

An Alternative to Technology Readiness Levels for Non-Developmental Item (NDI) Software

James D. Smith II

Carnegie Mellon Software Engineering Institute
jds@sei.cmu.edu

Abstract

Within the Department of Defense, Technology Readiness Levels (TRLs) are increasingly used as a tool in assessing program risk. While there is considerable evidence to support the utility of using TRLs as part of an overall risk assessment, some characteristics of TRLs limit their applicability to software products, especially Non-Developmental Item (NDI) software including Commercial-Off-The-Shelf, Government-Off-The-Shelf, and Open Source Software. These limitations take four principle forms: 1) “blurring-together” various aspects of NDI technology/product readiness; 2) the absence of some important readiness attributes; 3) NDI product “decay;” and 4) no recognition of the temporal nature of system development and acquisition context. This paper briefly explores these issues, and describes an alternate methodology which combines the desirable aspects of TRLs with additional readiness attributes, and defines an evaluation framework which is easily understandable, extensible, and applicable across the full spectrum of NDI software.

1. Introduction

Technology Readiness Levels (TRLs) have been used within the National Aeronautics and Space Administration (NASA) as part of an overall risk assessment process, since the late 1980s. By the early 1990s, TRLs were routinely used within NASA to support technology maturity assessments and comparisons of maturity between different hardware technologies [1, 2].

Within the United States Department of Defense (DoD), there has been considerable interest in using TRLs as part of risk assessments for entire systems, including both hardware and software. Current DoD guidance requires technology readiness assessments prior to entering System Development and Demonstration; TRLs are one approach to meeting this requirement [3]. The Air Force Research Lab has adapted the NASA TRLs for use in assessing the readiness of critical technologies for incorporation into weapon systems, and the Army Communications Electronics Command (CECOM) has developed a draft set of TRLs to support software technology management [4, 5].

Several sources cite the difficulties in applying TRLs to assess the readiness of software-based technologies and products [5, 6]. Some of the characteristics of TRLs that limit their utility in assessing Non-Developmental Item (NDI) software product (COTS: commercial-off-the-shelf, GOTS: government-off-the-shelf, OSS: open-source software) readiness are discussed in more detail in the following sections.

2. Relationship between quality and readiness

Understanding the need for an alternative to TRLs first requires an understanding of what is meant by “readiness.” Readiness, as used in this report, is a measure of the suitability of a software technology or product for use within a larger software-intensive system *in a particular context* (e.g., development of a management information system or sustainment of a deployed tactical information processing system). In other words, the readiness of the software product or technology reflects some measure of the risks of using it in the larger system: higher readiness denotes lower risk; lower readiness, higher risk. This can best be illustrated through the use of a recognized quality model, such as ISO/IEC 9126-1 (Software engineering—Product quality—Part 1: Quality model) [7]. In this model, software quality is defined in terms of six external and internal quality characteristics, each with a number of sub-characteristics and four “quality in use” characteristics. Readiness, then, can be thought of as representing some non-linear combination of these characteristics and sub-characteristics, in the context of a particular system.

It is important to note that “readiness” and “maturity”—though frequently used interchangeably—are not the same thing. A mature product may possess a greater or lesser degree of readiness for use in a particular system context than one of lower maturity. Numerous factors must be considered, including the relevance of the products’ operational environments (e.g., usage patterns, timeliness/throughput requirements, etc.) to the system at hand, product/ system architectural mismatch, as well as other factors that will be discussed later in this paper.

3. Understanding readiness in context

To better understand how context influences the determination of readiness in a software product or technology, a picture may be useful. In their paper, Hanakawa and colleagues model the knowledge growth experienced by an organization during software development. The resulting knowledge growth can be represented by a sigmoid (s-shaped) curve [8]. We can extend this model to a software-intensive system acquisition or development, and equate “knowledge” with some measure of system maturity, such as requirements satisfaction or technical performance measure improvement. We then find that a typical acquisition or development will mature slowly during initial concept exploration and technology development until some critical point is reached (e.g., fundamental science is understood or algorithms validated) at which point progress becomes more rapid. As a system moves towards greater maturity, and most—though probably not all—requirements are satisfied, progress tapers off. In Hanakawa’s model, the exact shape of this curve is dependent on the

- Statistical distribution of tasks (e.g., requirements to be satisfied or program milestones)
- Degree of task difficulty
- Knowledge/competence of the organization to perform the tasks
- Rate at which knowledge is accumulated through task performance

Thus, every acquisition and development will result in a unique “maturity profile,” as shown in Figure 1.

