A Generalized Reuse Framework for Systems Engineering

Gan Wang(✉)

Dassault Systèmes, Herndon, VA 20171, USA
gan.wang@3ds.com

Abstract. Reuse in system development is a prevalent phenomenon. However, how reuse is applied varies widely. The Generalized Reuse Framework is a strategic reuse model for systems engineering management in product development that addresses both investment and leverage of reuse through two interrelated and interacting processes: Development with Reuse (DWR) and Development for Reuse (DFR). This chapter summarizes the latest development of this framework by providing the taxonomic definition of DWR and DFR and analyzing the decision processes for reuse as applied to incremental development and product line engineering. It also describes how the framework is applied to the revision of the Constructive Systems Engineering Cost Model (COSYSMO), a parametric cost estimating model for systems engineering. With use case scenarios, it illustrates the approach to apply the framework and to quantify the economic impact of reuse vis-à-vis investment strategies.

Keywords: Reuse · Systems Engineering · System Development · Modeling · Parametric Estimating · Cost Estimation and Analysis · Project Planning

1 Introduction

As industry embarks on an accelerated digital engineering (DE) transformation journey aiming for more efficient and effective capabilities for developing and sustaining systems, organizations seek to invest in digital infrastructure, model-based business processes, and a digitally capable workforce. At the same time, they continue to search for ways to quantify returns on investment.

As widely accepted in the SE community, reuse is essential and sometimes even imperative for today's system development (De Weck, et al, 2003; Clements, 2015; Le Put, 2015). Reuse is fundamentally driven by economic necessities and ever-mounting pressure to deliver value and profitability to shareholders. A common rationale is cost-saving through reduced work and improved quality (Nazareth and Rothenberger, 2004; Selby, 1989, 2005). However, a more recent focus is the speed of delivery and time to market. In his keynote speech, Jan Bosch (Bosch, 2014) pointed out a relatively contemporary trend of modern systems from "built to last" to "built to evolve" to adapt to a faster pace of technology change and user expectations. Reuse, especially when

D. Krob et al. (Eds.): CSD&M 2023, LNEE 1085, pp. 9–24, 2023.
https://doi.org/10.1007/978-981-99-6511-3_2

strategically planned, can be a fundamental enabler for the evolutionary approach to system development.

Reuse can be *opportunistic* in that a designer or developer spends effort searching and discovering reusable resources when the need arises inside and outside their organization and then, if successful, attempts to use the artifacts they find. For example, someone can do a Google search for freeware and lift a code segment that appears to fit a purpose. In software development, it is sometimes called "code scavenging." The outcome is almost inconstant depending upon what is available. In some cases, the reused code has to be modified. In almost all cases, its behavior has to be tested and verified. If anything fails, the developer has to debug the issues and fix the defects. This ad hoc reuse approach involves a process of discovery, assessment, modification, integration, and testing. Its main effort focuses on leveraging benefits through opportunity. For smaller efforts, this approach can be successful. But it does not scale well. The reusability is generally low and uncertainty high, especially for larger and more complex systems.

On the other side of the spectrum, reuse is *planned* and *strategic* (Kim and Stohr, 1998; Hillhouse, 2011). In this case, the developer proactively and strategically invests in reusable resources through explicit reuse processes and standards. A major focus is the up-front investment effort to make an effort easier when the actual reuse occurs later. Examples that come to mind include object-oriented software development. An object class (typically called base class) is defined to instantiate other classes of objects (called derived classes), which inherit the properties of the base class. Another strategy is to create libraries that encapsulate particular objects and functionalities that can be used in multiple applications through a set of application programming interfaces (APIs). Modern programming languages like Python and MATLAB contain large libraries of reusable objects and patterns.

Developed through collaboration with the University of Southern California (USC) Center for Systems and Software Engineering (CSSE) and based on a series of studies (Wang, et al, 2014, Wang, 2016), the *Generalized Reuse Framework* (GFR) defines two interrelated and interacting reuse processes:

- *Development with Reuse* (DWR): a set of system development activities that focus on gaining benefits from utilizing or leveraging previously developed reusable artifacts, either in a planned process or an ad hoc manner
- *Development for Reuse* (DFR): a set of system development activities dedicated to developing reusable artifacts for future usages, generally in a planned manner or through an investment effort

DFR and DWR represent the two foundational processes that bridge reuse in system development projects or, in a broader sense, for any efforts that deliver products or services. The two processes are distinguished only by intent – production and consumption of reusable resources. DWR focuses on benefits gained from using reusable resources. The basic assumption is that DWR saves labor and improves the product quality at the same time for the system that leverages these resources. DFR, on the other hand, aims to create reusable products for future usage. Acting as a producer, the DFR process feeds reusable resources into the DWR process that consumes them in its development effort. See Fig. 1.

