Issues Related to the Explication of Process-Product Relationships in DoD-Std-2167 and DoD-STD-2168

R. Gregory Lavender

TR 88-36

Issues Related to the Explication of Process-Product Relationships in DoD-STD-2167 and DoD-STD-2168*

Technical Report SRC-88-006

31 March 1988

R. Gregory Lavender

Systems Research Center
and
Department of Computer Science

Virginia Polytechnic Institute

Blacksburg, Virginia 24061

*Work supported by IBM Federal Systems Division through the Systems Research Center under grant number 284291-WF.

Issues Related to the Explication of Process-Product Relationships in DoD-STD-2167 and DoD-STD-2168

R. Gregory Lavender

Systems Research Center and Department of Computer Science
Virginia Polytechnic Institute

Abstract

This paper is a discussion of issues related to the thesis entitled "The Explication of Process-Product Relationships in DoD-STD-2167 and DoD-STD-2168 via an Augmented Data Flow Diagram Model."

In particular, the major results of the above thesis are viewed in light of the draft standards DoD-STD-2167A and DoD-STD-2168 (both dated 1 April 1987), and the issue of *development objectives* is explored.

The ideas presented in this paper represent the author's opinion and are speculative in nature due to the fact that, at present, the revised DoD standards are in draft form, and the issue of development objectives has not yet been thoroughly investigated.

1. Introduction

This report focuses on issues that came about following the research reported in the thesis entitled "The Explication of Process-Product Relationships in DoD-STD-2167 and DoD-STD-2168 via an Augmented Data Flow Diagram Model" [LAVR88]. Since this report draws heavily on the ideas and results presented in this thesis, the reader is advised to review the referenced work before proceeding.

The two major issues resulting from the aforementioned research are:

1. how do the current draft standards DoD-STD-2167A [DODS87a], and DoD-STD-2168 [DODS87b] relate to the previous work done, and

2. what can be said about specific development objectives with regard to the results of the aforementioned thesis.

Section two of this report addresses the first issue by examining the major differences between the proposed draft standards and the current standards in light of the previous research. Furthermore, a preliminary assessment is made of the impact of both draft standards to previous results.

The discussion of the second issue is highly speculative since the notion of development objectives has not yet been fully researched. Section three offers conjectures on the relationship of development objectives to the Procedural Approach to the Evaluation of Software Development Methodologies [DANA87]. Since the issue of development objectives manifested itself at the conclusion of the research activity mentioned above, such conjectures are based solely on experience, intuition, and conclusions drawn from previous research.

1.1 Previous Work

In this section, the reader is briefly introduced to the fundamental ideas and relevant results of the author's thesis. The discussion in the following section focuses

on the differences between the June 1985 and April 1985 versions of DoD-STD-2167 and DoD-STD-2168 (respectively), and the April 1987 draft versions of both standards [DODS85a, DODS85b, DODS87a, DODS87b]. Previous work explicates the process-product relationships of DoD-STD-2167 (2167) and DoD-STD-2168 (2168) through a data flow analysis of both standards and related Data Item Descriptions (DIDs). A fundamental premise of this earlier work is that 2167 is *product oriented*. In other words, 2167 describe requirements for the production and acquisition of software and documentation and does not describe explicitly the development process.

Given that 2167 is product oriented, if one wishes to understand the processes by which products prescribed by 2167 are generated, one must first examine the product requirements in detail and infer the processes required to generate the products. As demonstrated by previous research, one can make a significant advance on this problem by adopting the perspective of the procedural approach to evaluating software development methodologies. This perspective is based on the notions of *objectives*, *principles*, and *attributes*. Furthermore, the evaluation procedure facilitates a bottom-up reasoning approach for inferring process requirements from product requirements through *linkages* which exist between objectives and principles, and principles and attributes.

The analysis of 2167 and 2168, based on the procedural evaluation approach, is facilitated by the use of an *Augmented Data Flow Diagram* (ADFD) model which offers a concise graphical tool that is extremely effective for capturing both explicit and implicit processes, products, and process-products relationships (data flow) found in both standards.

In the remainder of this report, references are made to ADFD analyses, previous results, and conclusions. The ADFD analyses pertain to sections or DIDs, described in 2167 and/or 2168, and are included in the previously referenced thesis. The results and conclusions are derived from these analyses. Hence, in order to fully appreciate the following discussions, it is imperative that the reader be familiar with the previous work.

