4.3.3.5 Testability

Return to Maintainability

- ☐ **[char]Please Review**

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*.

DIKW pyramid level	Structured Assurance Case	Term	Description

DIKW pyramid level	Structured Assurance Case	Term	Description
Understanding	Software Assurance (SwA)	Testability	**Testability** is about the documenting functionality and the requirements for the system or the program and verifying that the requirements are going-to or have-been met. Functional requirements are generally not a problem since most of these are directly measurable or observable. Often Functional requirements are directly measured or observed such as a data field needing to be provided, a relationship exists between two pieces of data, or that the relationship is a one-to-many or a many=to-many. For example, every person must have a company ID number, they may have multiple phone numbers or people can belong to multiple organizations. Other functional requirements are not so definite, but are expressed in terms of a range of acceptable value. For example, a Graphical User Interface (GUI) will respond in less than 5 seconds. The heart pulse rate is between 35 to 200. However, non-functional requirements are generally more abstract and relate to the quality of the system or program being delivered (i.e., portable, reliable, maintainable, secureable, scalable, et.) Often these are not directly measurable or observable but can be inferred from characteristics found in the delivered system's or product's architecture, design and implementation. These kinds of requirements require assurance. The requirements are specified as claims (i.e., the system is has a High Availability) which require sub-claims, arguments (i.e., Availability can be predicted by using the 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 a Performance or Functional Specifications and their is tendency to only describe hardware specifications, but the performance specification can also capture non-functional metrics as well. • **Note** Testability metrics are not limited to operational systems or programs but can also use system or program level artifacts describing architecture, design, discussion papers, outside references, software and executables. Another problem when reviewing requirements and trying to determine if they are actually “testable”. Here is a list of some common “mistakes” found in requirement documents[1]: • **Noise:** Text containing no information relevant to any aspect of the problem. For example, 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 the specification • **Over-specification:** Description of the solution rather than of 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 the 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. • **Weak phrases:** Causing uncertainty (“adequate”, “usually”, “etc.”) For Example, ◦ When possible, the systems shall do … ◦ The system shall collect their data (whose data?) • **Jigsaw puzzles:** Requirements distributed across a document and then cross-referenced • **Duckspeak:** Requirements only there to conform to standards • **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 is testable or not. All requirements can not be tested. Some requirements can be directly tested for by running specific tests (i.e., Unit Testing, integration testing, etc.) using test plans that exercise the portion of the system or program software responsible for providing the functionality. For example, the system is suppose to offer the choice of none, one, many. Another example might be that when an option is selected, a message is sent out over the network. Other requirements are not directly testable by design and therefore 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, but the algorithm must provide an explanation and a proof that no two sets of conditions can produce the same UUID. Often there is a risk of generating the same UUID, but the chances of the UUID being used in identical is even smaller. Another example would the reCAPTCHA which shows a series of photos and asks the user to identify the one with green peas in it. 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 the individual modules before they are integrated into the system or program. There are various requirement conformity checks that can be done 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	Is about 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, Sub-Claim or a Claim. Instead, one piece of data can support multiple Arguments, an Argument can support multiple Sub-Claims or Claims. That is why it is so important to have a Structured Assurance Case Model.

DDS Specifics

Return to Top

☐ *TBD - 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

From:
https://omgwiki.org/dido/ - **DIDO Wiki**

Permanent link:
https://omgwiki.org/dido/doku.php?id=dido:public:ra:1.4_req:2_nonfunc:20_maintainability:testability&rev=1622661387

Last update: **2021/06/02 15:16**