Bachelor thesis – Software Engineering and Management

How can code reuse help a Software Company

David Berglund & Gabriel Englund
Spring 2008

Project Coordinator: Björn Olsson
Supervisor: Björn Olsson

Table of Contents

1. Introduction

As software companies spend more and more money on development of new products, reuse becomes a more appealing solution. Reusable artifacts within software companies are called software assets, [Larsen], and can be executable code, source code, specifications on requirements, design and architecture, etc How can then reusing code help the company? Code reuse is a good way to put together a solution using tested and implemented code. It can also help a company put together the solution faster due to the fact that the code is already written and tested which makes implementation a simpler problem. This thesis will discuss the strengths and weaknesses of reusing code, it will also handle a case study where code reuse is of essence.
"Software reuse is the reapplication of a variety of kinds of knowledge about one system to another system in order to reduce the effort of development and maintenance of that other system" [Biggerstaff]

What are the possibilities and risks of introducing a new code reuse system?
Code reuse works in such manner that code from old projects is reused in new projects. This can save time for a company, but it can also waste the company's time [Meyer]. It all depends on how reusable the code is, how good the "reuser" is and how good of a reuse-tool is being used. The person that is to reuse the code has to have a good knowledge of what the code does, how it does it and why it is needed. Most code written today has never had the intention of being reused in later projects; therefore it might be hard to break out single functions from an old piece of code, due to poor documentation, variable names and such, while code written with reuse in mind will contain better documentation and descriptive names on variables. Some researchers [Tracz] require that code reuse is the use of code specially designed with future reuse in mind. Code reuse has, basically, existed as long as code, in some manner. Most developers have copied code from old projects and pasted into their own. This is a very straightforward way of code reuse. The tool used during code reuse is also a huge factor in the total time spent on finding the needed code, thus if it takes more time to find, adapt and implement the code needed, it is simply a waste of time for the developer to even try to reuse code.

To examine this question an action research oriented case study was conducted where the authors of this thesis actively participated in the development and implementation of a new code reuse system

This thesis is carried out in the following manner;

- **Literature review** focuses on previous studies in this field.
- **Research method** describes the method used during this thesis.
- **Problem statement** presents the purpose of this thesis.
- **Solution** explains how the problem at hand was solved using the method and theory
- **Conclusion** handles the conclusion, summary of contributions and future research.

2. Literature Review

The concept of code reuse was first introduced in 1968 [McIlroy] and has been discussed ever since. The idea of code reuse is brilliant, but the implementation, however, is the hard part. Code reuse, as mentioned above, never became a hit in software development due to the complexity to implement, but has every now and then been brought up for discussion in articles [Krueger]. Studies have also been made regarding why code reuse never became a standard software engineering practice [Krueger]. [Larsen] talks a lot about reusing parts of old projects into new projects, which gives a broad overview on the topic at hand. There was also some papers written about more specific problems and projects using reused code, such as reusing code within a compiler [Tabatabai], which is quite interesting but not fitting to this thesis since it's going to focus more on how a company can gain from it and not how software can become smaller after compilation.

2.1 Pros of code reuse

Code reuse benefits companies a lot, for example; when a common function is needed in a project, the developer can import the function knowing that it has been tested and works properly. This is particularly good when developing a new product family [Sommerville].

[Sommerville] states:

- Increased dependability
  - Previously used SW tends to have fewer defects.
- Reduced process risk
  - Building a component can fail more easily than reusing an existing one.
  - Particularly true for large, complex components (sub-systems).
    - Sometimes not true, if the adaptability of the reusable component is unclear.
- Effective use of specialists
  - Rare knowledge is encapsulated in a component and is then more readily and widely available.
- Standards compliance
  - Reusing components can promote the adherence to standards (e.g. user interface look-and-feel).
  - This improves usability of a component (for programmers and/or for end-users).
- Accelerated development
  - Often more important than anything else.

2.2 Cons of code reuse

[Stephen] describes in their Quality Impacts of Clandestine Common Coupling the dangers of code reuse. They have made an extensive research where they trace coupling and reuse of code throughout the Linux kernel history. They present results speaking of that same modules has been reused extensively and unchanged during a lot of versions. An example states why this can be hazardous to a project: "Suppose that a programmer is responsible for developing and then maintaining a module M. Suppose further that the programmer is fully cognizant of the deleterious effects of common coupling, and therefore makes every effort to minimize the common coupling between M and the rest of the product. The results of this paper show that, even if the programmer does not change M in any way, subsequent changes made by other programmers to other modules can increase the common coupling between M and the rest of the product."

