Using Code Reusability Analysis to Identify Reusable Components from the Software Related to an Application Domain

Guillermo Mayobre
Hewlett Packard
Grenoble Networks Division
5, Ave. Raymond Chanas
38053 Grenoble CEDEX 9

gm@hpgntol1,grenoble.hp.com

Abstract

The combination of SW metrics with domain expertise and economical evaluation of reuse costs, provides a way to select reusable workproducts from the software related to a domain of application.

1 Introduction

One of the major problems in software reuse is the lack of reusable components despite the large amount of existing software. Reuse efficiency and cost effectiveness highly depend on the number of available components. In fact existing software has captured past experience and knowledge, and that is particularly true under an application domain scope.

The idea of using that existing software to identify reusable components is very attractive. A methodology to identify and select reusable software components from an existing code is extremely helpful, not only to provide code components, but also to identify reusable workproducts that have been codified by the past.

The Code Reusability Analysis (CRA) combines three methodologies and an economical estimation to identify and select reusable workproducts from the code associated to a domain of application.

2 Overview

The first methodology involved is the one proposed by Caldeira and Basili [Cald 91]. It allows identification of reusable components based on software metrics.

The second one, the Domain Experience Based Component Identification Process (DEBCIP) is based on domain experience and uses a standard decision graph to help domain experts on the process of identifying reusable components.

The third one, the Variant Analysis Based Component Identification process (VABCIP) also based on domain experience, uses code metrics to estimate the specification distance between a new product and the existing software in the domain (or subset of software of the domain). This methodology select components from the existing software, whose specification distance, to the new targeted functionality implies a re-engineering effort lower than building the component from the scratch.

The economical estimation of the Component Return on Investment (CROI), uses economical models like Component Cost, Reuse Costs Investment and Benefits, Reuse Cost Effectiveness, to estimate the Return on Investment per Component.

Selecting components from the existing software to be reused globally on the domain of application differs from selecting components to be reused to build a new target functionality. The second case is a subset of the first one in the sense that it reduces scope of search -(functions of a new product against all possible functions of a Domain of Application)- .

The Code Reusability Analysis is useful to perform both types of selection depending on the way methodologies are combined.

DEBCIP is tailored to identify components with the highest reuse potential within the overall domain scope. VABCIP is performant to identify components to be reused on a new targeted functionality.

However those methodologies may be very expensive when applied to a large amount of software, in terms of time consumed by the experts and product marketing resources. C&B does not require neither expert nor product marketing resources, and in addition it may be automized completely. An interesting idea is then to use C&B before either DEBCIP or VABCIP to reduce the amount of code to be analysed by the experts. Once potential reusable components has been identified using C&B, experts review them from the perspective of the domain of application.

At the last step, an estimation of the Component Return On Investment of the candidates refines the final choice.

This paper explores several ways of combining those methodologies to find the better strategy to select reusable components from the software associated to a domain of application, in order to build a new target functionality, minimising the implementation effort and maximising the external reuse level.

3 Background

This chapter gives an overview of the methodologies and economical models included on the Code Reusability Analysis.

3.1 The Caldeira and Basili methodology

It defines three major attributes that characterises the reusability of a component: usefulness, cost to reuse and quality. It proposes four software metrics to estimate the attributes values: Volume (V), Cyclomatic Complexity (C), Regularity (R) and Reuse Frequency (RF).

Figure 1 is a fishbone diagram showing attributes and metrics relation:

The Volume

An Halstead's formula, represents the minimum number of bits necessary to code a module information:

$$V = (N1 + N2)\log_2(n1 + n2)$$

Where n1: total number of operators used by the program,
N1: total count of all usage of the operators,
n2: total number of unique operands defined and used by the program,
N2: total count of all usage of operands.

If the component volume is too small, the cost to reuse the component (extraction, adaptation and integration), may exceed the cost to building it from scratch.

