Best Practice Recommendation for Validation of Forensic DNA Software

Biological Data Interpretation & Reporting Subcommittee
Biology/DNA Scientific Area Committee
Organization of Scientific Area Committees (OSAC) for Forensic Science

OSAC Proposed Standard

Best Practice Recommendation for Validation of Forensic DNA Software

Prepared by
Biological Data Interpretation & Reporting Subcommittee
Version: 1.0
2018

Disclaimer:

This document has been developed by the Biological Data Interpretation & Reporting Subcommittee of the Organization of Scientific Area Committees (OSAC) for Forensic Science through a consensus process and is *proposed* for further development through a Standard Developing Organization (SDO). This document is being made available so that the forensic science community and interested parties can consider the recommendations of the OSAC pertaining to applicable forensic science practices. The document was developed with input from experts in a broad array of forensic science disciplines as well as scientific research, measurement science, statistics, law, and policy.

This document has not been published by an SDO. Its contents are subject to change during the standards development process. All interested groups or individuals are strongly encouraged to submit comments on this proposed document during the open comment period administered by the Academy Standards Board (www.asbstandardsboard.org).

Foreword

This document includes guidelines for the validation of software used in a forensic DNA laboratory that impacts the integrity of the evidence, the analytical process, interpretations and/or statistical conclusions. This document is not intended to be exhaustive and does not include specific recommendations regarding all aspects of good software engineering, development and testing. Additional guidelines and standards may be applicable to specialized software packages.

This standard was revised, prepared, and finalized by the DNA Consensus Body of the AAFS Standards Board. The draft of this standard was developed by the Biology/DNA Biological Data Interpretation and Reporting Subcommittee of the Organization of Scientific Area Committees. All hyperlinks and web addresses shown in this document are current as of the publication date of this standard.

Keywords: *software validation, forensic DNA*

Table of Contents

1 Scope 1

2 Normative References 1

3 Terms and Definitions 1

4 Requirements 3

5 Conformance 8

Annex A (informative) Bibliography 9

1 Scope

This best practice recommendation assists a laboratory in designing validation studies to evaluate the various software programs used in the forensic DNA laboratory.

Specifically, this guidance document applies to, but is not limited to:

a) Software used as a component, part or accessory of instrumentation
b) Software that impacts chain of custody documentation
c) Software that impacts the decision process and/or influences conclusions or reporting
d) Software created by the laboratory to assist with calculations and/or data transfers

2 Normative References

Federal Bureau of Investigation, (2011) *Quality Assurance Standards for Forensic DNA Testing Laboratories*, available at http://www.fbi.gov/about-us/lab/codis/qas-standards-for-forensic-dna-testing-laboratories-effective-9-1-2011.

Scientific Working Group on DNA Analysis and Methods, (2012) *Validation Guidelines for DNA Analysis Methods*, available at http://www.swgdam.org/

3 Terms and Definitions

3.1
boundary testing
Checking to confirm expected outputs are obtained when inputs are at the limits of the software (e.g. testing allele frequencies below the 5/2N minimum threshold or testing upper and lower limits for amplification setup calculations).

3.2
complex software
Software that contains many lines of code, complex algorithms, and/or interconnected modules.

3.3
critical software
Any software or modification that substantially influences the integrity of the evidence, the analytical process, interpretations, statistical conclusions, case file documentation, chain of custody documentation, accuracy of results, report wording, or any other item deemed integral.

3.4
functional testing
Checking to confirm that the software performs tasks as expected.

3.5
fuzz testing
Checking to confirm that invalid, unexpected or nonsensical inputs to a computer program or

module yield an acceptable response (e.g., an error message or other indication of a problem).

3.6
negative testing
Checking to confirm that incorrect or inverse inputs yield the expected output.

3.7
operating system
System software that manages computer hardware and software resources and provides common services for computer programs.

3.8
positive testing
Checking to confirm that the natural (or usual) inputs yield the expected output.

3.9
regression testing
Checking to confirm that changes or new functionality does not unacceptably alter or terminate a desired functionality that behaved correctly before the change was implemented.

