Confessions of a Used-Program Salesman: Lessons Learned

Will Tracz

Loral Federal Systems
Owego, NY
tracz@lfs.loral.com

Abstract

Software reuse is the second oldest programming profession. Ever since the first program logic board was wired, people have been looking for ways of saving time and money by building upon other's efforts and not "not re-inventing any wheels." This article summarizes the lessons I have learned as a used-program salesman. Using this analogy, I will examine efforts made to institutionalize software reuse.

Overview

Since 1983 [9] I have proudly worn the clothes of a used program salesman. (I am glad that plaid polyester leisure suites haven't gone out of style. :-) Over a dozen articles in *IEEE Computer*, *IEEE Software*, and *ACM Software Engineering Notes* and a newly published book (**Confessions of a Used Program Salesman: Institutionalizing Software Reuse**) [11] later, used programs are still "objects" of desire. (It is interesting to note the play on words in the last sentence. Object-oriented programming has contributed greatly to the spread of the reuse religion, even though, as pointed out by Martin Griss, the reuse rabbi, "objects are neither necessary nor sufficient for effective reuse" [4].) The sections that follow contain a brief overview of the following lessons I have learned in being a used program salesman. The paper concludes with a final confession, "*The Successful Reuse Diet.*"

SSR '95, Seattle, WA,USA

Lesson #1: Reuse is a management decision.
Lesson #2: The reuse rules of three are true.
Lesson #3: You need to reuse more than just code.
Lesson #4: The "glue" is the key technology.

Lessons Learned

Lesson #1: Reuse is a Management Decision

The notion of software reuse conjures up the notion of something from nothing. Reality reveals that, even though the rewards of reuse are great (e.g., 70% reduction in time to delivery [8]) the cost of institutionalizing reuse is substantial (e.g., 30-50% more development cost [5]). Therefore, management must be involved in providing resources and direction to bring this about. The major inhibitors to reuse are non-technical and can best be eliminated with management intervention.

Lesson #2: The Reuse Rules of Three are True

Ted Biggerstaff's proposed the following "Reuse Rules of Three" [10]. (Ted based them on Bob Lanergans's observations at Raytheon [6].)

1. *Before you can develop reusable software you need to have developed it* **three** *times.*
2. *Before you can reap the benefits of reuse, you need to reuse it at least* **three** *times.*

My experience, and collaborated as part of several other reuse testimonies [12], points out that reuse requires domain knowledge in order to recognize what to make reusable and how to make it (re-)usable. Furthermore, the break-even point for recovering the additional investment costs cited in the previous section still turns out to be around three (re-) usages of a particular component.

Lesson #3: You Need to Reuse More Than Just Code

Code is the easiest software artifact to reuse, but in order to achieve an order of magnitude improvement in software productivity, one must reuse other portions of the software development lifecycle. Table 1, originally used by Jim Neighbors in the **Reuse Results So Far** panel session at the *Third International Conference on Software Reuse* [7], November, 1994, illustrates possible reductions in the five stages in a typical software development lifecycle (Requirements Definition, Requirements Analysis, Design, Implementation, and Test).

	req	ana	des	imp	tst
base	100%	100%	100%	100%	100%
A	100%	100%	80%	20%	60%
B	100%	100%	40%	10%	35%
C	65%	85%	20%	5%	15%
D	50%	60%	10%	0%	10%
E	50%	50%	5%	0%	5%

Table 1: Lifecycle scenario vs. percentage effort from current lifecycle

Table 2 reflects the comparable reductions in workload, based on initial estimates found in [2]. One can observe the importance of reuse being pushed earlier and earlier into the software development lifecycle. One can also observe the necessity of using code generators and domain-specific software architectures [3] to achieve code and design reuse with minimal effort.

	req	ana	des	imp	tst	ttl
base	24	32	121	141	98	416
A	24	32	97	28	59	240
B	24	32	48	14	34	152
C	16	27	24	7	15	89
D	12	19	12	0	10	53
E	12	16	6	0	5	39

Table 2: Lifecycle scenario vs. labor months required in each phase of lifecycle

Lesson #4: The "glue" is the key technology

Finally, one can examine the success with Microsoft Visual Basic VBX componentry [1] to recognize the importance of common communications protocols and having a simple, well documented integration mechanism to bring about a "software components" industry. Putting components on the shelf is only going halfway down the path of successfully institutionalizing software reuse. One still needs the right glue to put them together with. CORBA (Common Object-Request Broker) also provides an integration mechanism that supports component reuse.

Closing Confession: The "Successful" Reuse Diet

In closing, I would like to offer the following analogy to those who have the desire to successfully institute reuse into their organization. I liken the goal of achieving software reuse to that of loosing weight.

