WIKIPEDIA

Software verification and validation

In software project management, software testing, and software engineering, **verification and validation** (**V&V**) is the process of checking that a software system meets specifications and requirements so that it fulfills its intended purpose. It may also be referred to as software quality control. It is normally the responsibility of software testers as part of the software development lifecycle. In simple terms, software verification is: "Assuming we should build X, does our software achieve its goals without any bugs or gaps?" On the other hand, software validation is: "Was X what we should have built? Does X meet the high-level requirements?"

Contents

Definitions
- Software verification
- Artifact or specification verification
- Software validation
- Artifact or specification validation
- Validation vs. verification

Related concepts

V&V methods
- Formal

Independent
- History
  - At ESA
- Methodology
  - Planning
  - Requirements verification
  - Design verification
  - Code verification
  - Validation

Regulatory environment

See also

Further reading

External links

References

Definitions

Verification and validation are not the same thing, although they are often confused. Boehm succinctly expressed the difference as[1]

- Verification: Are we building the product right?
- Validation: Are we building the right product?

"Building the product right" checks that the *specifications* are correctly implemented by the system while "building the right product" refers back to the *user's needs*. In some contexts, it is required to have written requirements for both as well as formal procedures or protocols for determining compliance. Ideally, formal methods provide a mathematical guarantee that software meets its specifications.

Building the product right implies the use of the Requirements Specification as input for the next phase of the development process, the design process, the output of which is the Design Specification. Then, it also implies the use of the Design Specification to feed the construction process. Every time the output of a process correctly implements its input specification, the software product is one step closer to final verification. If the output of a process is incorrect, the developers are not building the product the stakeholders want correctly. This kind of verification is called "artifact or specification verification".

Building the right product implies creating a Requirements Specification that contains the needs and goals of the stakeholders of the software product. If such artifact is incomplete or wrong, the developers will not be able to build the product the stakeholders want. This is a form of "artifact or specification validation".

Note: Verification begins before Validation and then they run in parallel until the software product is released.

Software verification

It would imply to verify if the specifications are met by running the software but this is not possible (e. g., how can anyone know if the architecture/design/etc. are correctly implemented by running the software?). Only by reviewing its associated artifacts, can someone conclude whether or not the specifications are met.

Artifact or specification verification

The output of each software development process stage can also be subject to verification when checked against its input specification (see the definition by CMMI below).

Examples of artifact verification:

- Of the design specification against the requirement specification: Do the architectural design, detailed design and database logical model specifications correctly implement the functional and non-functional requirements specifications?
- Of the construction artifacts against the design specification: Do the source code, user interfaces and database physical model correctly implement the design specification?

Software validation

Software validation checks that the software product satisfies or fits the intended use (high-level checking), i.e., the software meets the user requirements, not as specification artifacts or as needs of those who will operate the software only; but, as the needs of all the stakeholders (such as users, operators, administrators, managers, investors, etc.). There are two ways to perform software validation: internal and external. During internal software validation, it is assumed that the goals of the stakeholders were correctly understood and that they were expressed in the requirement artifacts precisely and comprehensively. If the software meets the requirement specification, it has been internally validated. External validation happens when it is performed by asking the stakeholders if the software meets their needs. Different software development methodologies call for different levels of user and stakeholder involvement and feedback; so, external validation can be a discrete or a continuous event. Successful final external validation occurs when all the stakeholders accept the software product and express that it satisfies their needs. Such final external validation requires the use of an acceptance test which is a dynamic test.

However, it is also possible to perform internal static tests to find out if the software meets the requirements specification but that falls into the scope of static verification because the software is not running.

Artifact or specification validation

Requirements should be validated before the software product as a whole is ready (the waterfall development process requires them to be perfectly defined before design starts; but iterative development processes do not require this to be so and allow their continual improvement).

Examples of artifact validation:

- User Requirements Specification validation: User requirements as stated in a document called User Requirements Specification are validated by checking if they indeed represent the will and goals of the stakeholders. This can be done by interviewing the stakeholders and asking them directly (static testing) or even by releasing prototypes and having the users and stakeholders to assess them (dynamic testing).
- User input validation: User input (gathered by any peripheral such as keyboard, bio-metric sensor, etc.) is validated by checking if the input provided by the software operators or users meets the domain rules and constraints (such as data type, range, and format).

Validation vs. verification

According to the Capability Maturity Model (CMMI-SW v1.1),[2]

- Software Validation: The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements. [IEEE-STD-610]
- Software Verification: The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. [IEEE-STD-610]

Validation during the software development process can be seen as a form of User Requirements Specification validation; and, that at the end of the development process is equivalent to Internal and/or External Software validation. Verification, from CMMI's point of view, is evidently of the artifact kind.

In other words, software verification ensures that the output of each phase of the software development process effectively carry out what its corresponding input artifact specifies (requirement -> design -> software product), while software validation ensures that the software product meets the needs of all the stakeholders (therefore, the requirement specification was correctly and accurately expressed in the first place). Software verification ensures that "you built it right" and confirms that the product, as provided, fulfills the plans of the developers. Software validation ensures that "you built the right thing" and confirms that the product, as provided, fulfills the intended use and goals of the stakeholders.