2. The Draft Standards

In this section, the major changes that appear in DoD-STD-2167A (2167A) and DoD-STD-2168 (draft 2168), relative to the previous versions, are reviewed. Furthermore, a determination is of the impact on the results of previous work.

In the discussion that follows, references to the original versions of these standards are in the abbreviated form 2167/2168, and references to the draft standards are in the form 2167A/draft 2168.

Similar to the previous research, the focus is on the detailed requirements expressed in section five of 2167/2167A, and the requirements for software quality evaluation found in 2168/draft 2168.

2.1 DoD-STD-2167A

The most striking change one observes in 2167A is the brevity with which the requirements are stated. Such brevity is in contrast to 2167. However, the surface changes can be misleading.

Many of the detailed requirements expressed in section five of 2167 are requirements restated from DIDs. In 2167A, this redundancy is removed. As indicated by the previous work, the DIDs contain the detailed requirements from which one can infer processes needed to generate a product. When using 2167A, one must rely more heavily on the DIDs for determining both explicit and implicit requirements.

The following sections discuss the major content and requirements changes of 2167A with regard to the previous version of the standard and previous research.

2.1.1 The Software Development Life Cycle

As in 2167, the software development life cycle remains as the model of software development for 2167A. In 2167A, however, the life-cycle model is extended to include two additional phases not defined by 2167.

The first new phase described by 2167A is the *pre-software development* phase. If one examines the requirements for this added phase in the context of the ADFD analysis in the aforementioned thesis, one finds that 2167A has split the software requirements analysis phase of 2167 into two distinct phases.

By examining the prior ADFD analysis of the software requirements analysis phase of 2167, one can easily identify in this ADFD the salient requirements of the pre-software development phase of 2167A. There is one major change however; the Software Quality Evaluation Plan (SQEP) [DID10] does not occur.

The other major addition to the software development life cycle is the inclusion of a *system integration and testing* phase which follows the CSCI testing phase. The requirements for system integration and testing in 2167A represent the separation of the Functional and Physical Configuration Audits that are part of the CSCI testing phase described by 2167. Again, one may examine the ADFD analysis of the CSCI testing phase to confirm the existence of the audit processes.

So far, the changes in the software development life cycle are just a finer partitioning of the phases of software development. A major conclusion of the previous research is that the phases in the life cycle are concurrent and there are several *interaction points*. .These interaction points are the *developmental configuration*, the *testing configuration*, and the *life-cycle support specifications*.

The formal partitioning of the development life cycle is (to the developer) academic, since in reality no such boundaries exist. However, if one is more interested in the delivery of products (as in 2167 and 2167A), then distinct boundaries provide a focal point for acquiring the deliverables.

2.1.2 Software Quality Evaluation

The omission of a SQEP from 2167A is the first indication of a change in the manner in which software quality requirements are expressed in the draft standards.

One may also observe that requirements for product evaluations are included in 2167A. The requirements for product evaluations are based on a set of *product evaluation criteria.* The product evaluation criteria do not represent an addition to the overall requirements, but a clarification. The product evaluation criteria were originally in 2168 but have now been shifted to 2167A.

A result of the previous research is that there exists a *duality of software quality evaluation.* This duality derives from the application of software quality evaluation activities to not only products, but also the processes and methodologies by which products are generated. An interpretation is that the shift of product evaluation criteria from 2168 to 2167A confirms the claim that 2167 and 2167A are product oriented. The requirements for the evaluation of the software development processes, that were in the original 2167, have been removed.

2.1.3 Formal Software Testing

An implication derived from the ADFD analysis of the original 2167 is the need for a formal testing configuration. As explained above, the testing configuration is a primary focal point for interaction of development processes across phase boundaries.

The importance of a testing configuration as a formal configuration item is confirmed by requirements in 2167A. In the ADFD analysis presented in the referenced thesis, the testing configuration is represented as being an implicit requirement. In the context of 2167A, one can view the testing configuration as an explicit requirement.