3.10
reliability testing
Checking beyond the functional aspects to measure the reliability of the software in the laboratory environment. This includes testing the impact on software performance when utilized by multi-user or multi-site scenarios and verifying network, server, and other applicable resources can handle the application's needs.

3.11
risk assessment
A systematic process for deciding the risk level associated with a particular software or module.

3.12
software developer
The legal entity or vendor company which created and/or provides a software program.

3.13
software module
Part of a software program. Programs are typically composed of one or more independently developed modules. Modules may be acquired as additions to a software program already in use, or they may be fully integrated into the software program.

3.14
software program
Instructions that cause a computer to behave in a desired way.

3.15
software reliability
Level of confidence of failure-free and correct software operation for a specified period of time in a specified environment.

3.16
software test
Individual trial designed to evaluate specific software functions.

3.17
software test types
Different categories of trials that comprise the software validation.

3.18
software validation
Confirmation, through the provision of objective evidence, derived from a series of documented tests, of the compliance of a software system with intended use and applicable guidelines.

4 Requirements

4.1. New software programs, modules or software modifications, such as a major functionality addition, that impact evidence integrity, the analytical process, the interpretations and/or statistical conclusions, should be validated prior to implementation.

4.1.1. There may be examples of enterprise level software that the DNA laboratory does not have autonomous control over, such as chain of custody software, for which the DNA laboratory may not be able to validate all of the modules. The DNA laboratory should still validate all of the modules over which they have control, and they should document the extent to which they rely on modules that they are unable to validate.

4.2 The validation of the software should be carried out for a given software environment which is recommended by the software developer (operating system, database management system, etc).

4.2.1 As the computing environment of the software evolves, the consequences must be evaluated, and, if applicable, a software risk assessment and validation should be conducted.

4.3 The forensic laboratory should choose a validation approach that includes one or more of the following:

4.3.1 The forensic laboratory performs the validation of the software.

4.3.2 The forensic laboratory uses a third party to perform the validation.

4.3.3 The forensic laboratory incorporates documented validations performed by the developer.

4.4 Validation should demonstrate that the software is fit for its intended use in the operational environment and define its limitations.
NOTE: Testing of software provides unique challenges and it is unlikely that a test can be designed for every use, case or scenario for all software programs or modules.

4.4.1 Software associated with an instrument may be validated in conjunction with the instrument.

4.5 When validating new software, the laboratory should rely on the software developer to explain functionality.

4.5.1 The laboratory should require the developer to provide documentation such as a user's manual to explain the intended uses and limits of the software.

4.5.2 The laboratory may rely on the results of testing conducted by the developer but the laboratory should also extend tests during their validation.

4.6 For software upgrades or modifications, the laboratory should require the developer to provide written documentation, such as release notes, to explain the purpose and scope of the modifications.

4.6.1 Every version of the software should be identified by a unique release number.

4.6.2 Information about recommended validation tests to assess the changes may also be requested from the developer.

4.7 The forensic laboratory should design and/or ensure that the developer has designed validation tests that evaluate the software's performance and the limits within which it performs properly.

4.7.1 If a third party is used, in whole or in part, to validate the software program or module, the forensic laboratory is responsible for determining that the validation design, testing procedures and documentation meet the requirements as detailed in this document.

4.7.2 If a validation incorporates the results of previously performed testing into the assessment, the forensic laboratory is responsible for determining whether the validation design, testing procedures and documentation meet the requirements as detailed in this document.

4.7.3 If the requirements have not been met, the forensic laboratory should identify what additional efforts are needed to establish that the software is sufficiently validated for its intended use in the laboratory.

