PharmaSUG 2015 - Paper MS03

How to Harness Your Company's Wiki for World Domination (or — Even Better — to Write the Best Reusable Code Possible)

Kurtis Cowman, PRA Health Sciences, Lenexa, Kansas

ABSTRACT

How can a group of laymen be smarter than the experts? From betting markets to Wikipedia, every day we see real-world examples of how large groups that are properly motivated and regulated can accomplish nearly impossible tasks. On "Who Wants to Be a Millionaire?" why was the "Ask the Crowd" option correct statistically more often than the "Ask an Expert" option? Daily we ask only our most senior staff, our experts, to design macros and reusable code without properly evaluating whether their effort is indeed the most successful.

This paper will show you how, by utilizing a company-accessible web application such as a Wiki, with proper management guidance, you can harness the knowledge of your global programming staff from top to bottom — with the end result of quite literally writing the best reusable code your company has to offer.

INTRODUCTION

The idea of a good macro is simple. Take code that we use over and over and turn it into a reusable program. Enclose the code in a simple SAS command of %macro and %mend, exchange a few variable names for general macro variables, and there you have it: a macro. However, in practice this is an entirely different exercise. Day after day we are plagued with the same issues. Does the macro correctly address the issues we have? Does it work in every program we need it to? Has someone changed the code along the way rendering it incompatible with older programs? Could it have been better but is now too late to change?

Wouldn't it be great if we could harness those years of hands-on knowledge obtained by actual program users to build a better macro before we even release it? In some ways you can. In a similar way to how the auto industry uses specific engineering tests to simulate hundreds of thousands of miles of wear and tear on a car we can do the same for macros. Using tools readily available to all companies, we can brainstorm, write code, test, and troubleshoot our macros.

The aim of this paper will be to explain how to gather information about what macros are needed, begin the programming framework, share the programming load among a global audience, vet program changes, test the early macro, and finally release your creation to the team. Using a simple wiki platform and conventional wisdom built on the study of psychology, you will understand more clearly the process involved in accomplishing these tasks.

WIKI

The idea of open source programming has been around for decades now. Even before it was given a name, the theory existed. Take all the great minds in a field, put them together, give them a problem, and see what happens. Generally this works out very well. But, in terms of business, the open source idea can be dangerous. How does one properly safeguard this knowledge? How does one go about paying for the knowledge received? Better still, how does one profit from knowledge that by its very nature is open to everyone?

The idea of open source programming can be slimmed down though. This idea that you can harness the knowledge of a group to solve issues that just one person can't is still valuable. So what lies at the heart of the open source process? You need several things. First you need a group of willing participants. Second, you need a problem to solve. Third, you need a place to facilitate the sharing of ideas. As long as your company has more than one employee and is willing to invest in some type of wiki platform, you're ready to begin. All you need is the problem to solve.

In 1994, Howard G. "Ward" Cunningham created software called WikiWikiWeb. As the inventor of the modern wiki software, Cunningham's original intention was to create a shared space for programmers to exchange ideas. One of the requirements of the software was it needed to be quickly and easily editable. After 20 years of use in many different forms, it's not a huge leap to conclude that he succeeded.

Knowing that Cunningham's original purpose was to have software where programmers could share ideas, we can build on this idea for our own purposes. Using a wiki platform along with general psychological knowledge we can begin to piece together a process for eliminating the guess work from much of macro design. Many different wiki platforms are available on the market, so the benefits of any particular software will not be discussed within this paper. However, we will note two important features that should be included in a wiki platform designed for coding.

First, and arguably most important, is the existence of an easy function for posting and displaying code. To make it as easy as possible for programmers to share code, the platform should ideally support a cut and paste feature to add code to a wiki page without the necessity to reformat. Any type of reformatting on either side (adding code to the page or copying code from the page) will increase the time it takes to make updates to the wiki and potentially deter programmers from participating in the task of completing open source work.

Secondly the ability to set administrator rights can prove beneficial as well. For the purposes of this paper, it is better to have all pages open to any participant; however, it may be helpful to have a way to restrict the write access to a page once all work has been completed. This can also be useful in restricting the editing ability of the macro requirements and criteria once they have been decided on by management.