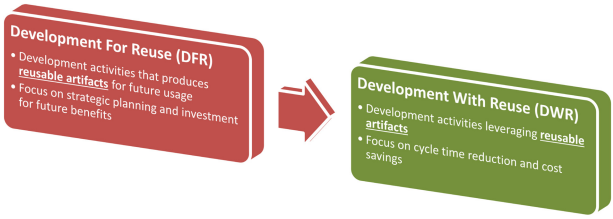


Fig. 1. The Generalize Reuse Framework consists of two interactive processes: the DFR process feeds reusable resources into the DWR process that leverages these resources

The basic premise, however, is that DFR may incur additional upfront costs than without such a consideration to gain the benefits in DWR. But in the aggregate, it will save cost from the lifecycle point of view. Table 1 contrasts the major characteristics of the two processes.

Table 1. Contrasting two reuse processes – "Development with Reuse" and "Development for Reuse"

	Development with Reuse	Development for Reuse
Role	• Consumer	• Producer
Purpose	• Consumption of reusable resources	• Production of reusable resources
Goal	• Improvement of product quality • Cost savings • Improved speed of delivery or time to market	• Investment for future benefits
Challenges	• Discovery of what to reuse • Decisions on how to tailor and integrate	• Plans for how to reuse • Design for reusability • Means to verify
Reusability	• If ad hoc, then generally low • If planned, then generally high	• Generally high

This process model addresses reuse as a central consideration for development strategy as applied to agile SE, product line engineering, and the evolution of system capability through incremental development. It assumes that a product or product line (whether it is a vehicle, aircraft, electronics, or software) is developed by a series of projects, each of which produces several articles or baseline versions. In each article or version release, there is a mix of the DWR and the DFR contents. With careful planning, the DWR and DFR content mix changes favorably with an increasing number of articles produced or release versions deployed, while the total incurred project effort (red line) decreases, as shown in Fig. 2. As the product line matures, the percentage of the DFR content decreases while relative DWR content increases. This phenomenon can be viewed as the investment paying off over time.

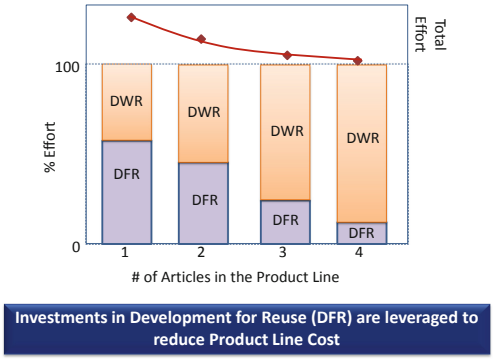


Fig. 2. A mix of DWR and DFR efforts in a project and the decreasing levels of DFR effort in the successive article releases of a project line

2 The Generalized Reuse Framework

2.1 Reusable Resources

What can be reused? A simple answer is almost everything. From an engineering perspective, however, it is the outputs from the SE processes that can be reusable, which include elements of the realized system. Then the question is, how do we express them?

We define a reusable resource as a collection of system artifacts that represent certain attributes of the system. System artifacts are physical and functional components (e.g., a piece of hardware, a software module) of a system, along with all the associated engineering and design data that specify the system at different stages of its life cycle.

A design specification can be considered as a collection of system attributes represented in requirements and logical, functional, and physical architecture descriptions. It can be functional or non-functional (performance) based. For example, a system attribute can be a system interface that is realized with a physical hardware connection or a software object that pulls or pushes data. Or it can be a function depicting a behavior in terms of an input and output relationship and that is realized by a hardware control logic or implemented by a software algorithm.

Together, these system attributes represent what the end system is and how it functions. As a system attribute is realized through a development life cycle, a set of system artifacts are produced and, over time, culminate in the actual system built and deployed. Reversely, a system is simply an integrated collection of system artifacts developed that, together, satisfy the specified system attributes.

For definitions of the GRF process coming up next, we use the term *system attribute* to represent all the reusable resources.

2.2 The Reuse Process

Development with Reuse. The DWR consists of six (6) categories, as shown in Table 2 below.

