OOPSLA 1993

Addendum to the Proceedings

Washington, DC—26 September–1 October, 1993

Panel Session

Developing Software for Large-Scale Reuse

Moderator:

Ed Seidewitz
NASA Goddard Space Flight Center

Panelists:

Brad Balfour, SofTech, Inc.
Sam S. Adam, Knowledge Systems Corporation
David M. Wade, Computer Sciences Corporation
Brad Cox, George Mason University

Introduction

This panel succeeded in both its goals. The first of these was, of course, to have a useful discussion about developing software for large-scale reuse. The second goal was to try to encourage greater communication between the Ada community and the object-oriented programming community. Two of the panelists (Brad Balfour and David Wade) were drawn from the Ada community, while the other two (Sam Adams and Brad Cox) were drawn from the object-oriented programming community. Further, the panel had been previously held in June at the Washington Ada Symposium. All the panelists agreed that this approach had been very useful, both for them personally and for the communities in general, and they hoped for further interaction in the future.

The position of each panelist is well-described by their position papers in the OOPSLA'93 proceedings. However, for completeness the presentation made by each panelist is very briefly summarized here. The remainder of this summary documents the active question and answer session that followed the presentations. Of necessity, all questions and answers have been paraphrased.

Brad Balfour

Brad was asked to focus on the influence of programming languages on achieving large scale reuse. He stated that the choice of programming language was a fundamental, influential decision. This decision impacts supporting technologies, such as development guidelines, asset certification and the process of using an asset. It also has an economic impact for maximizing the return the investment in reuse, because we currently don't really reuse software across languages, in practice. Finally, it influences the upstream products (i.e., in the analysis and design phases), because various advanced design and even requirements methods are easier to apply with some languages than others. Brad concluded that the choice of programming language will remain important for reuse until we achieve a technology that truly allows us to mix languages.

Sam Adams

Sam was asked to consider the methodological issues of large scale reuse. Sam first argued that reuse must occur in all phases of the life cycle, including maintenance. Most methodologies only cover the original development of software—just 20% of the life cycle. Large scale reuse involves both reusing software from one project to the next and also retrofitting off-the-shelf components into existing projects to replace custom software and reduce maintenance costs. Sam then discussed his idea of an enterprise as an internal reuse marketplace, in which a group working on a project would promote those components they developed that they think are reusable. An internal economy is needed for reuse, because the infrastructure is not yet their for commercial large-scale reuse. Sam next considered what kind of software gets reused: you have to be able to find it, it must be easily understandable, it must be easily integratable, it must be trustworthy and there must be a need to reuse it. Sam concluded by describing the Well Defined Object concept being developed at Knowledge Systems Corporation in which a reusable object encapsulates all such information that needs to be kept in sync to reuse that object.

David Wade

Dave was asked to talk about the practical experiences with reuse that he has had on the FAA Advanced Automation System project. Dave began by describing the complexity of AAS: it is really a group of cooperating systems for air traffic control spread over a large number of control centers. The project is so large, that they were

able to view these systems as comprising a domain and then do a domain analysis to glean commonality across the systems. He referred to this as internal reuse within a very large project, as opposed to reuse from one project to the next. In 1989 the AAS project formed a reuse working group, because they felt that some reuse opportunities were falling through the cracks. This group was initially viewed as simply providing coordination on reuse within the project, but eventually developed into a three-way effort: a strong emphasis on education, the collection of metrics and staff to develop those reusable components that were agreed to be necessary, but that no other group on the project was willing to develop (currently about 30,000 lines of code). Finally, Dave noted that they did this with very little automation, though they are now moving their taxonomy information from flat files to a PC data base system.

