Using Existing Software in a Software Reuse Initiative

Gregory W. Hislop

Working Knowledge, Inc.
738 Elgin Rd.
Newtown Square, PA 19073
Tel: (215) 359-9080
Email: hislopg@duvm.ocs.drexel.edu

Abstract

Existing software is often not well-suited for reuse. At the same time, it is a natural starting point for organizations trying to adopt a reuse strategy. Any efficient techniques for extracting value from existing software would ease the transition to a reuse-based approach.

My efforts have focused on trying to derive value for reuse from existing software. I have been exploring ways to apply to reuse ideas developed to detect plagiarism. I am also quite interested in the behavioral issues, both individual and organizational, related to starting and sustaining a reuse strategy.

Keywords: software reuse, software similarity, plagiarism detection

Workshop Goals: My goals for the workshop are to exchange information on research efforts and to discuss the status and results of formal reuse initiatives in commercial and governmental organizations.

Working Groups: Useful and collectable metrics; Reuse management, organization, and economics; Reusable component certification.

1 Background

I approach reuse from the perspective of both research and industry. I have recently completed a doctorate with a thesis focused on reuse. In addition, I am involved with fledgling reuse efforts at several commercial organizations.

Having a foot on both sides of the fence keeps me quite conscious of the gap between state of the art and state of the practice in software engineering. I have a general interest in technology transfer and ways to close that gap.

2 Position

Existing software contains a large amount of encoded knowledge about the organization and the organization's past software needs. This material could be very valuable for reuse if we could extract it efficiently.

Existing software is also a natural starting point for implementing reuse, especially for an incremental implementation. Existing software provides a familiar point of reference that may reduce uncertainty, and help create a feeling of ownership. Finally, it may be difficult to justify a major initiative that does not start with the existing software.

At the same time, working with existing software may involve significant problems. The quality of source code is uneven. The documentation is often inadequate. And existing software was probably not designed for reuse.

In short, while the idea of using existing software is appealing, the reality is difficult. We need special tools to support this work. We can extract gold from ore that looks like common rock. If we are going to use existing software, we need to develop more economical extraction techniques to find the flakes of gold in our software.

Intuitively we might be inclined to evaluate existing software by function, i.e., what it does. Reusable parts are generally selected by function, and functional categories are the usual basis for organizing parts collections. It would be logical to analyze the existing software by function too. Unfortunately, analysis by function is very difficult to automate and so tends to be quite expensive.

An alternative to evaluating by software function is to measure software form, i.e., what the software looks like. Software form includes characteristics such as size and structure. It is well suited to automated measurement, and there is a wealth of prior work that has studied various attributes of form.

One way to look at software form is to consider the broader question of software similarity. That is, are there parts in the existing software that are very similar to each other? A possible application of this type of analysis would be for locating instances of informal reuse with or without modification.

There is an existing pocket of research that measures software form to detect similarity. These studies are attempts to detect plagiarized student programming assignments. Clearly, this is quite close to our interest. Plagiarism, after all, is simply a socially unacceptable form of reuse.

A basic problem in many plagiarism studies stems from the fact that most software metrics are single values. For example, Non- Comment Source Statements (NCSS), is a single value. Single-valued metrics are very convenient for reporting and analysis, but their conciseness may not provide the discriminant power needed to detect similarity.

For example, similar programs might have similar NCSS values. However, it is easy to imagine programs with similar NCSS values that are not at all similar except in size.

An alternative approach is to represent attributes of software form in a less concise manner than a single-valued metric. Whale has proposed a structure profile which is a variable length descriptor based on control structure.[1] He reports better success at detecting plagiarism with this approach than with methods based on single-valued metrics.

I have taken concepts from the work on plagiarism and applied them to reuse. Student assignments are generally different from commercial software in size, complexity, and style, so it was not clear whether the concepts would transfer to this new application. I created a prototype tool, SoftKin, and conducted a series of case studies to test the concepts.

2.1 SoftKin Prototype

SoftKin is a prototype tool for exploring approaches to measuring software similarity. The name reflects the idea that SoftKin looks for related software parts.

SoftKin consists of a data collector and analyzer. The collector processes existing software and calculates measures of form for each module. The analyzer computes similarity measures for each module pair. The pairs of modules can then be ranked from most similar to least similar by each similarity measure.