Table 2. Definitions of the DWR categories

New	*A system attribute that is new or unprecedented, which requires developing from scratch; or from previously defined system design or constructed product components but requiring near-complete changes in system architecture as a result of that requires developing from scratch; or from previously defined system design or constructed product components but requiring near-complete changes in system architecture due to modified or extended system functionalities*
Design Modified	*A system attribute that is designed and developed by leveraging previously defined system concept, functional and logical reference architecture; or from previously designed physical architecture or constructed product components that require significant design and implementation changes or refactoring but without major changes in intended system functionality*
Design Implemented	*A system attribute that is implemented from an inherited, completed system design or a previously constructed product component that may require only limited design changes in the physical architecture to the extent that it will not impact or change the basic design, but that may require reimplementation of the component*
Adapted for Integration	*A system attribute that is integrated by adapting or tailoring (through limited modification of interfaces) of previously constructed or deployed product components without changes in the core architecture, design, or physical implementation (except for those related to interface), so that the adapted element can be effectively integrated or form fit into the new system. The change effort required is less than that of the Design Implemented category. Removing a system element from a previously developed or deployed system baseline is also included in this category*
Adopted for Integration	*A system attribute that is incorporated or integrated from previously developed or deployed product components without modification, which requires complete integration, assembly, test and checkout activities, as well as system-level V&V testing. This is also known as "black-box" reuse or simple integration*
Managed	*A system attribute that is inherited from previously developed and validated product components without modification or that the integration of such an element, if required, is through significantly reduced V&V testing effort by inspection or utilizing provided test services, procedures, and equipment (so called "plug and play"). Most of the SE effort incurred is a result of technical management*

The terminology for the category names is chosen in such a way that, colloquially, one can say a system attribute (e.g., requirement, interface, etc.) is "*New*," "*Design Modified*," "*Design Implemented*," and so forth.

The DWR categories capture the amount of work required to realize a system attribute in the final system by leveraging those system artifacts available for reuse at the time.

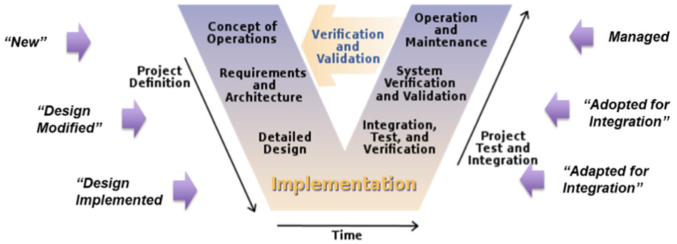


Fig. 3. Typical entry points for the work required by the DWR categories relative to the system "V" model, as a reference for different maturity level of the reusable resources

From a lifecycle perspective, this work typically corresponds to the lifecycle stage in which relevant artifacts are available, as shown in Fig. 3. The arrows indicate the typical entry point for most of the development work. For example, "*Adopted for Integration*" typically commences during the integration, test, and verification phase as it is possible to leverage artifacts from existing system implementation.

The concept is, in a sense, similar to that of the Technical Readiness Level or TRL. The use of system "V" model provides a convenient reference for the maturity of the system attributes under the reuse consideration. In other words, an attribute deemed "*Adopted*" must have sufficient maturity to be "*Adopted*" or integrated without any modifications. However, the DWR process is not limited to waterfall development and can apply other development models, such as agile development processes.

Development for Reuse. The DFR process consists of five (5) categories, as shown in Table 3 below.

Table 3. Definitions of the DFR categories

No DFR	No development for reuse within the planned work scope
Conceptualized For Reuse	*This category includes a set of front-end SE activities that produce conceptual, contextual, logical and/or functional architecture elements intended for future reuse, that must be further developed through a series of detailed design, implementation, verification and validation testing activities to realize the final deployable product*
Designed For Reuse	*This category includes a set of front-end system design activities that produce a complete system design or physical architecture elements intended for future reuse, that must be further developed through a series of implementation, integration, and verification and validation testing activities to realize the final deployable product*
Constructed For Reuse	*This category includes a set of system development activities that produce a physical product or component intended for future reuse, that has been implemented and independently verified through verification testing but has not been deployed or used in an end system. These activities include required efforts at all levels of design and development, just short of final system-level integration, transition, verification, and validation testing*
Validated For Reuse	*This category includes all system development activities that produce an end physical product or component intended for future reuse and operationally validated through its use in an end system*

Similarly, as in the case of DWR, Fig. 4 shows the general exit points for the DFR process relative to the system "V" model for different categories of reusable artifacts.

For example, if the development activity stops after the *Detailed Design* phase, the reusable artifacts generated would be at the level of "*Designed for Reuse.*" On the other hand, if a system component has been built and united tested, it should be categorized as "*Constructed for Reuse*", ready to be integrated into a future DWR process.

As in the case of DWR, the use of "V" model is only a reference for reuse maturity. The DFR process is not limited to waterfall development and can apply other development models, such as agile develop processes.

