*Software Quality Journal* **Vol. 13 No. 2, 2005**

# Editorial: Can Users Read Developers Minds?

Software development involves lots of mind reading. Developers continually try to read users' minds by imagining what users will try to do with software once it is released. Developers also try to read the minds of those in management to determine the "real" requirements, as opposed to what is in a written requirements document. Developers learn that hard way that they must satisfy unwritten and unspoken requirements. In addition, developers must guess what new features will be required, so that designs can be structured to be readily modified to include the new features. Of course, mind reading is not easy, and developers do not have greater telepathic powers than most people.

It turns out that, not only must developers read users' minds; there are times when users really need to be able to read developers' minds. Users may need to think like a developer to understand how to correctly access particular functionality. There is usually a gap between the system that developers have actually built and the system that users think that they are using. This gap can lead to both annoying and devastating problems, especially when users' expectations are reasonable, and the software limitations are not.

Problems are especially critical when they are in systems that control important aspects of users' lives, for example personal or business finances. One such software application domain is the software now commonly used in web-based, electronic bill paying systems. Bill paying software is connected to an overall banking system that allows customers and bank employees to examine the status of several accounts with a bank, define a set of "payees" for electronic payments, and schedule payments to be made to payees.

Here is an actual scenario that occurred to a customer, who will remain unnamed, of a major bank in the USA. This customer had a personal checking account, and had been using it successfully to pay bills both by check and via electronic payments. In order to better organize personal finances, this user opened a second checking account at the same bank to pay professional expenses. Such an arrangement makes it much easier to separate professional expenses from personal expenses and is especially useful when preparing US income tax returns, one of the most odious requirements of living in the USA. The customer would deposit regular income into the personal account, and deposit just enough funds into the professional account to pay expenses.

Using the on-line banking system, it is relatively simple to navigate between accounts to check balances, etc. When paying personal bills, the user navigates to the personal account and then selects the "schedule bill payment" option to set up the required payments. This method seemed to work fine for a while until a statement arrived with a series of overdraft charges piled onto overdraft charges adding up to several hundred dollars on the professional account. The customer called the bank toll free number and, after surviving phone software hell, learned that she had been inadvertently paying personal bills through her professional account rather than the personal one.

It turns out that after she opened the professional account with an on-line bill payment feature, the banking system automatically paid all bills from the professional account. The problem, from this customer's perspective, is an unstated, or well-hidden, system constraint – customers can only use one account for on-line bill payment, which cannot be changed by the customer on-line. Rather than allow the user to select an account to pay bills from, a default account for bill payment is set. The constraint is not obvious; it is not stated on the bank web page, and it appeared to be only known by those in the technical office. Local bank employees – those who advise customers when they open accounts and when they have problems – had no knowledge of this constraint. They were as surprised about this as the victim (the bank customer).

Consider this scenario. A customer navigates to their personal account, then selects the "bill pay" option, and proceeds to schedule payments. Most customers, other than programmers, would naturally assume that the personal account would be used. Unfortunately, this was not the case – the professional account was debited. The bill payment page does not indicate which account is to be used, and it appears impossible to tell from the provided on-line information. Perhaps this limitation is a security precaution. However, this behavior is clearly a problem for users with multiple accounts. And multiple accounts are common. Surely, there are thousands of bank customers who have or will run across this problem, which I classify as a serious program failure.

Depending on your perspective, the fault that is the root cause of the failure is either missing functionality or missing documentation. From the user's perspective, the fault is that users are unable to select between accounts when paying bills on-line. From the developer's perspective, the failure was caused by the missing documentation of the constraint that bills could be paid on-line from only one account per customer. Clearly the actual system behavior did not match the behavior expected by the customer or the local bank employee. A user would benefit, and possibly discover the hidden constraint, by reading the mind of the developer. But, to do this, they must have more knowledge of software design than most people.

Someone with a developer's perspective will look at the system and on-line bill paying process a bit differently than more naïve customers. Since the internet protocols are generally state-free, servers do not automatically know the relevant history of a client – the bill payment page software may not know that a client has just visited the personal account, so that it knows to pay bills from that account (a natural assumption for a naïve customer). Someone with a developer's perspective knows about the possible limitations, and is more likely to verify that the proper account is debited. However, most users are not developers, and are too trusting that software will do "the right thing."

Software failures that affect personal finances can be very serious and have safety implications. Customers can panic after receiving unexpected and unwarranted overdraft notices. A customer with a bad heart can end up in a hospital (or worse), and a mentally unbalanced customer may react in unpredictable and dangerous ways.

The actual scenario related here turned out fine. The local bank official apologized, and reversed all of the overdraft charges. The local bank official and the customer learned some important lessons about the limitations of the software. However, there should be easier ways to communicate key details about how a system works. Let's not force users to be mind readers.

Both developers and users are poor mind readers. Developers can't anticipate users' behavior and expectations. Users should not also have to read developers' minds. It's too bad that mind reading is necessary, since it is impossible. I guess that we must all wait for some astounding advances in clairvoyance and telepathy. Otherwise, software professionals must do a better job of satisfying the information and functionality needs of all stakeholders.

James Bieman  
Fort Collins, Colorado  
U.S.A