2.1.4 Configuration Management

Another notable difference between 2167 and 2167A is the distribution of configuration management requirements across each of the development phases. In 2167, explicit configuration management requirements are confined to a specific section. As demonstrated by the ADFD analyses, configuration management underlies each of the development activities and is embodied by the formal review process that occurs at the end of each phase. One can suppose that the intent of 2167A is to emphasis the need for a configuration management process at each stage of development.

As just mentioned, the need for a testing configuration is implicit in 2167 and explicit in 2167A. However, the main configuration item of interest in both 2167 and 2167A is the developmental configuration. One should also consider including the testing configuration as a main configuration item given the emphasis in 2167A on formal testing.

2.1.5 Conclusion

Other than organizational differences and clarifications, the only significant difference between 2167 and the draft 2167A is the change in software quality requirements. Particularly, the deletion of the Software Quality Evaluation Plan and the shift of product evaluation criteria from the original 2168 to 2167A. The conjecture is that this shift is due to an emphasis on product acquisition, not product quality. That is, the evaluation criteria represent more of a checklist approach to software quality.

The ADFD based analysis of the original 2167 is applicable to the draft version 2167A. As discussed, many of the changes are surface changes. Since the ADFD analyses capture the process-product relationships, the effects on the analyses are minimal. One need only modify the ADFDs to reflect the changes in the software life cycle.

For example, the removal of the SQEP as a product of the software requirements analysis phase affects only the ADFD analysis for that phase since in subsequent

analyses, the SQEP is considered part of a *virtual* specification (i.e., the Software Development Plan (SDP) [DID30]).

The ADFD model provides an abstraction mechanism called a *virtual data store* which is used to represent the relationship of a process to a collection of products. Hence, the syntax of the ADFD analysis is not affected by such a change, only the semantics associated with the SDP must be modified.

The implication is that one can view the draft 2167A in light of the ADFD analysis of the original 2167 and, with minor semantic modifications, have a clear indication of the requirements of 2167A.

2.2 DoD-STD-2168

The draft 2168 represents a major revision to the previous version of the software quality evaluation standard. The original standard is based on the software development life cycle, and the detailed requirements for software quality evaluation parallel the detailed development requirements of 2167.

In the draft 2168, one finds the reason for the removal of the SQEP (mentioned above) from the definition of software development plans during the pre-development phase of 2167A. The SQEP is replaced by a new DID called the Software Quality Program Plan (SQPP) [DIDXX].

Having noted earlier that product evaluation requirements have shifted from the original 2168 to 2167A, one can now seen that a previous development requirement has been shifted from 2167 to the draft 2168. Intuition suggests that an attempt is being made to more explicitly distinguish a separation of concern. In the following sections, the claim is made that the original 2167 and 2168 standards embody this separation of concern implicitly, and the ADFD based analysis of 2168 represents this implicit relationship.

2.2.1 The Duality of Software Quality Evaluation

A fundamental result of the previous research is that of a duality of software quality evaluation. Through the ADFD based analysis of 2167, there is an implicit relationship between the development processes and the software quality evaluation process. As explained in the referenced thesis, the ADFD analysis of 2167 fails to capture this relationship explicitly.

In the ADFD analysis of 2167, the argument is advanced that it is difficult to represent the relationship of the software quality evaluation process to any particular development process. The reason given is that software quality evaluation process "underlies" the software development process. In the ADFD analysis of 2168, the relationship of the software quality evaluation process to the development process is explicated by developing a higher level of abstraction from which one can view this relationship. The result is an explicit representation of the dual nature of software quality evaluation.

What emerges from the abstraction is a clear representation of a software quality process that has two levels of concern. The first is the evaluation of the products (i.e., the developmental configuration, the testing configuration, and the life-cycle support specifications), generated by any particular development process, based on product evaluation criteria.

Separate from the product evaluation process there exists a process concerned with the evaluation of the overall development process. The evaluation of a development process relies on *software quality factors* that are defined as part of the Software Requirements Specification (SRS) [DID25].

Considering the changes in 2167A and the draft 2168, the implication is that 2167A has the requirements for product evaluation and the draft 2168 has the requirements for process evaluation. The ADFD analysis of the original 2168 captures exactly this separation of concerns. The indication is that 2167A and the draft 2168 represent more explicitly the intent of the earlier versions of these standards.

