The Business Proposition for Developing Custom Tools

Robert Manna – Stantec Consulting Ltd.
David Spehar - Stantec Consulting Ltd.

BO3158 In the era of 2D CAD, many firms invested in custom tools and development staff to write and maintain them. A firm's overall experience varied in terms of the success of this investment and just about everyone has a story to tell about being "burned" from spending money developing custom tools. However a number of market forces are coming into play that continue to drive the need to customize the Autodesk® Revit® software platform, and the question is, "Do you rely entirely on third-party software or go back down the road of customization?" This class explores the benefits of investing in the development of custom tools for your firm, and shows how Revit is really "different this time" for those who have a horror story to tell. This class looks at the tools Stantec has developed in the last year and the return on investment we have already seen. We also examine our overall strategy and approach to development.

Learning Objectives

At the end of this class, you will be able to:

- Convey the value of custom tools for Revit
- Explain the difference between tools and utilities in Revit versus CAD
- Develop an internal strategy for development
- Review resumes of potential candidates who develop Revit tools

About the Speaker

Robert works for Stantec in the Boston office, he has been a key team member on multiple projects, and he now serves as the BIM R&D leader for the firm as well as providing business consulting services for clients implementing BIM. He has taught internally, provides high level support as well as planning and implementation of new tools. In 2013 he led the organization of the first annual RTC Design Technology Summit, he has spoken at: RTC NA, Autodesk University, has been a guest lecturer at the BAC, and has presented at BIM events hosted by the AIA, ACEC, Autodesk and resellers. He has written two articles about Revit for the AUGI AEC Edge magazine, and has written assessment questions for KnowledgeSmart. He maintains a personal blog dedicated to Revit & BIM. When he has time he hangs out with his wife and three year old daughter, and enjoys skiing, swimming and biking
robert.manna@stantec.com
david.spehar@stantec.com

Introduction

I am an API convert. I am one of those people who can remember Revit before there was an API and also believing "why do we need an API" what more do you want or need. I was also naive enough to believe that Autodesk would "fix all the problems" and "fill in all the gaps" with the toolset in Revit.

That was four or five years ago. Now, I realize that try as hard as they might (and they do try) the Factory is never going to fill all the gaps, and there are a multitude of reasons why. On top of that Revit is clearly evolving into a platform where we can use it to achieve many different ends. Lastly with the true adoption in the last three to four years of Revit MEP by engineers, there is simply too large of a base and too many needs for Autodesk to practically fill all the gaps.

What's an API?

API in some ways has become slang for referring to any software tool, application or utility that has been written to work with Revit (or any other multitude of software applications). API is actually an acronym that stands for "Application Programming Interface" numerous pieces of software include API's, Microsoft Office is a great example, but any number of other applications you use day in and day out have API's too. Effectively the Operating System itself provides an API through code structures like the .Net family of languages. Therefore, when someone says "we can do that with the API" that means the API provides the necessary functionality for a developer to write code that will do whatever it is you want it to do. On the other hand, if someone says "no we can't do that with the API" that means the developer hasn't been given the necessary access to the Application through the API to accomplish your goal.

Why develop tools? *(and how is this different from CAD?)*

Many firms in the past spent time and money developing complex tools to help users work in 2D CAD. The focus of many of these tools was to help users maintain consistent "clean" well organized CAD files by assuring that users followed various Layer standards, used proper text and dimension styles and various other graphic output controls to ensure consistency of graphic output. While the need to ensure graphic consistency is still an issue in Revit, to date we have not found it a significant enough issue to spend time developing tools focused on graphic consistency. Furthermore, the way in which Revit operates makes it much easier to enforce or check on graphic consistency using the tools that Autodesk provides either within Revit or as free API based tools.

Instead, we have found that many of our tools and our ideas for additional tools tend to focus on helping users to manage their project data. We have a number of tools that have the word "manager" in their name or "editor". Revit allows our project teams to have so much data at their finger-tips, but it can come up short when it comes to effectively managing data. We also look to build tools where they strategically make sense - is it likely to give us significant return on our investment (in any way) or is it an area where Autodesk is un-likely to focus attention in the relative near term? In some cases Autodesk can be very clear about where they don't foresee further feature development, in other cases you can infer their intentions based on the API features they've included or simply by paying attention to where their focus has been over multiple release cycles.