CROWD WISDOM

Knowing that getting a wiki up and running should be simple task, the attention can now turn to the process that will be used to create the macro. Conventionally, a macro is a solution to a problem. The problem is first identified by either an employee or a manager, and then the decision to create a macro is made. This decision is usually financial and is a driven by an analysis of time to create a macro versus time the macro will save. If the macro is efficient and robust, then the amount of time it will save — and in turn the money it will save — justify a greater amount of development overhead. Using this equation, we can see how taking the time to create a more robust code could lead to greater efficiencies over its full lifecycle.

In practice, many macros developed by a person or small team lack the perspective to be truly robust enough to use across all circumstances. Retrospectively one usually finds that the creator was focused on their own problem rather than the problems that all programmers might face. This cannot be viewed as a failing of the programmer simply for the fact he or she may not have the perspective to see all issues from their singular vantage point. The solution, then, is to create a better vantage point in order to increase the global effectiveness of our code.

The old solution to increase the height of the vantage point was to appoint a lead or manager to oversee the creation based on a supposedly broader or higher-level understanding of the problem. Similarly, in medieval times having a lookout in a tower was one of the most effective ways to survey the battlefield. However, in today's society, with hundreds of satellites roaming the skies and the area of the battlefield expanded to cover the entire world, the idea of just one spotter in a tower becomes just what it was: medieval.

Working with the assumption that having hundreds of satellites around the globe is the best way to survey any point on earth, how do we convert this idea to a something applicable in a business atmosphere? Assuming that every programmer has a good vantage point on the work they are doing, one can now conclude that, by involving every programmer in your company, you can in turn survey the entire landscape of your business globe in its entirety.

The next question becomes how can we successfully gather the wisdom of our collective knowledge? Here we turn to the wiki to do this work. But just having a place to post our information isn't enough. How do we know that the information we will receive is pertinent and correct? How do we know that those that choose to speak out will have the best conclusion? Finally, how do we know when the solution has been reached?

In his book "Wisdom of Crowds," James Surowiecki covers the idea of using wisdom sourced by a collective. Surowiecki summarizes that, to utilize a collective to its full potential, you must have four things:

- **Diversity of Opinion:** Each person should have private information and experiences that help them interpret the facts.
- **Independence:** Each individual's opinion is not determined by the opinions of those around them.
- **Decentralization:** Each person should be able to use their own knowledge and not a regurgitation of the knowledge of others
- **Aggregation:** A mechanism to compile all opinions into a collective solution.

The first three items Surowiecki mentions are subjective but can be assumed if certain criteria are met. First, assuming a work environment that is collaborative in nature and allows for differing opinions, one can assume that most humans do, in fact, have opinions. Getting them to share effectively is usually the common problem. Second, and again assuming a collaborative environment, as long as there is not one employee or group of employees that have for some reason banded together either by human nature or undue influence, one can assume independence. Last, we need decentralization. In a common corporate structure where training is given by experienced employees to new employees, it can be tempting to simply regurgitate information that has been handed down rather than by researching innovative new solutions.

Careful attention must be made to bolster an atmosphere where decentralization can thrive. This does not mean that none of your employees can reside in the same building. Rather, in this case we would think of decentralization in

respect to the mind. For example, if your company has an individual who is naturally able to influence people into their way of thinking or you have a group of people who have banded together despite the nature of the issue, then the idea of decentralization could be lost. One regularly sees this in mob mentality. As a group people tend to make poor decisions to loot and riot whereas individually the same people would not have behaved this way on their own. Another way this decentralization could be lost is if people tend to rely on their training instead of thinking through an issue fully. In the corporate setting where standard operating procedures and work instruction are the norm it is easy for employees to fault their rigid training as a reason to think alike. If everyone is trained in the same way then everyone naturally starts to think the same way. Having an atmosphere where there are rules but questioning them politely and logically is approved of can help keep decentralization.

