

Feature Articles

TopCoder @ Work: Introduction to upselling

[Discuss this article](#)


By **timmac** & rhudson
TopCoder Members

A developer's story:

It was a two-week project. That's all I could get on short notice. A lucrative contract programming gig had just fallen through, and I was out of work. The future was uncertain, but for the next 10 business days I would be developing Excel spreadsheets, muddling through financial data, and building VBA functions. It was hardly the work I was used to -- in previous jobs I'd done enterprise .NET and SQL Server tiered applications -- but it was all I had.

On the first day of work, I grumbled to myself, "I'd rather be doing web and database applications." At lunch, I idly sketched HTML and class diagrams. That night, I decided to turn those diagrams into a proposal. At the end of that first week, I had client approval to fund the web system over a period of four months.

By applying "upselling" techniques, formerly relegated to the domain of car dealers, I was able to extend a contract, expand my technical horizons, and develop an opportunity for future business with my client.

In this article, we'll describe how you can "sell" projects to both internal clients (if you're salaried), and external clients (if you consult), using a model we call UPSELL: Understand, Plan, Story, Estimate, Lock, and Leverage.

Understand

In discussing some of our experiences, we agreed that one of the greatest challenges we faced as developers was learning to listen to a client and understand what they were really saying. Barring those managers who fumble with technological language they barely understand, most clients speak in their own "business language." They know what they would like to accomplish, but aren't sure how to communicate it in terms of a system. In fact, many clients don't have a complete grasp of their own business process, inputs and outputs.

Here's a typical client statement:

Client: "I just want the application to be able to [insert vague or minimal-sounding words here]."

Qualifying words like "just," "only," and "minor" cause experienced developers to shake with the fear of creeping scope and client oversimplification. You may have heard some of the following:

What the client says...	What the client actually means...	What this means to you...
I just want to be able to download this data into an Excel spreadsheet.	Excel is so easy to use, so it must not be that hard to program. I can do formulas and put information wherever I want it.	This could either mean a simple dump to a CSV that the client could load into a basic spreadsheet, OR it could mean the client wants me to generate his complicated spreadsheet out of a database. I'm thinking about advanced exporting and whether or not a relational database can produce this funky spreadsheet without major manipulation.

I just need a way to calculate some things from this set of data.	I want to get something in place so that I don't have to keep doing this by hand. Let's just make this a simple utility -- it doesn't need to be anything complicated -- so we can get it done quickly.	Once the client gets a chance to use this utility, they might find it would be nice to do other calculations on other sets of data. It might be worth devising a solution that is expandable -- but still delivers the short-term goal of the desired calculation. Yet, an expandable solution would take more time initially...
I just want to see the results highlighted for these specific cases.	We've already got the cases on the screen; how hard is it to change the color of a few that I want?	Do the system rules support automatically choosing these cases, or are they determined by a complex calculation? What other rules, "special cases," or decisions will the software have to make? Can I build some kind of rules engine that will enable the client to highlight other elements of interest?
I just want this to work so that it stops accepting invalid registrations, and customers stop seeing the wrong results.	The system isn't working, and customers are complaining. I shouldn't have to pay for something that doesn't work.	Of course it doesn't work-- I barely had any time to test the thing, and had to hard-code the validations on every page for this "quick solution." There might be a more robust way to build in error-handling and validation...
I just want to keep this simple, so I don't want to convert to SQL Server, let's just leave it in an Access database. It works just fine like it is.	Why should I pay for a costly piece of software, and hours of developer time, for something that already works like it is.	How do I explain scalability to the client? The users are doubling every week, and we're importing more and more information into Access. This thing is a ticking bomb...

As you can see, what initially appeared to be a disgruntled client is really someone who requires an education in the benefits of spending money to improve a system. The most important way to "understand" the client is to observe working patterns, idle comments, and complaints. Ask questions like "what takes the most time to do?" and "where do you see lots of errors?"

Understanding the client's business is the key to upselling a solution that is presented in terms of the benefits to the client. Consider the hazards of inaction from the client's point of view, as well. Going back to our Access example, the client won't care if you look down your nose at Access, but once they understand the actual risks of database corruption to their business they might be more receptive to your ideas.

Plan

Rephrase each client remark in terms of a requirement, and devise a design and implementation that will solve the problem. Provide a phased implementation approach that gives the client options. Encourage your client to choose between two of your solutions, rather than choosing to do nothing.

At this stage, it's helpful to create user-centered documents, like business process flows and picture diagrams, that describe your solution concretely. Present cost savings in terms of estimated hours if you don't have concrete dollar amounts. Explain to the client that time is money -- and that time saved is time that can be applied to strategic purposes and increasing company income.

Storyboard

Add on to your process and persuasion documents with narratives and prototypes that depict what will "actually" happen. If you're extending the User Interface with a new reporting tool, show sample screenshots of the tool. Consider using a prototyping tool like Visio or Axure to make a "live" clickable prototype. Did we mention that clients like pictures that move around on the screen?

If you're pitching a "non-visual" solution, like a scalability increase (moving Access to SQL Server), prepare animations and slides that "show" data load and user increase. Consider visual metaphors like "overloading" and "exploding." This will make your presentation unique and stimulate discussion.

Estimate

Know about how much work is involved with each piece of the proposal. If your story goes well, this will be one of the first things the client wants to know. Remember that it is only an estimate, and that more detailed figures will be determined as more is finalized. Refer to our previous article on [estimating software](#) for advice on how to create a basic framework of costing. The article describes how to enumerate general costs and present a quantitative view of how you plan to implement the application. Again, think of implementation in terms of phases; the smaller the deliverable the less risk for you and the client. Minimize cost overruns by reducing the features you release at one time.

Lock

Put together a detailed list of features and functionality that will be built. The more specific you make this, the less room there is for confusion later. This is the point where finalizing requirements -- and removing all possible ambiguities -- is paramount. Seal the deal with the client, and be sure that the previously created list is part of the deal, to avoid any misunderstandings or scope creep. It helps to include "acceptance criteria" or a "testing checklist" that spells out what the product does. Offering a test plan to your client demonstrates your commitment to quality, and also structures the acceptance into a finite list.

Leverage

While it might seem that the hard work is done once the project is sold, this is really just the beginning. In terms of contract work, it is often much easier to get repeat business from an existing client than to seek out new clients. A successful implementation that delivers upon what was promised in your original pitch will bolster your credibility with the client. Additionally, as the client begins to see the predicted value (even if they were once blind to your suggestions), they will become more open-minded to other opportunities.

Convince the client that you aren't interested in "adding on" for the sake of increasing your bottom line. For example, you might integrate content management or wiki-like functionality that will empower the client to make changes to content that you were contracted to do previously. Though you lose the "constant updating" work, you win good faith, and the ability to make more interesting application changes with the full trust of your client.

As with all sales, upselling is still a game of chance. You won't always win: budget, timeframe, or corporate inertia may simply prevent the client from giving serious consideration to your proposals, no matter how well they are presented.

Even with a losing proposal, you've gained both presentation experience and a "toolbox" of upselling techniques and templates that you can reuse. Judging from what we've seen in the world of IT sales, a professional presentation with the elements we've mentioned is extremely rare; you might win business on presentation alone!

Watch for the [second part](#) of the article, coming soon, featuring some cases and a walkthrough on implementing the upsell approach.