As a software platform Revit offers one additional distinct advantage over CAD systems. In the process of ensuring consistency, custom CAD tools effectively worked to maintain a consistent data structure (layers), which lead to the desired graphic and data consistency. In Revit the base infrastructure is fixed, Categories can neither be added nor removed by end users; other portions of the "infrastructure" are fixed by way of the Family/Type relationship established by Revit. These features combine to create a platform where you can have certain expectations of the data structure and users must work within that framework. There are some holes in Revit, but they can also be dealt with far more effectively than in CAD where often users would create their own infrastructure by way of creating new layers/levels.

A critical shift has also occurred within the software industry in the last decade. In the eighties, nineties and into the early aughts companies like Autodesk or Bentley released new versions of their software every two or three years, in-between there were perhaps hot fixes, patches, updates, but not the introduction of significant new features. Therefore, when a new version did come out the changes were significant, multiple new features, potentially major structural changes to the code, new file format versions (after three years of a single format version). Such significant change also meant that the updating of any related tools required significant effort. In some cases the custom tools were rendered redundant by new features, or new features were similar, but not quite the same. In any case updating was never simple and required effort to update code, bug test, and finally deploy.

Now, in today's world, we've become more accustomed to annual software release cycles with incremental updates and changes, rather than far reaching modifications to the software platform. With the move to annual updates you are confronted with keeping the code of custom tools up-to-date regularly and more importantly you are less likely to become deeply entrenched in a whole series of custom tools, by having to regularly updating the tools you can continuously re-evaluate the effectiveness or need for any particular tool. If new features are added that "fill the gap" that the custom tool originally filled the choice can be made to either keep developing or cease development and determine how best to depreciate the custom tool for the user base.

Development Strategy

Targeted Value in developing software tools.

While I may have become more in favor of spending money to develop API tools I'm not in favor of wasting money either. Therefore in approaching the development of our own tools we've been careful to make sure we are investing our money in strategically successful ways. Therefore, the first question we ask ourselves when we, or anyone in the organization has an idea for a tool is;

> "Is there something that is similar or achieves the same results on the market now?"

After we ask that question we have to do the following more detailed analysis:

1. What tools are on the market with same or similar functionality
    a. Does it offer one to one functionality?
    b. Does it offer increased or additional functionality, what value does the increased functionality offer?
    c. Does it offer less functionality than we desire, but is it functionally sufficient for our purposes/needs?
    d. Is there an opportunity to work with the developer to improve, shape and guide further development of the tool?
2. How are updates to the tool handled?
3. What happens with new versions of Revit?
4. What kind of licensing is supported (fixed per workstation, fixed named user, network, unlimited)?
    a. If network licensing is supported, how is it implemented?
5. How much does the tool cost, are there bulk discounts?
6. How is the tool installed/distributed?

If in the process of answering any of the above questions we are led to the conclusion that we ***might*** be better off creating our own tool, we then need to know from our developer what the likely cost will be. Once we have an estimated cost to develop our own, we can compare that cost to what is on the market and weigh the various advantages and dis-advantages to developing our own versus buying something on the open market.

When asking our developer to estimate a cost we've learned from experience that it is very hard to get to an exceptionally accurate number. The exception being tools that are particularly simple in the process they have to perform and the related UI to make them work.

> For example we started out with a very simple tool for generating un-placed rooms in Revit from an Excel spreadsheet. Well, yes, it's simple until you say:
>
> - "we want to control what phase the rooms are created in"
> - "we want to set a Key value through our excel sheet"
> - "we want to create quantities of rooms"
>
> When it was just an excel spreadsheet that was a list of rooms with some parameter data, it was "easy" but easy is never what users want. Users want tasks to be "easy" which means more complex tools that respond to the type of input the user is likely to provide, or generates a result the user desires, without requiring many steps.

