Software Development - Process and Implementation: DOD-STD-2167A vs. Traditional Methodologies

Lt James W. McCord
System Avionics Division
Air Force Avionics Laboratory

Abstract

Successful software development is crucial to the acquisition and development of new weapon systems. Computer technologies are rapidly advancing and, as a result, weapon systems are becoming more dependent on computers and computer software. Present software engineering and software development techniques have fallen short of meeting the demands of mission critical software development. This paper addresses the current software development standard for the Department of Defense (DOD), DOD-STD-2167A, and makes recommendations for improving the delivered software product.

DOD-STD-2167 and DOD-STD-2167A were developed to reduce the number of cost and schedule overruns and to increase the quality, reliability, and maintainability of the computer software developed for DOD weapon systems. In fact, these standards address many of the problems encountered in software development as identified by Fathi and Armstrong in 1985. These include:

1) Poor technical and organizational structure
2) Inadequate tools for management including the lack of a well defined review plan and the identification of measurable milestones
3) The development of unclear and/or unambiguous specifications as well as changing requirements
4) Incomplete or vague test plans
5) Lack of user involvement throughout the duration of the product development process

Figure 1

Introduction

Software development is a serious challenge for the program manager. No matter how big or small the system under development, when software is involved, the program often becomes unmanageable. As computer usage expanded in the 1960's and software development became increasingly critical to the performance of weapon systems, it was realized that a systematic process for software development was needed. Unfortunately, due largely to the rapid changes and developments in the computer industry, little progress was made in the development of this process until the late 1980's. With the release of DOD-STD-2167 on 4 June 1985, and the release of DOD-STD-2167A on 29 Feb 1988, it became apparent that the DOD had made a commitment to improving the process of software development.

681 U.S. Government work not protected by U.S. copyright.
This list indicated that there was a real need for an organized software development process. Wasserman, in 1981, stated that an organized development process would provide benefits such as:

1) Improved software reliability and quality
2) Improved visibility for both management and the user by providing a well defined schedule with reviews and measurable milestones
3) Increased user satisfaction and confidence
4) Reduced software development and maintenance costs through improvements in software structure and documentation
5) Well-defined organizational responsibilities and better management control through the various phases of the project
6) A system which is more comprehensible and easier to maintain.

The standards implemented by the DOD offer many of these benefits; however, it isn't easy to explain why there are still significant cost overruns (C-17) and serious integration problems (B-1B). This paper will address some of the possible reasons why standards alone are not sufficient to solve the software problem.

The following sections will address the software development process as described by DOD-STD-2167A. This standard will be compared with a generic commercial software development process as described by Charette. These processes will be compared and suggestions on how to improve both the implementation of DOD-STD-2167A and the process described in DOD-STD-2167A will be made.

DOD-STD-2167A

The software development process as defined by DOD-STD-2167A is shown in Figure 1. This eight step process begins with System Requirements Analysis/Design and ends with System Integration and Test. From these two steps it is apparent that the software development process is closely associated with system development. The first and last steps, System Requirements Analysis/Design and System Integration and Test were not present in DOD-STD-2167.

The following paragraphs are brief descriptions of the phases involved in the software development process as defined by DOD-STD-2167A. As you read these paragraphs, don't feel discouraged if you don't understand the details. DOD-STD-2167A is a short standard but it is very confusing. The point that is trying to be made is that although this process is very structured and at time confusing, and it seems as though nothing has been left out, it lacks several key points which often makes it inappropriate for the development of computer software. These points will be addressed later in the paper.

This is the first step of the process described in DOD-STD-2167A. This step actually contains two parts, System Requirements Analysis and System Design as shown in Figure 2. The System Requirements Review (SRR) is conducted upon completion of System Requirements Analysis. The System Design Review (SDR) is conducted to evaluate the system design upon completion.

The results of this phase of the software development process should be 1) to allocate system requirements to hardware, software, and personnel, and 2) develop preliminary engineering, interface, and qualification requirements for each defined Computer Software Configuration Item (CSCI). The end product is a functional baseline on which the software requirements will be based.

During this phase (Figure 3) of the software development process, an analysis of the system specifications is performed. The system specifications are checked for adequacy, testability, understandability, validity, and completeness. Similar analysis is performed on the preliminary Interface Requirements Specifications (IRS) and the preliminary Software Requirements Specifications (SRS). The IRS and SRS are updated for integration into the preliminary design.
The final products resulting from this phase include the final Software Requirements Specifications (SRS) and the final Interface Requirements Specifications (IRS). The Software Specification Review (SSR) is conducted to evaluate the software specifications and, when approved, the allocated baseline is established. In short, the Software Requirements Analysis phase defines and analyzes a complete set of functional, interface, and qualification requirements for each Computer Software Configuration Item (CSCI).

