INTERNATIONAL STANDARD

ISO/IEC 23643

First edition
2020-06

Software and systems engineering — Capabilities of software safety and security verification tools

Ingénierie du logiciel et des systèmes — Capacités des outils de vérification de la sûreté et de la sécurité des logiciels

Reference number
ISO/IEC 23643:2020(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see http://patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 7, *Software and systems engineering*.

Any feedback or questions on this document should be directed to the user’s national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

Since a few decades, the importance of software safety and security verification tools has increased for several reasons: 1) rapidly increasing complexity of software applications and systems, 2) increasing number of safety-critical systems through growing integration between software applications and systems (e.g. in critical infrastructures), 3) the rapid increase of the number of cyber threats, and 4) the urgent needs of safety in high and medium critical software-driven systems (e.g. transportation, energy production, Internet of Things (IoT), and general purpose Operating Systems and middleware). Additionally, the number of products and system development cases, where the origin of all software components to be used is not exactly known, even for open-source applications, is increasing and thus making safety and security verification and validation (V&V) essential.

This document restricts its point of view to software and excludes computing and any other hardware from the context. In these other domains, other V&V methods and tools are used.

It is important to realize that verification of safety and security of software does not necessarily verify the system safety and system security of a system using the software as a component. However, if a system consists of software components which are not verified, the safety and security of the system cannot be guaranteed at any level.

“Continuous everything”, including continuous software development and thus versioning delivery, requires continuous software safety and security verification. At every new version, V&V needs to be redone. The popular “agile development processes” permit shorter development iterations and more frequent product delivery, and consequently this requires more frequent verification than traditional development approaches. Verification is needed during software development as well as during software maintenance, whenever safety or security of software can be endangered.

Validation answers the question “are we building the right product?”

Verification answers the question “are we building the product right?”

Software validation checks if the software product satisfies the intended use, such as defined in requirements and specifications. In other words (ISO/IEC 17029): “purpose of validation is to find out, whether a declared information (claim) is plausible”. Software verification checks if the specifications and requirements are met either by running the software (testing) or by reviewing its artefacts (specification, model, or pseudo code). The latter can consist of animating or analyzing statically one of its artefacts. ISO/IEC 17029 defines that the “purpose of verification is to find out, whether a declared information (claim) is truthful”. This document does not concern testing but animating and analyzing the artefacts, because “testing tools” is already well covered by and is the subject of ISO/IEC 30130.

This document is prepared as one of the series of single tool capabilities which are used with ISO/IEC 20741.

This document defines capabilities of and requirements for software safety and security verification tools.

This document is independent of the target application domains, as the languages, methods and associated tools are of general purpose, and can fit into different kinds of problems (e.g. functional specification languages can be used for any functional program).

Software and systems engineering — Capabilities of software safety and security verification tools

1 Scope

This document specifies requirements for the vendors and gives guidelines for both the users and the developers of software safety and security verification tools. The users of such tools include, but are not limited to, bodies performing verification and software developers who need to be aware and pay attention to safety and/or security of software. This document guides the verification tool vendors to provide as high-quality products as possible and helps the users to understand the capabilities and characteristics of verification tools.

This document introduces use cases for software safety and security verification tools and entity relationship model related to them. This document also introduces tool categories for software safety and security verification tools and gives category specific guidance and requirements for the tool vendors and developers.

2 Normative references

There are no normative references in this document.

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at http://www.electropedia.org/

3.1
application domain
well-defined set of applications

3.2
capability
quality of being able to perform a given activity

[SOURCE: ISO 19439:2006, 3.5]

3.3
certificate
attestation document issued by an independent third-party certification body

[SOURCE: ISO 22222:2005, 3.2]

3.4
defect
fault, or deviation from the intended level of performance of a system or *software* (3.19)

3.5
dynamic program analysis
process of evaluating a *software* (3.19) system or component based on its behaviour during execution

Note 1 to entry: The definition means that the software shall actually be compiled and run on a certain number of input data test cases. The physical response from the system is then examined and compared to expected results. Dynamic program analysis can be done manually or using an automated process. The results are examined either manually (e.g. with small input test data) or automatically using oracles.

3.6
entity
data concept that may have attributes and relationships to other entities

[SOURCE: ISO/TR 25100:2012, 2.1.3, modified — Note 1 to entry has been removed.]