Therefore, we accept that any early estimate our developer provides to us is likely to be low, particularly as find that we want additional features as the tool develops. Bug fixing and un-predictable behavior (user & Revit) can also increase the time spent on a tool. Once again, the more complex the task/process the more likely you are to have issues. Once the tool is able to perform the “simple” task it quickly becomes evident what additional tasks the tool might carry out, which then results in more feature requests which translates to more time.

The overall goal for us in making the decision to proceed with the development of a tool is to do our best to assure we are delivering the best value for our users and the company by investing in the places that make sense or purchasing off the shelf tools where it makes better sense.

MVP – Minimum Viable Product

MVP or Minimum Viable Product is a term from the software industry and is often associated with an approach to software development referred to as the “Agile[1]” method. Quite a bit has been written about Agile and if you know anything about LEAN processes you’ll find a number of similarities and common traits. The point of MVP is to focus on delivering the bare minimum product that will be functional and useful to some degree. This means temporarily leaving the bells and whistles off in favor of a solid base that works and users can use. It may not do everything users want, and never comes close to the original designer’s dream, but it gets a job done.

When you are a design firm spending your own revenue on developing software, it is important to maintain as tight a focus as possible; as discussed previously, software can get complicated fast, particularly as more features are layered on. We like to dream big, but we have found the best route is to make sure we stay focused on the original and primary goal. What we have also found is that if we do this, new features and expanded capability fall into place and the value of the additional features and the cost to implement them is more easily justified.

The challenge with the MVP approach is that you do not really have a guaranteed end product in terms of a specific feature set. Rather you have a guaranteed functional product at the end of a specific amount of time/money spent on development. While seemingly open ended, this guarantees that for some amount of money you end up with something that will be useful. The drawback is that the product may not do everything you originally wanted, but that is the idea of MVP, what is the base level of functionality that is absolutely required to make the application useful versus “this would be nice to have”.

The other important approach we have taken when determining what tools we should develop is an attempt to bucket tools to see if they overlap, have similar features or are very different from each other.. Alternatly when someone has a brilliant idea for a complex tool’ break it down into constituent parts which are really smaller, simpler tools that can eventually be combined to make a more complex “whole”. This assures delivery of tools that work, without getting bogged down in the complexities of attempting to trouble-shoot complex tools that do multiple functions simultaneously. Our goal is to get the small parts working, then look to combine the small parts into the more complex tools.

[1] http://en.wikipedia.org/wiki/Agile_software_development

KISS (Keep It Simple Stupid)

If we want our users to adopt custom tools and use them, they have to present an extremely low barrier of entry.

- If the custom tool duplicates existing functionality in some way, it has to make it easier for the user to do the task then what they had before.
- Whatever the tool does it has to be “simple”. Humans like gratification; a tool that quickly and easily solves a problem for them without significant input is the tool that will be adopted. Complex tools with multiple dialogs and requiring large amounts of input will initially give the impression that it is “harder” regardless of how much it may save the user.
- Tools cannot attempt to look too far forward. In the vein of gratification most users aren’t interested in a tool that will solve their problem(s) three months from now, they want tools that solve immediate problems and issues.
- Custom tools simply must be easier to use in every way, if the original tool provides a way that gives the impression of being easier users will not adopt the custom tool.
- To borrow a term from Autodesk, tools need to be “discoverable” and they should not require large amounts of documentation, users should be able to fairly quickly grasp the intent and function of the tool.

This philosophy affects all of our decision making when it comes to development. Our productivity tools in particular are driven by this approach, these are the tools that we think can provide the most users with the most productivity increase so we need them to be easy to use. We have developed some tools that are substantially more complicated, but they are also not intended for general consumption, rather to meet specialized needs that can still provide substantial benefits in the right project conditions.