What is needed is an atmosphere where all employees, be it those on their first day or their thousandth, associate or manager, must be able to relay their opinion and have it treated with the same weight as anyone else's. This is not to say that you should rank all employees the same and have a system void of hierarchy, but rather that, for the scope of the defined problem, all opinions should be given the same chance for evaluation.

With the correct environment in place, we are left only with the need for a method to aggregate the ideas. This is where the setup of the wiki comes into play. Given that the original purpose of Cunningham's wiki software is to share ideas between programmers, we already have an ideal tool for the job.

The idea of the wisdom of a collective is not a new one. As stated earlier, there have been many examples where a collective has been able to make predictions and solve problems quicker than or better than experts. However, this is not to say the system does not have its flaws.

The theory of collective wisdom is that, unlike the experts that generally have agreed on one best way to do things through years of trial and error, a collective will have to pool their knowledge. Some of the collective may know the answer as would the experts, others may only know what the answer is not, and yet still others may only be taking an educated guess as to the likeliest of outcomes. Generally, this results in an aggregate answer that is correct, but it is possible for this aggregate can be shifted by a corrupted environment. Imagine how easy it would be for those that thought they knew the answer to influence those that were ignorant to the particular topic. Whether they were correct or not, you have now lost the independence that is needed for this method to hit its peak effectiveness.

For this reason, it is vital that care is taken to both create and maintain an environment that emphasizes the four characteristics defined by Surowiecki.

DO WE HAVE A PROBLEM?

We now have the wiki software to support our aggregation and the work environment to allow our staff to get to the task of attacking the problem. But before we can do that, we have to know what the problem is. In the programming field we constantly run into problems that are outside of our knowledge base. However, just because we do not know how to resolve this issue does not necessarily define a true problem.

By first running our problem through the collective we can properly purge issues that can be addressed by those with more knowledge than ourselves or those in our close circle. But what about those problems that remain at large, items that other programmers are not able to address either? Sometimes posting these issues reinforces the fact that there is indeed a problem.

So what do we do when we in fact find out there is an issue? Surely we can't be expected to let people resolve the issue amongst themselves? Indeed we cannot. Remembering that the true function of any reusable code is to increase efficiency, wasting time searching for the answer without the problem being properly defined would lower the likelihood of achieving this efficiency.

In his article, "Digital Maoism: The Hazards of the New Online Collectivism," Jaron Lanier discusses his sociological theory behind collective thinking. "The collective is more likely to be smart when it isn't defining its own questions, when the goodness of the answer can be evaluated by a simple result (such as a single numeric value), and when the information system which informs the collective is filtered by a quality control mechanism that relies on individuals to a high degree. Under those circumstances, a collective can be smarter than a person."

Using Lanier's observations we can again examine the three points. The first issue occurs when the crowd defines its own questions. Indeed, we see evidence every day where, given infinite choices, groups are unable to effectively decide on an answer. Take the daily act of eating lunch. A group of people in an office who have decided to eat lunch together may not be able to decide on where to go to lunch when posed with a broad question like, "Where does everyone want to go for lunch?" But narrow that question down to "I am going to Bob's for pizza; who wants to join me?" and you have now made the process of deciding quick and easy.

By limiting the brain's need to access an entire list of food choices down to just deciding on one place, you have effectively made the decision process exponentially faster and more efficient. Before you were asking a person to first

access their brain to decide what they were hungry for and then where they wanted to go to get that food. Then there is the ever-present task of figuring out if someone else will like the same place and if not where are other places each of you would deem acceptable. By limiting the choices, you have now narrowed the decision down to whether or not one wants to go to Bob's Pizza. Even those who have not eaten at Bob's Pizza should be able to make the decision by weighing their like or dislike of pizza itself.

So too should we limit the definition of our problem. By saying, "What problem should we attack?" versus "Let's attack this problem," you risk overcomplicating the decision tree for your audience. Therefore, if we properly define what the problem is and what we are hoping to solve, even if the definition changes through the process, we have more effectively utilized the brain power of our collective.

