Feature Articles

TopCoder @ Work: Introduction to upselling, Part 2

By **timmac** & rhudson
TopCoder Members

Last week we attacked a common problem -- selling new ideas and functionality to clients -- with a clever acronym (UPSELL). Today, we'll demonstrate how to apply that methodology to a realistic situation combining common situations we've encountered into a complete story of upselling.

Suppose you are initially hired by a client to work a one-month contract, taking some information from an Access database, and displaying some reports on the web. Creating a few basic HTML reports from database tables proves to be trivial; you quickly finish your work. The techie in you speaks: "This system would be much better in ASP.NET and SQL Server 2005. A few third-party controls and some binary formatters would make this perfect." Yet, how do you reconcile your inner geek desires with what the client actually needs?

First, you've got to **understand** the client's business. He is an accountant, and has been complaining a lot about reporting. You hear him almost every day in the hallway outside the break room, chatting with his assistants. They grumble about the endless hours it takes to create the month-end reports in Excel. They get 50 different spreadsheets and have to cobble them together. Month-end overtime is a way of life; the five member staff just finished a 36 hour weekend on top of the 60 hour work week. The future looks grim -- the company just acquired two smaller firms and will have to reconcile their data each month on top of the current workload. Temporary workers seem to hinder rather than help -- most of the work is complex and the business processes are locked in the minds of the full-time staff. Your client breaks a cold sweat when someone asks: "What will you do if people quit?"

Based on this overheard conversation, you **plan** to assist the client in at least three different areas. To standardize the work and move away from the variety of Excel spreadsheets, you could develop a standard interface that accomplishes the same business goals. You might even consider Ajax or Web 2.0 functionality to provide more responsive client-side behavior for people used to the point-and-click widgets of Excel. Second, you could expand the functionality of your web application to include business objects that provide the essential calculations and operations for the month-end process. This would both make the process explicit, "unlocking" it from the minds of a few employees, and reduce the time required to manually combine the spreadsheets. Moreover, you could provide a robust reporting system that would generate Excel output for month-end processing. These approaches would address the short-term needs of the company.

In the long-term, however, you know that the Access database won't be able to handle the processing load of the data; the MDB file seems to grow exponentially each month as your client's firm acquires new companies and imports their accounting information. The database won't "explode" today, but will soon become a bottleneck for operations. You know that once your application is online, the user base will grow tremendously. Your plan must include an upgrade to a larger, more scalable database like Oracle or Microsoft SQL Server.

Now, you must convert your ideas into something presentable to the client. In a traditional life cycle, you would create requirements, or use cases, and a few flowcharts. More enlightened methodologies like agile development call for different documents and processes. In this case, however, your best approach, in our humble opinion, is to make it simple: show the client pictures. In other words, provide mock screens that depict how the application will look and function. Usability and information architects call these mockups **storyboards** or wire-frame prototypes. The site BoxesAndArrows.com provides both articles and templates for this process.

Use the existing month-end spreadsheets, if you can, to determine the rough requirements for the screens. It doesn't matter whether you've captured the process accurately at this point. You'll have enough to demonstrate to the client that you're listening. What you've heard, then, becomes something tangible for the client. Don't limit your storyboards to screens; also mock-up reports that approximate month-end information. As you *walk through* the screens with the client and his staff, elicit *business rules* and processes that would go with each screen. Listen for phrases like "I need to add more than one of those, " or "where are the associated .." Build a narrative from the client reactions to your screens. You can return later to edit and make the business processes more explicit.

At a surface level, nearly every client's first concern is what they can see in front of them immediately. This makes our second sell, the scalable enterprise database, more difficult. Though critical to operations, the database sits *in the background* and already appears to work. Yet, this functionality is only short-term. You know that an Access database will not be stable over time, but aren't sure how to present this to the client. How do you storyboard "database scalability"?