3.7
evaluator
competent person engaged in the *verification* (3.33) or *validation* (3.32) of a system or *software* (3.19)

3.8
false negative
true *defect* (3.4) that has not been observed

Note 1 to entry: The term is used for analysis tools producing defect information during the analysis of an application. In the presence of false negatives, the tool is said to be incomplete with respect to the real set of defects in the *software* (3.19) under analysis. False negatives can be due to several reasons such as 1) the use of heuristics for detecting defects, 2) too restrictive analysis data.

3.9
false positive
observed *defect* (3.4) which does not correspond to a true defect

Note 1 to entry: The term is used for analysis tools producing defect information during the analysis of an application. False positives appear during the analysis because of several possible reasons, such as lack of precision of the analysis rules.

3.10
formal verification
activity proving or disproving the correctness of intended applications with respect to a formal specification or a property, using formal methods of mathematics

3.11
malfunction
behaviour of a system or component that deviates from the specifications

3.12
protection
process to secure content

[SOURCE: ISO/IEC 15444-8:2007, 3.24]

3.13
risk
effect of uncertainty on objectives

Note 1 to entry: ISO 22538-4 defines risk as “probability of loss or injury from a hazard”.

[SOURCE: ISO 31000:2018, 3.1, modified — Notes 1, 2 and 3 to entry have been removed; a new Note 1 to entry has been added.]

3.14
safety
freedom from unacceptable *risk* (3.13)

3.15
safety-critical system
system whose failure or *malfunction* (3.11) may result in one (or more) of the following outcomes:

- death or serious injury to people
- loss or severe damage to equipment/property
- environmental harm

EXAMPLE Examples of safety-critical systems are critical infra-structures, medical equipment, transportation, and nuclear power plants. Safety-critical systems are also sometimes called life-critical systems.

3.16
security
resistance to intentional, unauthorized act(s) designed to cause harm or damage to a system

3.17
security level
combination of a hierarchical *security* (3.16) classification and a security category that represents the sensitivity of an object or the security clearance of an individual

Note 1 to entry: The minimum security level is an indication of the minimum *protection* (3.12) required.

3.18
semi-formal verification
method that is based on a description given in semi-formal notation

3.19
software
all or part of the programs, procedures, rules, and associated documentation of an information processing system

[SOURCE: ISO/IEC 19770-3:2016, 3.1.26, modified — Note 1 to entry has been removed.]

3.20
software item
identifiable part of a *software* (3.19) product, consisting of source code, object code, control code, control data, or a collection of these

Note 1 to entry: Software item is a generic term that designates well-identified parts of software source code, object code or data. A software item belongs to a syntactic category of the programming language in which the software is written. Examples are classes, variables, functions and types. A software item is an identifiable part of a software product.

3.21
software safety
ability of *software* (3.19) to be free from unacceptable *risk* (3.13)

Note 1 to entry: It is the ability of software to resist failure and *malfunctions* (3.11) that can lead to death or serious injury to people, loss or severe damage to property, or severe environmental harm.

Note 2 to entry: Software quality, including software safety, is achieved using software engineering. Software engineering for *safety-critical systems* (3.15) emphasizes the following directions:

- process engineering and management;
- selecting the appropriate tools and environment for the system; the principle of using the best tools fit to the purpose prevails as in most engineering disciplines;
- adherence to requirements.

3.22

software security

ability of *software* (3.19) to protect its assets from a malicious attacker

Note 1 to entry: According to software product quality model in ISO/IEC 25010, software security applies to software assets and is decomposed into the following set of properties: confidentiality, integrity, availability, authentication, authorization and non-repudiation.

3.23

software unit

smallest independent piece of *software* (3.19), which can be independently translated, and which can be tested with the relevant data on whether it performs to specification

3.24

static program analysis

sub-field of formal methods concerned by analyzing the properties of a *software* (3.19) code without executing this code in the target (binary) format

3.25

system safety

ability of a system to be free from unacceptable *risk* (3.13)

Note 1 to entry: A system is defined as a set or group of interacting, interrelated or interdependent elements or parts, that are organized and integrated to form a collective unity or a unified whole, to achieve a common objective. In a broader view the definition of a system consists in the hardware, *software* (3.19), human systems integration, procedures and training. Therefore, system safety is part of the systems engineering process and should systematically address all of these domains and areas in engineering and operations in a concerted fashion to prevent, eliminate and control hazards.