In their introduction they describe more about coupling and why this can be bad, "Although all types of coupling are sometimes useful in design, it has been demonstrated that some types have greater potential for introducing faults into software (Wulf and Shaw, 1973; Kafura and Henry, 1981; Troy and Zweben, 1981; Selby and Basili, 1991). Because some types of coupling are more likely to lead to faults than others, it is widely accepted that some coupling types should be limited in use."

[Sommerville] states:

- Increased maintenance cost
  - External changes to a reused component may force changes to the reusing software.
  - Required changes to a reused component may be more difficult to have than for a custom component.
- Lack of tool support
  - There is little support for maintaining a catalog of reusable components and for finding components.
  - Development tools do not support development with all kinds of reusable components well.
- Not-invented-here syndrome
  - Many software engineers tend to prefer rewriting a component over reusing it
- The reuse barrier:
  - A software developer will only search for a reusable component if s/he expects to find an appropriate one.
  - S/he will only find it if it is described in a way s/he can recognize as providing a solution.
  - S/he will only use it if s/he can understand it.
  - Use will only be successful if the component can be adapted to the problem at hand.

    - The whole process will start only if the developer expects all steps to be successful in the beginning.
- Reuse works well for many general-purpose components, but is difficult for highly domain-specific ones.

3. Research Method

This paper aims to study code reuse and how it can benefit a company. The primary goal is to write a research paper about findings in this field. The secondary goal will be to develop a functioning prototype that enables employees to search for already developed code within the company to be reused in current projects. This system will hopefully be used by Reveny, the company this thesis is carried out at, and perhaps Jeeves, the company Reveny is working for. The research method that will be used in this paper is action research (fig. 1).

“Action research can be seen as a variant of case research, but whereas a case researcher is an independent observer, an action researcher:

...is a participant in the implementation of a system, but simultaneously wants to evaluate a certain intervention technique.

...The action researcher is not an independent observer, but becomes a participant, and the process of change becomes the subject of research [Benbasat].”
[Westbrook].

Action research was chosen because a prototype is to be developed based on interview and literature material. The fact that action research is iterative makes it natural to update the prototype along with new interview- and literature findings. Each research cycle will take no longer than two weeks due to the fact that time is of the essence.
During the first month of the thesis, a lot of time was spent at Reveny in order to get familiar with their working methods. With this knowledge, the rest of the thesis could be carried out at the school, but with a continuous contact with the company. An interview was conducted with our contact person within the company. His position as a developer made him more suitable to answer the questions; that can be found in the appendix. We also received a product specification describing the requirements for the prototype that is to be developed. This specification can also be found in the appendix.

Fig. 1. Action research cycle [LLAS]

3.1 Research cycle

During the “Identify problems / issues for investigation” phase; the most critical problems will be identified so they can be taken care of in the current cycle. In the “Carry out research” phase; research in the problem area will be done to get a good idea of the issue at hand. “Formulate action plan” phase will consist of the actual “fixing” of the problem identified in the first phase. For the duration of “Reflect on and evaluate action” phase; changes made will be evaluated to see how big of an impact the modifications had regarding the issue at hand.

For example:
Listing of files and folders in the treeView is takes to long time and freezes the program until it is done listing.

- **Phase 1:** Through testing of the prototype; population of the treeView model is far to slow and freezes the program until it’s done.
  - 10 954 folders and 49 609 files takes two minutes to add to the treeView

- **Phase 2:** Look at the code that populates the treeView, and try to find out what part is making it slow;
  - Adding appropriate icons to all files takes long time.
  - Populating the treeView with full depth of the files and folders takes long time.

- **Phase 3:** Applying findings in “Phase 2”;
  - Assign one icon to all files and one icon to all folders. Add a checkBox to the GUI where the user can choose if the appropriate icons are to be shown.
  - When populating the treeView, files and folders with the depth of two, from the selected folder, will be

added to the treeView. When a folder that is not populated is pressed, the program will populate that folder with its files and folders to the depth of two.

- **Phase 4:** Testing of the treeView population, after updates, shows an improvement with one minute and 50 seconds.

4. Problem Statement

The goal with this thesis is to identify and try to solve the drawbacks of code reuse. The idea of code reuse has been around for some decades now, but has not yet reached its full potential. The goal is to make a solution that makes it easy for developers to get a hold of the specific piece of code they want to use, and implement it when they are at their clients. The problem is to make this easy. To find a way to gather all the code in one place, make it generic so you can use it over and over again and make it searchable.

There is today no standard tool for code reuse, so the authors of this thesis tried to solve the problem by implementing a prototype that could handle all requirements, found in the appendix, provided by the customer.