Brad Cox

Brad was asked to discuss the economic issues of large-scale reuse. Brad began by asking, "Why do we call it reuse, rather than buying, selling and owning like other engineers?" He answered himself by saying that reuse is a word applied to a liability, a waste product that you can't sell, so you reuse it! Brad identified the problem as that, for the first time in history, we have goods made of bits and not atoms. The normal market system does not work for software, because the markets ability to hang prices on the exchange of material assumes the conservation of mass, which does not apply to bits. Brad then described a concept call super-distribution that is based on the ability to copy and transport bits at the speed of light, rather than trying to deal with software like other products. In this approach, software is freely copied, but as copies are invoked, their use is monitored, usage rates are uploaded to a central system and the producer of the software is compensated based on this usage. Some of the usage fees for, say, a word processor would then automatically flow to the producers of reusable components and subcomponents used in the word processor, based on the relative usage of those components. Brad concluded by wondering if perhaps there was indeed a silver bullet: the mobilization of human energy to solve a problem. Unfortunately, Brad believes that for software this will require a paradigm shift, which is always chaotic and disruptive.

Questions and Answers

Question for Dave Wade: You mentioned that you have training for all your developers. Do you also have training for you managers, and what kind?

Dave answered that their education effort was broad based. They had a tailored class specifically for the management ranks. In fact, they eventually had three different kinds of courses, but its the course for the rank-and-file that has endured.

Question: What kind of solutions can we provide to the sociological aspects of large scale reuse, and the resistance to such reuse?

Brad Balfour mentioned that his company has been primarily supporting information systems groups within the DOD—people with 10 to 20 years of COBOL experience who really have no incentive to do reuse. He has found that in these situations the social changes are harder than the technical changes. The hardest change is the shift to a multi-system perspective. Brad said that this has been less of a problem in recent work his company has done for the Church of Latter Day Saints in Utah, because this is a centralized organization that can readily look across all its IS applications. Brad concluded by saying that he felt that 80% to 90% of the problem was non-technical.

Sam Adams noted that Smalltalk programmers tend to pick-up on reusing things quicker because they have so many great things to reuse. Sam felt that you can't make people reuse with a stick approach—you need a carrot. The problem is not getting programmers to reuse, its getting management commitment. This requires a change in the sociology of an organization. Current IS organizations tend to manage project-by-project, and in some organizations development teams stay together into maintenance, so they never have a chance to reuse software on the next project. Sam summarized by stating that barriers between projects cause a lot of the problem.

Dave Wade stated that his group avoided these problems with reuse. He felt that ultimately some paradigm shift to reuse will happen because of the pressure of economics. But he didn't see this happening in the short term on a large scale. His group has had success in coordinating reuse efforts within a large project, but they did find many problems to be insurmountable.

Brad Cox added that, to some extent, the social problems of reuse will remain almost insurmountable until we have addressed the underlying issue of market economics. Sam Adams agreed that this is one of the reasons reuse won't happen outside of an internal reuse infrastructure for a long time. But Sam felt that with the right management commitment and the right motivating crisis, organizations can make changes within themselves that would take generations to make in society at large.

Question for Dave Wade: Could you give some examples of cost savings, increased quality and schedules being met through your reuse effort?

Dave noted first that quality is absolutely essential on the AAS effort, so there is very strong quality assurance for the entire program. Dave then added that the reusable code on the project has additional quality because many people are using it in different ways and applications. He stated that the average number of times a component is reused on AAS is 4 to 5 times, with some components being reused 50 or 60 times. Dave also said that they kept track of cost avoidance and some statistics on staff months, but that it was difficult to quantify meeting schedules because of reuse of components. Dave admitted that the entire project was behind schedule anyway!

Question: What is the process and product of domain analysis? Comment on whether there is some consensus on what domain analysis is and why it is supposed to be so useful even though no one knows how to define it.

Brad Balfour answered that the Defense Information Systems Agency Center for Information Management reuse effort has focused a lot on domain engineering, consisting of domain analysis and design. SofTech has

put together a process for domain engineering and conducted a workshop to try to form a consensus. He gave a quick definition of domain analysis as a generalized form of requirements analysis in which, instead of developing object requirements for a specific application, the goal is to model a generalized set of requirements across a family of applications (the domain). Domain design is then the development of domain-specific architectures to implement applications in the domain.

Follow up: When you take any real system, how do you generalize it into a family of systems? A system can be generalized along many different dimensions.