Detailed Design

The Detailed Design phase (Figure 5) is used to extend the preliminary design to the Computer Software Unit (CSU) level. Each CSU will be designed and requirements, test cases, schedules, and stress tests for each CSU will be identified.

This phase of the DOD-STD-2167A software development process completes the software design to the lowest level, the CSU. The Critical Design Review (CDR) is performed to evaluate the updated design and the resulting documents produced in this phase.

Preliminary Design

The Preliminary Design phase (Figure 4) involves software design to the Computer Software Component (CSC) level. The software, design, and interface requirements are then defined for each computer software component. Testing requirements, including stress testing, for the integration of the computer software components are also established.

The Preliminary Design phase also identifies the qualification tests to be performed on each Computer Software Configuration Item (CSCI). The Preliminary Design Review (PDR) is performed near the end of this phase to review the work performed. In summary, the preliminary design phase develops a high-level design which reflects the specified system and software requirements.

Coding and CSU Testing

This phase (Figure 6) is where the actual coding at the Computer Software Unit (CSU) level takes place. Deliverable code is produced and tested. This code should be able to be regenerated and maintained by the user facility. Informal testing at the CSU level is conducted while procedures for conducting formal CSU and Computer Software Component (CSC) testing are defined.

No formal review is conducted during this phase of the development process. However, this phase requires close monitoring by the acquiring organization. The Coding and CSU Testing phase is where coding will officially begin. Each CSU will be coded and tested, at least informally, during this phase.

CSC Integration and Testing

The end result of this phase (Figure 7) is an integrated software product. The Computer Software Units (CSUs) are integrated into a complete software item. Integration tests are performed and recorded. Software Test Descriptions (STD) are completed for each Computer Software Configuration Item (CSCI).

The Test Readiness Review (TRR) is performed at or near the end of this phase. This review is performed to ensure that the integrated software has been properly integrated and tested and is ready for CSCI testing.
Summary of DOD-STD-2167A

DOD-STD-2167A is presented, in short, above. The process, although simplified for this paper, is still rather confusing to the computer novice. In fact, the process is rather confusing to the software and computer expert. This presents a major problem in the development of computer software. Vast amounts of time and man(woman)hours are spent in just fulfilling the requirements set under DOD-STD-2167A. I don't claim that structure is bad, in fact it is often necessary, but just maybe this standard overdoes it. In order to support this statement, a typical non-DOD-STD-2167A process is presented. Afterwards, the comparison of the processes and the problems with DOD-STD-2167A will be presented.

CSCI Testing

This phase (Figure 8) of the DOD-STD-2167A process should be conducted by independent test personnel and test recorders. This minimizes subjectivity in the evaluation process. Formal qualification testing is performed and recorded in this phase. Any deficiencies should be corrected and the tests performed again. Tests should prove that the software satisfies all specified requirements.

The Functional Configuration Audit (FCA) and the Physical Configuration Audit (PCA) are often conducted upon completion of this phase. However, these may be postponed and conducted after the System Integration and Testing Phase.

System Integration and Test

The objective of this phase (Figure 9) is simple. The Computer Software Configuration Item(s) (CSCI) must be integrated and the system validated to ensure that the complete system is properly integrated and satisfies the system requirements. Any problems in System Integration and Test may require the repeat of some or all of the phases of the DOD-STD-2167A software development process.

Typical Software Development

The goal of any software development effort is to build a system that performs its intended function adequately at a reasonable cost. The following software development process (Figure 10) identifies the major steps involved in order to achieve this goal. This process is defined by Charette (1986).
Requirements

The requirements phase of the software development process is actually a refinement of activities performed prior to beginning development. During this phase, the software product description (a description of performance features and characteristics), obtained through previous steps, is converted into a functional description. This functional description should provide a good understanding of what the software is required to do to meet the user's needs.

The requirements phase should be the result of a consensus among the people involved in the development of the system. A consensus between management, developers, users, and maintainers will prevent conflicts and misunderstandings later in the development process, promote teamwork, and minimize changes in the software's requirements.

Specifications

The software specification phase identifies the tasks to be performed in the development phase. This phase focuses on and clarifies the intended behavior of the software product as seen from an external viewpoint.

