the process
- Collect process metrics
- Benchmark the process
- Analyze process metrics and optimize the process
- Provide education to process users

Teams were encouraged to find new means of improving process efficiency. They often included both vendors and customers as part of their process improvement activities. The team approach has been ingrained in the Houston culture and is considered to be a normal business routine.

One of the first successes involved the development test team. Soon after they formed, the team identified the inability of unit test to discover errors. Inspection of the requirements, design, and code left virtually no errors in the software detectable by traditional unit testing. The few errors that remained consisted of interface problems and errors in rare execution scenarios that fell outside the scope of unit testing. The team responsible for the development testing process changed their testing philosophy from that of unit and functional test to scenario testing. Early results of this process change are encouraging, with increased error detection in this development stage. Detection of errors has been moved so that it is earlier in the development life cycle, reducing the cost of rework.

Leaders of the process teams are members of a Process Evaluation Team. The Process Evaluation Team meets regularly to discuss cross-functional process issues and to evaluate each process.[12] Improved communications has spread process concepts from one process to another. Ownership teams are accelerating the evolutionary optimization of the Houston processes.

Conclusion

Sophisticated processes to develop the space shuttle software evolved over many years. Several factors influenced the overall success of the project. Maturity grew out of practical experience and innovative ideas from industry and academia, as well as through trial and error. Disciplined application of the resulting processes has produced software systems that have been virtually error-free. Although these processes were developed for a complex aerospace application, many are fundamental for any type of software development. Strong program management, adherence to the process even during times of pressure, and procedural discipline have a significant positive influence on project results. None requires sophisticated technology. They are organizational, procedural, and cultural and can be implemented by managers and software professionals who have the desire to improve their development environment.

Houston demonstrated that disciplined use of program management, team inspections, independent testing, incremental development, requirements management, and measurement programs result in predictable product quality delivered on time and within budget. Additional important factors in attaining extremely low error rates were the use of product audits when process weaknesses were discovered, using independent testers as inspectors in requirements, design, and code inspections, and process evolution driven by problem causal analysis.

The software development life cycle is an integration of all the processes necessary to produce the software products. Process maturity comes from focusing on each of these processes and ensuring that all the steps are necessary, that the process is followed, and that the door is open to better ways of completing the activity. Maturity develops in both the processes and in the attitudes of those who must execute the processes.

Cited references and note

1. *Capability Maturity Model for Software*, CMU/SEI-91-TR-24, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213 (August 1991).
2. N. H. Madhavji, K. Toubache, and E. Lynch, *The IBM-McGill Project on Software Process*, Technical Report 74-077, Centre for Advanced Studies, IBM Canada Ltd., Toronto (October 1991).
3. J. F. Hanaway and R. W. Moorehead, *Space Shuttle Avionics System,* National Aeronautics and Space Administration Office of Management, Scientific and Technical Information Division, Washington, DC.
4. C. P. Lecht, *The Management of Computer Programming Projects,* American Management Association, New York (1967).
5. W. Madden and K. Rone, "Shuttle Operational Increments: Design and Development of the Space Shuttle Pri-

mary Flight Software System," Houston Technical Direction, IBM Federal Systems Company (1979).
6. E. Lee, "Using Testing Resources for Defect Prevention," *5th International Software Quality Week*, Software Research, Inc., San Francisco (May 1992).
7. W. S. Humphrey, Director, Software Process Program, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, stated, "We were delighted with the degree to which your experiences reinforce the SEI maturity framework and particularly by the way the continuous improvement culture seems to pervade your organization. . . . Four-step process for defect prevention was particularly impressive and represents a step beyond what we had been considering." (July 1990).
8. R. G. Mays, C. L. Jones, G. J. Holloway, and D. P. Studinski, "Experiences with Defect Prevention," *IBM Systems Journal* **29**, No. 1, 4–32 (1990).
9. "Use of Testing Resources for Early Defect Elimination," E. Lee, *Software Management News* **11**, No. 6 (November/December 1993).
10. N. F. Schniedewind and T. W. Keller, "Applying Reliability Models to the Space Shuttle," *IEEE Software* **9**, No. 4, 28–32 (July 1992).
11. F. P. Brooks, "No Silver Bullets ... The Essence and Accidents of Software Engineering," *Computer* **20**, No. 4, 10 (April 1987).
12. E. Lee, "Process Evaluation Teams," *8th Annual NASA/Contractors Conference and 1991 Symposium on Quality and Productivity*, ISSN 1049-667X, NASA (April 1992), Sec. 8.1.4, p. 127.