Conducting the literature review gave an understanding that having a tool like this leads to a performance gain for the company which in turn makes them more popular to their clients. A company without a code reuse system will have to reinvent the wheel over and over; code already developed in earlier projects is often developed again in a later project, thus costs time and money for the company. This is why it is so important that the companies have some sort of system and/or library that they can use out in the field. Upon implementation the company also gets a good and stable backup system.

5. Solution

This is where the case study company is presented and how the problem stated was solved.

5.1 Presentation of Reveny

Reveny System AB is a subsidiary company of Jeeves Information Systems AB. Reveny consists of about 20 people positioned in Göteborg, Växjö and Stockholm. They are a software company that develops business systems. Most of their work is consulting at their clients.

5.2 Problem solution

Having found the need for a solution in working with code reuse, a prototype was designed to help Reveny manage and make their code easily searchable. A number of factors were taken under consideration while constructing this prototype. The implementation of the prototype must take speed and usability as the main factor. Because if you have a slow program that is hard to use, no one will use it and makes both the implementation redundant and will not solve the problem at hand at all. [Nielsen]. The code should also be easily maintained even after this project time in terms of documentation and easy readable source code. A good design is therefore necessary and a good hierarchy within the code so you easily can extend it.

The prototype works in such a way that the user searches through a specified folder after certain keywords and this will result in a list of files containing said keywords. Upon parsing a specific file the user can see the code snippet he or she is searching for. All the code files are stored at the same place, a code library, which makes it possible for all the employees to have the most current files to work with. The goal of the prototype is to make the job easier for the consultants when they are working with their customers at their office. This is done by making it possible for them to access the code and make it easy for them to find the functions they are looking for as been said before. The code library in combination with the prototype does just this.

The code library can be setup however one wishes but the essence of it all is to store all the code in one place, making it accessible for everyone that needs access to it. A fairly simple way to do this is setting up either a SVN server which makes it possible to use version handling which is a very good thing working with code, but since the prototype in its current form have no SVN functionality forces Reveny to either implement this or have all employees working with the system install a SVN client. Another way of solving this problem is putting the code on one server and giving people access to it through disk sharing which make it easier to use through the prototype but removes the advantage of using version handling.

A properly working code reusing tool will benefit the company with:

1. Enables developers to produce more code per time unit.
2. Ensuring that the inserted code is working well, because it is already in use in other projects.
3. Saves the developers time when debugging and testing, thus the code has already been debugged and tested.
4. Creates a standard process in code storing.
5. If SVN is used, the company can reap the benefits of revision handling.

6. Conclusions

How can code reuse help a software company?
This question was the base for this thesis and has been answered in the way the problem was solved. Due to time restraints the prototype has not been fully implemented and quantitative nor qualitative tests can be done as of now whether or not this tool can help the company. The pros' discussed in the literature review however makes the prototype valid and as such proof that the problem has been solved in a good manner.

Time and usability was two main factors while designing the prototype. The first version of the prototype took too long to show files and folders. This time problem was solved with a smarter algorithm for population of files and folders. The contact person was amused with the usability and straightforward design of the GUI from the first version, so no updates has been done there.

In creating a tool, reduction of the negative sides where also more or less solved, such as lack of tools, maintenance costs and the reuse barrier. This is though mostly because the prototype was an in-house product and can be made more specific than generic.

6.2 Summary of contributions

Provided an understanding in how you can work with code reuse on a practical level.

6.3 Future research

Although there has been much written about this subject, there is still no generic tool or process to use within code reuse. This would be a great thing to research further and in the end implement.

As it is now, the prototype is developed in the language C#. If Reveny is to develop this system further, it would be a great function to have an "active directory listener" against the repository folder that in turns notifies other clients that the repository has been updated. If CVS functionality is implemented in the system, the system can then warn the user if one is trying to commit files to the repository when new files have been added.

7. References

[Larsen] Ö. Larsen, Vilka principer bör styra utveckling av komponenter för återanvändning, 2002

[Sommerville] Ian Sommerville, Software Engineering 6th & 7th edition, 2001, 2004

[McIlroy] M.D. McIlroy, Mass-Produced Software Components. In J.M. Buxion. P. Naur and B. Randell, editors, Software Engineering Concepts and Techniques; 1968 NATO Conference on Software Engineering, pages 88-98. Petrocelli/Charter, Belgium, 1976.