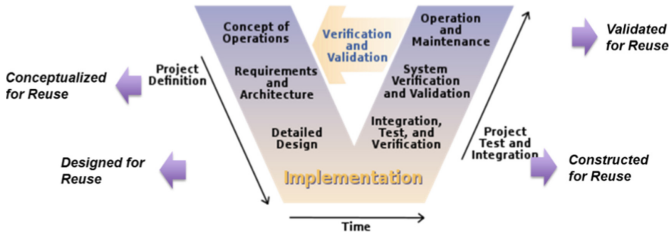


Fig. 4. The exit points of the DFR categories relative to the "System V" model, as an indication for different levels of reuse maturity

The DFR categories capture the work required to develop reusable artifacts at varying maturity levels. The DFR process can be a separate investment effort or occur in the same project as the DWR process. In the latter case, the same system attribute should be classified twice – once for DFR and second for DWR in an appropriate category.

2.3 GRF Usage Scenarios

We consider four basic scenarios of reuse strategy: reuse of 1) concept definition; 2) system design; 3) system implementation; and 4) valided system component. This achieved by managing different system artifacts between the DWR and the DFR processes. Figure 5 shows the four classifications in the DFR space and the logical paths to the respective DWR categories, depending on the reusability the system attributes or the amount of modifications required when applied to the DWR space. For example, if the system design is completely for a system attribute and it is classified as a "*Designed for Reuse*" in the DFR space, it can classified as "*Design Implemented*" for the DWR effort and we can directly proceed to implementation, if no additional design changes are needed. Otherwise, if design changes are needed, we must fall back to Conceptualized for Reuse and follow the logical path accordingly.

Importantly, the question should be asked in the reversed direction. If certain reusability is desirable in the DWR process, what level of investment is required for a DFR process? This thought process is critically important for systems engineers and product managers in planning strategic reuse.

The decision process associated with each case can serve as a guideline for a product line manager to consider in planning the work scope and estimating the development effort, both from an investment angle and leveraging that investment. This thought process is critically important for systems engineers and product managers in planning strategic reuse.

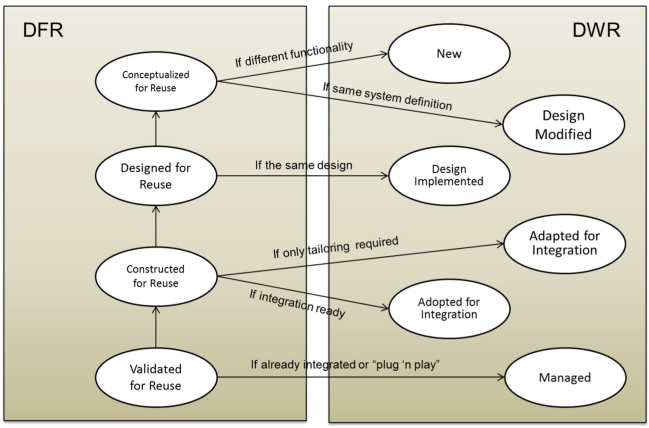


Fig. 5. Relationships of the DFR and DWR categories

3 An Illustrative Example – Application of the Generalized Reuse Framework to Cost Estimating and Analysis

Cost estimating and analysis is crucial to all modern systems engineering practices. It is an integral part of system development and acquisition. As a fundamental pillar for economic analysis and business decision-making, cost estimation is critical in evaluating the merit of system architecture and provides the essential criteria for design trade space.

Parametric estimating is based on statistical analysis of historical data. In essence, it uses parametric equations between cost (and effort) and one or more parameters. These parameters are derived from the characteristics of the system under estimation and may be physical, performance, operational, programmatic, or cost in nature. They are the independent variables into the equation and commonly known as *cost drivers.* The parametric equation is commonly known as *Cost Estimating Relationship* (CER). It is based on the statistical inferences of multiple similar systems or development efforts. Its output is cost or effort required for development of the system.

Parametric estimating is commonly recognized as one of the most effective and the most trusted cost estimating methods. It is often preferred by the source selection authorities for system acquisition.

3.1 COSYSMO

COSYSMO or the *Constructive Systems Engineering Cost Model* (Valerdi, 2005) developed at the University of Southern California, is a parametric model for estimating the end-to-end systems engineering and integration (SE&I) effort required in developing and deploying a system. Evolved from COCOMO, or the *Constructive Cost Model* (Boehm, 1981, Boehm et al., 2000), COSYSMO defines a CER that estimates the SE&I effort in a development project using four sizing parameters, also known as *size drivers*:

- System requirements (REQ)
- System interfaces (INT)

- Critical system algorithms (ALG)
- Operational scenarios (SCN)

The nominal effort is further adjusted by fourteen *effort multipliers*, also known as *cost drivers*, representing the product and project environment and complexity factors.

