4.3.3.2 Reusability

Return to Maintainability

☒

About

Reusability is a way to save time, resources and effort by reusing existing system or software assets that were created previously. Sometimes these assets maintain all the functionality they were originally constructed to provide (e.g., widgets in a Graphical User Interface (GUI)). Sometimes, well written modularized software is abstracted into generalized Modules that can then be used to serve one or more design patterns, e.g., a linked list, hash algorithm, etc.) When this is done, the software is considered to be reusable.

Reusability is defined as the *ability* of something to be used more than once. In contrast, *reuse* is defined as the *action* of using something more than once. Reusability promotes and enables reuse but does not ensure complete reusability.[1)]. Therefore, merely creating a system or program for Reusability is not important unless there is also a “marketing effort” to promote the reuse of the system or product. Marketing in this context does not mean commercial marketing. Often marketing can be achieved just through making the code available as Open Source.

The meaning and application of 'reuse' also depends on who you ask. To a Software Engineer, they naturally think about reuse of code, executables or even software patterns, etc. To a Systems Engineer, reuse makes them think about use cases, state transition diagrams, etc. To an Ontologist, reuse means glossaries, vocabularies and ontologies. To a contracting officer, they makes them think of contracts and performance specifications. Thus, it is obvious that even within a technology area, there are lots of different ideas about what, where, and how to re-use artifacts.

Jones[2)] identifies 10 different areas within a software project where reuse can be applied:

1. Architectures
2. Source Code
3. Data
4. Designs
5. Documentation
6. Estimates
7. Human Interfaces
8. Plans
9. Requirements
10. Test Cases

Types of Reuse

Return to Top

Reuse is a multi-faceted idea, which requires an overall classification for the various types of reuse. Frakes and Terry [3] have developed a taxonomy for reuse based on an extensive review of the literature on Reuse and Reusability. Table 1 depicts the two major elements of the reuse taxonomy: **Facets**, which cover all the types of reuse, and **Terms**, which are used to describe each **Facet**. For example, the **Development Scope Facet** has two possible **Terms** within it: **Internal** and **External**. When describing the **Development Scoping** used in a company or project, this can be **Internal**, **External** or both. The **Terms** associated with each **Facet** are defined in Table 2.

Table 1: Types of Software Reuse [3]

Facet					
Development Scope[1]	**Modification**[2]	**Approach**[3]	**Domain Scope**[4]	**Management**[5]	**Reused Entity**[6]
Internal (Private)	White Box	Generative	Vertical	Systematic (Planned)	Code
External (Public)	Black Box (vrbatum)	Compositional	Horizontal	Ad Hoc	Abstract Level
	Adaptive (porting)	In-the-Small			Instance Level
		In-The-Large			Customization Reuse
		Indirect			Generic
		Direct			Source Code
		Carried Over			
		Leveraged			

[1] **Development Scope** refers to whether the reusable components are from a source external or internal to a project.
[2] **Modification** refers to how much a reusable asset is changed.
[3] **Approach** refers to different technical methods for implementing reuse.
[4] **Domain Scope** refers to whether reuse occurs within a family of systems or between families of systems.
[5] **Management** refers to the degree to which reuse is done systematically.
[6] **Reused Entity** refers to the type of the reused object.

Clear definitions of types of reuse are necessary prerequisites to measurement. Table 2 provides a faceted classification of reuse definitions gathered from the literature Each column specifies a facet, with the facet name inbold.

Table 2: Definitions of Types of Reuse [3]

Type of Reuse	Description
abstract-level	Abstract-level reuse is the use of high-level abstractions within an object-oriented inheritance structure as the foundation for new ideas or additional classification schemes.
ad-hoc	Ad-hoc reuse refers to the selection of components that are not designed for reuse from general libraries or where reuse is conducted by an individual in an informal manner
adaptive	Adaptive reuse is a reuse strategy that uses large software structures as invariants and restricts variability to low-level, isolated locations. An example is changing arguments to parameterized modules.
black-box	Black-box reuse is the reuse of software components without any modification. See **verbatim**.
Carry-Over	Carry-Over Reuse is when software used with one version of a software component is carried over and used as is in a subsequent version of the same system.
compositional	Compositional reuse is a reuse strategy that uses small parts as invariants and then uses variant functionality to link these parts together. Programming in a high level language is an example. Compositional reuse is the use of existing components as building blocks for new systems. The Unix shell is an example
customization	Customization reuse is the use of object-oriented inheritance to support incremental development. A new application may inherit information from an existing class, overriding some methods in that class and adding new behaviors.
direct	Direct reuse is reuse without going through an intermediate entity.
external	External reuse level is the number of lower level items from an external repository in a higher level item divided by the total number of lower level items in the higher level item. See **Public**.
generative	Generative reuse is reuse at the specification level using application or code generators. Generative reuse offers the “highest potential payoff.” The Refine and MetaTool systems are state of the art examples.
generic	Generic reuse is reuse of generic packages, such as templates for packages or subprograms.
horizontal	Horizontal scope reuse is reuse of generic parts in different applications. Booch Ada Parts and other subroutine libraries are examples.
In-the-large	Reuse-in-the-large is the use of large, self-contained packages such as spreadsheets and operating systems.
In-the-small	Reuse-in-the-small is the reuse of components that are dependent on the environment of the application to achieve full functionality. Favaro asserts that component-oriented reuse is reuse-in-the-small.
indirect	Indirect reuse is reuse through an intermediate entity. The level of indirection is the number of intermediate entities between the reusing item and the item being reused
instance-level	Instance-level reuse is the most common form of reuse in an object-oriented environment. It is defined as simply creating an instance of an existing class.
internal	Internal reuse level is the number of lower level items not derived from an external repository and used more than once divided by the total number of lower level items not derived from an external repository. See **Private**.
leveraged	Leveraged reuse is reuse with modifications.