This article has used the strict or narrow definition of verification.

From a testing perspective:

- Fault – wrong or missing function in the code.
- Failure – the manifestation of a fault during execution. The software was not effective. It does not do "what" it is supposed to do.
- Malfunction – according to its specification the system does not meet its specified functionality. The software was not efficient (it took too many resources such as CPU cycles, it used too much memory, performed too many I/O operations, etc.), it was not usable, it was not reliable, etc. It does not do something "how" it is supposed to do it.

Related concepts

Both verification and validation are related to the concepts of quality and of software quality assurance. By themselves, verification and validation do not guarantee software quality; planning, traceability, configuration management and other aspects of software engineering are required.

Within the modeling and simulation (M&S) community, the definitions of verification, validation and accreditation are similar:

- M&S Verification is the process of determining that a computer model, simulation, or federation of models and simulations implementations and their associated data accurately represent the developer's conceptual description and specifications.[3]
- M&S Validation is the process of determining the degree to which a model, simulation, or federation of models and simulations, and their associated data are accurate representations of the real world from the perspective of the intended use(s).[3]
- Accreditation is the formal certification that a model or simulation is acceptable to be used for a specific purpose.[3]

The definition of M&S validation focuses on the accuracy with which the M&S represents the real-world intended use(s). Determining the degree of M&S accuracy is required because all M&S are approximations of reality, and it is usually critical to determine if the degree of approximation is acceptable for the intended use(s). This stands in contrast to software validation.

V&V methods

Formal

In mission-critical software systems, formal methods may be used to ensure the correct operation of a system. These formal methods can prove costly, however, representing as much as 80 percent of total software design cost.

Independent

Independent Software Verification and Validation (ISVV) is targeted at safety-critical software systems and aims to increase the quality of software products, thereby reducing risks and costs through the operational life of the software. The goal of ISVV is to provide assurance that software performs to the specified level of confidence and within its designed parameters and defined requirements.[4][5]

ISVV activities are performed by independent engineering teams, not involved in the software development process, to assess the processes and the resulting products. The ISVV team independency is performed at three different levels: financial, managerial and technical.

ISVV goes beyond "traditional" verification and validation techniques, applied by development teams. While the latter aim to ensure that the software performs well against the nominal requirements, ISVV is focused on non-functional requirements such as robustness and reliability, and on conditions that can lead the software to fail.

ISVV results and findings are fed back to the development teams for correction and improvement.

History

ISVV derives from the application of IV&V (Independent Verification and Validation) to the software. Early ISVV application (as known today) dates back to the early 1970s when the U.S. Army sponsored the first significant program related to IV&V for the Safeguard Anti-Ballistic Missile System.[6] Another example is NASA's IV&V Program, which was established in 1993.[7]

By the end of the 1970s IV&V was rapidly becoming popular. The constant increase in complexity, size and importance of the software led to an increasing demand on IV&V applied to software.

Meanwhile, IV&V (and ISVV for software systems) consolidated and is now widely used by organizations such as the DoD, FAA,[8] NASA[7] and ESA.[9] IV&V is mentioned in DO-178B, ISO/IEC 12207 and formalized in IEEE 1012.

At ESA

Initially in 2004-2005, a European consortium led by the European Space Agency, and composed by DNV, Critical Software SA, Terma and CODA SciSys plc created the first version of a guide devoted to ISVV, called "ESA Guide for Independent Verification and Validation" with support from other organizations.[10] This guide covers the methodologies applicable to all the software engineering phases in what concerns ISVV.

In 2008 the European Space Agency released a second version, having received inputs from many different European Space ISVV stakeholders.[10]

Methodology

ISVV is usually composed by five principal phases, these phases can be executed sequentially or as results of a tailoring process.

Planning

- Planning of ISVV activities
- System criticality analysis: Identification of critical components through a set of RAMS activities (Value for Money)
- Selection of the appropriate methods and tools

Requirements verification

- Verification for: completeness, correctness, testability

Design verification

- Design adequacy and conformance to software requirements and interfaces
- Internal and external consistency
- Verification of feasibility and maintenance

Code verification

- Verification for: completeness, correctness, consistency
- Code metrics analysis
- Coding standards compliance verification

Validation

- Identification of unstable components/functionalities
- Validation focused on error-handling: complementary (not concurrent) validation regarding the one performed by the development team
- Compliance with software and system requirements
- Black box testing and White box testing techniques
- Experience based techniques

Regulatory environment

Software often must meet the compliance requirements of legally regulated industries, which is often guided by government agencies[11][12] or industrial administrative authorities. For instance, the FDA requires software versions and patches to be validated.[13]

See also

- Compiler correctness
- Cross-validation
- Formal verification
- Functional specification
- Independent Verification and Validation Facility
- International Software Testing Qualifications Board
- Software verification
- Software requirements specification
- Validation (drug manufacture)
- Verification and validation – General
- Verification and Validation of Computer Simulation Models
- Independent verification systems
- Software testing
- Software engineering
- Software quality
- Static code analysis
- Requirements engineering
- Safety-critical system
- Katherine Johnson Independent Verification and Validation Facility

