Background

Demand for software quality continues to increase as the software market matures and as competition increases among software producers and vendors. The quality attributes of a software component are related to its dependency, which is a reflection of its reusability [1] and maintainability [2] [3].

Coupling is a measure of the degree of dependencies between two software components [4]. In object-oriented software, coupling can be categorized into three distinct types: *parameter coupling, inheritance coupling, and common coupling* [5]. Different types of coupling have different effects on component maintenance and reuse.

Common coupling should be deprecated in object-oriented software because it allows one class to access variables (attributes) of another class directly, rather than through the message passing. This defies the principle of design by contract. Inheritance coupling is a unique property of object-oriented software with the objective to achieving class-level reuse. However, inheritance coupling should be designed with caution in the context of software maintenance, because any changes to a base class will affect all of its derived classes. Parameter coupling is usually considered as a weak form of coupling.

Method

Two important parameters in describing component dependency are the type of coupling between components and the type of the dependent component. We classify component dependency in object-oriented software based on these two parameters.

First, we divide dependent components into three different types. The first type is language API or compiler library, which is well designed, tested, and maintained. We call this type of a component as an *intrinsic component*. The second type is developer designed and implemented components, which are targeted to specific applications in a specific domain. We call this type of a component as a *self-designed component*. The third type is software components produced by a third party, such as commercial-off-the-shelf (COTS) components or open-source components. We call this type of a component as a *third-party component*. Accordingly, different types of dependent component have different effects on component dependency.

Combining the three types of coupling with the three types of dependent components, we classify dependencies between components into nine distinct types as shown in Table 1 and we

define four levels of severity of effects that a dependency may have on software maintenance and reuse to describe these nine types of dependency:

- *Prevent*: This kind of dependency is dangerous and should never be allowed.
- *Avoid*: This kind of dependency should be avoided if possible.
- *Allow*: This kind of dependency is allowed.
- *Strive*: This kind of dependency should be promoted.

Obviously the level of severity increases from strive to prevent. A component dependency should aspire to belong at the strive severity level instead of the prevent severity level. The severity levels and the effect on maintenance and reuse of these nine types of dependency are summarized in Table 1.

Table 1. Component dependency and severity levels of effect.

	Coupling type		
Dependent component type	Parameter coupling	Inheritance coupling	Common coupling
Intrinsic component	*Strive*	*allow*	*prevent*
Self-designed component	*Strive*	*allow*	*prevent*
Third-party component	*Allow*	*avoid*	*prevent*

To measure the component dependency, we present the following component metric for object-oriented software.

$$D\_D = (1*N\_S + 2*N\_AL + 3*N\_AV + 4*N\_P) / N\_C$$

In the above, D_D is the degree of the dependency of a component, N_S is the number of *strive* level coupling within the component, N_AL is the number of *allow* level coupling within the component, N_AV is the number of *avoid* level coupling within the component, N_P is the number of *prevent* level coupling within the component, and N_C is the number of classes in this component. We used different multipliers (1, 2, 3, and 4) to state that coupling of different severity

levels have different degrees of effect on dependency. Considering the size difference between the components evaluated, we use the number of classes (N_C) in each component as a normalization parameter.

Case Study

We analyze 11 java components from Jakarta. We expect to find a strong correlation between the dependency metric of a component and other external quality factors of the component. The validation contains two steps. First, we study the coupling in the 11 Jakarta components. We calculated the degree of dependency (D_D) of these components. Second, we assign these 11 components to four teams to reuse these 11 components. Based on their experience, each team assesses the reuse effort for adapting each component and ranks the effort from 1 to 10.

We expect the reuse effort increases as the degree of dependency increases and we test the following null hypothesis: *H0: There is no linear relationship between the degree of dependency and the reuse effort of a component.*

The Spearman's rank correlation test shows that there is strong correlation between the degree of dependency and the reuse effort of a component, which means that the classification of component dependency and the subsequent metric are valid indicators of the external quality properties of the component.

Contributions

Our contributions made in this paper can be summarized as:

- We analyzed the effects of different dependent components on the software dependency and divided them into three types: *intrinsic, self-designed, and third party*.
- We presented a classification of component dependency in object-oriented software based on the coupling type and the dependent component types. This classification provided guidelines for quality software design.
- Based on this classification, we derive a component dependency metric for object-oriented software. The metric is then validated in a case study of open-source Java components.

References

[1] Card D N, Glass R L. *Measuring Software Design Quality*. Prentice-Hall, Upper Saddle River, NJ. 1990.

[2] Gibson V R, Senn J A. System structure and software maintenance performance. *Communications of the ACM*, 1989, 32(3): 347–358.

[3] Banker R D, Datar S M, Kemerer C F, Zweig, D. Software complexity and maintenance costs. *Communications of the ACM*, 1993, 36(11): 81–94.

[4] Offutt J, Harrold M J, Kolte P. A software metric system for module coupling. *Journal of System and Software*, 1993, 20(3): 295–308.

[5] Briand L C, Morasca S, Basili V R. Defining and validating measures for object-based high level design. *IEEE Transactions on Software Engineering*, 1999, 25(5): 722–743.