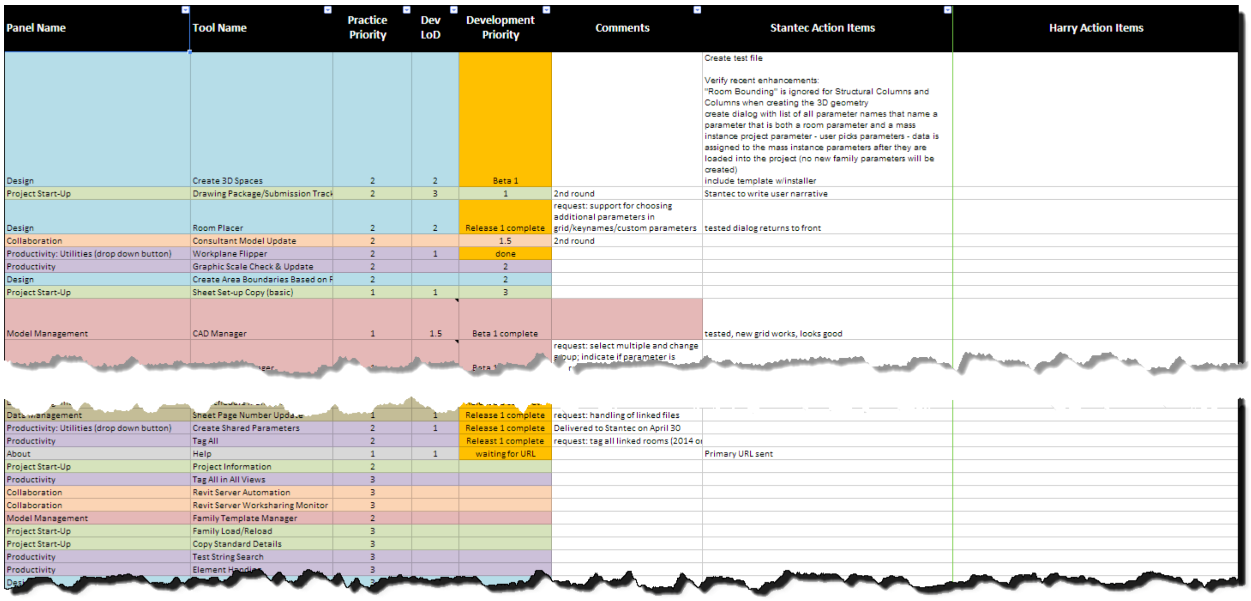
Determining what to Develop

Our justification for developing tools and utilities is to save our project teams time in managing the data associated with their models, or save them time in dealing with time consuming multi-step tasks. In both cases we can clearly articulate where our teams will see time savings and even make a reasonable estimation of the time savings.

To determine what makes sense to actually develop we started with a wish list of ideas from a variety of users and by also looking at what was available on the market but, as discussed previously, would be cost prohibitive to make available to all of our staff. Employing our MVP strategy when evaluating tools, we also made sure to focus on what is the most critical functionality presenting the greatest benefit to our users.

With all of our criteria in mind we assembled a roadmap of potential tools to evaluate with our developer and determine which would be the easiest tools to develop while also understanding which ones may require more time. We were then able to prioritize estimated cost against estimated value to arrive at a list of tools that would be developed.