The second of Lanier's observations pinpointed the necessity of evaluating the quality of the answers. This can be a difficult task but one that is not impossible. As we have discussed, the goal of any reusable code is to maximize efficiency. Therefore, it is possible that evaluating the quality of the answer is as simple as, can you improve on the time it takes to do this task? Something as simple as automating a folder setup process would fall under this category. But, maybe the problem is harder to define. What if the task is to improve output quality? How does one measure this?

The answer to more subjective measurements falls on the shoulders of management to guide its staff to what they believe is the best result. For some problems, it may be as simple as looking at processing speed. For others, it may get as complicated as looking through outputs to see which answer produces the best and most reliable results. Or it could be as simple as putting a deadline on the task completion. For example: "What is the best answer we can come up with by Tuesday to use for an audit on Wednesday?" No matter what the criteria used to evaluate the answer, the fact is that criteria must indeed exist.

Last on Lanier's list was the need for a filtering system for the information gained. This is the simplest of the tasks he discusses. Almost all companies will have work instructions and standard operating procedures. Using these as a guide, no matter what information system employees pull from, be it other programmers inside or outside the company, the internet, or many of the valuable resources at our industry conferences, it will be properly filtered by the needs and restrictions of the company.

GETTING TO THE CORRECT ANSWER

So we have now set up a wiki, enhanced our work environment, learned how to derive and define the problem, and evaluate the solution, but how do we get the collective to generate resolutions? This becomes an even bigger issue. Questions will arise like, "How do we drive traffic to the wiki site for this problem? How do we keep the collective from bickering and losing site of the task? And how do we know the best answer will rise to the top?"

Jim Worley covers some of these items in his paper "Managing and Sharing Information through the Use of a Wiki." Worley states early on that, "The biggest fear is often the ability for misinformation to be posted and spread though this can be reduced or eliminated." Worley's stance on reducing this misinformation is to enact and enforce solid user policy. His suggestions include having users log in to post, establishing boundaries for the platform's use, having policies on appropriate content and interactions, as well as planning for how users will correct faulty information.

In fact Worley goes as far when speaking about user interactions as to say, "Laying out a plan for users to correct faulty information will help encourage them to do so without feeling like they are messing with someone else's article." Two issues exist with Worley's stance though. One, is there is an assumption made that any article belongs to any one person. In practice, all articles belong to the wiki as a whole. The only ownership would be to the company that hosts the wiki platform. Second, the purpose of a wiki is to share information, so the fact that any plan is needed for a change to be made deviates from the nature of a wiki entirely. One should expect when they post an article that changes can and will be made.

But what stops someone from making an incorrect change? The existence of a social Darwinian process nicknamed "Darwikinism" is noted in the article "Wikis, Blogs, and Podcasts: a new generation of Web-based tools for the virtual collaborative clinical practice and education" by authors Maged N. Kamel Boulos, Inocencio Maramba, and Steve Wheeler. In the article, the authors surmise, "'Unfit' sentences and sections are ruthlessly culled, edited and replaced if they are not considered 'fit', which hopefully results in the evolution of a higher quality and more relevant page. Whilst such openness may invite 'vandalism' and the posting of untrue information, this same openness also makes it possible to rapidly correct and restore a 'quality' wiki page."

So what is found in practice is that the need for correctness overwhelms the need of a few to post errant information. In fact, a pleasant side effect of the ease of editing a wiki and the fact that most wiki software now tracks the last user to make an update is that you can actually discuss with someone why they posted errant information. It may turn out that the individual in question was just misinformed or that what was once thought to be errant by most has now become correct and the user is just the first to recognize it. This can commonly be the case in the pharmaceutical industry where constantly changing standards are the norm. It has actually been observed that in some cases the

overall need to be correct has put the quality of some Wikipedia pages near in quality to that of the Encyclopedia Britannica.

Due to the nature of a business wiki, though, some controls are of course needed. However, conventional wisdom that more controls are better may not apply in this case. One could hypothesize that a good way to ensure the quality of information is to restrict access to wiki pages and employ a tactic of utilizing a large number of administrators. Camille Roth, Dario Taraborelli, and Nigel Gilbert, authors of the paper “Measuring Wiki Viability: An empirical assessment of the social dynamics of a large sample of wikis,” found in their research that the reverse may actually be true.