We've made an attempt here:
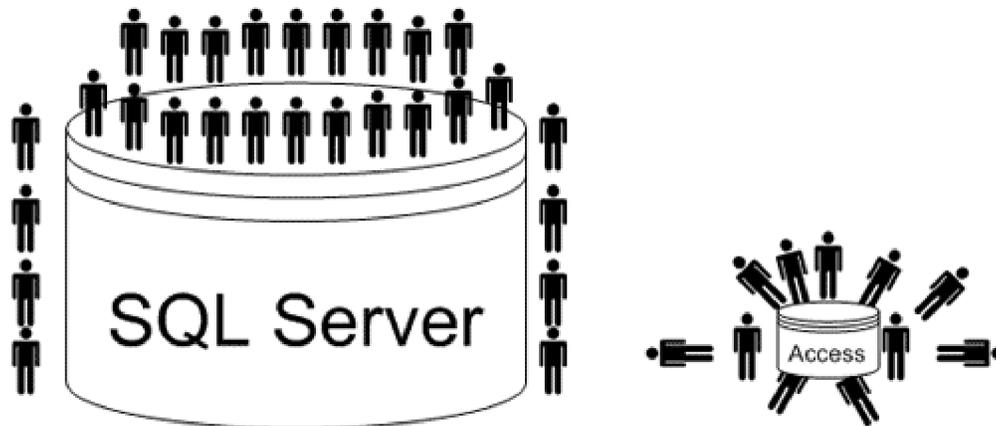
This chart illustrates that SQL Server is stable, and capable of supporting many users. Access, however, is unstable, and is clearly hazardous to clients. Support this image with documentation and case studies from companies who have made the Access to SQL Server migration. Use white papers, vendor recommendations, and your experience in the field. The client will likely identify with and remember the illustration.

Once you've created and presented your planning documents to the client, ensure that you have enough information to estimate the effort required to actually produce the system that you've pitched. Before you begin, however, be sure to finish gathering and documenting requirements and business processes that support your storyboards. You can do this for the entire scope, or for the first part of the system that you'll build. For estimating tips, see our recent TopCoder article: "How long will this take, anyway?"

Your **estimate** should be divided into phases, that is, the smallest functional deliveries that add value to the system. You might begin by enhancing the entry screens in your application to accommodate user interface requirements gathered from observing Excel spreadsheets. Next, you might integrate the SQL Server and the reporting infrastructure. Each subsequent delivery may include more screens, or more reports. Keeping delivery schedules small reduces the risk of cost overruns and failure.

Once you've developed and divided your estimates into deliverables (according to phases), **lock** the first phase with the client. In other words, ensure that the client agrees, or signs off on the requirements and time frame for the particular deliverable you're completing. Work with the client to develop acceptance criteria, like "In this phase, the system must calculate month-end totals for the debit accounts." Plan a time to demonstrate the completed phase to the client, and a time to discuss subsequent phases.

Six months have passed and you're preparing to finish the final statement of work with your client. As you predicted, the user base has increased from the original five staff members to seventy-five people, including managers and executives. The latter group was so impressed with your work that they've added an online newsletter to the set of requirements. You anticipate an endless stream of content update requests; you'll be busy indefinitely. Yet, as a developer you're bored with the idea of modifying an application for each content tweak. The tedium of redeploying for each minor change is almost unbearable. At the same time, you don't want to turn down work.

Or should you? Why not develop dynamic updating, or wiki, capability for these news-oriented sections of the application? While it might seem that you're losing money by relinquishing the work, you're actually **leveraging** your relationship with the client and building trust. Providing real-time, do-it-yourself features communicates that you are not scavenging the client for every scrap of free work. Offloading the work to the client empowers the users and also frees you to develop more advanced applications.

Upselling won't always work, and since you're on your own time, in a sense, at least initially, you might burn hours only to be met with a courteous "No, thanks." Not all of the work is wasted, however. You've probably built a set of templates and boilerplate that you can

customize for each client. At the same time, though, be sure not to give away "free work" to the client. Requirements, designs and prototypes are valuable; we consider them to be billable work product. Without an arrangement in place, a client can simply take your designs and either complete the work internally or search for a cheaper vendor. Some clients don't believe that they need the documentation; we commonly hear: "I don't need you to tell me how my business works." Clearly, many of these clients **do not** have a systematic concept of their own business.

Don't let these warnings dissuade you. Building relationships and creating future business opportunities with clients can be an exciting process. As a software developer, you gain an understanding of the actual impact of technology on real people. This experience translates directly into better work and a better job experience. Your client also gains familiarity with the abilities and constraints of software to solve problems. Ultimately, you both will grow from the cooperative engagement of creating a solution to a problem; you'll gain experience and cash, and your client will save time and money.

Part 1