

Assessing Module Reusability

Chris Lüer

Computer Science Department

Ball State University

Muncie, IN 47306, USA

clueer@bsu.edu

ABSTRACT

We propose a conceptual framework for assessing the reusability of modules. To do so, we define reusability of a module as the product of its functionality and its applicability. We then generalize the framework to the assessment of modularization techniques.

1. INTRODUCTION

Reusability is one of the most important benefits of modularization. One of the purposes of a module is to act as a unit of reuse. Reusability means that the module can be copied and used in projects other than the one it was originally developed for. While not all modules may be reusable, copying, adapting and reusing modules such as functions, classes, aspects, or components is a frequent activity in any development project. Ideally, the need to adapt the module is minimal. Typical modularization techniques actively support the reuse of modules by giving a module clearly-defined boundaries in the source file, or even requiring that the module be put into a file of its own.

Unfortunately, similar to most software qualities, reusability of a module is an abstract property that is hard to assess or measure. Not only is it an abstract property, it is also a subjective and context-dependent property. Nonetheless, a method of assessing the reusability of modules – and, going further, a method of assessing the reusability of modularization techniques – is needed.

In order to be able to better choose between alternative modularization techniques – such as functions, components, or aspects – one needs to be able to assess the reusability of a modularization technique. In order to assess the reusability of a modularization technique, one needs to be able to assess the reusability of individual modules built using this modularization technique.

It has been observed before that a measure of reusability should be based on the context in which code can be being reused, and not only on the internal structure of the code [3].

2. ASSESSING REUSABILITY

We model reusability as a function of applicability and functionality.

The *functionality* of a module is the number of situations in which a client project might want to use the module, based on its specification. A module that addresses many general, important problems will have a high functionality, while a module that addresses a few specific, rare problems will have a low functionality. Functionality is roughly correlated to the size of the module: larger modules will have more features, which will make them attractive in more situations. However, functionality is based on the specification of the module, not its code size, since the size of the code itself does not directly contribute to the usefulness of the module.

Applicability measures the number of situations in which a module can be reused. The applicability of a module is 100% if in any situation that calls for features provided by the module, the module can actually be applied. If the module can only be applied in 50% of the situations where its specified features are needed, then its applicability is 50%. There are many reasons why a module's applicability might be less than 100%, including:

- technical limitations, such as programming languages and platforms. The module cannot be used because its language or platform is incompatible to the required language or platform;
- incompatibilities through details of the interface, such as incompatibility of basic types (e.g., 32-bit integer support is required, but the module can deal only with 16-bit integers);
- incompatibilities through external dependencies. The module may depend on another module, which in turn is incompatible with the requirements in some way;
- problems caused by unspecified nonfunctional qualities, e.g., insufficient performance;
- architectural mismatches [1], e.g., a business-tier module performs user I/O, thus violating an architectural constraint that may be expected by the requirements.

Note that both applicability and functionality are dependent on the specificity of the module's specification. For a given module, if the specification is made more specific by introducing more detail, its functionality will decrease, because more specific requirements mean that the module is less general and fewer problems will be solved. However, its applicability will increase, because there will be fewer situations of hidden incompatibilities that were not made explicit by the module's specification.

Table 1. The table shows the relative applicability and functionality of two concrete modules (not italicized), and several types of modules (italicized).

	error printing aspect	Gui library	<i>procedure</i>	<i>aspect</i>	<i>class</i>	<i>component</i>
applicability	very high	medium	medium	high	medium	low
functionality	very low	very high	low	low	medium	very high

We define reusability as the product of applicability and functionality. Thus, a module is most reusable when both its applicability and its functionality are high. See Table 1 for a comparison of the relative reusability of several sample modules.

Sometimes, a tradeoff occurs between applicability and functionality. The larger a module becomes, the higher its functionality, but at the same time its applicability may suffer, since larger modules tend to have more hidden incompatibilities. For example, larger modules often have more dependencies on other modules, which is likely to reduce their applicability. Also, larger modules are more likely than smaller modules to have constraints on performance and architecture.

As a consequence, when one wants to maximize reusability one needs to attempt to maximize applicability and functionality. While there sometimes is a tradeoff between the two, this is clearly not always the case (see Table 1), so for any given module, there may be an optimum. However, the power of the designer of an individual module in a given project is limited: the designer is severely constrained by the conventions and constraints of his or her language and environment. When one looks at modularization techniques as opposed to individual modules, the number of variables that can be manipulated increases drastically.

Note that a well-known method to increase the applicability of a module is to minimize its interface [2]. The smaller the interface, the smaller is the probability that there are hidden incompatibilities or complex dependencies. The pipe-and-filter architecture of Unix shells is a famous example [4]: the interface of filter modules is extremely simple, since the only data type that can be passed through a pipe is a plain text file consisting of a sequence of space-separated strings. But minimizing the interface of a module may at the same time reduce its functionality, exemplifying the discussed tradeoff.

Both measures, applicability and functionality, depend on the number of situations in which a module can be used. They give a measurable number if one is interested in actual cases of reuse. However, generally designers are interested in future potential for reuse rather than in past cases. To assess future potential, one cannot avoid the need for a certain amount of speculation. Thus the real (i.e., interesting) measure of module reusability will have to be based on some expectation of the future use of the module. As all design, design for reusability is based on predicting the future in the form of future requirements.

To assess the reusability of a modularization technique, we have to abstract one step further. Instead of looking at the future reuse potential of actual modules, one must look at the future reuse potential of potential modules that could be implemented using the given modularization technique. Thus, to assess the reusability of a modularization technique, one has to perform the following steps:

1. Determine likely modules that can be implemented using the modularization technique.
2. Determine likely situations in which these modules can be reused.
3. Determine the functionality of the modules based on these situations.
4. Determine the applicability of the modules based on these situations.
5. Multiply functionality and applicability to calculate reusability.

3. CONCLUSIONS

We proposed and discussed a conceptual framework for assessing the reusability of modules and modularization techniques. The assessment may be precise in the context of actual modules and actual reuse situations. In practice, however, one will be interested in assessing the future reusability of modules – and this introduces a large element of intractability into the method. Nonetheless, we believe the proposed framework may be useful in framing and structuring discussions about module reusability.

4. REFERENCES

- [1] Garlan, D., Allen, R. and Ockerbloom, J. Architectural Mismatch. *IEEE Software*, 12, 6, (1995) 17-26.
- [2] Meyer, B. *Object-Oriented Software Construction*. Prentice Hall, Hemel Hempstead, 1988.
- [3] Poulin, J.S., Measuring Software Reusability. In *Proceedings of the Third Conference on Software Reuse*, (Rio de Janeiro, 1994), 126-138.
- [4] Shaw, M. and Garlan, D. *Software Architecture*. Prentice Hall, Upper Saddle River, 1996.