[Tabatabai] A-R. Adl-Tabatabai, T. Gross, and G-Y. Lueh, Code Reuse in an Optimizing Compiler, 1996

[Krueger] C.W. Krueger, Software Reuse, 1991

[Biggerstaff, 1989] Ted Biggerstaff and Allan Perlis. Software Reusability: Concepts and Models, volume 1, chapter 1. ACM Press, 1989

[Meyer, 1997] Bertrand Meyer. Object-Oriented Software Construction. Prentice-Hall, 1997.

[Tracz, 1995] Will Tracz. Confessions of a Used Program Salesman: Institutionalizing Software Reuse. Addison-Wesley Publishing Company, 1995.

[Benbasat] Benbasat, I., Goldstein, D.K. and Mead, M., “The case research strategy in studies of IJOPM information systems”, MIS Quarterly, September 1987, pp. 369-86. 15,12

[Westbrook] R. Westbrook, Action research: a new paradigm for research in production and operations management, 1994

[Stephen] STEPHEN R. SCHACH, BO JIN and DAVID R. WRIGHT,GILLIAN Z. HELLER, JEFF OFFUTT, Quality Impacts of Clandestine Common Coupling, 2003

[Nielsen] Jakob Nielsen, Usability Engineering, 1994

[LLAS] Figure found at: http://www.llas.ac.uk/projects/project_item.aspx?resourceid=2837

[Tortoisesvn] http://tortoisesvn.tigris.org/

8. Appendices

8.1 Interview with the contact person at Reveny:

1. How are you working today?
   - How do you search after functions?
     We ask our fellow developers, or look at code locally saved on the computer.
   - How long time can it take to find a needed function?
     Everything from one second till a point where it is not worth looking for it in ones old code.
   - Do you have any code storage system?
     No. not mode than that we store the code locally on our computers or on the clients servers.
   - Do you have standardized code documentation
     Yes, since February.
   - How often do you need to access the code?
     On a daily basis.
   - How often are you reusing code
     In 10% of all cases, at most.
   - How often do you start looking for code to reuse, but give up before it's found, and wrights the same code your self?
     It's hard to approximate, thus we mostly don't look around for code today.
2. What will you gain with a new work procedure?
   - Time?
     The gain is hard to estimate before we have implemented the system, although we estimate that the gain will be greater in a long term perspective. Thus we won't have that much code in the system at the start, it will however be added as the system is used.
   - Money?
     The advantage with code reuse is that we will be able to work faster, but with the same invoice as before. Alternative lower our costs and get more effective to give us a great advantage against our competition on the market.
   - Customer service?
     Code reuse does not support customer service in particular. Although if all code is stored on one place, one will always be able to see the customers code even when the customers servers are down. We will have few standard solutions, although in those cases that we do bugs will be well documented, thus debugging time will be decreased.
3. What exactly is it that you need?
   - Functionality

We need; a common code storing server that can be reached from a client, even though they have a firewall, a search function that enables one to quickly find code and functionality allowing one to store code locally on the computer.

- Who will use a new system?
  Consultants and developers.
- When?
  At implementation time at the customers.
- How?
  Through searching after wanted code and get to the implementation phase as fast as possible.

8.2 Product specification

Utöver versionshantering med plugin för ut och incheckning av kod till visual studio så behövs sökfunktionalitet.
All kod producerad av företaget har ett standardiserat kommentarhuvud, med en beskrivning av vad funktionen gör, vem koden producerats för, vad koden är kopplad till (D.v.s. händelse eller program som exekverar koden), sökord, när koden är skapad och av vem samt när den är ändrad och av vem.
Ett exempel:

```
/**************************************************************************
*Beskrivning: Uppdaterar kundstatus till aktiv                            *
*Kund: Ica                                                                *
*Kopplad till: Inget                                                      *
*Sökord: kus, kundstatus                                                  *
*Skapad av: 080310.kaot                                                   *
*Ändrad av:                                                               *
**************************************************************************/
```

Behovet finns av ett sökverktyg som kan söka igenom dessa huvud för att snabbt få fram information om de olika kodbitarna, men även fulltextsökning behövs.
I mån av tid kan detta verktyg utökas med funktionalitet så att det lokalt lagrar all kod i klienten, antingen i en databas eller i en mappstruktur. När man sedan får kontakt med servern ska koden kunna synkas.

Sökverktyget bör kunna kommunicera över tcp/ip med valfri port för att kunna nå servern ifrån externa nätverk.
Verktyget ska preliminärt utvecklas som ett fristående verktyg men i mån av tid och intresse ifrån kunden så kan den implementeras som en plugin i visual studio.
Verktyget ska därför utvecklas i .net plattformen.

Fotnot.
Huvudet kan vara utformat på annat sätt men med samma innehåll .
T.ex. med xml taggar för lättare parserimplemetering.