Mathematically, the effort, under a nominal schedule, is described as a function of weighted counts of the four size drivers Eq. (1):

$$PM_{NS} = A \cdot \left(\sum_{k} (w_{e,k}\Phi_{e,k} + w_{n,k}\Phi_{n,k} + w_{d,k}\Phi_{d,k}) \right)^{E} \cdot \prod_{j=1}^{14} EM_j \qquad (1)$$

where,

PM_{NS} = effort in Person Months (Nominal Schedule)
A = calibration constant derived from historical project data
k = {REQ, IF, ALG, SCN}
w_x = weight for "easy," "nominal," or "difficult" size driver
Φ_x = quantity of "k" size driver
E = represents (dis)economies of scale
EM_j = effort multiplier for the jth cost driver; the geometric product results in an overall effort adjustment factor to the nominal effort.

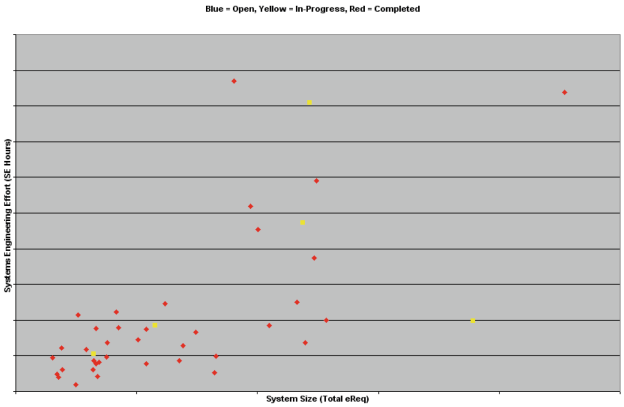
On an intuitive level, the weighted sum term in the equation above describes how "big" a system is and represents the "size" of the job for the development effort. We call it the "*system size*," which aggregates the effect of four size drivers into a single quantify, with a unit of measure called "*eReq*" or equivalent requirements. The size drivers are each counted at three levels of difficulty.

For a detailed description of the model, including the definitions of the size and cost drivers, as well as the quantitative aspects of its CER, please refer to (Valerdi, 2005). The remaining discussion in this section assumes the reader is familiar with the basic model definition.

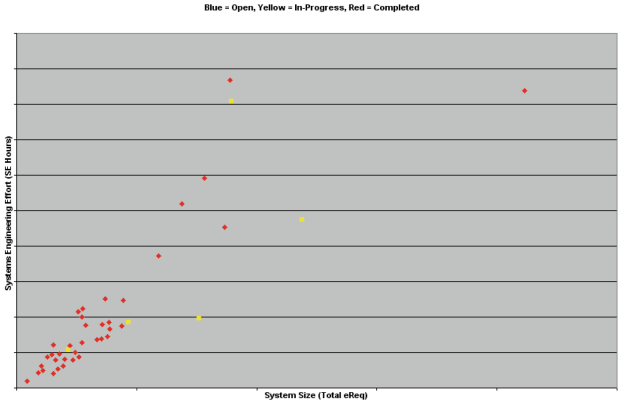
This model definition, known as COSYSMO version 1.0, does not consider reuse. As represented in Eq. (1), the CER implicitly assumes that all system developments start from scratch – not a very realistic and practical situation in today's world.

The lack of reuse semantics proved to be problematic in practical applications. It severely affects the estimating accuracy and cost realism (Wang, et al, 2008). An initial effort was to provide a formal construct for reuse similar to the DWR definition today (Wang, et al, 2010) and to augment the COSYSMO model definition. Practical applications of this extended COSYSMO proved effective and significantly improved the estimating accuracy. Figure 6 compares a set of historical project data, collected and validated in COSYSMO (Wang, et al, 2008), with two scatter plots showing the same set of projects captured before and after the reuse extension. Figure 6(a) shows the projects captured in COSYSMO version 1.0, without consideration of reuse and there is no distinctive trend observed. Figure 6(b) shows the same data set, now captured with an application of GRF, and the data converges and displays a distinct trend.

Additional studies (Wang, et al., 2011, 2013–14) expanded the model to account for the entire scope of reuse – both the investment and the leverage of reusable resources – leading to the GRF today.



(a) Without consideration of reuse and all size drivers are effectively counted as new



(b) With consideration of reuse and size drivers are classified according to respective reuse categories

Fig. 6. Scatter plots of the same dataset representing a group of historical system development projects, before (a) and after (b) implementing a reuse extension in COSYSMO

3.2 The GRF-Based Cost Estimating Relationship

The Generalized Reuse Framework extends the COSYSMO model definition to account for both the efforts of DWR and DFR. Thus, better characterizes the modern system development process and as a result, significantly improves the fidelity of the estimated cost.