3. Development Objectives

In this section the issue of development objectives is examined. Development objectives are in contrast to the set of objectives defined by the procedural approach to the evaluation of software development methodologies, as discussed in the referenced thesis

The objectives defined below are defined by the evaluation procedure and correspond to the software quality factors mentioned in the previous section:

1. Adaptability - the degree to which software can adapt to change.

2. Correctness - the degree to which software adheres to specified requirements.

3. Maintainability - the degree to which software corrections or modification can be made.

4. Portability - the degree to which software can be transferred to another environment.

5. Reliability - the degree of error-free performance over time.

6. Reusability - the degree to which software can be used in other applications.

7. Testability - the ability to evaluate conformance with requirements.

A conclusion of the previous research indicates that one may be able to define objectives for development that differ from the objectives listed above. In order to distinguish between the two types of objectives, the ones listed above are called *system objectives*

Development objectives are embodied in the procedures and methods used during a particular development process. A requirement exists in 2167 for defin-

ing these procedures and methods (i.e., methodologies), used during each phase in the development life cycle, in the Software Standards and Procedures Manual (SSPM) [DID11]. Hence one would expect to find development objectives defined in the SSPM. System objectives apply to the entire development process and are defined in an SRS for a particular software configuration item.

The following is a list of particular development objectives that one may have for software development:

1. more design at the outset of software development,
2. less testing during software development,
3. more documentation,
4. self-documenting software,
5. interface driven design, and
6. separation of development and integration.

In the remainder of this section, speculation is offered on the significance of each of these development objectives to the software development processes prescribed by 2167.

3.1 More Design

Current development strategies focus more on the early design stages since a common belief is that more attention applied to design tasks implies less time spent in development, integration, and testing. Such a statement defies formal proof, yet many practitioners can attest to the success of this strategy.

A manifestation of the objective of incorporating more design into the development process is the *prototype implementation*. It is not the intent of this paper to

argue the pros and cons of prototypes. The point of view is adopted that a prototype is a *result* of having (either explicitly or implicitly) more design as a development objective. As an example for discussion, however, the prototype implementation is useful since it is so well known.

Consider that, throughout the software development life cycle, requirements often change. The impact on cost and schedule due to a change in a critical requirement can be significant. Worse yet, a requirement may have been initially misunderstood or omitted. Even the most elegant, efficient, and cost effective design can be destroyed as the result of such events. Hence prototype implementations are often used as a vehicle for proof of concept and requirements verification. Typically, the developer must perform some portion of the overall software system design at the outset. However, a conjecture is made that, because of various constraints (e.g., cost), early design activities do not occur under the same guidelines as do full-scale development activities. The claim is made that pre-development design activities should be governed by the development methodologies similar to those used during full-scale development.

The methodologies and tools required for pre-design activities (such as a prototype implementation) can be defined in the context of the meta-methodology prescribed by 2167. In fact, doing so is in concert with the new requirements found in 2167A (section 5.1).

The SSPM is the vehicle for defining the methodologies and tools applied to each phase of software development. The addition of a pre-development phase to the life cycle prescribed by 2167A implies that one should address separately the methodologies and tools to be applied at this stage.

3.2 Less Testing

The objective of less testing is closely related to the objective of more design. Consider the types of software testing that typically occur during the software development life cycle: unit testing, CSC testing, and CSCI testing.

In terms of cost (say for resources involved) one might argue that unit testing is less expensive than CSC testing, which is less expensive than CSCI testing. The objective of less testing might then be realized by employing better unit testing procedures and spending more time during unit testing. This incremental strategy may result in less overall time spent in testing. Such a strategy make sense if one has designed and built software in a top down manner. That is, intuition suggests that an incremental design and build should be incrementally tested.

In terms of the requirements of 2167, the analysis of the coding and unit testing phase suggests that a formal review of the unit testing process be incorporated even though none is required by 2167. Hence, if less testing is a development objective, then additional processes beyond those required by 2167 must be employed.

In addition to establishing a coding and unit testing formal review process, previous results demonstrate the importance of establishing a *testing* configuration as a separate configuration item.

The testing configuration represents a focal point of interaction across all phases of software development. In order to assess how well an objective such as less testing is being achieved, one must focus on the processes that have a relationship to the testing configuration (in the work done previously, the ADFD model identifies these processes).