Roth, Taraborelli, and Gilbert found, after looking at data surrounding wikis, that having a large number of administrators in relation to users actually stunted the growth of both users and pages. It was also found that having no access controls to the editing of pages actually favored growth in both pages and users. None of this research mentioned the accuracy of the information contained within; however, using the key principles proposed by Surowiecki one could make the leap that the increase in users actually increases the accuracy of the information. Additionally, interestingly, page growth also increased with less control indicating that users were more apt to use the wiki interactively and share a broader range of information when controls were loosened.

PUTTING THE PLAN INTO ACTION

Now that we have looked at all the conventional wisdom on how using a wiki to build better programs should work, let’s examine the actual execution. As with many things in the business world, there are many ways to achieve the same goal. The same is true for using a wiki to build better programs. You will find with using this method that, as the programming progresses, the methods and criteria will change. So to should the method by which you resolve them. Before judging the method too hastily you must give the method time to grow. As Rome was not built in a day, neither can a good wiki. In practice, building a robust wiki and user base that is needed to produce the outcome of this method can take many years. You will find though that with these many years will come knowledge that will advance all of your programmers tremendously.

Defining a problem

Let’s start at the beginning. How do you identify a problem? As with the conventional method for writing a macro, you may already have some problems in mind. This is a good start but, as shown earlier, your problem may actually be unique, and someone else may already have a solution. The best way to attack this problem is to begin a wiki page with a list of issues.

By pointing your entire staff to this page, you can now begin the process of culling what is a real problem and what has already been solved by others. The instructions to your staff should be to address each problem with tact. Even though the problem may seem trivial to some, others may have encountered this problem previously and been afraid to speak out. As with anything business-related, the person you respect most may be the one you are choosing to unintentionally mock.

If programmers are able to effectively resolve issues, have them start a wiki page devoted to just that one problem. Have both the original poster and the person who offered the resolution work together so that both the problem and the solution are properly defined. Make sure to include searchable terms and links to pages that reference methods you used in resolving the issue. If your problem is directly related to a method discussed in another wiki page make sure to link to your page from that page so that people know you have resolved some issues that may come up. Remember that sometimes staff may be reading the site specifically to resolve the same issue you are having and not just informing themselves before using that method.

Once you have reached a consensus on what problems have not already been solved, you will now have a list of prospective candidates for reusable code. Remember that the intention is to make programmers more efficient, so evaluate the problem based on whether creating code will actually create efficiency. Some one-off problems will need to be resolved manually, and the act of writing code may actually take more time and therefore be less efficient.

Defining code criteria

The next step in the process, once you have decided which problem to expend resources on, is to decide what criteria will drive the code. If you are creating code to solve an issue with dates for example it may be easy to get lost in the idea of writing a macro to resolve every date possible and derive missing and partial dates as well. However, using the idea of object-oriented programming, you may actually be altering your efficiency equation by creating a macro that is too robust. If 99 percent of the users are just going to use the macro to join separate day, month, and year variables, is the extra time needed to create code to derive missing dates really necessary?

Defining the criteria to which the code should adhere can be a lengthy and difficult task. One must take the time to examine all possible uses and determine what is really necessary. This brings us to the problem highlighted earlier by

Lanier. We must not leave it up to the collective to make these decisions as there is no one correct answer. Rather, a leader or manager should be the one that makes the final decision based on what is most prudent for the company at any given moment.

Once you have decided on the criteria, release it to a wider audience by starting a wiki page devoted to the issue. Make sure to link it from your problem's brainstorming page so that staff can use the main page as a landing page. This will allow them one central place to see what problems are out there that they may have the knowledge to assist with. Also, make sure as a courtesy to your audience to include reasoning behind the criteria. Knowing why you chose the criteria you did will help them make their own decision as to whether the knowledge they have will be helpful. It will also give them guidance when they stumble upon problems and need to use their independent judgment to resolve an issue.