The generalized reuse framework inherently implies two parallel efforts in a development project, each driven by the DWR and DFR processes. Therefore, the total project effort is the sum of the two, as below:

$$PH_{Total} = PH_{DWR} + PH_{DFR} \tag{2}$$

where,

PH_{Total} = total SE effort in person-hours for the entire project
PH_{DWR} = total SE effort in person-hours spent on the DWR process

PH_{DFR} = total SE effort in person-hours spent on the DFR process

In essence, PH_{DWR} is the total effort dedicated to developing the end system with benefit of reuse. It is less than what it would be if it started from a "clean slate". PH_{DFR} is the effort devoted to developing reusable artifacts that could be reused either within the same project or in future projects.

Thus, the total effort can be expressed as

$$PH_{Total} = A_{DWR} \cdot SS_{DWR}^{E_{DWR}} \cdot CEM_{DWR} + A_{DFR} \cdot SS_{DFR}^{E_{DFR}} \cdot CEM_{DFR} \tag{3}$$

where, SS represents the system size under development in the DWR and the DFR processes, respectively. It is with the same unit of measure of "*eReq*" or equivalent requirements, as in Eq. (1). The respective system size is expressed mathematically as

$$SS_{DWR} = \sum_{k} \left(\sum_{r} w_r (w_{e,k} \Phi_{e,k} + w_{n,k} \Phi_{n,k} + w_{d,k} \Phi_{d,k}) \right) \tag{4}$$

and

$$SS_{DFR} = \sum_{k} \left(\sum_{q} w_q (w_{e,k} \Psi_{e,k} + w_{n,k} \Psi_{n,k} + w_{d,k} \Psi_{d,k}) \right) \tag{5}$$

where,

Φ_x = quantity of "k" size driver, accounted for in the DWR process
Ψ_x = quantity of "k" size driver, accounted for in the DFR process
$k = \{REQ, IF, ALG, SCN\}$
$e = \{Easy, Nominal, Difficult\}$
$r = \{New, Design Modified, Design Implemented, Adapted for Integration, Adopted for Integration, Managed\}$
$q = \{No DFR, Conceptualized for Reuse, Designed for Reuse, Constructed for Reuse, Validated for Reuse\}$
w_x = weight for "easy," "nominal," or "difficult," for the respective size driver
w_r = weight for defined DWR levels of the respective size driver
w_q = weight for defined DFR levels of the respective size driver
A_{DWR} = calibration constant for DWR, typically derived from historical project data
A_{DFR} = calibration constant for DFR, typically derived from historical project data
E_{DWR} = nonlinearity for the DWR productivity curve, representing a diseconomy of scale
E_{DFR} = nonlinearity for the DFR productivity curve, representing a diseconomy of scale
CEM_{DWR} = composite effort multiplier for DWR
CEM_{DFR} = composite effort multiplier for DFR

Putting it all together, we can express the CER for the total project effort, including both the DWR and DFR efforts, as

$$PH_{Total} = A_{DWR} \cdot \left[\sum_k \left(\sum_r w_r (w_{e,k}\Phi_{e,k} + w_{n,k}\Phi_{n,k} + w_{d,k}\Phi_{d,k}) \right) \right]^{E_{DWR}} \cdot CEM_{DWR}$$
$$+A_{DFR} \cdot \left[\sum_k \left(\sum_q w_q (w_{e,k}\Psi_{e,k} + w_{n,k}\Psi_{n,k} + w_{d,k}\Psi_{d,k}) \right) \right]^{E_{DFR}} \cdot CEM_{DFR} \quad (6)$$

This cost estimating relationship captures the total project effort, including the part for investment and the part with the benefit of reuse. It shows that a development project may contain both the DFR and DWR efforts, in different proportions. Reversely, a system attribute may purposefully be developed in both the DFR and the DWR processes. When counting the size drivers for a system or a project, we classify them in the corresponding DWR and DFR categories, respectively, to accurately account for the collective effort.

For example, a critical system algorithm may be implemented as part of a standard library and it would be classified as *Constructed for Reuse* in the DFR process. It can then be used in the same project and would be classified as *Adopted* in the DWR process, if it can be directly integrated into the end system.

As any experienced systems engineer would say, "Not all requirements are created equal!" COSYSMO extended by the GRF is a powerful proof of that statement.

3.3 The DWR and DFR Weights

The reuse weight values in Eq. (6) were obtained through a series of wide-band Delphi Analyses and further validated with the dataset shown in Fig. 6. They are elaborated next.

3.3.1 The DWR Weights

Table 4 provides the derived values for the DWR category weights. The percentage values in the first row are the relative weights for the six DWR categories, with *New* at 100% and other categories at an incrementally lower level. Mathematically, they are the numerical values for the coefficients, w_r, in Eq. (6). At an intuitive level, they represent the partial set of SE activities required to realize a size driver (REQ, IF, ALG, or SCN) in the end system due to leveraging reuse, relative to the complete set of end-to-end activities corresponding to New. The rest of the 24 (4x6) decimal values in the table correspond to the individual DWR weights for each of the four size drivers, in respective rows, at the nominal level of difficulties.