3.3 More Documentation

The motivation for having the development objective of producing more documentation may be an indication of the desire to disseminate more information about the design and development activities. However, there is typically a substantial cost associated with maintaining documentation.

In the course of the design and development of a software configuration item, there are several levels of documentation involved. Using the requirements of 2167 as an example, one first develops software development plans (in particular

the SSPM), an allocated baseline (requirements), a developmental configuration, a testing configuration, and life-cycle support specifications. Ideally (but not usually), all documentation is kept up to date and reflects the latest requirements and design. In theory, at any point during the development, one can establish the traceability of a particular function of a component back to the requirements via this paper trail.

In addition, the documentation provides a focal point for interaction. The formal review processes that accompany the phases of software development rely on the documentation to assess how well the requirements are understood. Hence more documentation would seem to imply that the developer has a better understanding of the requirements. However, one must balance the usefulness of the documentation against the cost of maintaining the documentation, especially in the later stages of the development life cycle.

One of the results of the previous research is that software development files (SDFs) typically represent the current state of development of a particular software component. It is often the case that resources are not available to maintain the previous development documents in light of a large number of requirements and/or design changes. Since the SDFs are the repository for much of the detailed design information, it is not unusual for the SDFs to eventually become the final product specification. In fact, the draft 2168 standard makes note of this fact by requiring, as part of software quality evaluation, a quality assessment of the SDFs.

The issue of more documentation is dependent on the particular environment in which software development is occurring. If the resources are available to develop and maintain the documentation, then more documentation may prove to be useful for disseminating more information. In an environment characterized by continual requirements and design changes, maintaining additional documentation may be cost prohibitive.

3.4 Self-Documenting Software

Recent emphasis has been given to the issue of developing maintainable software. A reasonable assumption is that self-documenting software can contribute to the

maintainability system objective. However, this assumption may be naive for applications in which significant technical expertise is required to understand the software.

If one could achieve self-documenting software, a significant advance could be made with regard to establishing the quality of a software product. Consider that one of the goals of software quality evaluation is to establish the correctness of a particular software component. Establishing whether or not a software component is correct relies on the ability to trace a function of the software to a particular requirement.

The procedural approach to the evaluation of software development methodologies offers a perspective that would allow one to establish requirements traceability. The procedural approach, as discussed in the referenced thesis, adopts a unique perspective based on the idea that one can identify attributes in software. Linkages exist between attributes and principles, and principles and objectives.

The objective of self-documenting software may be more explicitly defined as ensuring that attributes are readily identifiable in the software. The perspective of the procedural evaluation then offers a framework for assessing the quality of the software.by tracing linkages from an attribute to system objectives (requirements).

3.5 Interface Driven Design

One of the perceived deficiencies of 2167 is the lack of explicit detailed interface design requirements. The meta-methodological nature of 2167 allows a developer to adopt methodologies and tools that can be used to facilitate an interface driven design. Experience has shown that the interface points among software components are where many integration discrepancies occur.

One may argue that the objective of producing and interface driven design is both advantageous and disadvantageous. Consider an interface driven design that requires global conformance to an interface definition. For example, a centralized

task manager which requires that all interfacing tasks conform to a given interface. This design approach may alleviate interface definition conflicts, but at the same time create a bottleneck since all tasks must use the task manager.

In practice, there are alternatives to the previous example; however, the claim is made that having the development objective of an interface driven design leads one to consider implementation alternatives such that the development objective is eventually achieved. For example, given the same development objective, a more elegant approach is to define interface protocol sets. This approach has proved to be widely successful in the implementation of heterogeneous communications networks, such as the ARPANET. The advantage to the protocols approach is that there is a strict interface definition to which all tasks must adhere, yet there are few, if any, internal constraints. Each interfacing task has a virtual interface; hence, the physical interface can be implemented separately, thus reducing the likelihood of inconsistent interfaces.

The development objective of an interface driven design can be realized by defining principles that one could use to achieve this objective. As just described, abstraction (in the form of a virtual interface) appears to be a key to achieving an interface driven design. Mechanisms for achieving abstraction exist in many forms; the most obvious being functional decomposition. One could possibly define others.

If one accepts the meta-methodological characterization of 2167, then one can augment the methodologies and tools defined in the SSPM such that the development methodologies embody principles that lead to the objective of an interface driven design. Further one can define define specific tools to facilitate the process by which the software and documentation are produced. Confirmation that the objective is met can be deduced by defining the attributes one would expect of the products, and then examining the products in light of these attributes.

