A Quagmire of Terminology: Verification & Validation, Testing, and Evaluation

Valerie Barr
Department of Computer Science
Hofstra University
Hempstead, NY 11550
vbarr@hofstra.edu

Abstract
Software engineering literature presents multiple definitions for the terms verification, validation and testing. The ensuing difficulties carry into research on the verification and validation (V&V) of intelligent systems. We explore both these areas and then address the additional terminology problems faced when attempting to carry out V&V work in a new domain such as natural language processing (NLP).

Introduction
Historically verification and validation (V&V) researchers have labored under multiple definitions of key terms within the field. In addition, the terminology used by V&V researchers working with intelligent systems can differ from that used by software engineers and software testing researchers. As a result, many V&V research efforts must begin with a (re)definition of the terms that will be used. The need to establish working definitions becomes more pressing if we try to apply verification, validation, and testing (VV&T) theory and practice to fields in which developers do not normally carry out formal VV&T activities. This paper starts with a review of terminology that is used in the software engineering/software testing areas. It then discusses the terminology issues that exist among V&V researchers in the intelligent systems community and between them and the software engineering/software testing communities. Finally, it explores the terminology issues that can arise when we attempt to apply VV&T to other domain areas, such as natural language processing systems.

Terminology Conflicts – First View
The first term to tackle in the terminology of software testing is the term testing itself. Unfortunately this word is used to refer to several activities that take place at very different levels in the software development process. In one usage, the term refers to testing in the small, the exercise of program code with test cases, with a goal of uncovering faults in code by exposing failures. In another usage, the term refers to testing in the large, the entire overall process of verification, validation, and quality analysis and assurance.

The term V&V, for verification and validation, is also used in both high level and low level ways. In a high level sense, it is used synonymously with testing in the large. V&V can refer to a range of activities that include testing in the small and software quality assurance. More specifically, V&V can be used as an umbrella term for activities such as formal technical reviews, quality and configuration audits, performance monitoring, simulation, feasibility study, documentation review, database review, algorithm analysis, development testing, qualification testing, installation testing (Wallace & Fujii 1989; Pressman 2001). This is consistent with the ANSI definition of verification as the process of determining whether or not an object in a given phase of the software development process satisfies the requirements of previous phases (ANSI/IEEE 1983b), as cited in (Beizer 1990)). In this view, V&V activities can take place during the entire life-cycle, at each stage of the development process, starting with requirements reviews, continuing through design reviews and code inspection, and finally product testing (Sommerville 2001). In this sense, software testing in the small is one activity of the V&V process. Similarly, the National Institute of Standards and Technology (NIST, formerly National Bureau of Standards) defines the high level view of VV&T as the procedure of review, analysis, and testing throughout the software life cycle to discover errors, determine functionality, and ensure the production of quality software (NBS 1981).
Verification & Validation

In a low level sense, each of the terms verification and validation has very specific meaning and refers to various activities that are carried out during software development. In an early definition, verification was characterized as determining if we “are building the right product” (Boehm 1981). In more current characterizations, the verification process ensures that the software correctly implements specific functions (Pressman 2001), characteristics of good design are incorporated, and the system operates the way the designers intended (Pfleeger 1998).

Note the emphasis in these definitions on aspects of specification and design. The definition of verification used by the National Bureau of Standards (NBS) also focuses on aspects that are internal to the system itself. They define verification as the demonstration of consistency, completeness, and correctness of the software at each stage and between each stage of the development life cycle (NBS 1981).

Validation, on the other hand, was originally characterized as determining if we “are building the right product” (Boehm 1981). This has been taken to have various meanings related to the customer or ultimate end-user of the system. For example, in one definition validation is seen as ensuring that the software, as built, is traceable to customer requirements (Pressman 2001) (as contrasted with the designer requirements or specifications used in verification). Another definition more vaguely requires that the system meets the expectations of the customer buying it and is suitable for its intended purpose (Sommerville 2001). Pfleeger adds the notion (Pfleeger 1998) that the system implements all of the requirements, creating a two way relationship between requirements and system code (all code is traceable to requirements and all requirements are implemented). Pfleeger further distinguishes requirements validation which makes sure that the requirements actually meet the customers’ needs. These various definitions generally comply with the the ANSI standard definition (ANSI/IEEE 1983a) of validation (as cited in (Beizer 1990)) as the process of evaluating software at the end of the development process to ensure compliance with requirements. The National Bureau of Standards definition agrees in large part with these user-centered definitions of validation, saying that it is the determination of the correctness of the final program or software with respect to the user needs and requirements.

