Dev Roy

droy@governmentcio.com

Abstract

This document provides a guideline and general best practices for Artifacts Reusability

SOFTWARE Artifacts Re-usability and best practices

# INTRODUCTION

Introducing software reuse at a corporate level represents one of the most promising means of addressing the rising costs that are plaguing the software industry. A series of mechanisms are needed for shortening development cycles and providing reliable software of high quality which will be more maintainable and flexible for future extensions.

This document describes a POV for implementing a reuse methodology focusing on the reusable software artifact library system and its role inside the reuse program. The goal for reuse program is to make software reuse a significant and systematic part of the software process. To institutionalize a software reuse process that incorporates reuse-specific activities all along the object oriented software development process, drawing heavily on a reusable software artifact library system which has been designed to support the classification, management and search for artifacts to be employed in reuse efforts.

# Definition of Reusability

Artifacts can be reused in multiple contexts in a project, including collections and modules. One can edit artifact content in a module or by opening the artifact outside the module. Content changes are reflected in all contexts; however, links, tags, and comments are specific to the context in which they are created.

A broad definition of *software reuse* is using existing software artifacts to construct a new software system. A *software artifact* can be defined as a piece of formalized knowledge that can contribute to the software development process.

There are two types of software artifacts:

1. Software products that are created as deliverables during the development process
2. Development knowledge that is applied to the process.

The most commonly reused *software product is* *source code*, which is the final and most important product of software development. In addition to code, any intermediate life cycle products can be reused, which means that software developers can pursue the reuse of requirement documents, system specifications, modular designs, test plans, test cases, and documentation in various stages of software development.

*Reusable software development knowledge* *and experience* exist at different abstraction levels:

* The architecture level,
* The modular design level
* The program (or code) level.

A *software architecture style* describes the formal arrangement of architectural elements, and can be reused by software developers to construct their new software systems once the style is well defined. For example, the domain-independent *multifaceted architecture* is an architecture style for domain-oriented design environments, which has been reused in and refined through the development of many generations of design environments for different domains.

Reusable knowledge of modular design can be codified in design patterns and frameworks. A design pattern is the description of a solution to recurring problems. It specifies a problem to be solved, a solution that has stood the test of time, and the context in which the solution works. Design patterns provide a common vocabulary for software developers to discuss their designs and can be passed from one developer to another developer for reuse. The concept of the framework comes from object-oriented programming languages. A framework describes the interaction pattern among a set of collaborative classes or objects, and can be represented as a set of abstract classes that interact with each other in a particular way. Programmers can reuse frameworks directly in their development after providing implementations for those abstract classes. Framework reuse is a mixture of knowledge reuse and code reuse.

Programming knowledge at the level of code is represented as program plans that can also be reused by programmers if a suitable representation form is defined.

# Software Process for Reuse

Software process for reuse includes the set of activities that allow, during the course

of a project, the early identification of artifacts to be developed that might exhibit a high reuse

potential either within or in other iterations of projects. Besides, added costs and time (incurred for reuse addressed specific activities) estimation techniques will be defined.

We will discuss two types of reuse here:

* Vertical reuse (i.e., the reuse of software artifacts within a specific domain or application area)
* Horizontal reuse (i.e., the reuse of software artifacts across domains or application area).

However, it puts a major emphasis on the latter, since horizontal reuse provides the highest payoff. The reusable components can be organized hierarchy in a number of product layers, each playing a well defined technical and market role. This reusability framework goal is to set the strategic direction for structuring organization’s software products to support reuse activities. The four layers are defined as below:

* *Operating Environment (layer 1):* it addresses the technological foundation of the software products. The architectural components of this layer reflect the selection of basic database, graphics, and communications enabling technologies for the Operating Environment.
* *Application (layer 2):* it characterizes software components specifically designed to

solve application problems for a broad telecommunication market (e.g., Billing, Service Pro-visioning, Customer Network Management, etc.). This layer is likely to contain a large number of reusable software components.

* *Market Segment Specific (layer 3):* it reflects the specificity of a market segment related to a specific problem-domain (e.g., the Service Provisioning Application can be customized for small PTfs, large end-users, and mobile telephony market segments).
* *Customer Specific (layer 4):* it represents those software components which are highly specialized to address application features specific to a particular customer. Components at this layer have typically a low reuse potential.

# Software Process with Reuse

Software process with reuse includes the set of activities that aim at maximizing reuse

of existing components classified into the framework. The approach supports the earliest possible identification (at requirements or design stages) of candidate reusable components since reuse is most effective when applied at early stages of the life-cycle. The principle, well practiced by hardware engineers, is to specify and design aiming at reusing existing components. The earlier these components are identified, the greater is the possibility to tailor requirements and design to reuse those components, rather than developing new ones. Since the definition of reuse is based on applying existing solutions to new problems, one can succeed in identifying something to reuse only if:

* The domain of available solutions are complete with respect to new, emerging problems;
* The description of both the problem and the solution are expressed at similar levels of granularity, for assessing their match.

# Training and education:

The framework defines and enacts the policies to disseminate the reuse culture

and to encourage the adoption of reuse practices among project teams. It is principally

through this mechanism that the organization learns how to "reuse" effectively.

