Finding Objects:

What to do when the Cards don't work.

Ward Cunningham
Wyatt Software

Good decompositions of complex problems can be found by allocating responsibilites among collaborating objects. We have presented an informal method for making such allocations using ordinary index cards (1). A key part of the method is the assesment of goodness made by working through computational scenarios using cards as props and collegues as judges. This process depends on their being sufficient collective experience within the group to actually solve the problem. First time object-programmers are advised to adjust the scope of the problem or the breath of the team to insure this condition.

In practical programming situations such adjustments are not always possible. At times a group is simply not prepaired to solve a problem. This becomes apparent when scenario judges find they have no opinion. No opinlon being counter to the norm and usually accompanied by an uneasy feeling in the stomach. Continued work with index cards becomes uninformed speculation.

Cards don't always work. Cards do correctly identify where experience is in short supply, but, they don't always lead to good decompositions when that happens. An experienced programmer will take heart in knowing that even poor decompositions can be made to work and that he will soon have a great deal of experience. My advice then is to forge ahead with any objects. When simply quiting is not an alternative take whatever you have and proceed. Programming will reveal its strenghts and weaknesses.

One can expect a poor choice of objects to slow development, enlarge code and limit reuse. These are all signals that further innovation is required. The inventive programmer will set upon such programs with the hope of finding objects. In particular, he will search for the *missing* objects. Once identified, they can be inserted into the otherwise deficient program with modest effort and dramatic result.

I would not make this suggestion had object-oriented programming not been accompanied by truly outstanding programming environments. The best offer sufficient mechanism to recover from almost any organizational mistake, a property we have found possible to extend into deployment.

Organizational mistakes should not be confused with program errors. The latter are failures to meet specifications, the former only missed opportunities for efficiency. Organizational mistakes have a substantial cost that must be figured into the economics of program development. They slow development, enlarge code and limit reuse. All all signs of missing objects.