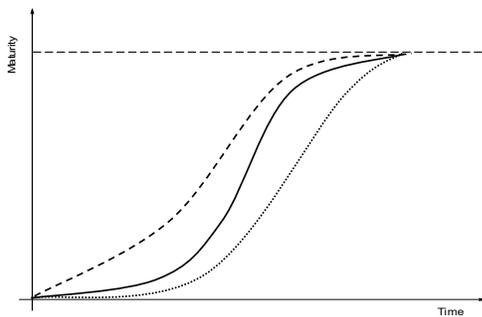


Figure 1. Maturity profile curves

These maturity growth curves provide some insight into the system development context, and permit an understanding of how the individual contributors to product or technology readiness vary in importance during the course of the program. For example, early in a program’s life cycle, the fact that a software technology or product is projected to be unsupported some-time during the system’s operational lifetime is probably of

much less significance than if that product or technology is so closely tied to the system’s architecture or implementation that replacing it would send you “back to the drawing board.” On the other hand, during the post-deployment sustainment phase, the impending retirement of a product or technology may become as important—or possibly more important—than how closely tied it is to the system’s design. The key to this approach is that, while the absolute values of the individual contributors to product or technology readiness cannot be defined, it is possible (in fact, it is necessary) to articulate the importance of one aspect (e.g., importance) relative to another, using “fuzzy” definitions like “as important as,” or “much less important than.” This provides the basis for the evaluation framework described later in this paper.

4. Limitations of TRLs

Given the origin of TRLs, it is unsurprising that organizations experience difficulty in using them to assess the readiness of software-based products. Characteristics of TRLs include the blurring, or blending together of multiple components of readiness; the lack of any built-in mechanism to deal with issues such as the “criticality” of a technology or product; NDI product “aging”; and varying sensitivities to different contributors to readiness experienced at different points in the development/acquisition life cycle. These characteristics complicate TRL use in assessing the readiness of NDI software products. These issues are discussed in more detail in the following sections.

4.1. Blurring contributions to readiness

One of the difficulties with using TRLs in programmatic and technical risk assessments is the manner in which TRL definitions combine several different aspects of, or contributors to, technology and product readiness. For example, CECOM’s draft software TRLs defines TRL 7 as follows:

Represents a major step up from TRL 6, requiring demonstration of an actual system prototype in an operational environment... Algorithms run on processor of the operational system and are integrated with actual external entities. Software support structure is in place. Software releases are in distinct versions. Frequency and severity of software deficiency reports do not significantly degrade functionality or performance. VV&A completed [5].

Thus, TRL 7 combines aspects from across all the product external quality characteristics: for example, functionality (“Algorithms run on [the] processor of the

operational system and are integrated with actual external entities.”), maintainability (“Software support structure is in place.”), and reliability (“Frequency and severity of software deficiency reports do not significantly degrade functionality or performance”), as well as several quality-in-use characteristics. The manner in which these combine makes it difficult, if not impossible, to understand how any one aspect contributes to, or influences the overall readiness of the product or technology.

4.2 Product criticality

Just as importantly, TRLs leave out such considerations as the degree to which the technology is critical to the overall success of the system (including how difficult it would be to replace it, or assume some fallback posture, should the technology in question prove unacceptable), or the suitability of the technology in question to its intended use within the system.

Some attempts have been made to deal with this through the use of “correction factors” to adjust the TRL for a given technology if that technology is critical to the success of the system (as measured by the percentage of the total system capability provided by the technology in question), or the technical complexity of the technology. For example, a program may adjust a TRL downward by some amount if a particular technology or product comprises more than some threshold, measured as a percentage of the functionality of the system [9]. Other techniques include normalizing technology readiness to the relevant environment for the different life-cycle phases of an acquisition or development (e.g., for a laboratory “bench top” test, a product or technology with a TRL of 3 or 4 may be acceptable) [4, 5, 9].

4.3 Software aging

TRLs were designed to measure the maturation of technologies as a way to gauge their readiness for use in a specified context. In this view, a technology (e.g., as used in a spectrometer) that has been “flight proven” through successful operation in space would be evaluated as being at TRL 9. Absent any changes to the way in which the technology is employed, it remains at TRL 9.