Be wary but open to challenges to your criteria. Remember the entire purpose behind sourcing the collective to solve your problem is to find the best answer. You may not be asking the best question though. Make sure that you have a strong opinion of what the criteria are trying to accomplish. This is one time that letting the crowd steer you may not be the most efficient. If the consensus seems to be that you are asking the wrong question, then maybe taking your idea back to the drawing board may be best. Keep in mind that the goal is to solve a problem not to be the one with the correct answer.

Before setting the collective to task it will also be helpful to determine ahead of time how the best answer will be judged. Letting the collective know that processing power, fewest log issues, successful run of the code on extremely errant data, or some other criteria will be the deciding factor in how the collective can help. In order to self-regulate the collective will need to know how the best answer should be evaluated and by whom. If processing time is chosen, simply turning on the option to print the processing time in a log could help the collective regulate themselves. Programmers can simply post logs showing that their code ran faster than the previous version. Every situation will be different, but be sure to release these criteria before work has begun.

Using the collective to write and release code

Now that you have a problem and criteria to drive the resolution of that problem, it's time for the collective to work their magic. Again, sticking to Lanier's observation, providing a base code for others to springboard from can be very helpful. In programming especially, there are many different paths to the same location. Some programmers prefer arrays while some prefer transpose functions. Neither path is more correct and in some cases can be very equivalent. To keep the efficiency of the collective high and remove any need for overthinking of problems, it can be helpful to release an initial version of the code. This can be written by yourself or your standard macro team but it should stay within the criteria you have released. Remember, it is easier for people to find where they are going once you give them a map.

This will lead some to believe that this method simply piggybacks on the efficiency of having one group of highly skilled programmers writing code. But going back to the example of the medieval tower, remember that the collective will be able to produce a better answer than just one programmer. Giving the collective a head start will make the process more effective. As shown with Lanier's observations it would be much easier for a collective to say what code does not work or is inefficient and replace it with one that is rather than creating a suitable code in the first place. Given the infinite ways a code problem can be solved, remember it is easier to decide if you like Bob's Pizza than it is to decide where to go for lunch.

Utilization of a core macro team will also need to change when using this process. In the past, the core team may have struggled with some ideas and instead of reaching out to the collective may have burned many hours researching bits of code to resolve their issue, or worse yet scrapped code entirely to start over with a different method. This burned time does not need to be used with this method as the collective will naturally take its place. The core group must now know when it has reached its limitations and the edge of its capacity for efficiency. When to turn code over to the collective can be a hard decision to reach and one that may need input from leadership or management. Remember that the goal is to be more efficient not for the glory of solving the problem on your own.

Administration of the wiki page

Once you have created a page for your problem, posted the criteria and base code, and the collective has begun work on your page, what do you do? Now is the time to monitor the progress of the collective. You may find it helpful to have anyone that makes a change to a page log their name, changes made, and hours spent, especially if your wiki software does not already track this. Having this information will help you monitor the success of the page. Using a discussion topic area on the page is also recommended. This would be a place for programmers to post their questions or observations. These could be as simple as to compliment people on how they resolved a certain issue or as complicated as requesting rationale for using one bit of code over another.

Keep in mind this is still a business environment, and employees should handle themselves as such. To grow your wiki, employees should feel as though they are able to question their superiors but this must be done respectfully and

in a work-appropriate manner. You may find that knowledge will come from very unlikely places. Conventional wisdom may have been leading you astray, and, given the chance to express an opinion, you may find that a better way does exist. Remember for centuries people knew the world was flat and they were deemed the experts.

Concluding the work

Timing of when to end the collective work can be difficult. In some cases finding any solution may be the resolve you were looking for. However, generally knowing when you have reached the most efficient solution will be a lot like cooking microwave popcorn. By the time you hear no more pops your budget and efficiency is all burnt up. Knowing when progress has slowed to the point that your balance of efficiency has not been lost will take time and practice. Be sure to give the first few tries their due course. Better to lose efficiency and prove that the method may have merit than to end the experiment too early. Remember, if you end too early, you will still be left with a problem unsolved, and any time you let your programmers spent will be wasted hours.