If the volume is too large, the qualification will probably be lower and the component more error prone.

The Cyclomatic Complexity

It computes the maximum number of independent paths:

$$C = e - n + 2$$

Where e: number of edges,
n: number of nodes.

The bigger the complexity the higher the number of tests necessary to walk through all the code, and thus the effort to qualify the module.

Once again a very low complexity may not repay the cost of extraction, adaptation and integration. A high complexity will probably imply a lower than expected qualification and a higher risk of error.

The Regularity

Is the ratio between the expected length and the actual length of a component:

$$R = \frac{n1\log_2 n1 + n2\log_2 n2}{(N1 + N2)}$$

Figure 1. The Basic reusability attributes model.
(Caldeira & Basili)

In other words it measures readability and non redundancy of component implementation. A regularity close to one indicates a well implemented component.

The Reuse Frequency

It measures the potentiality of a component to be reusable. It is the ratio between the number of static calls to a component with respect to the average number of static calls to a system module. The higher the reuse frequency , the bigger the reuse potentiality of a component.

$$RF = \frac{n(C)}{\frac{1}{M}\sum_{i=1}^{M} n(S_i)}$$

Where M: number of system modules,
n(C): number of static calls to the component C,
n(Si): number of static calls to the system module Si.

Ranges must be defined for every metric to measure the results. They may differ depending of the type of analysed code. Range values we used in our practical example are given in the case study section.

3.2 DEBCIP and VABCIP

The Domain Experience Based Component Identification Process DEBCIP and the Variant Analysis Based Component Identification Process VABCIP, illustrated in Figure 2 and 3 respectively, are both based in similar concepts.

Both methodologies evaluate reuse potentiality based on specification distance and expected number of reuse instances. DEBCIP needs an important investment in domain analysis. Domain bounds must be well defined and domain model must be available to be able to evaluate the reuse potentiality of a component. Existing software functionality is compared to all the possible functionalities described in the domain model to evaluate expected number of instances and specification distance.

The specification distance evaluation on DEBCIP is relatively complex. The first step is to define and intersection between all the target functionalities that we want to cover with a component, we call it the reduced functionality. Then, the specification distance from the existing software is evaluated to the reduced functionality.

As the reduced functionality is very sensible to product strategies, market pressures and even resources fluctuation, the specification distance is used on DEBCIP more as an information parameter rather then a decision parameter.

The biggest emphasis is put on the estimation of the expected number of reuse instances.

VABCIP reduces the scope of search. Reuse potentiality of components is evaluated regarding a unique or a very few new targeted functionalities that are very close one to each other. On that case the major effort is put in the evaluation of the specification distance from the functionality of the existing software to the new targeted one. A well defined domain model is not crucial, and thus the prior investment on domain analysis is lower.

Figure 2. DEBCIP decision graph.

Figure 3. VABCIP Decision graph.

R_VABCIP (Reduced VABCIP) is a variant of VABCIP. The software whose functionality is compared to the new target one is a subset of the existing software related to the domain. R_VABCIP process is also illustrated by Figure 3.

Collecting decisional information:

Decision graphs provide a frame to collect decisional information: cost evaluation for changes on the analysed component to made it reusable, and expected number of reuse instances. (Figure 3 shows where cost of changes are evaluated in the VABCIP: {SC}, {DC}, {CC}).

Computed decisional information is used to feed the economical models to estimate the component Return On Investment.

Evaluating costs:

Costs to change, what we called adaptation costs, are evaluated based on software metrics. We commonly use Cyclomatic complexity to evaluate the productivity.

To each phase of software development: specification, design, code, unit testing, integration testing and tools development, we associate a productivity expressed in:

Nb.of branches (or Independent paths = Cyclomatic Complexity)/ Engineer.week.

Once a change is identified at any part of the graph, it is estimated on Cyclomatic Complexity. The adaptation effort to made the component reusable is then evaluated according to the type of change (specification, design, code,..) by applying the corresponding productivity metric.