Our "Roadmap" (spreadsheet) for tracking tool development



Panel Name	Tool Name	Practice Priority	Dev LoD	Development Priority	Comments	Stantec Action Items	Harry Action Items
Design	Create 3D Spaces	2	2	Beta 1		Create test file Verify recent enhancements: "Room Bounding" is ignored for Structural Columns and Columns when creating the 3D geometry create dialog with list of all parameter names that name a parameter that is both a room parameter and a mass instance project parameter - user picks parameters - data is assigned to the mass instance parameters after they are loaded into the project (no new family parameters will be created) include template w/installer	
Project Start-Up	Drawing Package/Submission Track	2	3	1	2nd round	Stantec to write user narrative	
Design	Room Placer	2	2	Release 1 complete	request: support for choosing additional parameters in grid/keynames/custom parameters	tested dialog returns to front	
Collaboration	Consultant Model Update	2		1.5	2nd round		
Productivity: Utilities (drop down button)	Workplane Flipper	2	1	done			
Productivity	Graphic Scale Check & Update	2		2			
Design	Create Area Boundaries Based on F	2		2			
Project Start-Up	Sheet Set-up Copy (basic)	1	1	3			
Model Management	CAD Manager	1	1.5	Beta 1 complete		tested, new grid works, looks good	
	ager			Beta 1	request: select multiple and change group; indicate if parameter is		
Data Management	Sheet Page Number Update	1	1	Release 1 complete	request: handling of linked files		
Productivity: Utilities (drop down button)	Create Shared Parameters	2	1	Release 1 complete	Delivered to Stantec on April 30		
Productivity	Tag All	2		Releast 1 complete	request: tag all linked rooms (2014 o		
About	Help	1	1	waiting for URL		Primary URL sent	
Project Start-Up	Project Information	2					
Productivity	Tag All in All Views	3					
Collaboration	Revit Server Automation	3					
Collaboration	Revit Server Worksharing Monitor	3					
Model Management	Family Template Manager	2					
Project Start-Up	Family Load/Reload	3					
Project Start-Up	Copy Standard Details	3					
Productivity	Test String Search	3					
Productivity	Element Handi	3					
Desi		3					

In developing our roadmap we also had to take into consideration that our practitioners stretch across continental North America (and further) and that we have staff practicing engineering, architecture and interior design in just about every market segment available. Therefore it was important to make sure the tools would benefit a variety of end users and not overly benefit any particular segment.

Paying for It

Paying for things like custom tools is always the first question from anyone in management "great that sound fantastic; how much is "it" going to cost?" Furthermore when management asks that question, they are not actually asking for a number value (well they are), what they really need to understand is the intrinsic value it will bring to the business. Any decent business person will write a large check if they think the Return on Value from their investment will more than make up for the investment, the bigger question is, how large is the check going to be compared to the returned value, and how long will it take to regain that value?

The initial math is pretty easy, take the average rate of the employees, determine approximately how much time can be saved by the tool over the course of some unit of time (day, week, month, year) then divide to figure out how long it takes for the users using the tool to pay for the cost of the tool. After that time period the tool is theoretically returning value to the company and decreasing the "per person" cost of the tool.

The problem with this approach is that fundamentally it's quite hard to know for sure how much time someone is going to save so your estimates have to be conservative. Secondly, unless you are a reasonably small company it's hard to assure that the majority of your users will use the tool and at the end of the day the tool may not be directly beneficial to all employees.

The simple ROI calculation also fails to account for the in-tangible benefits, for instance how does the tool improve quality, avoid mistakes, errors or omissions, how does it allow employees to spend less time on low value tasks/labor and instead focus their attention on higher value tasks and labor that improve the overall project? Without a huge amount of performance metric data from before and after, these questions are almost impossible to calculate and answer. You just have to realize that these benefits do exist and there is often no good way to measure them, particularly at the point where you have no tool(s) in hand and you are holding out the hat in hand asking for money to build the tool(s) in the first place.

For these reasons, this is perhaps why there is a reasonably (for AEC) robust “app market”, where the development risk has been eliminated and instead you can go out and incrementally buy tools, show a positive result then continue the investment in purchasing more tools. The gap that must be spanned with leadership, is that sometimes it may be worth it take a similar investment approach, but instead build your own tools that do exactly what you need and that you have full control over.

As tough as it is to say, at this point leadership has to have a certain level of trust in you, not to say that you should not arm yourself with as many facts and figures as you can, a roadmap and value proposition are key, but at the end of the day you are asking your senior leadership to take some level of risk by making an investment with no absolute guarantees. This is one reason why we find it better to focus on Minimum Viable Product, we have earned enough trust from our leadership that we will deliver something, but we are not required to demonstrate up front exactly what we will deliver, simply that we will deliver tools that provide value to our “customers” over the course of one or two years of availability and access.

To fund our “basic” Stantec Utility Ribbon (as we refer to it) we were able to allot a portion of our BIM budget to development costs. Borrowing from the AGILE method, we set the original contract up as an eight week development period of twenty hours per week (our developer assured us we would have something in hand that would be usable within this time span/investment). We were also fairly certain we would extend the contract another eight weeks, but this allowed us to incrementally invest, get comfortable with our developer and see how far we got; if the whole plan had gone awry, we would not have been too deep in the hole.