A graphical representation is provided in Fig. 7 that gives an intuitive impression of the relative weight distributions for the four size drivers in each of the six reuse categories. The height of the bars corresponds to the decimal values in the table above in the respective reuse categories. The downward trend for each of the size drivers (shown in a different color) represents a decreasing level of required development effort due to the benefit of increasing levels of reuse (from *New* to *Managed*), as previously elaborated.

Table 4. The numerical weights of the six DWR categories for each of the four COSYSMO size drivers at the nominal level of difficulty

	New	Modified	Impl'ted	Adapted	Adopted	Managed
	100.00%	66.73%	56.27%	43.34%	38.80%	21.70%
System Requirements	1.00	0.67	0.56	0.43	0.39	0.22
System Interfaces	2.80	1.87	1.58	1.21	1.09	0.61
Critical Algorithms	4.10	2.74	2.31	1.78	1.59	0.89
Operational Scenarios	14.40	9.61	8.10	6.24	5.59	3.13

When resolving the weight values for the three-dimensional weighted sum term in Eq. (6), we get four sets of 18 (3×6) weight values, one for each of the four size drivers, or 72 values in total. This is the result of expanding the six DWR categories to consider all three levels of difficulties at the same time.

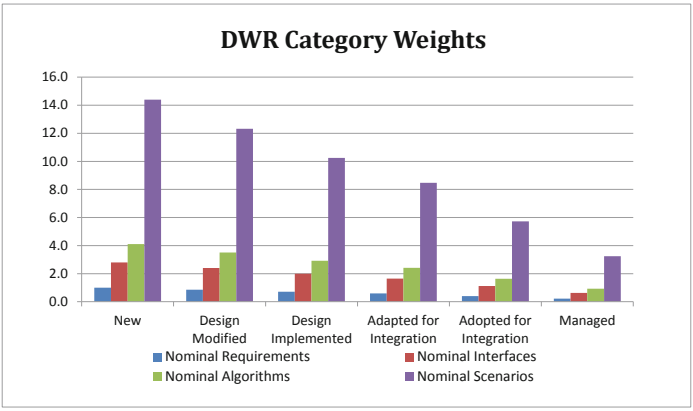


Fig. 7. The graphical representation of the six DWR category weights for each of the four COSYSMO size drivers at the nominal level of difficulty

3.3.2 The DFR Weights

Similarly, the values for the DFR weights are listed in Table 5. Mathematically, these values define the numerical values for the coefficients w_q in Eq. (6). At an intuitive level, however, they indicate an increasing level of SE effort required to realize a size driver (REQ, IF, ALG, or SCN) to a higher level of reusability. In particular, the percentage

values in the first row represent the derived DFR category weights, with 0% for *No DFR* and other categories at increasing levels up to 94.7% for *Validated for Reuse.*

Table 5. The numerical weights of the five DFR categories for each of the four COSYSMO size drivers at the nominal level of difficulty

	Nom. Weights	No DFR	Conceptualized for Reuse	Designed for Reuse	Constructed for Reuse	Validated for Reuse
		0.00%	36.98%	58.02%	79.15%	94.74%
System Requirements	1.00	0.00	0.37	0.58	0.79	0.95
System Interfaces	2.80	0.00	1.04	1.62	2.22	2.65
Critical Algorithms	4.10	0.00	1.52	2.38	3.25	3.88
Operational Scenarios	14.40	0.00	5.33	8.36	11.40	13.64

Figure 8 represents the DFR category weights for the four size drivers (No-DFR not represented). Each of the four groups represents the weights of a category for the four size drivers, each shown by different color bars. Once again, the height of the bars corresponds to the decimal values in the table above, in the respective reuse categories for all the drivers. Contrary to DWR, the upward trend in the graph, from *Conceptualized for Reuse* to *Validated for Reuse*, represents an increasing level of SE effort for all four size drivers.

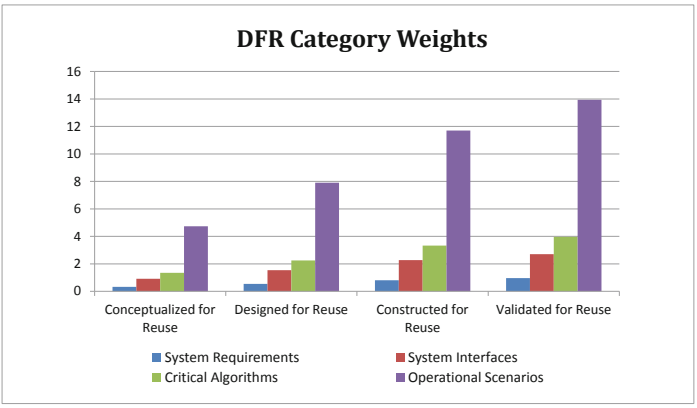


Fig. 8. The graphical representation of the five DFR category weights for each of the four COSYSMO size drivers at the nominal level of difficulty

After fully resolving the weight values for the three-dimensional weighted sum term in (2–2), we get four sets of 15 (3×5) weight values, one for each of the four size drivers, or 60 values in total. These values are the result of expanding the five DFR categories to consider all three levels of difficulties at the same time.