As other terms within software engineering are more carefully defined, there is a subsequent impact on definitions of V&V. For example, the “requirements phase” often refers to the entire process of determining both user requirements and additional requirements that are necessary for actual system development. However, in new texts on software development (for example (Hamlet & Maybee 2001)) this process is broken into two phases: the requirements phase is strictly user centered, while the specification phase adds the additional requirements information that is needed by developers. This leads to confusing definitions of V&V which necessitate that first the terms “requirements” and “specifications” be well defined. In (Hamlet & Maybee 2001) the issue is addressed directly by defining verification as “checking that two independent representations of the same thing are consistent in describing it.” They propose comparing the requirements document and the specification document for consistency, then the specification document and the design document, continuing through all the phases of software development.

Testing

We next return to various attempts in the literature to define testing. Most software engineering texts do not give an actual definition of testing and do not distinguish between testing in the large and testing in the small. Rather, they simply launch into lengthy discussion of what activities fall under the rubric of testing. For example, Pfleeger (Pfleeger 1998) states that the different phases of testing lead to a validated and verified system. The closest we get to an actual definition of testing (Pressman 2001) is that it is an “ultimate review of specification, design, and code generation”. Generally, discussions of testing divide it into several phases, such as the following (Pressman 2001):

- unit testing, to verify that components work properly with expected types of input
- integration testing, to verify that system components work together as indicated in system specifications
- validation testing, to validate that software conforms to the requirements and functions in the way the end user expects it to (also referred to as function test and performance test (Pfleeger 1998)).
- system testing, in which software and other system elements are tested as complete entity in order to verify that the desired overall function and performance of the system is achieved (also called acceptance testing (Pfleeger 1998)).

Rather than actually define testing, Sommerville (Sommerville 2001) presents two techniques within the V&V process. The first is software inspections which
are static processes for checking requirements documents, design diagrams, and program source code. The second is what we consider testing in the small, which involves executing code with test data and looking at output and operational behavior.

Pfleeger breaks down the testing process slightly differently, using three phases (Pfleeger 1998):

- testing programs,
- testing systems,
- evaluating products and processes.

The first two of these phases are equivalent to Pressman's four phases listed above. However, Pfleeger's third phase introduces a new concept, that of evaluation. In the context of software engineering and software testing, evaluation is designed to determine if goals have been met for productivity of the development group, performance of the system, and software quality. In addition, the evaluation process determines if the project under review has aspects that are of sufficient quality that they can be reused in future projects. The overall purpose of evaluation is to improve the software development process so that future development efforts will run more smoothly, cost less, and lead to greater return on investment for the entity funding the software project.

Peters and Pedrycz (Peters & Pedrycz 2000) present one of the vaguer sets of definitions. They define validation as occurring "whenever a system component is evaluated to ensure that it satisfies system requirements". They then define verification as "checking whether the product of a particular phase satisfies the conditions imposed at the beginning of that phase". There is no discussion of the source of the requirements and the source of the conditions, so it is unclear which step involves comparison to the design and which involves comparison to the customer's needs. Their discussion of testing provides no clarification as they simply state that testing determines when a software system can be released and gauges future performance.

This brief discussion indicates that there is a fair amount of agreement, within the software engineering community, on what is meant by verification and validation. Verification refers, overwhelmingly, to checking and establishing the relationship between the system and its specification (created during the design process), while validation refers to the relationship between the system's functionality and the needs and expectations of the end user. However, there are some authors whose use of the terms is not consistent with this usage. In addition, all of the key terms (testing, verification, validation, evaluation, specification, requirements) are overloaded. Every effort must be made in each usage to provide sufficient context and indicate whether a high-level or low-level usage is intended.

**V&V of Intelligent Systems**

The quagmire of terminology continues when we focus on the development of intelligent systems. As discussed in (Gonzalez & Barr 2000), a similarly varied set of definitions exists. Many of the definitions are derived from Boehm's original definitions (Boehm 1981) of verification and validation, although conflicting definitions do exist. It is also the case that, in this area, the software built is significantly different from the kinds of software dealt with in conventional software development models. Intelligent systems development deals with more than just the issues of specifications and user needs and expectations.

The chief distinction between "conventional" software and intelligent systems is that construction of an intelligent system is based on our (human) interpretation or model of the problem domain. The systems built are expected to behave in a fashion that is equivalent to the behavior of an expert in the field. Gonzalez and Barr argue, therefore, that it follows that human performance should be used as the benchmark for performance of an intelligent system. Given this distinction, and taking into account the definitions of other V&V researchers within the intelligent systems area, they propose definitions of verification and validation of intelligent systems as follows:

- Verification is the process of ensuring 1) that the intelligent system conforms to specifications, and 2) its knowledge base is consistent and complete within itself.

- Validation is the process of ensuring that the output of the intelligent system is equivalent to those of human experts when given the same inputs.

The proposed definition of verification essentially retains the standard definition used in software engineering, but adds to it the requirement that the knowledge base be consistent and complete (that is, free of internal errors). The proposed definition of validation is consistent with the standard definition if we consider human performance as the standard for the "customer requirements" or user expectations that must be satisfied by the system's performance.