Note 2 to entry: A system safety concept helps the system designer(s) to model, analyze, gain awareness about, understand and eliminate the hazards, and apply controls to achieve an acceptable level of *safety* (3.14). The systems-based approach to safety requires the application of scientific, technical and managerial skills to hazard identification, hazard analysis, and elimination, control, or management of hazards throughout the life-cycle of a system, program, project or an activity or a product. Hazop is one of several techniques available for the identification of hazards.

3.26

target of verification

TOV

software (3.19), or a set of *software items* (3.20) or *units* (3.23), to be verified (e.g. in terms of *safety* (3.14) and *security* (3.16))

Note 1 to entry: Target of evaluation (TOE) is a commonly used term in systems security techniques. TOE is defined as a set of software, firmware and/or hardware possibly accompanied by guidance.

3.27

target software

final product of a *software* (3.19) development process, containing at least the binary code able to run on the target computer

Note 1 to entry: Target software may consist of several files, including binary and source files, libraries, installation and compilation script files, documentation and data files. The target software often relies on underlying layers of software, that are not part of the target software, but that are necessary to be executed on the target platform, for instance libraries.

3.28

target system

complete computing platform capable of running the *target software* (3.27)

Note 1 to entry: A target system consists of hardware resources and *software* (3.19) resources installed on the hardware.

3.29
toolbox
set of tools completing each other in terms of capabilities, to cover a larger area of their intended use

3.30
trust
degree to which a user or other stakeholder has confidence that a product or system will behave as intended

[SOURCE: ISO/IEC 25010:2011, 4.1.3.2]

3.31
use case
specification of a sequence of actions, including variants, that a system (or other *entity* (3.6)) can perform, interacting with actors of the system

[SOURCE: ISO 15745-1:2003, 3.35, modified — The domain "<class>" at the beginning and "[UML]" at the end of the definition has been removed.]

3.32
validation
confirmation, through the provision of objective evidence, that the requirements for a specific intended use or application have been fulfilled

EXAMPLE *Safety* (3.14) validation has been defined as an assurance, based on examination and tests, that the safety goals are sufficient and have been achieved (ISO 26262-1).

[SOURCE: ISO/IEC 25000:2014, 4.41, modified — Note 1 to entry have been removed; EXAMPLE has been added.]

3.33
verification
confirmation, through the provision of objective evidence, that specified requirements have been fulfilled

[SOURCE: ISO/IEC 25000:2014, 4.43, modified — Note 1 to entry have been removed.]

3.34
verification method
method for producing objective evidence that specified requirements of a system have been fulfilled

3.35
verification tool
instrument that can be used during *verification* (3.33) to collect information about the *target of verification* (3.26), to perform interpretation of information or to automate part of the verification

3.36
vulnerability
potential flaw or weakness in *software* (3.19) design or implementation that could be exercised (accidentally triggered or intentionally exploited) and result in harm to the system

Note 1 to entry: The CVE classification (see Reference [25]) defines the de-facto standard classes of the known software vulnerabilities.

Note 2 to entry: A vulnerability is exploitable if it can be activated in practice.

4 Abbreviated terms

AASPE	automated assurance of security policy enforcement tool
ACSL	ANSI/ISO C specification language
ARM	advanced RISC machine
BDD	binary decision diagram
BNF	Backus–Naur form
CASE	computer-aided software engineering tool
CC	common criteria
CSPN	Certification de Sécurité de Premier Niveau
CTL	computation tree logic
dns	domain name server, domain name system
DREAD	damage, reproducibility, exploitability, affected users, and discoverability risk assessment model
EAL	evaluation assurance level
FMEA	failure mode and effects analysis
HAZOP	hazard and operability analysis
LTL	linear temporal logic
OSINT	open-source intelligence
RM	reference manual
SAT	satisfiability
SCL	Safety Culture Ladder
SDLC	software development life-cycle
SMT	satisfiability modulo theory
SQL	structured query language
STRIDE	spoofing, tampering, repudiation, information disclosure, denial of service and elevation of privilege
SWOT	strengths, weaknesses, opportunities and threats
UML	unified modeling language
VDM-SL	Vienna development method - specification language
V&V	verification and validation
XSS	cross-site scripting