Software, on the other hand, is continually changing. As noted by Basili, a COTS software product generally “...undergoes a new release every eight to nine months, with active vendor support for only its latest three releases” [10]. Furthermore, software ages as a result of maintenance activities. In their paper, Eick and colleagues discuss three mechanisms of maintenance-induced software aging:

1. “Span of changes,” which is shown to increase over time

2. “Breakdown of modularity,” which manifests loss of architectural integrity of the software

3. “Fault potential,” which indicates the probability that modification introduces new faults into the software [11].

Compounding these effects is the fact that a system developer using NDI software as part of a larger system has little or no control over the scope or timing of these changes. Similarly, other forms of NDI software (i.e., GOTS and OSS) experience analogous decay processes.

4.4 Readiness in context

The above-mentioned issues, coupled with the realization that context varies throughout the life cycle of a system, introduce a fourth problem area: different aspects of technology or product readiness contribute, in varying degrees, to system risk at different times, and for different types of acquisitions. For example, the fact that there is an “end of life” announcement for a product that is critical to a given system is probably more significant if the system is fielded and operational, than if the system is a laboratory prototype not intended for operational use.

5. An alternative approach

The previous section briefly outlined some of the issues related to using TRLs in assessing the readiness of NDI software products for use in a particular system. The remainder of this paper will introduce a new approach that addresses these issues, and show how this can complement and extend the current TRL process to provide greater insight into the technical and programmatic risks facing a program.

Given that the readiness of a software product reflects some combination of quality characteristics in a specific context, then reasoning about readiness requires the definition of some attributes of readiness. Addressing the issues raised in the previous discussions on TRLs, these attributes should

- Provide coverage of the quality attributes most important to determining readiness.
- Be “orthogonal.” In other words, one criterion should not be a function of another one.

While TRLs combine various quality aspects in a way that it is impossible to directly discern the contributions of any particular aspect to the overall readiness of a product or technology, they do provide useful insights into two key contributors to readiness:

1. Degree of functionality provided
2. Fidelity of the environment (to the intended operational environment) in which this functionality has been demonstrated

Other key contributors to readiness that are missing from the TRLs include product/technology criticality in the context of the system under consideration, and the effects of software aging. There are two aspects of aging that are of particular interest in this context: the maturity of a product or technology—which varies by its “domain” (e.g., COTS, GOTS, or OSS)—and its availability.

The remainder of this section will describe a set of proposed readiness attributes, with definitions for various “levels” within each attribute, and attempt to show how these attributes satisfy (or at least improve upon TRLs) the requirement for coverage of the salient quality characteristics. Orthogonality of these attributes, and a proposed evaluation framework, will be discussed in the next section.

5.1 Requirements satisfaction

This attribute, shown in Table 1, describes how well the requirements, including functional (e.g., throughput, accuracy, latency) as well as non-functional (e.g., reliability, maintainability) allocated to a given software product or technology are satisfied by it. For functional requirements, this includes not only how many requirements are satisfied, but also any provided functionality that is not required. As previously mentioned, this is one of the two attributes which is derived from the definitions of TRLs.

Table 1. Requirements satisfaction attribute

<i>Requirements (R)</i>	
<i>Evaluation</i>	<i>Definition</i>
<i>Ideal</i>	“Perfect” fit between requirements and product/technology capabilities. (This rarely occurs in practice.)
<i>Good</i>	Requirements satisfied, but there are some minor “fit” issues.
<i>Fair</i>	Deficiencies in one or more second/third tier requirements, with work-around possible.
<i>Limitations</i>	Deficiencies in one or more second/third tier requirements, with no work-arounds.
<i>Major Limitations</i>	One or more major requirements unsatisfied; system performance degraded.
<i>Unsatisfactory</i>	One or more major requirements unsatisfied with no work-arounds.

There are a number of techniques to determine the “fit” between the allocated requirements and the capabilities of a product or technology, including the “Risk Misfit” process described by Wallnau and colleagues and the “Gap Analysis” methodology described by Ncube and Dean [12, 13].

5.2 Environmental fidelity

This attribute (Table 2) describes how faithfully the environment in which the software product under evaluation has been demonstrated reproduces the target operational environment. This provides some insight into a product’s ability to satisfy the allocated requirements based on observed performance in another context.