4 Conclusion

Reuse is a fundamental feature of SE. However, only planned reuse with a proactive product strategy likely yields economic benefits. The Generalized Reuse Framework provides an effective tool for product-line managers and systems engineers to manage reuse in a system development effort. We provided the definitions of the DWR and DFR processes and showed how they can be applied to tradeoffs of incremental development strategies and reuse planning in product-line engineering for large-scale system development projects.

We also described an application of this framework to cost estimating. Specifically, we describe an extension of COSYSMO model definition by applying the GRF, including the modified cost es-timating relationship with a set of coefficients calibrated by the reuse weights. The extended COSYSMO more closely captures actual system development processes, cost and efforts, and significantly increases the fidelity of the cost model and the accuracy of its estimates.

With the competitive market environment and continuous drive for improved productivity and efficiency, reuse has become a primary consideration by business enterprises, not just a cost-saving measure but a strategic approach that can positively impact various aspects of system development, ultimately contributing to their success in the market. Systematically integrating reuse into the end-to-end systems engineering process is essential for maximizing the advantages it offers and realizing the promises of improved architecture understanding, better development de-cisions, increased agility and collapsed cycle time, and reduced lifecycle cost for the complex systems we develop today and tomorrow.

References

Boehm, B.W.: Software Engineering Economics. Prentice Hall PTR (1981)

Boehm, B., et al.: Software Cost Estimation with COCOMO II. Upper Saddle River, NJ, Prentice-Hall (2000)

Bosch, J.: Examining the need for change in strategy, innovation methods and R&D practices. In: Keynote, the 24th INCOSE International Symposium, Las Vegas, NV (2014)

Clements, P.C.: Product line engineering comes to the industrial mainstream. In: Proceedings of the 25th Annual INCOSE International Symposium. Seattle, WA (2015)

De Weck, O., Suh, E.S., Chang, D.: Product family and platform portfolio optimization. In: Proceedings of the ASME Design Engineering Technical Conferences – Design Automation (2003)

Hillhouse, B., Ishigaki, D.T.: Strategic Reuse: A Fundamental Approach for Success in E/E Engineering. IBM Rational webinar (2011)

Kim, Y., Stohr, E.A.: Software reuse: survey and research directions. J. Manag. Inform. Syst. **14**(4), 113–147 (2015)

Le Put, A.: Systems Product Line Engineering Handbook. Association Francaise d'Ingenierie Systeme (2015)

Nazareth, D.L., Rothenberger, M.A.: Assessing the cost-effectiveness of software reuse: a model for planned reuse. J. Syst. Softw. **73**(2), 245–255 (2004)

Selby, R.W.: Quantitative studies of software reuse. In: Biggerstaff, T.J., Perlis, A.J. (eds.) Software reusability: vol. 2, applications and experience, pp. 213–233. ACM, New York, NY, USA (1989). https://doi.org/10.1145/75722.75733

Selby, R.W.: Enabling reuse-based software development of large-scale systems. IEEE Trans. Softw. Eng. **31**(6), 495–510 (2005)

Valerdi, R.: The Constructive Systems Engineering Cost Model (COSYSMO), PhD Dissertation, University of Southern California (2005)

Wang, G., et al. COSYSMO reuse extension. In: Proceedings of the 18th INCOSE International Symposium. Utrecht, the Netherlands (2008)

Wang, G., Valerdi, R., Fortune, J.: Reuse in systems engineering. IEEE Syst. J. **4**(3), 376–384 (2010)

Wang, G., Rice, J.: Considerations for a generalized reuse framework for system development. In: Proceedings of the 21st INCOSE International Symposium. Denver, CO (2011)

Wang, G., Valerdi, R., Roedler, G., Pena, M.: Quantifying systems engineering reuse – a generalized reuse framework in COSYSMO. In: Proceedings of the 23rd INCOSE International Symposium, Philadelphia, PA (2013)

Wang, G., Valerdi, R., Roedler, G., Pena, M.: A generalized systems engineering reuse framework and its cost estimating relationship. In: Proceedings of the 24th INCOSE International Symposium. Las Vegas, NV (2014)

Wang, G.: The generalized reuse framework – strategies and the decision process for planned reuse. INCOSE Int. Symp. **26**(1), 175–189 (2016)