4.8 Validation should follow a predefined strategy as depicted in Figure 1.

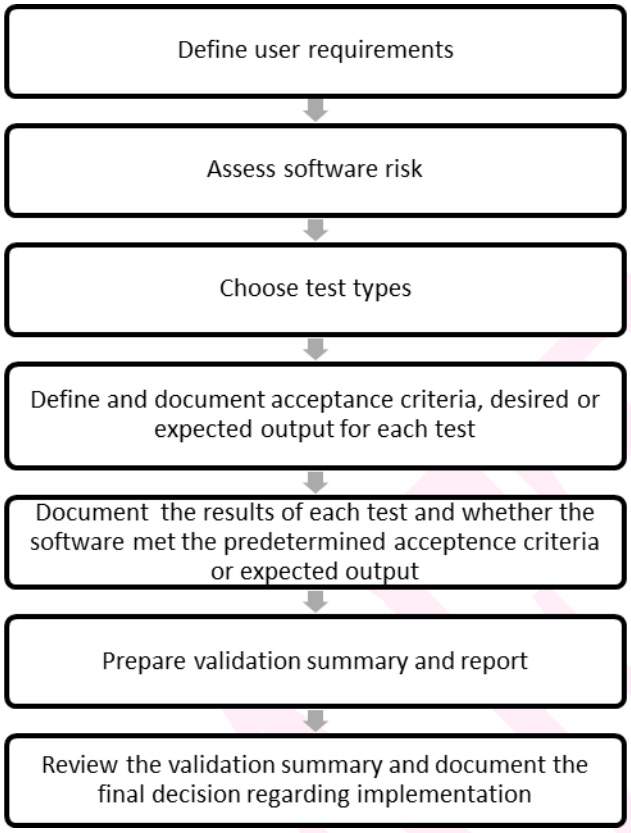

Figure 1

4.8.1 User requirements for the software program and/or module should be defined and documented.

4.8.2 A risk assessment should be performed to make an objective assessment of the level of criticality and complexity of a software program or module.

4.8.2.1 Critical: Any software or modification that substantially influences the integrity of the evidence, the analytical process, interpretations, statistical conclusions, case file documentation, chain of custody documentation, accuracy of results, report wording, or any other item deemed integral.

4.8.2.2 Not Critical: Software that provides information to the analyst that aids the analysis, interpretation, or reporting process, but does not substantially affect the analytical process, conclusions or documentation.

4.8.2.3 Complex: Software that contains many lines of code, complex algorithms, and/or interconnected modules. In cases where an already validated software program or module has been updated, the complexity of the change or update should be evaluated based on the complexity of the software modification.

4.8.2.4 Not Complex: A simple software program, module or upgrade that does not satisfy the definition of complex software. This may include simple spreadsheets or equations easily confirmed by the user.

4.8.2.5 Whenever a new version of the software program or module is released, or when there are software modifications or updates, a risk assessment should be conducted.

4.8.3 The required test types should be chosen based upon the determined level of criticality and complexity established during risk assessment. The most complex software programs or modules with the highest level of criticality warrant the highest level of validation testing.

	Not Critical	**Critical**
Not Complex	1. Not Critical Not Complex	2. Critical Not Complex
Complex	3. Not Critical Complex	4. Critical Complex

Figure 2. Software Risk Assessment Decision Matrix

4.8.3.1 The following testing types should be performed during validation:

4.8.3.1.1 Functional testing including the following sub-types (asappropriate):

a. Positive testing
b. Negative testing
c. Boundary testing
d. Fuzz testing.

4.8.3.1.2 Reliability Testing

4.8.3.1.3 Regression Testing.

4.8.3.2 The type and number of tests should be based on the risk assessment. The following tests, at minimum, are recommended for given criticality/complexity levels:

4.8.3.2.1 Not Critical/Not Complex (e.g. already validated software with a misspelled word corrected):

a. Functional testing: Positive

4.8.3.2.2 Critical/Not Complex:
a. Functional testing: Positive, Negative, Boundary
b. Regression testing: If applicable (i.e., software updates)
c. Reliability testing: If applicable (e.g., multi-user environment)

4.8.3.2.3 Not Critical/Complex (e.g., module with quality flags used to aid analysis was

updated):

a. Functional testing: Positive, Negative, Fuzz
b. Regression testing: If applicable (i.e., software updates)

4.8.3.2.4 Critical/Complex:
a. All Functional, reliability and regression testing

4.8.3.3 Test cases, which are designed to challenge the system and evaluate its performance against pre-determined criteria, especially for its most critical parameters, should be created.

4.8.3.3.1 The test cases should cover the full range of operating conditions so that the system can encounter a wide spectrum of conditions and events that will typically be encountered at the user site.

4.8.3.3.2 Complex, critical analytic software (e.g., probabilistic genotyping systems) should not be trained and tested using the same data set.