Table 2. Environmental fidelity attribute

<i>Environment (E)</i>	
<i>Evaluation</i>	<i>Definition</i>
<i>Full</i>	Subject product/technology demonstrated through use in the actual operational environment under “fully-stressed” conditions
<i>Partial</i>	Use in a less than fully-stressed operational environment demonstrated.
<i>Simulation</i>	Use in a simulated operational environment demonstrated.
<i>Comparable</i>	Product/technology demonstrated through actual use in a comparable environment.
<i>Integration</i>	Software product integrated with other components in a development/integration environment.
<i>Standalone</i>	Product used in a standalone environment.

5.3 Product criticality

This attribute is concerned with the degree to which the target system is dependent upon, or inseparable from the product or technology. For example, if the system is architected and partitioned so that the only interface between a product under evaluation and the target system is a simple asynchronous messaging interface, then the criticality of the product to the system is probably minimal.

Table 3. Criticality attribute

<i>Criticality (C)</i>	
<i>Evaluation</i>	<i>Definition</i>
<i>Minimal</i>	At least one alternate product/technology can be easily substituted within the target system.
<i>Low</i>	At least one alternate can be substituted; reintegration required with minimal software changes.
<i>Moderate</i>	At least one alternate can be substituted; moderate reintegration required with pervasive software changes necessary.
<i>Strong</i>	Substitution possible; significant architectural and/or implementation changes required, limited to a single aspect or partition of the system.
<i>High</i>	Significant, wide-ranging architectural and/or implementation changes required; good candidate for re-factoring/re-design.
<i>Fixed</i>	No flexibility: any changes to the product/technology under evaluation would require a complete redesign of system.

On the other hand, if the system depends on some proprietary capabilities contained within the product for its correct performance, or the interface consists of numerous, complex application programming interfaces (APIs), then the ability of the system developer to substitute another product is diminished—and the

criticality of the product to the system is correspondingly greater. Attribute definitions are shown in Table 3.

5.4 Product aging

There are two aspects of interest to product “aging”: Availability, and maturity. These are discussed in more detail in the following sections.

5.4.1. Product availability. The Availability attribute (Table 4) provides some insight into this aspect by comparing a product’s lifespan with the requirements of the system under development. Is it available now? When needed? For how long? If it is being retired, has a replacement been announced?

Table 4. Product availability attribute

Availability (A)	
Evaluation	Definition
Lifespan	Product/technology available over the intended lifespan of the system under development. (This will almost <i>never</i> occur in practice.)
Probably	Available by system “need date,” but will probably be replaced during the system’s life.
End-Of-Life (EOL) With Replacement	Available by system “need date,” but product End-Of-Life (EOL) with replacement announced.
EOL Without Replacement	Available by system “need date,” but EOL without replacement announced.
Alternate	Not available by system “need date,” but suitable alternate exists in the interim.
Unavailable	Not available by system “need date,” and no suitable alternate exists.

5.4.2 Product maturity. The second aspect of product aging is the maturity of the software product or technology (Table 5). Unlike the case with the other attributes, there are several distinct modes, or domains, of NDI software with their own maturation mechanisms, each of which have differing implications to readiness.

While these three domains represent different maturation mechanisms, some rough equivalence across domains can be made.

6. Evaluation framework

We’ve seen that readiness is defined by multiple attributes and their importance relative to one another. Multi-Criteria Decision Making (MCDM) theory provides numerous methods for determining the optimal solution in the presence of multiple criteria. These methods fall into two broad classes: Multi-Objective Decision Making (MODM) and Multi-Attribute Decision Making (MADM), with subclasses based on the data used (i.e., deterministic, stochastic, or “fuzzy”) and the number of decision makers (i.e., single or group) [14].