At the end of the graph walk through, the reuse potentiality of a component may be: None (non reusable) or Reusable with or without an associated adaptation cost and a number of reuse instances.

4 The economical evaluation

Reuse is cost effective when reuse cost benefits are bigger than reuse cost investments. Cost effectiveness of software reuse is an essential decision parameter that must be estimated at the earliest stage of product development.

If an early estimates indicates that the total reuse benefits will be very low, we should do only a limited investment in reuse technologies. If at the contrary the estimates shows a big reuse benefits we should make substantial investment in advanced reuse technologies.

On reuse oriented software development we can distinguish two main types of activities: product development and component development. Product development may be almost completely classified on the design WITH reuse side, while component development may almost completely classified on the design FOR reuse side. From this perspective reuse is cost effective when benefits of designing WITH reuse are bigger than investments on designing FOR reuse.

In addition to the cost effectiveness another important parameter to evaluate the performance of an investment is the Return On Investment which is the ratio between the reuse costs benefits and the reuse costs investments.

The Component Return On Investment is estimated for every candidate component to decide whether or not it is effectively reusable.

Economical models like component cost, reuse costs investment and benefits, reuse cost effectiveness, are needed to estimate the component return on investment.

The reusable component cost model The cost to produce a reusable component may be expressed as:

$$RCC = Eor * CC + DBC$$

Where RCC: Reusable Component Cost. Measured in Eng.week.
CC : Component Cost. Cost of the component as designed not for reuse.Measured in Eng.week.
Eor: Reuse expansion (or multiplication) factor.
DBC: Data Base Cost per component. Measured in Eng.week.

Eor includes all additional costs to implement a reusable component. Additional documentation costs, specification costs, design costs , code costs, ...

DBC includes the Data Base maintenance cost (system backup, configuration, maintenance) and also the Librarian activities. Librarian activities includes tasks as work in reusables assets to made them visible to potential users, marketing functions to identify users needs, component purchase, ...

Reuse cost investment

Is the additional cost needed to made a component reusable. From that perspective it may be expressed as the cost difference between developing a component to be reusable against developing the same component not for reuse.

$$RI = (Eor - 1) * CC + DBC$$

Where RI: Reuse Cost Investment. Measured in Eng.week.

Reuse costs benefits Benefits of reusing a component may be calculated as:

$$RB = CC - A - DBR$$

Where RB : Reuse Costs Benefits. Measured in Eng.week
A : Adaptation cost. Measured in Eng.week. Is the cost to adapt the component to integrate it to the new application.
DBR: Cost to search the component in the Data Base.

Regarding the above definitions we may express the Component Return On Investment as the ratio between the sum of individual benefits for each reusing activity and the reuse costs investment to made the component reusable.

$$CROI = \frac{NbInst(CC - DBR) - \sum_{i=1}^{NB.inst} A_i}{(Eor - 1)CC + DBC}$$

Where CROI : Component Return On Investment.
NBinst: Number of expected reuse instances.
Ai : Adaptation cost for integrating the component to an activity i.

Cost effectiveness of a reuse activity

It may be also interesting to evaluate the reuse cost effectiveness of a project involved in a reuse oriented software development process considered in terms of the balance between the reuse cost benefits of designing WITH reuse against the reuse cost investment in designing FOR reuse:

$$Ceff = \sum_{i=1}^{NB.CNS} \{CC_i - (A_i + DBR_i)\} - \sum_{i=1}^{NB.PROD} \{(Eor - 1)CC_i + DBC)$$

Where NB. CNS:
Is the number of consumed reusable components when designing WITH reuse.
NB. PRD:
Is the number of produced reusable components when designing FOR reuse.

5 The Experience

The experience was conducted over a case study to compare the usage of C&B combined with R_VABCIP against the usage of VABCIP.

5.1 Context:

The case study is on the context of the software reuse program at Hewlett Packard Grenoble Networks Division.