4.8.3.3.3 If a set of test data are used to set parameters and establish initial boundary conditions for acceptable performance, the system should then be tested on a fresh data set. This step is necessary to avoid inadvertently fitting the software performance to the characteristics of a single data set.

4.8.3.3.4 Whenever a new version of the software program or module is released, or when there are software modifications or updates, the changes with reference to the previous version should be identified and their consequences should be evaluated to determine the extent and impact of the change on the entire system. Due to the complexity of software, a seemingly small local change may have a significant system impact.

4.8.3.4 If the laboratory chooses to use or incorporate some aspects of the developer's validation testing and results as part of their own validation, then the laboratory should perform a subset of the evaluations performed by the software developer at the laboratory.

4.8.3.4.1 The forensic lab should additionally conduct some of their own tests to demonstrate that the software was installed properly and functions in the context of their operational environment. This is to make sure that there are no incompatibilities with other applications that might not have been present in the developer's software environment. This may be facilitated by the software developer or vendor furnishing the user with test data sets to be used for this purpose while providing the user with expected results such that the outputs can be compared. The software may have built-in tests that the laboratory can exercise.

4.8.4 Acceptance criteria should be defined and documented prior to performing the tests.

4.8.4.1 The acceptance criteria should be chosen based on either documented performance characteristics of the software, or against some known/expected values or outcomes.

4.8.5 The results of each test and whether the software met the predetermined acceptance criteria or expected output should be documented.

4.8.5.1 Failing some of the tests does not necessarily mean invalidation. The laboratory may decide

that some failures represent minor inconveniences that do not invalidate the software, or alternatively that a single critical or combination of moderate failures are intolerable and decide the software cannot be accepted.

4.8.5.1.1 Records of any system failure should be maintained.

4.8.5.1.2 The records should detail whether the failure impacts the software's fitness for use.

4.8.5.1.3 If the failure is one that impacts the software's fitness for use, then the laboratory should document the actions taken to ensure the source of failure was resolved and the test(s) repeated with expected results.

4.8.6 A validation summary and report should be prepared.

4.8.6.1 The validation summary should objectively confirm whether the software is validated for its intended use.

4.8.6.1.1 The validation summary should include, at a minimum, the following information:

a) Organization that conducted testing
b) Defined user requirements
c) Risk assessment decision and reasoning
d) Software name including version number
e) Test cases
    a. Date of test(s)
    b. Operating system
    c. Input Data
    d. Acceptance criteria
    e. Test result(s)
f) Record of any system failure and actions taken to ensure failure was resolved
g) Record of formal acceptance or rejection
h) Date the software was approved for use, if implemented

4.8.6.1.2 The validation summary should be maintained by the laboratory and made readily available to interested parties.

4.8.6.2 If the laboratory chose to use a third party testing service, or use the developer's validation testing and results, the laboratory should ensure that a validation summary has been prepared and is available on site at the laboratory.

4.9 The laboratory should complete its assessment prior to use on evidence samples, casework reference samples, or database samples.

5 Conformance

Documentation demonstrating conformance with the recommendations described in this document should be signed and dated by the laboratory's DNA technical leader and made readily available in hard copy and/or electronic form for review by an assessor.

Annex A
(informative)

Bibliography

This is not meant to be an all-inclusive list as the group recognizes other publications on this subject may exist. At the time these standards were drafted, these were the publications available to the working group members for reference. Additionally, any mention of a particular software tool or vendor as part of this bibliography is purely incidental, and any inclusion does not imply endorsement by the authors of this document.

1) ANSI SES1:2002. Recommended Practice for the Designation and Organization of Standards. Date of Issue: 2002-08-01
2) FDA. General Principles of Software Validation; Final Guidance for Industry and FDA Staff. Date of Issue: 2002-01-11
3) NPL (National Physical Laboratory). NPL Report DEM-ES 014. Software Support for Metrology Best Practice Guide No. 1; Validation of Software in Measurement Systems v. 2.2. Date of Issue: 2007-01
4) International Civil Aviation Organization. DOC 9906 AN/472. Quality Assurance Manual for Flight Procedure Design; Vol 3. Flight Procedure Design Software Validation. 2010.