The software specification phase is intended to simplify the design phase by developing a more precise, understandable description of the requirements document(s). Specifications must possess the following characteristics to be effective:

1) Correct and Complete - Incomplete and/or incorrect specifications make effective design and implementation impossible
2) Consistent - Specifications which are contradictory will lead to confusion in design and development
3) Unambiguous - Specifications must be described in detail in order to avoid confusion or misinterpretation.
4) Verifiable - The specification must meet system requirements. The end product must also be testable against the specifications.
5) Traceable - Each requirement must have a path defined to the specification.
6) Minimal - Only critical information should be included in the specification.
7) Modifiable - The specifications must be modifiable when changes occur.

Since specifications are the standard for measuring the correctness of the software product, they must possess all of these characteristics to be considered effective.

Design

During the design phase, the software specifications are developed into a software architecture which represents the physical implementation of the software product. This software architecture is virtually a blueprint to be followed during the implementation phase. There are two activities performed during this phase. These are the external and internal design.

External design is the definition of the externally observable characteristics of the software product. These two definitions compose the architectural design. Architectural design defines the program structure and the interfaces and connections among the software modules. Detailed design defines and selects the algorithms and data structures to be used in the implementation phase.

Implementation

The implementation phase results in the production of the product structures, data structures, algorithms, and interfaces developed in the design phase. If the earlier steps in the system specifications phase have been conducted properly, this is a straightforward process. However, this is rarely the case.

The implementation phase is often chaotic and difficult for the project manager to track. This is due, largely, to the fact that even very effective requirements analysis, specifications, and design cannot predict all the problems or technical difficulties encountered in a major software development project. In reality, design errors lead to incorrect assumptions by the implementer which must later be corrected. Also, the intricacies of various computer operating systems and programming languages are often beyond the expertise of the product design team and, thus, are not considered in the design process. There are many types of errors that do not appear until the implementation phase. These include human errors, domain problems, logic errors, and translation problems.

Whenever an error is encountered, the implementer is faced with the task of making tough decisions on design and constraint tradeoffs. Most decisions are non-functional in nature making it difficult to predict the outcome of the decision. The tradeoffs in the implementation phase depend on how effectively it can be executed on the chosen computer system architecture. Several examples of some tradeoffs that the implementer must make are reliability, cost, maintainability, efficiency, and timeliness.

Test

The testing phase follows the implementation phase and attempts to locate and isolate physical and/or logical errors in the product. Both formal and informal testing is conducted. Informal testing is usually conducted during both the implementation and testing phase. This type of testing is usually performed by the implementer and is generally done on a modular basis. When the implementer completes a particular software module or routine, he/she then informally tests it to assure that it functions as intended.
Formal testing begins as each module is completed. This testing attempts to identify exactly what the module or subroutine is doing as compared to the specifications. Once all modules or subroutines are tested, the modules are connected and tested as a unit, thus, as the finished product. Various tests are conducted and the results compared against expected results.

Evolution

The evolution phase is often called operations and maintenance. This phase follows the product throughout the rest of its functional life. Product changes such as functional enhancements or increased performance are necessary to increase the short life of computer software. There are three basic types of evolution in software products. These are perfective maintenance, adaptive maintenance, and corrective maintenance.

Perfective maintenance is product enhancement. Suggestions or requirements by the user, designer, or researcher may identify the need for this type of maintenance. Adaptive maintenance is the planned modification of the product. These planned modifications usually are enhancements that were not considered feasible for the original product. Corrective maintenance refers to the modification of the product due to deficiencies or errors in the product after its release. The software product usually does not go back through the entire software development process when maintenance is performed. However, major changes may require the redevelopment of the software.

Comparison of Development Cycles

On the surface, the goals of these two software development cycles would seem the same— to produce a working, maintainable product. However, this really isn't the case. The life of a software product in the commercial world is short, in general, compared to that of the life of DOD software. As mentioned earlier, the goal of most commercial software development is to produce a system which performs its function adequately at a reasonable cost. This goal, although very good for commercial software development, is not quite as applicable for software development in the DOD. Lifecycle costs must be considered and short-run savings often cause long-run problems and system deficiencies. From this discussion, it is apparent that these development cycles are inherently different. Figures 11 and 12 illustrate the differences in the distribution of effort for DOD-STD-2167A and a typical software development process. Oddly enough, although the goals differ, from these figures it is apparent that there is little difference in effort over the life cycle of the products. Both processes spend about 70-75% of the effort or cost in maintenance (75% evolution for a typical process - 10% error correction + 60% enhancement for DOD-STD-2167A). Both processes use 6% resources for coding (implementation) and approximately 12% for design (design + requirements/specifications). Testing varies from 8-12% between processes.