This program has been running since 1.5 years. Models and methodologies proposed in addition to the Caldeira and Basili's one, has been developed by the reuse staff (apart from the project staff) during the reuse program development.

5.2 Problem Statement

It may be expressed generically as: Given an Application Domain with the associated workproducts and architecture, and given the specifications of a new product within the Domain, implement it, in such a way to maximise the external reuse level and minimise the implementation time. Of course that implies reusing, already existing components, and new selected ones, not yet recorded as reusables.

5.3 The Case Study

The analysed code is a subset of the software developed for the Datacomunications Front End Application Domain (DTC-FE). The subset was delimited according to architectural considerations: for example we do not analyse code involving X.25 staff, because we are leading mostly with Telnet Protocol and MPE/XL terminal IO.

Size of the analysed code is 35K NCSS.

The new product specification are the one of the pilot project ECRINS, involved in the Software Reuse Program.

ECRINS project information:

- Staff: 7.5 engineers.
- Estimated Code Size: 40K NCSS (Non Commented Source Statements). (ECRINS is applying Formal Reuse Oriented Software Development Process, based on the results of the Code Reusability Analysis study).

The Code Reusability Analysis takes four weeks. It involves a Domain Expert (architect), a Domain Analyst and a SW Engineer expert in SW metrics.

5.4 Results

Results are summarised in Figure 4.

Column 1 shows the results of applying C&B for two different Reuse Frequency: .5 and .3, and with 800< V ¡4000, .65< R < 1.35 and 5< C <16.

Column 2 shows the results after experts performed R_VABCIP on the components previously selected by C&B.

Column 3 measures the accuracy of C&B by computing the percent of components retained by the experts from those detected by C&B.

Those results shows C&B as very accurate. Even for the case of lower reuse frequency the percent of code complexity retained after the experts advise of the code selected by C&B is 88%.

Column 4 shows the result of applying VABCIP directly. Column 5 and 6 compares results between C&B + R_VABCIP and VABCIP. As shown, the amount of selected components, code complexity and total effort avoided are higher when using VABCIP. But the selection time is 5 times longer.

The Component Return On Investment estimation was 1.66 in average, and none of the components had a CROI lower than 1.

Figure 4. Results.

Figure 5. Component Reusability Analysis Strategies.

6 Conclusion

The strategy to follow to select reusable workproducts from the software related to a domain of application is shown in Figure 5.

At the left side of the dashed line is shown the selection of workproducts globally reusables for the domain. That case was not considered on the case study, but, for medium to large systems we recommended to go first through C&B. Then use DEBCIP for the retained components to estimate the expected number of reuse instances and specification distance, and finally do CROI to evaluate the component return on investment.

At the right side of the dashed line is shown the selection of workproducts reusables on a new target functionality. In that case the strategy is the following: Use C&B + R_VABCIP as a first estimation. If you are about or better than the reuse objectives: shortage,... go directly through CROI. If you are far behind, perform VABCIP.

In between, evaluate the cost of VABCIP with respect to the effort needed to develop from scratch the non detected components.

References

[Bigg 89] Biggerstaff and Perlis, *Software Reusability*, ACM Press, 1989.

[Cald 91] Caldieri and Basili, "Identifying and Qualifying reusable software componentsi" *IEEE Computer*, February, 1991.

[Booc 87] Booch, G., *Software Components with Ada*, Benjamin/Cummings, 1987.

[Barn 91] Barnes and Bollinger, "Making reuse cost effective", *IEEE Software*, January, 1991.

[Boeh 83] Boehm, B., *Software Engineering Economics*, Prentice-Hall, 1983.

[Chau 91] Chauvet, "Une analyse economique de la reutilisabilite", *AFCET Interfaces*, Mai/Jun, 1991.

[Prie 91] Prieto-Diaz and Arango, "Domain Analysis and System Modelling".