Therefore, we can apply the usual definitions of V&V to intelligent systems with slight modifications to take into account the presence of a knowledge base and the necessity of comparing system performance to that of humans in the problem domain.
Applying V&V in a New Area

As shown, the area of VV&T is based on overloaded terminology, with generally accepted definitions as well as conflicting definitions throughout the literature, both in the software engineering field and in the intelligent systems V&V community. The questions then arise, how should we proceed and what difficulties might be encountered in an attempt to apply VV&T efforts in a new problem domain? In this section we discuss the difficulties that arose, and the specific terminology issues, in a shift into the area of natural language processing (NLP) systems.

Language, as a research area, is studied in many contexts. Of interest to us is the work that takes place at the intersection of linguistics and computer science. The overall goal (Allen 1995) is to develop a computational theory of language, tackling areas such as speech recognition, natural language understanding, natural language generation, speech synthesis, information retrieval, information extraction, and inference (Jurafsky & Martin 2000).

We subdivide language processing activities into two categories, those in which text and components of text are analyzed, and those in which the analysis mechanisms are applied to solve higher level problems. For example, text analysis methods include morphology, part of speech tagging, phrase chunking, parsing, semantic analysis, and discourse analysis. These analysis methods are in turn used in application areas such as machine translation, information extraction, question and answer systems, automatic indexing, text summarization, and text generation.

Many NLP systems have been built to date, both for research purposes and for actual use in application domains. However, the literature indicates (Sundheim 1989; Jones & Galliers 1996; Hirschman & Thompson 1998) that these systems are typically subjected to an evaluation process using a test suite that is built to maximize domain coverage. This immediately raises the questions of what is meant by the term evaluation as it is used in the NLP community, whether it is equivalent to testing in the small or to testing in the large, and where it fits in the VV&T terminology quagmire.

NLP systems have largely been evaluated using a black-box, functional, approach, often supplemented with an analysis of how acceptable the output is to users (Hirschman & Thompson 1998; White & Taylor 1998). The evaluation process must determine whether the system serves the intended function in the intended environment. There are several evaluation taxonomies (Cole et al. 1998; Jones & Galliers 1996), but the common goals are to determine if the system meets objectives, identify areas in which the system does not perform as well as predicted or desired, and compare different approaches for solving a single problem.

What becomes apparent is that there are several key differences between testing and evaluation. One obvious difference between testing and evaluation is that evaluation takes place late in the development life cycle, after a system is largely complete. On the other hand, many aspects of testing (such as requirements analysis and inspection, unit testing and integration testing) are undertaken early in the life cycle. A second difference is that evaluation data is based on domain coverage, whereas some of the data used in systematic software testing is based on code coverage.

The perspective from which a system is either tested or evaluated is also very important in this comparison. In systematic software testing a portion of testing involves actual code coverage which is determined based on the implementation paradigm. For example, there are testing methods for systems written in procedural languages such as C, in object oriented languages such as C++ and Java, and developed using UML. However, NLP systems are evaluated based on the application domain. For example, a speech interface will be evaluated with regard to accuracy, coverage, and speed (James, Rayner, & Hockey 2000) regardless of its implementation language.

Finally, we contrast the respective goals of testing and evaluation. As stated above, the goal of program level testing is to ultimately identify and correct faults in the system. The goal of evaluation of an NLP system is to determine how well the system works, and determine what will happen and how the system will perform when it is removed from the development environment and put into use in the setting for which it is intended. Evaluation is user-oriented, with a focus on domain coverage. Given its focus on the user, evaluation is most like the validation aspect of VV&T.

As part of evaluation work, organized (competitive) comparisons are carried out of multiple systems which perform the same task. For example, the series of Message Understanding Conferences (MUC) involved the evaluation of information extraction systems. Similarly the Text Retrieval Conferences (TREC) carry out large-scale evaluation of text retrieval systems. These efforts allow for comparison of different approaches to particular language processing problems.

Functional, black-box, evaluation is a very important and powerful analysis method, particularly because it works from the perspective of the user, without concern for implementation. However, a more complete methodology would also take into account implementation details and conventional program based testing. Without this we can not be sure that the
domain based test data adequately covers the actual system code, and the system performance statistics present a false view of the expected system performance. In new research we are exploring the utility of implementation based testing tools for NLP systems. Our contention, which we expect to prove through experimental work, is that implementation based testing will facilitate elimination early in the development process of software problems and will lead to improved analysis of the expected performance and reliability of NLP systems.

Conclusions
What are the lessons for those of us who do VV&T work and would like to apply it to a new area? The most important step is to investigate activities normally undertaken in the new area. We must learn what developers in that area do to analyze their systems, and learn the terminology that they use to refer to their own work. Next, we can build a lexicon of the relevant words and concepts in the new area, identifying their relationship to the standard VV&T terminology.

Only after these steps are taken is it possible to determine if there is an opportunity to exploit actual VV&T tools and methods in the new area.

References