SoftKin can calculate similarity based on a variety of measures of software form. These include single-value metrics (such as McCabe's Cyclomatic Complexity), a metric composite, and also a structure profile that is a slight variant of the profile proposed by Whale.

The case study goal was to see if any of the approaches to measuring similarity show promise for identifying candidate parts that might be reengineered for a reusable parts collection.

To evaluate the various similarity rankings, I focused on the task of locating informal reuse. In each of the case studies, we were able to identify a set of instances of reuse in the existing software. This allowed me to evaluate each measure by where the known instances of reuse fall in the similarity ranking. A good ranking is one that places the actual reuse instances near the top of the list.

This evaluation provides a relative measure of various methods of similarity detection. As a production tool, SoftKin would guide an analysis of existing software. The ranking by similarity would suggest where to focus attention in looking for candidate reusable parts. If SoftKin is successful, the best candidates for a parts collection would be clustered near the top of the list.

2.2 Case Studies

The case studies analyzed existing software from 3 large commercial organizations. The applications were all commercial data processing systems including areas such as finance, sales, and distribution. The analyzed software consisted of about 360 modules totaling 156,000 NCSS.

The case study results are quite interesting. The results of the single-value metrics are all fairly poor. In fact, a ranking by similarity of size provides as good a result as a ranking by any of the other single-value metrics. However, the average similarity, which is a composite of the single-value rankings, provides substantially improved performance.

The ranking by structure profile is the best method for locating instances of actual reuse. This is particularly true for the top part of the ranking, which is the part that has practical value. For example, a sizeable percentage of the known cases of reuse fall in the first 100 ranked pairs. (The actual results are 37, 50, and 63 percent, respectively, for the three case studies.)

We can test the statistical significance of the result based on the structure profile vs. the other metrics. Using Dunnett's T to control for Type I error across the set of comparisons, the structure profile shows a significant performance improvement versus the single-value metrics at the 0.05 level. The improvement over the average similarity, while substantial, is not significant at the 0.05 level.

The pattern of results is very consistent across the three case studies. This is particularly encouraging since the case study settings vary substantially including differences in programming language, development methodology, and organization size.

In general, concepts carried over quite well from plagiarism detection to reuse. Single-value metrics show poor results for locating the instances of informal reuse. On the other hand, a composite based on these metrics performed much better. Finally, the structure profile approach provided the best performance.

3 Comparison

A number of studies have applied measures of form to software reuse. Selby conducted a general analysis to determine the characteristics of software that is known to have been reused. [2] This work provides some general guidelines that may indicate attributes that are desirable for reusable software.

Calderia and Basili make extensive use of form metrics in their strategy for locating reusable parts in existing software in the Care system.[3] They propose a reusability attributes model for determining probable reusability for a part. The major attributes are:

- Quality of the part.
- Costs of using the part.
- Usefulness of the part.

It is easiest to associate measures of software form with the quality and cost attributes of the model. It is more difficult to propose automated measures to predict usefulness of the part.

For the Care system, "reuse frequency" is taken as a predictor of usability. This value is based on the number of static calls to a software part. This approach may be quite effective in some environments. However, it requires that a given part have only one name. In addition, it does not consider cases in which there are variants of a part each with a different name.

The software similarity approach that SoftKin explores could be used to complement the reuse frequency measure of the Care system.

References

[1] G. Whale, "Software Metrics and Plagiarism Detection," vol. 13, pp. 131–138, 1990.

[2] R. W. Selby, "Quantitative Studies of Software Reuse," in *Software Reusability, Vol II* (T. Biggerstaff and A. Perlis, eds.), pp. 213–233, ACM Press, 1989.

[3] G. Calderia and V. R. Basili, "Identifying and Qualifying Reusable Software Components," *IEEE Computer*, February 1991.

4 Biography

Greg Hislop has held technical and management positions in software organizations for almost 20 years. He is principal of Working Knowledge, Inc., a company he founded in 1987 to provide services and education to information systems groups.

His current work involves software reuse, downsizing mainframe applications, I/S tools and techniques for distributed organizations, and practical application of software metrics. He also provides services related to measuring and analyzing I/S workloads and charging for I/S services.

Dr. Hislop holds degrees in Economics from Georgetown University, Computer Science from Queen's University, and Information Studies from Drexel University.