3.6 Separation of Development and Integration

It may be advantageous to separate the software development activities from the software integration activities. A perceived benefit is that the development team

is required to submit the product for peer review, which may result in a higher degree of quality in the product. In effect, psychological pressure is applied to the development team which may induce a competitive attitude, leading to higher quality output by the team. This statement is highly speculative and is based solely on intuition and experience.

On a more concrete foundation, the ADFD analyses of the previous research indicate that the phases of software development typically overlap. Furthermore, one can identify specific interaction points for development processes that occur concurrently. Once again, the interaction points are: the developmental configuration, the testing configuration, and the life-cycle support specifications.

If one has the objective of separating the development activities from the integration activities, then the ADFD analyses offer a mechanism for identifying those processes required by 2167 which should be allocated to the development team and the integration team. In addition, the interaction points identified above represent the critical interface points.

Having separated the overall development process into development activities and integration activities, the next step is to implement a strong configuration management program as the mechanism for managing the interaction points.

Consider again the discussion of an interface driven design presented above. One could develop a metaphor in which the development team and integration team are separate tasks with a virtual interface. Extending this metaphor further, the configuration management activity then represents the physical interface. That is to say, the primary interface is between the development team and integration team. However, the configuration management activity underlies this interface and provides the mechanism that ensures the integrity of the interface.

This metaphor demonstrates a conclusion from the previous research: that configuration management underlies all of the development activities prescribed by 2167. Hence, the separation of development activities and integration activities within the context of 2167 is permissible provided one considers the importance

of configuration management in maintaining the integrity of the development process as a whole.

3.7 Conclusion

As previously stated, the discussion of the development objectives is highly speculative. However, having offered such speculation, there appears to be evidence to support the claim that development objectives can reasonably be defined in the context of the development process prescribed by 2167 (and 2167A).

In contrast to the system objectives defined earlier, the development objectives may be of interest only to the developer. However, a preliminary hypothesis is that through additional research, one could establish linkages from development objectives to system objectives similar to the linkages between principles and system objective defined by the procedural evaluation. The advantage of establishing the relationship of development objectives to system objectives provides one with the foundation for establishing a *methodology evaluation process* which is shown, in the ADFD analysis of 2168, to be an integral part of the evaluation of the software development process. Hence, development objectives appear to enhance one's ability to assess how well the development process is being performed.

As suggested in the referenced thesis, additional research is need before the relationship of development objectives to methodology evaluation is completely understood. The suggestion is that one use the previous research as the foundation for beginning such an activity, using an actual development project as a case study.

References

DANA87 Dandekar, Ashok V., "A Procedural Approach to the Evaluation of Software Development Methodologies," *Masters Thesis*, Virginia Polytechnic Institute and State University, September, 1987.

DID10 DI-MCCR-80010, *Software Quality Evaluation Plan*, United States Department of Defense, 4 June, 1985.

DID11 DI-MCCR-80011, *Software Standards and Procedures Manual*, United States Department of Defense, 4 June, 1985.

DID25 DI-MCCR-80010, *Software Requirements Specification*, United States Department of Defense, 4 June, 1985.

DID30 DI-MCCR-80030, *Software Development Plan*, United States Department of Defense, 4 June, 1985.

DIDXX DI-MCCR-8XXXX, *Software Quality Program Plan - Proposed Draft*, United States Department of Defense, 1 April, 1987.

DODS87a DoD-STD-2167A, *Defense System Software Development - Draft*, United States Department of Defense, 1 April, 1987.

DODS87b DoD-STD-2168, *Defense System Software Quality Program - Draft*, United States Department of Defense, 1 April, 1987.

DODS85a DoD-STD-2167,*Defense System Software Development*, United States Department of Defense, 4 June, 1985.

DODS85b DoD-STD-2168, *Software Quality Evaluation*, United States Department of Defense, 26 April, 1985.

LAVR88 Lavender, Robert G., "The Explication of Process-Product Relationships in DoD-STD-2167 and DoD-STD-2168 via an Augmented Data Flow Diagram Model," *Masters Thesis*, Virginia Polytechnic Institute and State University, March, 1988.