4.3.3.5 Testability

Return to Maintainability

About

Return to Top

Testability, *Testable*, *Testing* and *Test* are not synonyms for each other. Just because a system or program is undergoing *testing* using various *tests* does not necessarily mean that the system or program is actually *Testable*. The following table provides definitions for each of these four terms, associates each with the appropriate Structured Assurance Case, as well as, level in the Cognitive Model's Science and Knowledge Management DIKW (Data, Information, Knowledge and Wisdom) pyramid.

DIKW pyramid level	Structured Assurance Case	Term	Description

DIKW pyramid level	Structured Assurance Case	Term	Description
Understanding	Software Assurance (SwA)	Testability	**Testability** is about documenting the functionality and requirements for a system or program and verifying that these requirements will be or have been met. Functional requirements are generally not a problem since most of these are directly measurable or observable. Functional requirements are often directly measured or observed: a data field needing to be provided, a relationship existing between two pieces of data, or a relationship that is one-to-many or a many-to-many. For example, every person must have a unique company ID number but they may have multiple phone numbers and also belong to multiple organizations. Other functional requirements are not so definite, but expressed in terms of a range of acceptable values. For example, a Graphical User Interface (GUI) will respond in less than 5 seconds or the heart pulse rate is between 35 to 200 beats per minute. In contrast, non-functional requirements are generally more abstract: they relate to the quality of the system or program being delivered (i.e., portable, reliable, maintainable, securable, scalable, etc.) and are usually not directly measurable or observable but must be inferred from characteristics found in the delivered system's or product's architecture, design and implementation. These kinds of requirements require ways to characterize assurance and are specified in terms of claims (i.e., the system has a High Availability), sub-claims, and arguments (i.e., Availability can be predicted using a Mean Time To Repair (MTTR) of 5 minutes, 15 seconds or less of downtime in a year for all components). These kinds of requirements are generally specified in Performance or Functional Specifications. These specifications tend to focus on hardware specifications; however, performance specifications can also capture non-functional metrics. • **Note** Testability metrics are not limited to operational systems or programs but can also take advantage of system or program level artifacts that describe architecture, design, discussion papers, outside references, software and executables. Here is a list of some common “mistakes” found in requirement documents[1)] that can make it difficult to determine if requirements are actually “testable”: • **Noise:** Text containing no information relevant to any aspect of the problem. For example, a requirement on a standalone application that does not need access to the Ethernet ◦ The system shall conform to IPV6 … • **Silence:** A feature not covered by any text within the Requirements documents or specifications • **Over-specification:** Description of the solution rather than the problem. For example, ◦ The distributed system must use blockchain. (blockchain is one of many distributed technologies used by Cryptocurrencies) ◦ The system must use a checkbox to select the appropriate option • **Contradictory:** Mutually incompatible descriptions of the same feature. For example, ◦ The system shall not record any personal information ◦ The system shall record all transactions and parties participating in the transaction • **Ambiguity:** Text that can be interpreted more than one way ◦ The system shall support real-time operations (what is real-time?) • **Forward reference:** Referring to a feature not yet described ◦ The system shall publish all information on a topic (but topic has not been officially defined yet) • **Wishful thinking:** Defining a feature that can’t be validated ◦ The system shall initialize all values with intelligent default choices. (what's the metric for “intelligent”?) • **Weak phrases:** Causing uncertainty (“adequate”, “usually”, “etc.”) For example, ◦ *When possible*, the systems shall … ◦ The system shall collect their data (whose data?) • **Jigsaw puzzles:** Requirements distributed across a document and then cross-referenced • **Duckspeak:** Requirements included merely to conform to standards that have no or little relationship to the problem at hand. Perhaps required as part of a boilerplate. • **Terminology invention:** “user input/presentation function”; “airplane reservation data validation function”. For example, ◦ The system shall use a double blind logged journal entry (huh, what is that?) • **Putting the onus on developers and testers:** to guess what the requirements really are. ◦ The system shall use a right-handed approach when presenting data

DIKW pyramid level	Structured Assurance Case	Term	Description
Knowledge	Claim	Testable	A **Testable** attribute of a system or program is a functional or nonfunctional requirement that may be testable or not. Some requirements can be directly tested for by running specific tests (i.e., Unit Testing, integration testing, etc.) using test plans that exercise a portion of the system or program software responsible for providing specific functionality. For example, the system is supposed to offer the choice of none, one, and many. Another example might be that when an option is selected, a message is sent out over the network. By design, some requirements are not directly testable, i.e, are untestable. Often, these requirements are met through the use of mathematical proofs or demonstrations. For example, the generation of a Universally Unique IDentifier (UUID) can not be tested directly; instead, the algorithm used to generate them must provide an explanation and proof that no two sets of conditions will produce the same UUID. Often there is a risk of generating the same UUID, but the chances of the same UUID being used in identical domains or environments is even smaller. Another example would be the reCAPTCHA, which shows a series of photos and asks the user to identify the ones with green peas in them. The order of the photos and the thing it is asking you to identify are randomly assigned.
Information	Argument	Testing	Testing is a process that generally involves the execution of the system or program under scripted, controlled situations. The scripts can be human instructions in documents or they can be captured in text files that a testing engine uses to drive the software. Sometimes, a Unit Test is used to test individual modules before they are integrated into the system or program. Below are the various requirement conformity checks that can be performed to verify functional requirements: 1. Unit Testing 2. Integration Testing 3. End-to-End Testing (E2E Testing) 4. Smoke Testing 5. Sanity Testing 6. Regression Testing 7. Acceptance Testing 8. White Box Testing 9. Black Box Testing 10. Interface Testing 11. Interoperability Testing
Data	Evidence	Test	*Test* refers to the act of collecting the evidence used to support arguments, sub-claims and claims made about the system or program. There is not a one-to-one relationship between a Test, an Argument, a Sub-Claim or a Claim. Instead, one piece of data can support multiple Arguments and an Argument can support multiple Sub-Claims or Claims. That is why it is so important to have a Structured Assurance Case Model.

DIDO Specifics

Return to Top

To be added/expanded in future revisions of the DIDO RA

1)

Achieving Requirements Testability, ProlificsTesting, 10 October 2018 Accessed on 9 August 2020 https://www.prolifics-testing.com/news/achieving-requirements-testability