Table 5. Product maturity attribute

Maturity (M)			
Evaluation	NDI Software Domain-Specific Definitions		
	Commercial-Off-The-Shelf (COTS)	Government-Off-The-Shelf (GOTS)	Open-Source Software (OSS)
Off-the-shelf	Widespread commercial use; available as COTS	System has achieved FOC	Product is in large-scale public use
Deployed	Limited or first commercial use	System at IOC	Product is in limited public use
System Test	Product undergoing public beta/release candidate testing	System in OT&E	Product undergoing public beta/release candidate testing
Subsystem/Component Test	Product in limited or private testing	System in DT&E	Product in limited or private testing
Prototype	“Engineering tool” (“opportunistic” re-use)	System in development	Product under development
Concept	Product announced (see “vapor-ware”)	System planned/budgeted, but not yet started.	Product announced, but not yet started

MODM applies to problems in which the decision space is continuous; MADM, in contrast, is used in decision problems with discrete decision spaces. The methodology described in this paper falls into the deterministic, single decision maker MADM class. Saaty’s Analytic Hierarchy Process (AHP), supported by several commercially available tools (e.g., Expert Choice), is recommended as an evaluation approach [15].

AHP defines a process for evaluating multiple criteria, using a hierarchical structure (i.e., goal, attributes and sub-attributes, and alternatives) and pair-wise comparisons to determine the alternative that best satisfies the desired goal. The use of ordinal values, such as “x is much more important than y” or “x has roughly the same importance as y,” works well in the context of software intensive system acquisition and development where cardinal values (e.g., “Criticality” = 7.5) cannot be defined with any degree of confidence. One issue with using AHP is that the evaluation criteria must be orthogonal for the results to be valid. In other words, one criterion cannot be dependent on another criterion. The criteria described in this report, on the other hand, do present at least the appearance of orthogonality: none of the attributes (i.e., criticality, requirements satisfaction, product availability, product maturity, and environmental fidelity) are expressed in terms of any other attribute, nor does the evaluation of an attribute imply anything about any other attribute.

While AHP provides a method to reason about the contributions of various attributes to satisfying a desired goal, neither AHP nor the approach described in this paper define how the relative rankings of the criteria are obtained. Just as the SEI Capability Maturity Model® (CMM®) framework leaves the definition of appropriate processes to the implementing organization, this

framework leaves the criteria evaluation definitions to the developing or acquiring organization.

7. Example application

To see how this could work in practice, a very simple hypothetical system provides the context within which a single software technology/product choice is examined. First, the evaluation is made using TRLs alone, then is repeated using the criteria and evaluation framework described in this paper. Finally, the results of the two approaches are compared.

In this system, two NDI software products are being considered for potential use. The first of these, "Product A," has the following characteristics:

- It is nearing deployment, and is currently undergoing release candidate testing.
- All of the system threshold requirements, and most of the objective requirements allocated to this component are satisfied. The missing functionality can be provided through the use of some operational workarounds.
- The product can be replaced fairly easily. That is, it is sufficiently decoupled from the rest of the system that changes in this product should not require changes elsewhere in the system.
- It is projected to be available by the "need date" for the system under development, but will probably have to be replaced sometime during the development system's life cycle.
- Its capabilities have been demonstrated in an environment that partially replicates the intended operational environment for the final system.

Similarly, "Product B" exhibits the following traits:

- It is currently available as a COTS product.
- All threshold requirements are satisfied. Most objective requirements are satisfied, but some functions are missing and cannot be satisfied through any combination of work-arounds.
- The integration of this product and the system would necessitate a moderate reintegration effort, with widespread—though relatively moderate—software changes to the target system if the product had to be replaced.
- The product is available now—and will be available when needed for the development system—but there has been an "end of life" announcement, with a replacement planned by its developer.
- Its capabilities have been demonstrated in the target system's intended operational environment.

7.1 TRL assessment

Applying CERDEC's draft software TRLs to evaluate the readiness of these two products, using the TRL calculator from AFRL, results in the following:

Product A: Evaluated as being at TRL 7

Product B: Evaluated as being at TRL 9

7.2 Alternative assessment

The first step in this process is to determine the relative importance of the criteria in context. The context for a given system development is determined by the interactions of many complex variables, and cannot be ascertained by the application of any "cookbook" or prescriptive process. Among the factors to be considered in this determination are

- Where is the system in its life cycle?
- What is the development program's risk tolerance?
- How important is it for the NDI product to be stable?