Brad replied that the processes that are successful do it bottom-up from an existing family of systems. For example, in France, CSF Thompson generalized from a set of air traffic control products. They generalized along the lines of variations they saw between their products.

Ed Seidewitz added that they had been doing domain analysis at Goddard for four years and are just figuring out what it is. He stated that one thing that is particularly important is to bound the domain: you have to decide what you are generalizing over, the strategic direction you see your organization going. He noted that this is not necessarily a fixed decision, that it may change over time. Ed stressed that an organization has to be careful not to become overwhelmed by analyzing an expanding domain. To avoid this the organization must decide what is useful to generalize over and document it.

Sam Adams took the view that through domain analysis you are describing the problem space in a way that can be used to describe it to others. This is an abstraction process. Sam felt that you want something that is in the common conceptual vocabulary of all the people you are trying to service. For example, the concept of an account will occur a lot in bank applications, and the concept of a document re-occurs in PC applications. Sam noted that these are not really real world things but are rather abstractions, concepts we choose to invent.

Brad Cox made an admittedly sarcastic comment: What's the difference between doing domain analysis and not doing it? It's the difference between thinking and not thinking!

Question for Dave Wade: You mentioned that you kept track of the number of times a component was reused. What other metrics did you keep track of?

Dave answered that when they started their metrics program, they wanted to be as unobtrusive as possible, so they keep very few metrics. They keep the standard metrics such as source lines of code, time to develop and effort to develop, of course. They also keep a data base of initial requests for components and their current users. The emphasis has been to keep it simple. Dave concluded with the comment that if they didn't know they needed some information, they didn't ask for it.

Question: Suppose you have 10,000 or 100,000 very nicely packaged, well-documented, reusable objects, how do you find the ones you want? There seem to be only two choices: text retrieval or a formal specifications language.

Sam Adams stated that the problem with specifications languages is that they get too detailed and complicated. Reuse is not about precise matches, but fuzzy matches. At Knowledge Systems they capture the class-responsibility-collaboration information on objects using the Well-Defined Object concept. When you do this, you see a common responsibility language emerge with repeatable patterns for describing abstractions. Sam hoped that such patterns could be used to provide a quick match against a large number of objects to identify just a few objects for a more detailed search.

Brad Balfour mentioned that SofTech's tools use the faceted classification scheme developed by Rubin Prieto-Diaz. This classification approach is used to create a domain terminology. The faceted scheme is superior to a hierarchical approach because it is much more bottom-up expandable and combinable for searches.

Brad Cox, claiming to be non-sarcastic, said that the question was a lot like asking "Where do I find a good Mexican restaurant?" Brad noted that we manage to answer this question without too much problem. The point is that human beings are good at organizing information, given an incentive to do so. Right now we don't have such an incentive for software, so we try to compensate with high-powered data base technology, and that will never work.

Follow up to Brad Cox: What about the software IC concept you espoused not long ago?

Brad said that he had done a lot of work on specification techniques and inspection gauges a few years ago. He is more convinced than ever of the importance of these techniques. However, he couldn't make it make sense to do such things unless we addressed the income issues.

Ed Seidewitz remarked that the original question assumes that first we figure out what system we want to build, and then we find reusable components that fit the concept. In Ed's organization, they have a large number of systems that have a great deal of similarity, but vary from one satellite mission to the next. As a product of their domain analysis they are coming up with a general specification map of the functionality that the organization needs to support as an institution. This map will be used to define how to support a mission using only the institutional functionality that already exists. and then tailor this as necessary for mission-specific needs. This approach shifts the emphasis from trying to find reusable components and justifying their use, to identifying what is not reusable and justifying not reusing institutional capabilities. Ed concluded by noting that in a situation of 80% to 90% reuse (which his organization has already achieved in certain limited areas), it is actually easier to only have to identify what is not reusable, given a standard map of reusable capabilities.

Sam Adams agreed that any development process must consider reuse heavily in all phases of the life cycle in order to make reuse happen. He felt that most of the processes out their are about clean-slate design, not assuming the availability of large reuse libraries.