Another major difference in the software development processes is where the actual software development cycle begins. In the commercial cycle, software development begins after the product (or system) has been defined. The software is then designed to meet the system requirements; thus, the software is considered a means to accomplish the product requirements. DOD-STD-2167A begins the software development process with System Requirements Analysis/Design. As mentioned earlier, this phase, as well as the last phase, is new to DOD-STD-2167A and did not exist in DOD-STD-2167. This indicates that the software product is not developed as a means to achieve system requirements but is an integral part of system design and development. In other words, DOD-STD-2167A takes a systems engineering approach to software development. As a result, software requirements are addressed early in system design. This is a difference driven by the variance in goals.

Although the process goals are different and the exact implementation of these processes differs, many of the steps are similar and there is little difference in the overall distribution of effort for the life-cycle of the software products.

![Figure 11](image1.png)

![Figure 12](image2.png)
Problems with DOD-STD-2167A

It has been shown that there is little difference between the life-cycle costs of software development under DOD-STD-2167A and the distribution of effort under a typical commercial software development project. This is somewhat surprising due to the fact that DOD-STD-2167A is a very structured approach to software development. The question remains: Why isn't there a significant improvement in life-cycle costs when using DOD-STD-2167A?

There are many possible reasons why DOD-STD-2167A falls short in gaining significant improvements over a typical process. Many experts argue that the most apparent is, of course, the application. Many DOD applications are time-critical. Therefore, the level of complexity is greater than that of the commercial world. This theory is invalid since many commercial software development projects are at the systems level and just as time critical.

One possible reason for the astounding life-cycle costs of software under DOD-STD-2167A is the documentation requirements. DOD-STD-2167A is a very structured process which goes to great lengths to make sure that every step of the development process is documented and recorded. This practice is extremely expensive and not always necessary. Careful tailoring of this standard eliminates much of the excess documentation and, therefore, negates this argument.

The chief problem in DOD-STD-2167A is not in the process itself. This paper has gone to rather great depths in, hopefully, proving this. DOD-STD-2167A is not a bad software development process. It offers structure and provides the documentation necessary for the maintenance of deliverable code. So, if the process is not the problem, what is? The Air Force's implementation of DOD-STD-2167A is the problem.

Air Force personnel, in general, are not technically oriented. Many have technical backgrounds; however, once on the job, they are forced down an administrative/managerial path where technical skills are quickly lost and outdated. As a result, project/program managers and engineers are forced to take a "checklist" approach to contract monitoring. The responsible managers and engineers are, thus, only capable of monitoring the progress of the contract such as fund expenditures and milestones. They lack the expertise necessary to effectively evaluate the quality of the product (code) being produced. This is not to indicate that Air Force personnel must evaluate each line of code produced; however, Air Force personnel should be able to evaluate code when necessary and be technically competent enough to discuss the details of the software being produced.

The personnel in the Air Force are not to blame for this shortcoming. Most are overworked and cannot afford the time to take both the managerial and the technical courses necessary to effectively manage a project. For this reason, the Air Force emphasizes the managerial (acquisition) aspects of the job.

Air Force policy would have to change to make technical abilities a priority. This is not likely to happen soon. However, steps can be taken to improve the situation.

The Air Force can easily change the way software acquisition is taught. Recently, at the System Acquisition School's Computer Resource Acquisition Course, a guest speaker (a Colonel in charge of a major system acquisition) stated that Air Force personnel did not need to know anything about software to effectively acquire a software product. It is this type of thinking that must be overcome to solve the problems in implementing DOD-STD-2167A.

Solutions and Recommendations

In order to effectively develop and acquire computer software under DOD-STD-2167A, the Air Force must place more emphasis on the technical aspects of computers and software. More emphasis must be placed on the quality of the product. In order to do this, more emphasis must be placed on the abilities of Air Force personnel to effectively evaluate this software. This can be accomplished by two means. Number One - the Air Force should not impede or discourage technical excellence. Historically, military members which have elected to remain in technical fields have not fared as well as their management-oriented counterparts. The Air Force must encourage technical excellence in all personnel involved in the acquisition of computer software. Number Two - the Air Force must overcome its shortsided thinking. The fact that the DOD-STD-2167A is followed to the letter does not guarantee that the software delivered will be good quality software. In order to ensure that the software is of good quality and that it meets Air Force standards and system specifications, technically competent and fully qualified Air Force engineers should follow the development at lower levels.

Summary

DOD-STD-2167A is a valid, well-designed process for software development. However, following this process does not guarantee success. The Air Force must strive to manage not only the process, but also the quality of the software. In order to accomplish this, Air Force personnel must strive to maintain high levels of technical competence in the software field. DOD-STD-2167A is a process by which quality software can be obtained. In order to ensure the acquisition of quality software, the process must be implemented correctly by technically oriented personnel.

References