Type of Reuse	Description
private	Private reuse is"the extent to which modules within a product are reused within the same product." See **Internal** .
public	Public reuse is "the proportion of a product which was constructed externally." See **External**.
source-code	Source code reuse is the low-level modification of an existing object-oriented class in order to change its performance characteristics.
systematic (planned mode)	Systematic/planned mode reuse is the systematic and formal practice of reuse as found in software factories.
verbatim	Verbatim reuse is reuse of some item without modifications. See **Black-Box**.
vertical scope	Vertical scope reuse is reuse within the same application or domain. An example is domain analysis or domain modeling.
white-box	White-box reuse is the reuse of components by modification and adaptation.

Types of Metrics and Models

Return to Top

Frakes and Terry categorize the kinds of metrics and models that can be used for evaluating reuse.

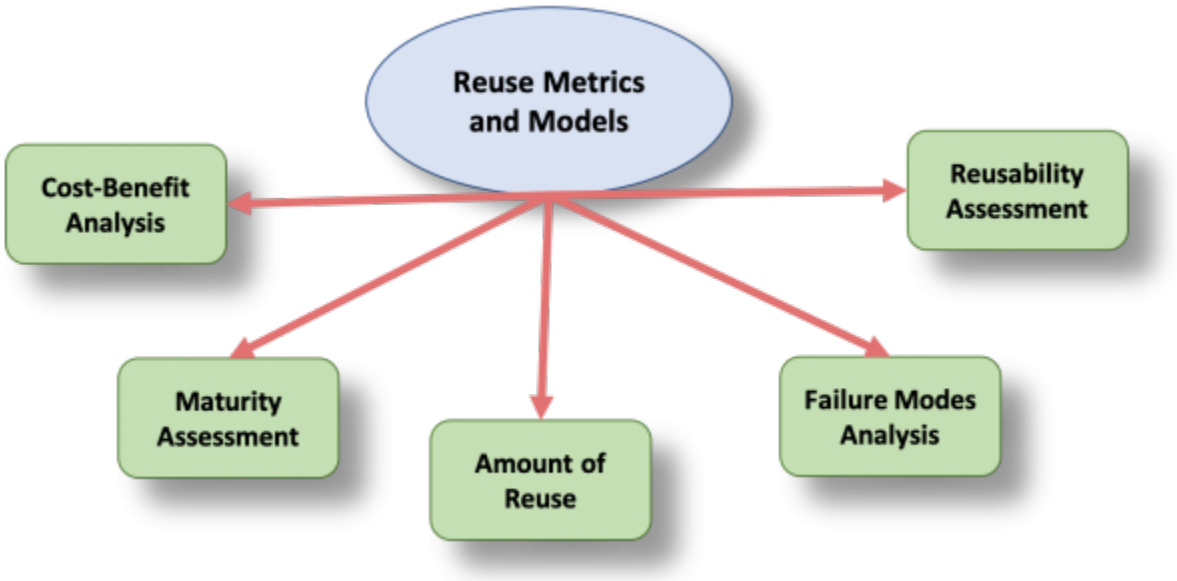

Figure 1: Categorization of Reuse Metrics and Models [3)]

Table 3: Descriptions of Reuse Metrics and Models [3)]

Metric or Model	Description
Cost-Benefit Analysis	Uses **models** to include economic cost/benefit analysis, as well as, quality and productivity payoffs. Maturity assessment models categorize reuse programs by how advanced they are in implementing systematic reuse.
Maturity Assessment	Uses **models** to categorize reuse programs by how advanced they are in implementing systematic reuse.
Amount of Reuse	Uses **metrics** to assess and monitor a reuse improvement effort by tracking the percentages of reuse for life cycle objects.

Metric or Model	Description
Failure Modes Analysis	Identifies and orders the impediments to reuse in a given organization.
Reusability Metrics	Indicates the likelihood that an artifact is reusable.
Reuse Library Metrics	Manages and Tracks usage of a reuse repository.

* **Note:** Organizations often encounter the need for these metrics and models in the order presented.

DIDO Specifics

Return to Top

Startup initialization is minimal because of DDS Discovery.
☐ *TBD - May be added/expanded in future revisions of the DIDO RA*

1)
Luis Zafra, Know the Difference: Reusability vs. Reuse, Global Logic, 22 September 2015, Accessed 3 August 2020, https://www.globallogic.com/latam/blog/know-the-difference-reusability-vs-reuse/
2)
Jones, C. Software return on investment preliminary analysis, 1993, Software Productivity Research, Inc.,
3)
William Frakes and Carol Terry, Software Reuse and Reusability Metrics and Models, Virginia Tech, Accessed on 4 August 2020, https://pdfs.semanticscholar.org/53d3/37f49c7d1ef98968ae5ed7e699096974db10.pdf