The BIM funding is good seed money but we cannot do it all, nor do we think that the BIM group in Stantec should be in the habit of writing blank cheques for development, not to mention as pure overhead, getting more money is always a challenge. Therefore we are continuously looking out for projects that can help to directly support the development of additional tools. This type of work can come in all sorts of shapes and sizes.

For one project we had the developer spend a couple of hours creating a tool that would automatically create parameters in a project based on an Excel file, this easily saved the project team many hours of labor and the tool that was developed has not even been “released”, we have as an option the ability revisit it and clean it up to make it more usable in the future should we choose to, but the tool did its job and paid for itself.

For a second project, we once again had a manually intensive task, our developer looked into it, and while he could not completely automate the task due to limitations in the API, he was able to create a couple of small tools that helped to eliminate picks and clicks and therefore sped up the process from about ten minutes per file to closer to five minutes per file. Once again, these small “one-use” tools have not been included as part of our Utility Ribbon, but instead are located on our “beta” ribbon tab which we can enable on anyone’s machine should we choose to.

For even larger projects we have gone so far as to scope out full blown applications and not just utility tools that aid some simple production issues. In one case we spent several months developing a full blown analysis application for a large project that was required to determine more than a thousand different paths of travel. Once again, in reviewing the market there was no tool available that would meet our needs and the decision to develop our own was not just a strategic decision for the project itself, but for our work in general; with the expectation that the tool will continue to be used into the future.

Our last source of potential funding is our own internal Research Grant program where Stantec awards grant money based on an evaluation of grant applications to different “projects”. This is not a guaranteed source of funding, but it is something we can attempt to tap into to help fund projects that we feel are beneficial. To date we have funding for one development project related to Facilities Management and BIM and we are working to attempt to secure funding for a second effort.

Who Does the Work?

For us we were fortunate to come in contact with an experienced Revit API developer who also happens to have known Revit since Version 1. In our case we have not been able to justify bringing our developer on board as a permanent full time employee, and possibly that is for the best. Our BIM budget cannot support forty hours a week and there is not enough project funded work to fill the gaps. As an independent contractor our developer can work for us “as needed” for the most part, and we’ve developed a very solid relationship with him. As the results continue to show value we hope that we can continue to increasingly employee him more often.

The real issue is finding the folks who are talented enough to do the work. Many companies have users who dabbled in writing AutoCAD lisp routines or maybe even some simple Visual Basic routines for AutoCAD or Microsoft Office products. The reality is that the Revit API and Revit itself is a far more complex beast. You have to use a .Net language, either Visual Basic or C# (pronounced Sea-Sharp), C# is often preferred but effectively you can achieve the same results in both languages. The critical factor is that coding and developing for Revit is substantially “more” than working with AutoCAD lisp.

When looking for someone there are some key traits that you should be looking for:

- Experience with .Net languages, preferably C# and not just Visual Basic (VB)

- Experience with structured data and databases. They do not necessarily need a lot of database experience (it depends on your goals) but at its heart Revit is a database (not a perfect one) so having a developer that has some understanding of databases and structured data is beneficial.
- Experience with Revit, makes it harder to find a developer, but certainly a plus.
    - If no experience in Revit, the attitude and personality that lends itself to learning about something new from an experienced Revit user on your staff.
- Experience in the building industry, also limits the fields of candidates, but the same caveat as Revit experience applies. Someone who demonstrates a high level of intelligence and willingness to learn, may be a better candidate then someone who has worked in the industry for multiple years.
- Experience developing for other data driven applications, particularly data driven applications that also use geometry. Note, that while you can do a lot with AutoCAD, in general I don't think AutoCAD fits this description, tools like Inventor, Solid Works, Pro Engineer, etc. are all examples of data drive applications that also use geometry.
- Demonstrates problem solving skills and creative thinking. Sometimes there are limitations in the Revit API, but sometimes some creative thinking will help you solve the problem regardless of the limitations (other times its not worth it). Having a developer who can approach a problem from multiple angles and give you good advice/feedback helps to improve the process.

