4.3.3.2 Reusability

Return to Maintainability

- ☐ **[char]Please Review**

About

Reusability is one way to save time, resources and effort by reusing existing system or software assets that were created previously. Sometimes the assets maintain all the functionality they were originally constructed for (i.e., widgets on a Graphical User Interface (GUI)). Sometimes, well written, modularized software is abstracted into generalized Modules that solve a particular pattern. In these cases, the software is made reusable to support new purposes (i.e., a linked list, hash algorithm, etc).

Reusability is the ability of something to be used more than once. Reuse is the action of using something more than once. Reusability promotes and enables reuse but does not ensure it.[1)]. Therefore, creating a system or program for Reusability is not important unless there is a “marketing effort” to help the system or product be reused. Marketing in this context does not mean the same as commercial marketing. Often the marketing can be done just through making the code available as Open Source.

The meaning of 'reuse' depends on who you ask. To a Software Engineer, they naturally think about reuse of code, executable 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. Finally, ask a contracting officer about reuse and they will think of contracts and performance specifications. So, obviously 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 just within a software project where reuse can play a part:

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 that requires a classification for the types of reuse. Frakes and Terry [3] have done an extensive review of the literature on Reuse and Reusability and have developed a taxonomy for reuse. Table 1 captures types of reuse called **Facets** in the taxonomy and uses **Terms** to describe each **Facet**. For example, the **Development Scope Facet** has two possible **Terms** within it: **Internal** and **External**. When describing the the **Development Scoping** used in a company or project, it can be **Internal**, **External** or both. The **Terms** used within 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-levelreuse	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 which are not designed for reuse from general libraries; reuse is conducted by the individual in an informal manner

Type of Reuse	Description
adaptive	Adaptive reuse is a reuse strategy which 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 Reuse	Carry-Over Reuse is when software that occurs when one version of a software component is taken to be used as is in a subsequent version of the same system.
compositional	Compositional reuse is a reuse strategy which uses small parts as invariants; variant functionality links those 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 reuse	Customization reuse is the use of object-oriented inheritance to support incremental development. A new application may inherit information from an existing class, overriding certain methods 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 with 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 scope	Horizontal 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 which are dependent upon the environment of the application for 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 reuse	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 from an external repository which are used more than once divided by the total number of lower level items not from an external repository. See **Private**.
leveraged	Leveraged reuse as reuse with modifications.
private	Private reuse as “the extent to which modules within a product are reused within the same product.” See **Internal** .
public	Public reuse as “the proportion of a product which was constructed externally.” See **External**.

Type of Reuse	Description
source-codereuse	Source code reuse is the low-level modification of an existing object-oriented class to change its performance characteristics.
systematic(planned mode)	Planned reuse is the systematic and formal practice of reuse as found in software factories.
verbatim	Verbatim reuse as reuse of some item without modifications. See **Black-Box**.
vertical scope	Vertical 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 also go on to develop a categorization of the kinds of metrics and models that can be used for evaluating reuse.

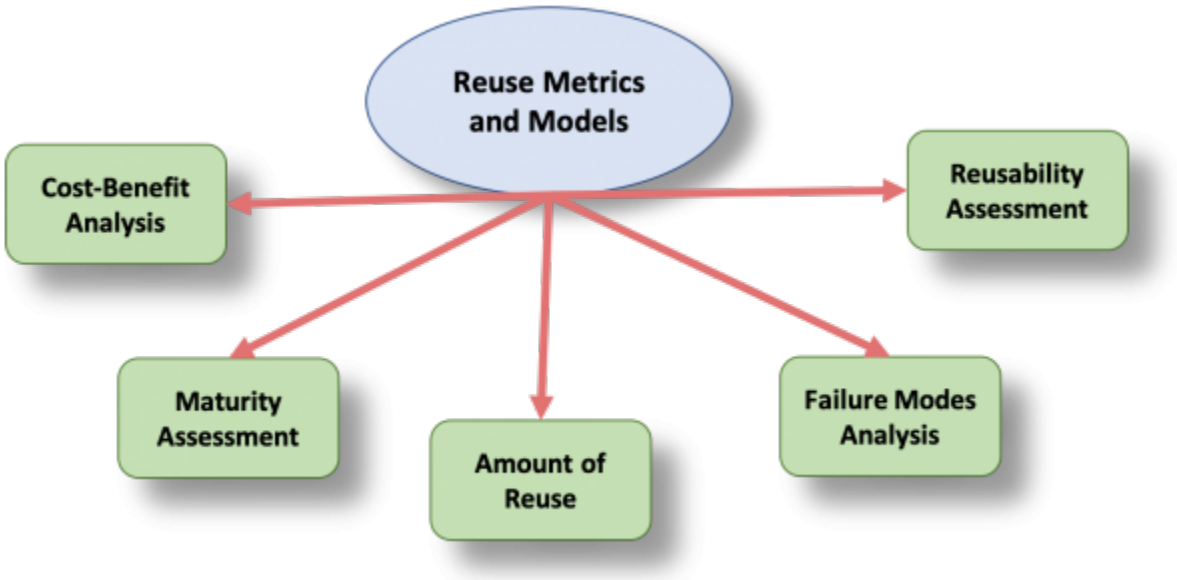


Figure 1: Categorization of Reuse Metrics and Models [3)]

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 payoff. 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 percentages of reuse for life cycle objects.
Failure Modes Analysis	Identifies and order the impediments to reuse in a given organization.

Metric or Model	Description
Reusability Metrics	Indicates the likelihood that an artifact is reusable.
Reuse Library Metrics	Manage 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

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:reuseability&rev=1622857254

Last update: **2021/06/04 21:40**