“Transition to OO” - The BaBar experience

Bob Jacobsen

LBL, Berkeley, USA

1 What has to built

The BaBar collaboration is trying to build a long-lived, flexible offline software system from scratch. To some extent, this has meant moving away from the infrastructure available to the more traditional HEP software systems, and striking out toward a new set of solutions at many levels:

– Language features (C+ vs. FORTRAN77)
– Language extensions (class libraries & databases vs ZEBRA and CERNLIB)
– Code management tools (CVS vs. Historian, Patchy)
– Standards, practices, policies
– Programmer skill, commitment and ingenuity

To create an entire system in the three years we have available, we have to make progess on all these areas simultaneously. In particular, we have to maintain a strong focus on developing and involving our collaborators. Its our experience that very few of our collaborators are a priori interested in OO technology; rather, they are neutral to the software engineering aspects, but motivated to make things work within some provided software structure. We can engage them in programming, but there is significant resistance to “learning software engineering” as a separate skill.

The software development plan has to take this reality into account, and simultaneously develop our design skills, our system design, and our code base. To do this, we started a four-prong attack on the project:

2 The four prong approach

Architecture: By defining a simple but flexible architecture for the overall system, we could immediately start identifying where people could contribute, and into what context their efforts would fit. This also reduced the huge number of possibilities available to a startup effort, which allowed us to put a lot of religious arguments behind us.

Start an iterative design & implementation process: A balance must be struck between "Getting something in place" and design work, because we can't successfully design a system on paper. We must build a significant part of the system, expose it to users, examinine the result with a critical eye, design improvements and implement them. Each turn of this wheel improves the design, improves the resulting code, and improves our abilities. All of those are important. Note that we are deliberately using evolutionary pressure to improve the result: there are several areas where we have multiple solutions under construction, which are continuously being compared as working systems, and mined for new ideas. This process has resulted in significant change to the architecture, to the high and low level designs, and to a lot of the code.

Control that process: "Software engineering process" is uninteresting to the average physicist, and even occasionally a target of ridicule. Rather than debate it, we choose to control and improve that process through a software release system. Starting with CVS as a code repository, we have built a layer of tools on top to control monthly or bi-weekly working releases of the entire offline system. We are using this system to quasi-statically introduce quality control and documentation standards, and to ensure that the separately-developed parts of the system continue to work together. More importantly, the pulse-beat of the releases gives people something to contribute to, and hence to "join". By *expecting* that people will contribute their efforts to a large, growing common set of code, we reduce the probability of people going off in their own, incompatible direction; instead, we can get them involved in design discussions, standardization efforts, etc, quite early in their development.

Team-building: beyond the obviously necessary efforts to get people involved, its also necessary to continuously motivate them to upgrade their design skills and efforts. A balance has to be struck between formal training, direct experience, and exposure to the efforts of others. By relentlessly focusing group discussions on the design level issues and on the performance of the evolving system, rather than at the code details, we simultaneously get people to think in those terms while solving real problems. For example, the number of people who are drawing Booch ("fluffy") diagrams has continuously increased as people begin to understand their utility; we are beginning to see a similar ramp in the use of CASE tools.

3 Experience with this appraoch

Over the past year, this approach has been a qualified success. Many of the more active developers have significantly upgraded their skills. The design discussions within the various working groups continue to improve, becoming both more comprehensive and more effective. The experience gained by building the first iteration of the system has been invaluable: our understanding of the physics issues and their interactions with the code design has improved. In turn, our experience with the technical details has taught us a lot about what's important and what's not when building larger OO systems in the HEP context. Our standards make a lot more sense, our designs are now focused on things that matter, and the people interested in religous debates have moved on to other projects. We think this improvement is on an exponential growth curve, and hope it will last at least until BaBar starts to run.

In the end, what will matter is the actual system that we deploy during the summer of 1998. Will the software work? Will the people be able to maintain and upgrade it? To that extent, the real proof will be *using* the system. Because we always have a working system, even if incomplete, we are already in the position that physicists outside the development group can use it to start work on their analyses and studies. A formal "Mock Data Challenge" is planned for each summer until the detector runs, with the goal of getting feedback from these users by getting them excited about looking for "hidden physics" in large Monte Carlo data samples. Running a large number of events through a production version of the code, and then asking non-developers to study resolutions, efficiencies, etc, risks a lot of embarrassment: Few things work well the first time. But it also generates a lot of familiarity, buy-in, and eventually enthusiasm for the effort, and appears to be generating more effort from new people than its costing.

4 Summary

If it had to be summed up in a few words, the lesson of BaBar's transition seems to be:

Although standards, documents, and studies are important, they pale in comparison with getting your people involved in the messy process of building a system in a new way. Once they are engaged in that continuing effort, you can work with them to upgrade their approach, and they will adopt anything that they see helps them meet their goals.