4.3.3.4 Modifiability

Return to Maintainability

About

Return to Top

Modifiability is characteristic of a system to successfully support:

- Extensibility
- Updateability
- Deletability

In other words, it is a system or products ability and receptiveness to adapt to future and often unexpected changes. Planning for Modifiability is a bit like looking into a crystal ball about the product, its place within the ecosystem and its ability to adapt to changes in the environment. An easy way to consider Modifiability of a system or product is to ask the following series of questions suggested by McGovern et al. [1)]:

Table 1: Questions to consider when assessing Modifiability.

Questions proposed by McGovern et al.	Considerations
How often is it expected that a system change will be required?	This question is trying to understand the maturity (see 4.3.2.1 Maturity) associated with the system or product. It is not just about the product, but the maturity of the domain knowledge surrounding the system or product. For example, the accounting domain has been around for thousands of years and is well documented and understood while the use of applications to address Blockchains is less than a decade old.
What is the usual extent of the change?	This question also relates to maturity (see 4.3.2.1 Maturity) of the domain associated with the system or product. It also has to do with how conservative the attitude is towards changes. For example, changes made to an end-user entertainment application such as TikTok are easily tolerated while changes made to accounting records, which can adversely effect an individual's wealth, are frowned upon.

Questions proposed by McGovern et al.	Considerations
Who is expected to make the changes?	If changes are made by a single individual with minimal review and testing, the ability to modify the system is high, but the risk of failure is increased. Modifiability must consider these risks in the context of the domain. In essence, the better the governance over changes, the higher the probability of success (see 3 Governance). An individual can be extremely disciplined and positive when it comes to adapting to changes, while organizations can be sloppy. So, the maturity of the domain is important and the organization (even if it's a single person) is important. See ISO 90003-2018 and Capability Maturity Model Integration (CMMI).
Is it necessary for the system to use current platform versions?	This addresses the End-of-life (EoL) issues associated with any product (see 4.3.5 Manageability and 4.3.1.3 Replaceability). As the system ages, more and more EoL problems arise. As more Commercial Off-The-Shelf (COTS), Government Off-The-Shelf (GOTS), Modified Off-The-Shelf (MOTS), and NATO Off-The-Shelf (NOTS) products are used by the system and the longer the system exists the risk to the system increases because each subsystem, component or modular needs to be managed. As a case in point, in mid-2020, roughly 200 million PCs worldwide will still be running older Windows versions, mostly Windows 7[https://www.zdnet.com/article/how-many-pcs-are-still-running-windows-7-today/]]. Many of these are probably not modifiable any more.

Zarnekow et al. [2)] did a detailed study of 30 applications in 2015 and found the following time and cost characteristics and that over half (55%) of the cost of the projects can be attributed to maintenance and support. These findings underline the importance of Modifiability. All too often when a project gets started, too many problems are “kicked down the road” with the idea that “we'll cross the bridge when we get there”.

Table 2: Summary of time and cost characteristics found in 30 projects (Zarnekow et al.)

			Time			Actual Cost (in Mill of Euro)					
	# of Users	# of Transactions/yr	Total	Init Dev	Prod	Total Cost	Planning	Init Dev	More Dev	Prod	Shutdown
Minimum	160	23,900	2.0	0.3	0.4	0.50	0.01	0.13	0.00	0.14	0.00
Maximum	135,500	91,250,000,000	16.4	3.0	12.4	137.33	50.00	85.00	38.00	117.8	0.13
Average			5.6	1.7	3.1	35.74	2.97	11.57	5.52	15.6	0.08
% of total			100.0%	30.3%	55.3%	100.00%	8.31%	32.37%	15.44%	43.65%	0.2%

Another paper published by Björklund [3)] reported the cost of software maintenance as 67%.

Table 3: The Cost of Software Maintenance for one project

Lifecycle Phase	Percent of Cos
Requirements Definition	3%
Preliminary Design	3%
Detailed Design	5%
Implementation	7%

Lifecycle Phase	Percent of Cos
Testing	15%
Maintenance	67%

* **Note:**

- Another study found at least 50% of the effort spent on maintenance
- Another study found between 65% and 75% on maintenance
- In embedded real-time systems, maintenance costs may be up to 4 times development costs

Regardless of the actual numbers for a system or a program, all the numbers point to one conclusion: the cost of maintenance is a major driver in the total cost of ownership for systems or projects. Much of the maintenance cost for many projects can be traced back to not planning or considering Modifiability throughout the project lifecycle, especially early on. Modifiability is closely correlated to creating layered, modular and loosely coupled systems or programs. Fortunately, there are tools which can analyze a system or a program during all its phases (see 4.3.3.1 Modularity and 4.3.5 Manageability).

Layering involves separating the system or programs based on technical responsibilities, usually using an N-Tier Architecture. Generally, these tiers are referred to as the *presentation tier*, *processing tier* and *data management tier*. Often the tiers are both logically and physically separated, with each tier running on its own dedicated platforms. In a distributed system, the tiers do not follow the client-server architecture but use a Peer-to-Peer (P2P) architecture. However, the peers can be categorized as fulfilling presentation, processing, and data management functionality.

In addition, systems or programs that are declarative and configurable are more modifiable, especially if the configuration describes the details of connectivity between the modules (or peers). In other words, these descriptions provide the context and should address the 5-Ws of **who**, **what**, **when**, **where** and **why**; as well as, the **how**. For example:

Table 4: How the 5-Ws and H can be used to help ensure Modifiability

W	context
who	Who can access the module (peer) including privileges: developer, a business user, an analyst, or some combination of these is responsible
what	What is the module (peer) name, version, download URI
when	When can the module (peer) be accessed: event, calendar, time, etc.
where	Where can the module (peer) be found: paths, endpoints, etc.
why	Why is the module (peer) defined: documentation, rules, filters, etc.
how	How is the module (peer) accessed: Library, RESTFul, Remote Procedure Call (RPC), DDS, Message queue, etc.

Expectations of frequent changes driven by business-related changes can be more modifiable if the rules are not codified into software but stored as machine readable rules that can be interpreted at run time using, for example, rule engines. However, the downside of a data-rule driven system is that changes in data or rules can lead to crashes and adverse impacts to stability (see 4.3.5 Manageability).

DIDO Specifics

Return to Top

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

1)

James McGovern, Sameer Tyagi, Michael E. Stevens and Sunil Mathew, Java Web Services Architecture, 2003, ISBN 978-1-55860-900-6, Accessed 5 August 2020 https://www.sciencedirect.com/book/9781558609006/java-web-services-architecture

2)

Ruediger Zarnekow and Walter Brenner, Distribution of Cost Over the Application Lifecycle - A Multi-Case Study, University of St. Gallen, 22 July 2015, Accessed 5 August 2020, Researchgate

3)

Carl Björklund, App Maintenance Cost Can Be Three Times Higher than Development Cost, 15 April 2019, Accessed 5 August 2020, https://www.econnectivity.se/app-maintenance-cost-can-be-three-times-higher-than-development-cost/