Planning The Software Industrial Revolution
Supply-side Economics of Software Reuse

Brad Cox
Information Age Consulting

71230.647@@compuserve.com

Abstract

Software Reuse is like Nuclear Fusion. Both address major problems (software crisis, energy crisis). Both already work at certain (coarse) levels of granularity (shrinkwrapped applications, fusion weapons). And neither scales to smaller levels of granularity because of the same unresolved issue; the supply-side return-on-investment is negative. So long as there is no robust infrastructure capable of incentivizing people to invest in building truly reusable software, research on demand-side issues (languages, repositories, classification schemes) will continue to achieve disappointing results.

An infrastructure for admistering a pay-by-use revenue stream for software is presented, which is loosely modeled after the role ASCAP/BMI play for the music industry.

1 Position

In 1982, a decade ago next June, I helped to initiate a large-scale experimental study of whether software reuse could get at the root causes of the software crisis. This was not an academic study of a stripped down subset of the problem, but a well funded experiment on the largest possible scale. The experimental apparatus was a full-scale commercial enterprise, originally called PPI and later Stepstone. Its subject was the software development industry as a whole. I'll present interim conclusions of this unique experiment in the Practices section of this workshop. And in the Research section, since my role of Chief Scientist made me responsible for key technical and scientific challenges in this endeavor, I'll present these challenges along with suggestions for further research.

The key conclusions of this experiment are a good news/bad news pair. The good news is that Stepstone's and its customers' experiences have left me more convinced than ever that software reuse can be a silver bullet for the software crisis, just as interchangeable parts were a silver bullet for the problems that armament consumers faced before the industrial revolution. The bad news is that these benefits will not come easily. Achieving them on a broad scale will require a paradigm shift, a software industrial revolution, comparable in magnitude and time-frame to the industrial revolution, with comparable trauma to well-entrenched value systems.

The software crisis is not over now that yet another decade has passed, nor will it over in the next decade or two. During the industrial revolution, the armories' pioneering attempts at interchangeable parts were consistent failures for twenty-five years and it took more than fifty years to build them routinely. This was for tangible goods that anyone could sense with their natural senses and which manufacturers could sell by the copy and thus guarantee a robust income stream. Although things do change faster today, the difficulties arising from software's peculiarities provide little grounds for believing that the software industrial revolution will proceed significantly faster.

Since this assessment may seem unduly pessimistic (and I certainly hope that events prove it wrong), allow me to use this workshop's title, Software Reuse, to emphasize the magnitude of the paradigm shift that we're facing. The very word, 'reuse', implies that we're talking about a waste product, a liability; something that garbage-pickers recycle and reuse. Certainly not an asset, something that quality-conscious engineers would pay good money to buy. People pay money for word processors, spreadsheets, and programming languages; tangible stuff that they can poke with a cursor and see it respond on the screen, and even then software piracy is a major concern. Small-granularity software components; the spreadsheet functions in Excel, the macros in Nisus, the subroutine libraries of Fortran or the Software-IC libraries of Objective-C; are waste products, not assets. Waste products are products that cannot be sold, just reused. At best they can be given away by hiding them inside something of tangible value such as a compiler. People only reuse stuff that is free, and even then whether they'll reuse it or throw it away is far from certain. The gut reaction that reuse evokes is "Not in my backyard!"

This is Obstacle #1, the fundamental non-technical socioeconomic problem that must be solved before other, admittedly more technically interesting problems, can have any but academic merit. The software reuse problem will be supply-limited, regardless of programming language, operating system, browser, repository or networking technology, so long as we have no crisp answer to the fundamental motivational question that any urchin in a souk will understand, "Why should I build expensive reusable waste products when non-reusable ones are cheaper and faster?".

I'll mention three additional obstacles, but they are all insignificant, almost negligible, compared to this one. Fred Brooks speaks of the complexity/mutability obstacle (Obstacle #2) and the intangibility obstacle (Obstacle #3) in his paper, No Silver Bullet.. The single-threadedness (Obstacle #4) of the Von Neumann machine architecture is also an obstacle in that it is the source of our obsessive concern that any component can crash an entire system.

But these three obstacles all have easy, i.e. technical, remedies, at least when considered individually in isolation from the others. The complexity obstacle has been successfully attacked in mature domains such as plumbing by organizing a broad and deep specialization of labor hierarchy, whose motive force is a robust market in pre-fabricated plumbing components. The intangibility obstacle can be addressed through tangibility technologies such as browsers and interpreters. We could also be doing far more than we are today with explicit software specification/testing tools, tools that make the external behavioral interface of intangible software components visible for inspection, regardless of their internal implementation. They play the role of Galileo's telescope, which by making the heavens accessible to scientific observation, brought on the demise of the myth and religion of the pre-Copernican philosophers. The single-threadedness obstacle can be at least partially addressed by adopting well known techniques, such as lightweight multi-tasking and exception handling, as mainstream concepts alongside the object-oriented technologies that are already in place today.

But compared to our inability to conceive of a commercially robust market in something as intangible as small-granularity software components, these obstacles are at best of minute interest to those who are allowed by their management to ignore pressing and far more fundamental problems.

In the practices section of this workshop, I'll present several possible avenues for putting software reuse on a commercially robust footing. These are based on analogies with other domains of human endeavor, such as the music industry, which has been face to face with the replicability obstacle for nearly a century.

In the research section I'll describe an inheritance-based software specification/testing technology that captures the abstract specification of large libraries of Software-ICs in separate tools called 'gauges'. The gauges determine whether Software-ICs comply to their specifications when the implementation is changed in any way, such as after a repair, extension or port.

2 About the Author

Dr. Cox is the author of the book, *Object-oriented Programming, An Evolutionary Approach*, and the originator of the Objective-C "System-building Environment and many of its Software-IC" libraries. He is presently writing a second book to be titled *Object-oriented System-building; A Revolutionary Approach.*

The objective of Cox's work is bringing about a software industrial revolution in which software is produced, not by fabricating everything from first principles, but by assembling interchangeable (reusable) software components which are in turn supplied by lower-level echelons of producers.

Prior to co-founding The Stepstone Corporation, he worked for Schlumberger-Doll Research labs where he applied artificial intelligence, object-oriented, Unix, and workstation technologies to practical problems in the oil field services business. Before that he worked in the Programming Technology Center at ITT, where he applied Unix and object-oriented technologies in support of an extremely large, highly distributed telephone exchange, System 1240.

Dr. Cox's received his Ph.D. from the University of Chicago for theoretical and experimental work in the area that has since become known as neural networks. He also carried out post-graduate experimental studies at the National Institutes of Health and at the Woods Hole Marine Biological Laboratories.