If you have administrator rights, now would be the time that it would actually be appropriate to lock the page. This will exhibit to the collective that an acceptable solution has been reached. Remember, you still have a problem main page and if the solution you reached is not sufficient, staff can always utilize that open page to post a follow-up problem. You may find that the follow-up problem does not match the criteria of the original problem and can be treated separately. This new problem may need to be resolved manually as opposed to burning efficiency on an already accepted resolution.

Be sure to thank all contributors to the solution and add a link to where the code can be found if it is residing in your analysis system before locking the page. Many times you will find that the best compensation to employees is merely highlighting that someone was integral to resolving an issue. This will help to drive traffic to other problem pages. Also, make sure to let staff know that a solution has been reached so that they can implement the new code as needed. Nothing is more frustrating than finding out a solution to your problem has existed all along.

The lifecycle of a mature product

As discussed earlier, once you have reached a resolution, there will be a need to re-evaluate whether the solution is working. There will be some cases where your code is not all-encompassing. Problems will exist that will need to be resolved manually. This cannot be avoided, but remember you are going for the most efficient solution and not the all-encompassing solution. If a small number of users have to resolve their problems manually, this may still be more efficient than spending more time writing code.

Make sure that any re-evaluation adheres to the original criteria. There was a reason that the criteria existed in the first place and abandoning it can be costly. If you find that in most cases your solution is not working, you may not have correctly specified the criteria by which the code was written. If this is the case, it may be worth opening a new page and treating this as a new and separate problem with updated criteria.

Most of all, remember the first line of any follow-up criteria should be that the code is nondestructive. If the updated macro uses the same name as the original macro, all calls to the original macro should function the same when calling the new macro. This will guarantee that any updated version can be used in place of older version without the need to change any other code programs. If you can't adhere to these criteria consider changing the name of the macro so that it can be used to this point forward and the use of the original version can continue.

CONCLUSION

You have now seen ways to properly evaluate the need for reusable code, harness the knowledge of a collective by utilizing a wiki platform, and implement your newly found code. You also know some of the pitfalls that can entrap you, from letting the crowd do too much to delaying a decision on when enough is enough. But utilizing this newfound knowledge should pay dividends in the efficiency that your programmers can now identify and resolve issues that could have become black holes for departmental budgets.

Remember that this newfound knowledge can be scary to others. Not everyone is going to be jumping at the chance to invest company funds in a wiki platform and expend time and energy it takes to launch it. Remind them that science and research is on your side. Studies have proven that crowds can be wiser than even the top experts, and efficiencies can be gained by taking advantage of the staff you already have.

REFERENCES

Boulos, Maged N Kamel, Maramba, Inocencio, and Wheeler, Steve. "Wikis, Blogs and Podcasts: A New Generation of Web-based Tools for Virtual Collaborative Clinical Practice and Education." Biomedcentral.com. August 15th, 2006. Available at http://www.biomedcentral.com/1472-69220/6/41/

Cunningham, Howard G. "Front Page." C2.com. December 23rd, 2014. Available at http://c2.com/cgi/wiki.

Gilbert, Nigel, Roth Camille, and Taraborelli, Dario. “Measuring Wiki Viability: An Empirical Assessment of the Social Dynamics of a Large Sample of Wikis.” Nitens.org. September 8, 2008. Available at http://nitens.org/docs/wikidyn.pdf

Lanier, Jaron. “Digital Maoism: The Hazards of the New Online Collectivism.” Edge.org. May 29th 2006. Available at http://edge.org/conversation/digital-maoism-the-hazards-of-the-new-online-collectivism

Surowiecki, James. 2005. The Wisdom of Crowds. pp xv. New York, United States; Anchor.

Worley, Jim. 2010. “Managing and Sharing Information through the Use of a Wiki” PharmaSUG 2010. Chapel Hill, NC: PharmaSUG. Available at http://www.lexjansen.com/pharmasug/2010/MA/MA05.pdf

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Kurtis B. Cowman
PRA Health Sciences
9755 Ridge Drive
Lenexa, KS 66062
913-410-2271
cowmankurtis@prahs.com