Interface design as part of writing the specification

Vlad Golovach and Alexander Belyshkin, 27 June 2003. Translated from Russian; the original is on guimachine.ru.

Putting a business automation system into use is, as any specialist in the field knows, far from simple. Building the system is, generally speaking, not very hard technically: the average system can hardly be said to be full of the most complex algorithms. Getting it adopted, though, demands uncommon skill, persistence and resourcefulness. Many of the problems are rooted in the requirements specification. As the saying goes, “we built what we planned”, and then it sometimes turns out that the plan itself was wrong.

Many technologies and methods have been invented for the problems that arise when a specification is written and surface when the system is rolled out. Their sheer number shows that none of them guarantees complete success. Many also share a basic flaw: they add work at one phase, even if to save it at another, and require a serious investment in training staff. UML is a typical example.

There is, however, an approach that needs no special qualification and makes adoption much easier without adding to the work of preparing the specification. It rests on one idea: designing the interface is not part of development. It is part of specifying the system.

Two things need to be made clear here. First, the interface will have to be designed in any case; practice shows that customers are for some reason reluctant to pay for functions without an interface. Second, designing it requires nothing special, and the same resources can be used as for ordinary development. With apologies to the specialists who earn their living by designing ergonomic interfaces, the interface here does not even have to be ergonomic: adoption will be easier all the same. An ergonomic interface would of course make it easier still, but it takes longer to make and costs more.

What the approach gives

The approach has the following advantages.

First, it removes the difference between how the customer and the contractor see the task. Specifications for any system of some complexity are too abstract. Even their authors struggle to keep them in their heads, and nobody understands them fully, including the key person, the customer. Many people even believe that incomprehensible specifications exist to impress the customer and extract more money. But it is naive to suppose that a customer who readily signed an unintelligible specification will accept the finished system just as readily. Interface prototypes are the one document a customer can understand and judge, and, having understood and judged it, sign knowingly.

Second, it makes adoption easier. In a well-executed project a large share of the adoption problems comes from an interface that is formally correct but does not match what the customer had in mind. No kind of specification other than an interface prototype can capture requirements of this sort. A plain example: any specification can state that “the system has an address book consisting of such-and-such data and such-and-such functions”. It cannot formalise how the address book should actually work, which functions should be brought to the surface and which can be tucked away, or, in the end, what the address book should look like. When the contractor then appeals to the signed specification (the functions are listed here, and here they all are), it usually does not work. With a little ingenuity a specification can always be read in very different ways where the user interface is concerned. Only an interface designed in advance insures against claims of this kind.

Third, it reduces the rework caused by a mismatch between what the system does and what the client expected. Only on seeing the system can a customer really understand what it can do and judge their own needs. For the customer the software product and its interface are in a sense the same thing. So showing the customer the interface while the specification is being prepared reduces the number and size of the changes. (It should be said that such changes are usually no problem for developers, who tend to insist on being paid extra for them.)

Fourth, it lowers the risk that the customer will be dissatisfied with the proposed interface and that the system’s functions will have to be reworked as a result. When an interface is developed there is no guarantee at all that the customer will accept it. A description of a system’s functions is binary: a function is either there or not, and proving that it is there rarely takes argument. An interface can only be good enough or not good enough. Once relative terms come into play everything gets harder, and conflicts can follow. Needless to say, when an interface that is not good enough is redone, the functions the system already has change too, and the developer is not paid for that work.

Problems

So there is an objective case for treating interface design as a stage of writing the specification, not a stage of development. But how is it to be done? At first sight the task looks hard, both organisationally and technically.

The organisational side first. It would seem that the customer expects something “real” to be done as soon as the contract is signed. Practice shows, however, that visible intermediate results, namely interface prototypes shown on the second day of work and not after several weeks, put the client in a good mood. Work on an ordinary specification is not really visible to the customer (“so a couple of clauses were added”). Interface prototypes are easy to take in, and progress is plain to see. The second organisational problem is that two contracts have to be signed: one for the specification (that is, the interface) and one for developing the system’s functions. The second is postponed for as long as the interface takes to design, which stretches the project out. In principle this

problem cannot be solved, but much depends on how it is seen: yes, there are two contracts, but the wording of the second is far more precise, because with the interface in hand the effort is easier to estimate.

The technical problem is the difficulty of making a prototype. Normally the interface is created in the development tool itself, and building a prototype that way does not pay. An interface is made through many iterations, and redoing what is already done becomes expensive. Special tools for prototyping interfaces have appeared fairly recently (Norpath Studio, for example), but they are still rather raw. The way out is to use general-purpose graphics and presentation editors. A few years ago the main tool of this kind was Microsoft PowerPoint; today Microsoft Visio is the most convenient. Complex screen forms, though, are still easier simply to draw on paper.

And finally the main problem. Developers themselves often see a longer specification phase as an unqualified evil. The habit of doing first and thinking afterwards is traditionally strong in the Russian IT business too. Sadly, only hard-won experience can change this custom. For now, at least.

Designing the interface while the system is being specified is not a cure-all, of course. It does not improve the quality of development as such: it does nothing, for instance, to reduce the number of programming errors. Nor can it always be applied. The interface of a complex system cannot be designed in full from the start: a working beta has to be built first and the interface finally corrected on that basis. In many projects, for reasons that are nobody’s fault, the specification phase cannot be stretched, because the customer wants to see some results tomorrow. Yet given how little the method demands to get started, nothing like the bureaucratic overhead that comes with UML, designing interfaces at the specification stage almost always proves an extremely successful way of solving the problems of adoption.