For this example, the relative importance of the criteria was determined to be as follows:

- Criticality (C) is slightly more important than Requirements (R). In other words, it is somewhat less important in this context that the product satisfies every requirement allocated to it, and more important that the system not be too dependent on any particular product choice.
- R is significantly more important than either Maturity (M) or Environmental Fidelity (E). This means that, for this stage in the system's development, it is much less important that the product be a true "off-the-shelf" product, or that it has been demonstrated in the intended operational environment, than it is for it to satisfy the allocated requirements or be easily replaced.
- M and E are, in turn, more important than Availability (A). This means that the likelihood that the product will be replaced during the life of the system is less important than its level of "productization," or how closely its demonstration environment matches that of the target system. These relations can be expressed as

$$C > R \gg \{M, E\} > A$$

In the AHP, this relation is converted into a pair-wise comparison matrix (PCM), where each entry in the matrix represents the comparison between the row attribute (X) and each column attribute (Y) using the values shown in Table 6 [13]:

Table 6. Pair-wise comparison matrix values

<i>X Compared to Y</i>	<i>X Preferred to/more important than Y</i>	<i>Y Preferred to/more important than X</i>
Equally preferred	1	1
Moderately preferred	3	1/3
Strongly preferred	5	1/5
Very strongly preferred	7	1/7
Extremely strongly preferred	9	1/9

The second step is to determine, within each of these attributes, the relative preference for the various levels within the context of a particular development. In this example, comparing the importance/desirability between a rating of “Minimal” for the Criticality attribute, and other levels within that attribute, resulted in a determination that “minimal” was

- Moderately preferable to “low“
- Strongly preferred to “moderate”
- Very strongly preferred to “strong”
- Somewhat more strongly preferred to “high” than to “strong”
- Extremely more preferable than “fixed”

Completing the remaining comparisons, these judgments are then converted to a PCM:

	Mi	L	Mo	S	H	F
Mi	3	5	7	8	9	
L		3	5	7	9	
Mo			3	5	7	
S				3	5	
H					3	
F						3

This process is repeated for the remaining attributes (i.e., R, A, M, and E).

The next step is to evaluate each candidate product against each readiness attribute, resulting in a separate PCM for the candidates for each attribute. As discussed in an earlier section, this methodology neither prescribes nor proscribes any particular evaluation techniques: each development program is unique, and the implementation of this approach must be tailored accordingly. In this example, evaluating both products against the Criticality attribute results in the PCM:

$$PCM_{Criticality} \begin{matrix} & A & B \\ A & \begin{bmatrix} 1 & 6 \end{bmatrix} \\ B & \begin{bmatrix} 1/6 & 1 \end{bmatrix} \end{matrix}$$

Similarly, the remaining product/attribute PCMs are calculated:

$$PCM_{Requirements} \begin{matrix} & A & B \\ A & \begin{bmatrix} 1 & 4 \end{bmatrix} \\ B & \begin{bmatrix} 1/4 & 1 \end{bmatrix} \end{matrix}$$

$$PCM_{Availability} \begin{matrix} & A & B \\ A & \begin{bmatrix} 1 & 2 \end{bmatrix} \\ B & \begin{bmatrix} 1/2 & 1 \end{bmatrix} \end{matrix}$$

$$PCM_{Maturity} \begin{matrix} & A & B \\ A & \begin{bmatrix} 1 & 1/6 \end{bmatrix} \\ B & \begin{bmatrix} 6 & 1 \end{bmatrix} \end{matrix}$$

$$PCM_{Environment} \begin{matrix} & A & B \\ A & \begin{bmatrix} 1 & 1/4 \end{bmatrix} \\ B & \begin{bmatrix} 4 & 1 \end{bmatrix} \end{matrix}$$

Finally, applying the AHP with these PCMs produces weighted scores for the candidate products as shown:

Product A: 0.654
Product B: 0.346

7.3 Comparison of results

From this extremely simple example, the effect of system and development context on product readiness is apparent. Using the existing (draft) software TRL definitions and the AFRL TRL calculator, Product B—which is available as a COTS product, and has been used in the target system’s intended operational environment—is determined to be at a higher degree of readiness than Product A (TRL 9 versus TRL 7). When context is taken into account—reflecting management and engineering estimations about the relative importance of the readiness attributes, as well as value judgments about preferences within each attribute—Product A is seen to have higher readiness.