Further reading

- *1012-2012 IEEE Standard for System and Software Verification and Validation*. 2012. doi:10.1109/IEEESTD.2012.6204026 (https://doi.org/10.1109%2FIEEESTD.2012.6204026). ISBN 978-0-7381-7268-2.
- Tran, E. (1999). "Verification/Validation/Certification" (http://www.ece.cmu.edu/~koopman/des_s99/verification/index.html). In Koopman, P. (ed.). *Topics in Dependable Embedded Systems*. Carnegie Mellon University. Retrieved 2007-05-18.
- Menzies, T.; Y. Hu (2003). "Data mining for very busy people". *Computer*. **36** (1): 22–29. doi:10.1109/MC.2003.1244531 (https://doi.org/10.1109%2FMC.2003.1244531).

External links

- Chapter on Software quality (including VnV) (http://www.computer.org/portal/web/swebok/html/ch11) in SWEBOK

References

1. Pham, H. (1999). *Software Reliability*. John Wiley & Sons, Inc. p. 567. ISBN 9813083840. "Software Validation. The process of ensuring that the software is performing the right process. Software Verification. The process of ensuring that the software is performing the process right." Likewise and also there: "In short, Boehm (3) expressed the difference between the software verification and software validation as follows: Verification: ‘‘Are we building the product right?’’ Validation: ‘‘Are we building the right product?’’."
2. "CMMI for Software Engineering, Version 1.1, Staged Representation (CMMI-SW, V1.1, Staged)" (https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=6217). *resources.sei.cmu.edu*. Retrieved 2021-03-20.
3. "Department of Defense Documentation of Verification, Validation & Accreditation (VV&A) for Models and Simulations". Missile Defense Agency. 2008.
4. Rogers, R. (1981-10-26). "Planning for independent software verification and validation" (http://arc.aiaa.org/doi/10.2514/6.1981-2100). *3rd Computers in Aerospace Conference*. San Diego,CA,U.S.A.: American Institute of Aeronautics and Astronautics. doi:10.2514/6.1981-2100 (https://doi.org/10.2514%2F6.1981-2100).
5. Ambrosio, Ana; Mattiello-Francisco, Fátima; Martins, Eliane (2008-05-12). "A Independent Software Verification and Validation Process for Space Applications" (http://arc.aiaa.org/doi/10.2514/6.2008-3517). *SpaceOps 2008 Conference*. Heidelberg, Germany: American Institute of Aeronautics and Astronautics. doi:10.2514/6.2008-3517 (https://doi.org/10.2514%2F6.2008-3517). ISBN 978-1-62410-167-0.
6. Lewis, Robert O. (1992). *Independent verification and validation : a life cycle engineering process for quality software* (https://www.worldcat.org/oclc/74908695). New York: Wiley. ISBN 0-471-57011-7. OCLC 74908695 (https://www.worldcat.org/oclc/74908695).
7. Asbury, Michael (2015-03-09). "About NASA's IV&V Program" (http://www.nasa.gov/centers/ivv/about/index.html). *NASA*. Retrieved 2021-03-20.
8. Balci, O. (2010). "Golden Rules of Verification, Validation, Testing, and Certification of Modeling and Simulation Applications" (https://www.semanticscholar.org/paper/Golden-Rules-of-Verification%2C-Validation%2C-Testing%2C-Balci/a78f010dd6e17f8da30bf92232d9e7ecb6a3f071). *undefined*. Retrieved 2021-03-20.
9. "Flight Software Systems Section (TEC-SWF)" (https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Software_Systems_Engineering/Flight_Software_Systems_Section_TEC-SWF). *www.esa.int*. Retrieved 2021-03-20.

10. Iavva.pt. "New ISVV Guide for Space in the Works" (https://www.criticalsoftware.com/en/news/new-isvv-guide-for-space-in-the-works). *www.criticalsoftware.com*. Retrieved 2021-03-20.
11. "General Principles of Software validation; Final Guidance for Industry and FDA Staff" (https://www.fda.gov/downloads/MedicalDevices/DeviceRegulationandGuidance/GuidanceDocuments/ucm085371.pdf) (PDF). Food and Drug Administration. 11 January 2002. Retrieved 12 July 2009.
12. "Guidance for Industry: Part 11, Electronic Records; Electronic Signatures — Scope and Application" (https://www.fda.gov/downloads/Drugs/GuidanceComplianceRegulatoryInformation/Guidances/UCM072322.pdf) (PDF). Food and Drug Administration. August 2003. Retrieved 12 July 2009.
13. "Guidance for Industry: Cybersecurity for Networked Medical Devices Containing Off-the Shelf (OTS) Software" (https://www.fda.gov/downloads/MedicalDevices/DeviceRegulationandGuidance/GuidanceDocuments/ucm077823.pdf) (PDF). Food and Drug Administration. 14 January 2005. Retrieved 12 July 2009.