The fact is, it is hard to find candidates that fit all these criteria, and some of it will depend on what your goals for development are. If you can't find a specific person to hire, then there are multiple resellers and consulting companies that can provide the same services. Whom you hire and how is dependent on what you want to gain from the process, the level of competency of your staff and how much control you want to exert over the process.

The other choice is to grow talent in house. This will take time, but if you have a staff member with the right aptitude, then it might be better to give them the time and resources to expand their knowledge and help your firm. There are resources available for getting started with the Revit API, a particularly good one is the blog "Boost Your BiM" which is focused on the beginner, "part time" coder.

If you do go out of house for development, there are a few items to consider in negotiating the business relationship:

- Who "owns" your software? Ownership of software and intellectual property in the context of software development continues to be a very grey area, where multi-million dollar corporations battle it out in the courts. Make sure your agreements have some explicit language in this regard.
- In this context you may also want to think about the source code. I suggest including in your agreement that the developer must also provide the source code at your request, once again a

bit of a grey area particularly given the nature of being an external consultant that focuses on software development for multiple clients.

- How will you handle updates for new versions? Not to suggest that your developer should update the tools for free, but what is the internal plan, do you have a budget for it? Updating tools from version to version is typically not time consuming or expensive, particularly if you do not want to attempt to leverage new features, but it still must be done and planned for.
- With regards to updating, will your developer be around? Will they have the time, how many clients are they servicing, and how busy are they, what is your typical deployment schedule for a new version compared to release date?

Justifying the Cost *(delivering value).*

Earlier we discussed the justification to develop your own tool(s) because the equivalent tool that you could buy would cost more than it would to develop your own. However, there are other ways to justify development costs. For our custom ribbon the key business value we presented to leadership was increased productivity. Given our size and number of users if we were to theoretically create a tool that could save each user five minutes each week, it would have a huge impact on the bottom line. If we assume an average rate of $75 / hour, and we assume 300 users, the five minutes per week calculates to a labor savings of just shy of $100,000 per year ($97,500 to be exact).

- Will that translate to $100,000 in more profit? No.
- Does it mean that with some luck at least $50,000 worth of labor went to more useful tasks? Yes.
- Does $100,000 of annual savings more than pay for our $50,000 investment in one year? Yes it does.
- Is that a highly theoretical ROI calculation? Yes it is, but it sets a very low bar in terms of achieving a good return on investment.

We know that some of the tools we have developed will save some users significantly more time than five minutes per week and other tools will save more users shorter periods of time in a week.

The challenge for smaller companies with fewer users is that the math gets much harder. Effectively development is a set cost; a developer would likely be unable to develop the same series of tools for less than the money we have spent. A smaller firm does have an advantage in an easier adoption curve where you can make sure the users use the tools, thereby justifying the investment. A smaller firm may also choose to develop less and simply get more return on what you do develop by way of increased adoption rates. As a large firm our difficulty is that we cannot assure adoption, the best we can do is develop tools that are easy to use and hopefully are easier to adopt then doing the same thing in a different way in Revit. This also has the effective benefit of forced adoption of “standards” since some of our tools are dependent on how Project Templates are set-up and configured. Adoption of best practices also results in time and cost savings that help to improve our performance and quality, thereby compounding the investment in the first place.

In Conclusion

There are many factors that can influence a firm's decision to invest in the customization of tools for platforms such as Revit - firm size, funding, available tools on the market, return on investment, etc. Defining your goals and developing a strategy that focuses on them is paramount. Before venturing down the API path ask yourself a few simple questions. Do you want to improve efficiency, and how much do you stand to gain? Do you want to enforce standards and best practices, and what is the benefit? Do you need to improve quality? Do you want to find innovative solutions that can help to maintain (or create) a competitive advantage? Are you merely staying one step ahead of the software development cycle? Do you want to get into the software business and market your own tools? How you answer and rank your responses can help you develop a roadmap that is commensurate with your goals (and budget).