8. Conclusions

While there is a growing body of evidence that using TRLs as part of an overall risk assessment can lead to an improved understanding of the technological and programmatic risks in a system development or acquisition, there are several difficulties in applying “traditional” TRLs to the evaluation of software technologies. This is especially true for NDI, including COTS, GOTS, and OSS, where TRLs neither provide any way to discriminate between mature technologies or products, nor take into account the inevitable decay which all software experiences. Finally, the existing TRL framework lacks any explicit mechanism to deal with various aspects of the system and development context, including the time-varying effects of the various contributors to technology and product readiness.

The methodology described in this paper provides an alternative to using TRLs for NDI software products and technologies that directly addresses these shortcomings. The methodology allows the evaluation criteria to be tailored to the particulars of any system development, including judgments about acquisition and development risk. As a result, a more nuanced determination of product or technology readiness is possible. On the other hand, considerably more effort is required to perform this evaluation than simply assessing TRLs.

So far, this methodology has not been applied to an actual system development and, thus, remains purely theoretical. It is planned, over the next year or so, to apply this to one or more case studies to see how well this approach is able to “predict the past.” After some refinement, it should then be possible to pilot this methodology in an actual system development.

9. References

1. Eisman, M. and Gonzales, D.: *Life Cycle Cost Assessments for Military Transatmospheric Vehicles*, <http://www.rand.org/publications/MR/MR893/>, 1997.
2. Mankins, J.: “Technology Readiness Levels – A White Paper”, <http://advtech.jsc.nasa.gov/downloads/TRLs.pdf>, 1995.
3. Department of Defense.: *Operation of the Defense Acquisition System*. [http://dod5000.dau.mil/DOCS/DoDI%205000.2-signed%20\(May%2012,%202003\).doc](http://dod5000.dau.mil/DOCS/DoDI%205000.2-signed%20(May%2012,%202003).doc), 2003.
4. General Accounting Office.: *Better Management of Technology Development Can Improve Weapon System Outcome*, <http://www.gao.gov/archive/1999/ns99162.pdf>, 1999.
5. Graettinger, C., Garcia, S., Sivi, J., Schenk, R. and Syckle, P.: *Using the Technology Readiness Levels Scale to Support Technology Management in the DoD’s ATD/STO Environment*. http://www.sei.cmu.edu/publications/documents/02_reports/02sr027.html, 2002.
6. Graettinger, C., Garcia, S. and Ferguson, J., *TRL Corollaries for Practice-Based Technologies*, <http://www.acq.osd.mil/sis/Conference%20Presentations/TRL%20Corollaries%20for%20Practice%20Based%20Technologies.pdf>, 2003.
7. International Organization for Standardization, *ISO/IEC 9126-1 Software Engineering—Product Quality—Part 1: Quality Model*, Geneva, Switzerland, 2001.
8. Hanakawa, H., Morisaki, S. and Matsumoto, K., “A Learning Curve Based Simulation Model for Software development”, *Proceedings of the 20th International Conference on Software Engineering*, IEEE Computer Society, Washington, DC, 1998, pp. 350-359.
9. Wong, B. “NASA Cost Symposium – Multivariate Instrument Cost Model-TRL (MICM-TRL)”, <http://ipao.lare.nasa.gov/symposium/MICM-TRL-Wong.pdf>, 2000.
10. Basili, V. and Boehm, B. “COTS-Based Systems Top 10 List”, *IEEE Computer*, IEEE, May 2001, pp. 2-4.
11. Eick, S., Graves, T., Karr, A., Marron, J. and Mockus, A. “Does Code Decay? Assessing the Evidence from Change Management Data”, <http://www.cs.umd.edu/class/spring2003/cmsc838p/Evolution/ecay.pdf>, 2001.
12. Wallnau, K., Hissam, S. and Seacord, R *Building Systems from Commercial Components*, Addison-Wesley, Boston, MA., 2002.
13. Ncube, C. and Dean, J. “The Limitations of Current Decision-Making Techniques”, *Proceedings of the First International Conference on COTS-Based Software Systems*, Springer-Verlag, New York, NY, 2002, pp. 176-187.
14. Tryantaphyllou, E., Shu, B., Sanchez, N. and Ray, T. “Multi-Criteria Decision Making: An Operations Research Approach”, *Encyclopedia of Electrical and Electronics Engineering*, John Wiley & Sons, New York, NY, 1998, pp. 175-186.
15. Saaty, T. *The Analytic Hierarchy Process*, McGraw-Hill, New York, NY, 1980.