Accepted for publication September 8, 1993.

Note: At the time of publication, Federal Systems Company, now a unit of Loral Corporation, was an IBM-owned company. Addresses for authors may still be considered valid.

Cyndy Billings *IBM Federal Systems Company, 3700 Bay Area Boulevard, Houston, Texas 77058-1199.* Ms. Billings is a consultant with the IBM Federal Application Development Consulting Practice, specializing in test process and methodology for high-quality software. She has over a decade of experience in the testing and verification of complex embedded software systems, with five years of management experience in space station and Onboard Shuttle Software testing.

Jeanie Clifton *IBM Federal Systems Company, 3700 Bay Area Boulevard, Houston, Texas 77058-1199.* Ms. Clifton began her career with IBM in 1981 in Tucson, Arizona, working in a product reliability, availability, and serviceability group. She later worked in the Tucson Customer Support Center, the first of its kind within IBM, solving customer computing problems and instructing them in the use of many IBM software and hardware products. Following the Space Shuttle Challenger accident in 1986, Ms. Clifton transferred to IBM's Federal Sector Division and played a key role in revalidating the Onboard Shuttle Software. She has been a technical lead within the Onboard Shuttle Software project, being the chairperson of the Development and Inspection Process group. This process is recognized around the world as the key to developing the "zero defect" shuttle software. She is currently a member of the IBM Federal Application Development Consulting Practice, helping software development laboratories both internal and external to IBM.

Barbara Kolkhorst *IBM Federal Systems Company, 3700 Bay Area Boulevard, Houston, Texas 77058-1199.* Ms. Kolkhorst is a senior systems engineer with the IBM Federal Application Development Consulting Practice. Her work has focused on evaluating and improving the software development process to produce affordable, highly reliable software systems. Ms. Kolkhorst has extensive experience as both a software development manager and a technical leader in the development of software products for NASA's space shuttle program. She has over 30 years experience in all phases of the software development life cycle, developing highly reliable software for applications supporting manned space flight, nuclear energy, and modeling for business economics.

Earl Lee *IBM Federal Systems Company, 3700 Bay Area Boulevard, Houston, Texas 77058-1199.* Mr. Lee has 27 years of experience on large, complex, data processing systems. These have included the semi-automation of the FAA's Nation Air Space Enroute Air Traffic Control System and the Space Shuttle Onboard Data Processing System. He has been involved in software development for 18 years in both technical and management roles, and his experience spans the entire software development life cycle. As a manager, he led the establishment of software process evaluation methods on the space shuttle project. Mr. Lee is currently a member of the IBM Consulting Group.

William Bret Wingert *IBM Federal Systems Company, 3700 Bay Area Boulevard, Houston, Texas 77058-1199.* Mr. Wingert joined IBM as an aerospace engineer in 1982 where he worked in the Space Shuttle Onboard Flight Software Guidance, Navigation, and Control (GN&C) Requirements Analysis and Performance Verification organization until 1988. In 1989, he worked on various upgrade strategies for onboard avionics and ground systems. He also investigated ways to reuse shuttle software technology on other manned and unmanned launch systems. In 1991, he began managing the Space Shuttle Onboard Flight Software GN&C organization. He currently acts as a consultant to various internal and external organizations on testing and requirements analysis.

Reprint Order No. G321-5532.