Question: How does this panel get the word out to the maintenance programmers who do not attend OOPSLA, but who are involved in 80% of the software life cycle.

Sam Adams agreed that 60% to 80% of the budgets of IS organizations are for maintenance. Thus the real bang for the buck is in this area, not just creating new software. Sam argued that we have to start viewing software as an asset, not a liability. Software must become a long term financial medium that pays back dividends and that accountants can put on the bottom line. Until then we can't do reuse properly. Sam pleaded for a change in the accounting rules to allow this to happen. Given this, software organizations will eventually change into ones that deal with the constant evolution of software.

Brad Balfour noted that there is a group of people in the Ada community that are in fact currently working to get the accounting rules for software changed. Brad also felt that there is a natural grass roots movement of reuse into the maintenance area. If code is reused in several projects, then the maintainers tend to see it over and over gain. There is then a realization that they can get a maintenance benefit from being able to deal with this common code. Brad admitted that this takes time, though.

Question for Sam Adams: You described your idea of an organization with a whole lot of projects, trying to bubble up reusable code from those projects and then market this internally in the organization. How do you do that without having programmers dealing with blue sky issues, instead of the problems that the client is paying to have solved?

Sam agreed that this was an important management issue. Sam related that he had heard at the Washington Ada Symposium that some organizations in the Ada world have required projects to produce their deliverables with 30% of the code reusable on other projects. He hadn't realized that anyone was that bold! However, what does tend to work in general is to put a member on a project team to focus on reuse. Nevertheless, Sam admitted that it can be seductive with an object-oriented approach to try to come up with a generic solution and never get your own project done.

Question for Brad Cox: We already have the technology to sell software as you described in which you would get a product with a little boot device that lets you use it a limited number of times, then it prevents you from using the product any more until you buy more uses from the producer. This would let you try things out cheaply and only pay more for the products you liked. Is there anything like this today, and if not, why not?

Brad Cox stated that the pieces of even the most ambitious implementation of what he proposed do exist today. Brad felt that the approach suggested by the questioner, however, was too much like copy protection schemes that have already been rejected as persecuting the honest. He is more interested in solving the problem right using pieces that exist today, like computers, communications, the ability to write metering software, credit card companies as models and encryption software and hardware. Brad admitted that this will be a major cultural change, but perhaps the world is just becoming ready to think these thoughts.

Brad Balfour then voiced some reservations with the concept of usage fees. He felt that the approach advocated by Brad Cox could actually be a disincentive to reuse. If it is necessary to pay every time you use a reusable component, then the perception of programmers will be that they can write their own version cheaper, rather than paying over and over again. Having done the calculations, he realized that this is not really true, but he emphasized that it will be perceived that way. Brad concluded with the comment that having to pay for using reusable code would not help promote the necessary culture shift from the fun of writing new code.

Sam Adams related an experience that Knowledge Systems Corporation had when it tried to sell some simple reusable components at OOPSLA'87. They originally priced the package at $250 for a standard source license. However, no one would buy the product because they thought they could just write it themselves. Sam estimated that it would take him about 2 weeks to completely reproduce the package they were selling—at a cost of a lot more than $250! Nevertheless, KSC dropped the price to $99, about as low as they could go. The reaction then was that the software must not be worth much because it was so cheap! Sam's point was that there won't be large scale reuse until you can establish a marketplace to determine the price of reusable components. Right now, we don't know how to value such components, so we don't know what we should pay for it.

Question for Dave Wade: In your project, how did you gauge the quality of the components you collected?

Dave replied that it is very difficult to measure quality. The only tangible measure of quality they have on his project is the program trouble reports they receive on the software. Dave didn't think this was a particularly good measure of quality, but it is all they have. Nevertheless, initial statistics show a surprisingly low trouble report rate for the reusable software developed by his reuse group, relative to other software on the project. One reason for this may be that at one time they had as many as four different development platforms. The reusable component developers had to test their software on all four platforms, so their software got more testing.