1. **You need a good reason to loose weight.**

   One needs to realize that loosing weight will make you look and feel better. It will make you more competitive and improve the time it takes you to "run through the paces." If one has no desire to improve oneself, then one needs someone to explain the benefits.

   In the case of software reuse, the analogy is obvious; management should be the one to point out the benefits of change (i.e., improved productivity and quality). If not, the programmer should recognize the need him/herself and be personally motivated. (But being motivated is not enough to bring about change, as the remaining observations reveal!)

2. **You need realistic and measurable goals.**

   One needs to have a realistic "before-and-after" picture in ones mind about how much weight one wants to loose and how long one is willing to work at it. This implies the use of a very low-tech tool, a scale! Without a scale, one has no idea if any progress is being made. (Of course a tape measure could be used also, but that is comparably low-tech.) Also, one needs to determine how long one is willing to wait for results. The faster results are desired, the more effort is involved.

   The reuse analogy is the same in that unless realistic metrics are in place, one has no way to tell if the reuse program is successful or if changes in the approach need to be made during the transition period. Just as it helps the dieter to get positive feedback, it helps the programmer to know in what areas success is being achieved. Finally, just as no one can loose 100 pounds in a month (unless they chop off both legs) achieving reuse levels of 50% or greater in a matter of months is unrealistic. Reuse levels of greater than 20% require at least one year or more to achieve.

3. **You need a diet plan.**

   There are dozens of diet plans currently available. Some may be more suited for a particular individual's life style than others. All of them "work" if

taken seriously. The secret of success is sticking to the diet.

In the case of software reuse, there are several software development methodologies that can be adapted to support reuse. The key to success is disciplining oneself to follow the design and documentation guidelines necessary for identifying and developing the reusable resources in the first place.

4. **You need some support along the way.**

   Loosing weight is not easy. Sometimes one is tempted to slip back to one's old ways. To avoid setbacks, the involvement of others helps (especially if they provide exercise equipment, dietary supplements, education classes and books, or bring in motivational speakers). In the end, misery loves company and having others to help (or reward) you along the way, makes a big difference.

   Institutionalizing software reuse, as with any technology insertion program, is not done in a vacuum. Management plays a key role in providing the necessary resources (e.g., guidelines, tools, training, and even rewards) to support reuse.

In closing, the words of Winston Churchill come to mind. After the war, Churchill was asked to address the preparatory school he had attended as a child. The headmaster informed the students of this noteworthy orator's profound wisdom and had set aside a large portion of the school day for his address. Churchill arrived, took the podium, and began, "Don't ever, ever give up!", then sat down. These are also inspiring words for software developers and used-program salesman facing the challenges of institutionalizing reuse.

References

[1] T.J. Biggerstaff. Is 'Technology' a Second Order Term in Reuse's Success Equation? In *Proceedings of Third International Conference on Software Reuse*, page 190, November 1-4 1994.

[2] B.W. Boehm. *Software Engineering Economics.* Prentice Hall, Englewood Cliffs, NJ, 1981.

[3] M. Graham and E. Mettala. The Domain-Specific Software Architecture Program. In *Proceedings of DARPA Software Technology Conference, 1992*, pages 204–210, April 1992. Also published in CrossTalk, The Journal of Defense Software Engineering, October, 1992, pages 19-21, 32.

[4] M.L. Griss. PANEL: Object-Oriented Resue. In *Proceedings of Third International Conference on Software Reuse*, pages 209–213, November 1-4 1994.

[5] T.C. Jones. Economics of Software Reuse. *IEEE Computer*, 27(7):106–107, July 1994.

[6] R.G. Lanergan and C.A. Grasso. Software Engineering with Reusable Design and Code. *IEEE Transactions on Software Engineering*, SE-10(5):498–501, September 1984.

[7] J. M. Neighbors. Reuse so Far: Phasing in a Revolution. In *Proceedings of Third International Conference on Software Reuse*, pages 191–192, November 1-4 1994.

[8] Inc. QSM Associates. Independent Research Study of Software Reuse. Technical report, QSM Associates, Inc., Pittsfield, MA, September 1994.

[9] W. Tracz. Confessions of a used program salesman. *Computer*, 16(4):100, April 1983.

[10] W. Tracz. RMISE Workshop on Software Reuse Meeting Summary. In *Software Reuse: Emerging Technology*, pages 41–53. IEEE Computer Society Press, 1988.

[11] W. Tracz. *Collected Confessions of a Used Program Salesman – Institutionalizing Software Reuse.* Addison-Wesley Publishing Co., New York, NY, 1995.

[12] K. Wentzel. Software Reuse, Facts and Myths. In *Proceedings of 16th Annual International Conference on Software Engineering*, pages 267–273, May 16-21 1994.