* *Provide expertise:* whenever possible, engineers work with the project teams to better support them in achieving both development "for reuse" and "with reuse".
* *Development of reusable components:* dedicated teams are in charge of producing reusable components by either re-engineering the "normal" software components or by developing new components whose reuse potential has been assessed during domain analysis. The act of developing reusable components asynchronously with project team activities is called an "a-posteriori (or backroom)"approach.
* *Maintenance of reusable components:* reusable components may change as flaws are corrected and enhancements are made.
* *Maintenance and administration of the framework*: the central repository is managed by a single team which controls the nature and quality of the information stored in the repository. This activity also deals with the population of the library of artifact descriptors, including setting policies for artifact acquisition and maintenance (e.g., evaluation, certification, classification, and weeding). In addition, it has to define and maintain a coherent classification scheme for the stored artifacts, weeding old artifacts, announcing incoming ones, user management (e.g., add/delete users, set user privileges), and reporting on artifact usage.

# Classification of an artifact

The three classification methodologies implemented in the reusability framework are different in the way the artifacts are classified and retrieved.

1. When a new artifact is inserted, it has to be classified in all of the methodologies supported by the system. In fact, if this is not done, the artifact will not be found when a search based on the missing methodology is performed.
2. For the free text classification, the artifacts are classified associating them with a text-description of arbitrary length which describes the artifacts themselves.
3. In the keywords classification, each artifact in the system is associated with a set of keywords that characterize the artifact itself and are chosen from a well-defined dictionary. The dictionary can be expanded by the system administrator, who has the capability to remove unused keywords

# Retrieval of an artifact

1. An artifact stored using the reusability can be retrieved by navigating through the relations established between the artifacts or performing a search based upon one of the classification methodologies implemented in the system.
2. The search is fundamental even when navigating through the relations in fact, it is necessary to find an initial artifact from which to start the navigation. The search is done in different ways depending on the classification it uses:
   * The user has to do a query in a format which depends on the different kinds of classification. If the results of the search are not satisfactory, the query can be relaxed or modified in some way and the search

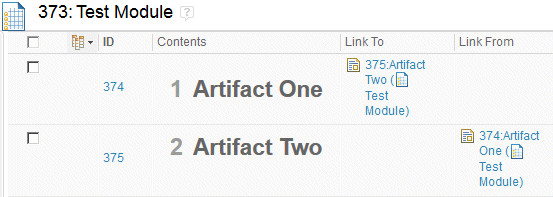
# APPROACHES FOR REUSE

Reuse approach can be of two types:

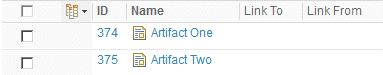
* + Generation-based: The generation-based approach reuses the process of previous software development efforts, often embodied in computer tools that automate a part of the development life cycle.
  + Composition-based: The composition-based approach reuses existing software products in a new system to avoid repetitive work. As mentioned in the previous section, many types of software products can be reused. However, because this research focuses on the reuse of components, the discussion here is limited to component reuse, although many problems and solutions discussed can be extrapolated to the reuse of other software products. Component reuse is also known as *component-based development*. Based on the role that components contribute to the programming process, component reuse is further divided into three categories.
* **Black-Box Reuse:** In black-box reuse, a component is directly reused without modification. A component can be reused as it is or reused through inheritance if the programmer creates a specialized subclass of an existing class component.
* **White-Box Reuse:** In white-box reuse, programmers reuse the component after they have modified the components to their needs. White-box reuse does not contribute as much to the easier maintenance and evolution of software systems as black-box reuse does, but it can reduce development time.
* **Glass-Box Reuse:** In glass-box reuse, programmers do not directly reuse the component; instead, they use it as an example for their own development. For instance, programmers can look at examples to find out how a program plan is realized and build their own system through analogy.

# STORING ARTIFACTS IN MODULES

* To reuse artifacts in a module, existing artifacts should be added to a module or create new artifacts in the module. The artifact can be reused by copying it from one module and pasting into another module.
* After addition or creation of artifacts in modules, work can be done with the artifacts in the module or on the Artifacts page. When you work with an artifact outside a module, you are working with the *base* artifact. Changes to the artifact name, content, and attributes are reflected in all contexts: in the base artifact and in all modules that include the artifact.
* Creation of links, tags, and comments for artifacts in the context of a module, those contextual elements are applicable and visible only in the module where are they created. For example, in the following module, artifact 374 is linked to artifact 375 and artifact 375 is linked from artifact 374:



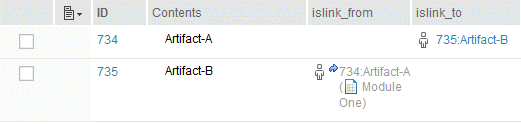
* However, the base artifacts 374 and 375 on the Artifacts page, are not linked:



## Using bi-directional links in a module

The user should be able to link one artifact in a module to another artifact in a module using a bi-directional link type, the link will appear with a blue icon next to it when you select Include links, tags and comments from base artifacts to indicate that the link is from a base artifact.

If user Include links, tags and comments from base artifacts, an indication must be shown as below:



# GENERAL GUIDELINE:

* When user creates links, tags, and comments for base artifacts, those elements apply to the artifact in all contexts in the project. However, those elements should not be shown by default in modules.
* The user should be able to override the default display settings in a module and show the links, tags, and comments for the base artifacts in addition to the links, tags, and comments that are created in the module.
* If you create links between *base* artifacts, those links are available in every module that the base artifacts are used in.
* If you create links between artifacts in the context of a module, the links exist only in that module and in any copies you make of the module.
* If you want to link to an artifact that is reused, when you create the link, you can access the artifact by selecting from the **Rows in the module** list.

# REUSABILITY FRAMEWORK

Domain Analysis

Software Architecture Development

Reusable Component Development

Domain Model

Structural Model

Repository Reusable Artifacts/ Components

\

Analysis

Architectural Design

Component Engineering

Testing

Component Qualification

Component Adaptation

Component Composition

Component Update

Component-Based

Development

# SOURCES

1. Source 1: Work in progress