Listening to the Voice of the Customer when Deploying Your Application: Using SAS and Design Methodologies to Create a Pleasing User Installation Experience

Joe Perry, Perry & Associates Consulting, Oceanside, CA

ABSTRACT

We have all had to install software from a simple package like WinZip® or something more complex like SAS® and these installations create a lasting first impression about the quality of the software vendor's product and the competency of their coding. Work for a large company planning an internal deployment or a small company deploying your software at multiple client sites you have limited opportunities to create a good impression and continue the business relationships you have made. The fact is that whether you like it or not, just like meeting other people first impressions count and the installation and first use of your software is your opportunity to make that impression a good one. This paper explores how to think like a naïve SAS user and not like a programmer while planning your next deployment of your SAS-based software applications. This paper is intended for users of moderate to advanced SAS skills.

BACKGROUND

A story, not of SAS but; you'll get the picture later… You would like to use some new communication technologies available through the web and friends have recommended that you try Skype. So, you download Skype, install it, register a new account with a distinct password (carefully typing it into a separate document) and, typing twice into the designated space on the website, you are good to go, simple right? Now, log into this new account with that password still sitting in that document you have open and… usually, the whole process works without a glitch but somehow… somehow, the password is inexplicably wrong.

You try to log in again and… the Skype window tells you that the account/password combination is incorrect. Now, you are not amused and you are getting annoyed because, well, the password is sitting right *there*! Worse yet, you know that if you try calling Skype, there is a good chance that you will probably be spoken to like a third grader because, as everyone knows it *must* be operator error. Really, Skype has millions of users and if this were really a problem in the Skype application they couldn't possibly have that many users so, a system with a problem as fundamental as inability to get an account set up can't possibly exist, right?

But, you call anyway, they *do* talk to you like a third grader but, hey, at least they don't tell you that you have to reload your operating system and cause you to lose all of the data on your hard disk. You swallow your pride, listen to the lecture, admit that maybe, just maybe you might have typed the password wrong (whilst you are thinking that THAT is about as likely as you being hit by a meteorite in the next second) but, hope springs eternal and you dutifully listen to the technical support advice and promise yourself that you'll try this… once!

So, you *are* told to try changing your password; you follow the directions on the site and the system sends you a temporary new password, you log back onto the Skype site and go with a 'new' password, one that you can really remember but, hey, why not take that same password you have in your desktop document, re-enter it, verify it and now you know that this will work. You are now thinking to yourself that you are really sure that the password in that silly document is the one that the system took right? it must be, isn't it? You typed it twice, it looked just like the one in the document so, memory, and your age, couldn't cause a problem.

So, the system tells you the password has been changed and you log off the website and start a Skype session on your desktop; logging in with the password that was entered into the website, twice, you carefully type the password in with two fingers so you don't make a mistake and… dang, the password is once again rejected! You want to call up the technical support site again but you figure if they can't even get the password part of the application right then there's probably other functions in the application that will be as poorly designed as this one so, you might as well forget Skype and try some other application.

After all, your time is valuable, something (you think) that a $10/hour tech support person doesn't quite get and, even though Skype is free, free isn't worth it if it does not work as advertised, right? Now, given that there is free enterprise system and there are a lot of smart, ambitious programmers out there, there's a very good chance that there'll be other, similar applications to Skype and you might as well try one of those because (hopefully) someone appreciates the client's point of view so, you just figure 'next' and move on!

Back to SAS-based development… Change the name of the application from Skype to one that you have designed or have work on… change the price from free to a higher figure, an amount that will ensure the beginnings of financial

security for you and your family and the above is a pretty likely example you might have a part in, i.e. a 'dream' software package's installation and first use[1] cycle nightmare; a nightmare that will stop your dreams in their tracks. It may seem unreal to you, as an *application programmer,* but it is a nightmare that is all too familiar to the naïve *users* of software, including some that you might have had a hand in developing.

THE MORAL OF THE STORY: USER-CENTRIC MINDEDNESS

The moral of the above story is: users and programmers of software often have different perspectives on performing their jobs. Users don't want technical challenges when they use the software – they just want to *use* the software; programmer-type people (i.e. those that: develop, install and setup software packages, read the 'Skype' programmers) accept technical challenges as a necessary part of their work. The 'technical' people tolerate installation glitches but; these 'glitches' are not tolerated by the users because they are seen as, quite possibly, a reflection of the applications' quality. Users have a different view of software than the 'normal' view that the programmer-types have of the same issues.

The take away from this story is this: To get the greatest use of yours or any application: system engineers, developers and programmers need to change their frame of reference from a developer-centric to a user-centric frame of reference.

Some products ***are*** clunky and difficult to use but, though difficult, are used anyway because of business requirements and/or some other external rationale; but they are not by free choice and alternative products are constantly being sought out and undesirable performance characteristics are reason enough to find a replacement.

The truth seems to be that, only when an application is: developed, produced and supported to and with user- or customer- centric systems will users get products that they will really *want* to use rather than products that they *have* to use… only then will your product have a real chance in the free marketplace.

BECOMING USER-CENTRIC: LESSONS FROM THE SERVICE INDUSTRY

In the example above, assuming that a real error existed in the password sub-system, it can be reasonably argued that the technical support system had a bias *against the user* and in favor of the system design and programming.

This bears repeating: ***Many, if not most, technical software support systems have a bias against the user and in favor of the system design and programming organizations***!

This fact stands in sharp contrast to the claims of many companies that they 'care' about their customers yet, given a reasonable person's interpretation of a bias towards the technical organization, that claim must be taken with a grain of salt. The fact is, the customer always loses when a company's user support structure favors the company's product delivery systems over the customer, if a bias exists in the systems those biases must be in favor of the customer, not the company.

It should be clear that being or becoming user- or customer- centric requires more than just words and good intentions; it requires working structures within the company that places the technical production and user support systems subservient to user's wants and needs; anything less and the company's claims fall short of reality.

One fundamental rule from the service industry is to never give the technical delivery group sole control over what's delivered to the client; they tend to produce what is comfortable for them to deliver rather than going the extra mile to create a pleasant first use experience. In the SAS world, we are used to being able to do anything and everything via code yet programmers, if left to their own devices, will stop enhancing code when they feel it is good enough; the problem is that it probably *is* good enough -- for a *programmer* but, probably not good enough to produce a pleasant *user experience*!

This then is the conundrum facing the company's management; the application developer or programmer is seldom given installation and first use performance criteria yet they have to make decisions on when to stop enhancing the code for the customer's experience, assuming that customer experience was even considered at all. If management fails to provide that information to the technical delivery team, management is ceding control of the critical user experience system elements to the technical delivery team; a team whose mission and production processes seldom aligns with the needs of the customer.

It is this lack of other, user-centric, controlling standards combined with the need to work within a departmental budget that ensures that their production-centric performance decisions will win out over any user-centric

[1] 'First use' is defined as all of the steps after the extraction and placement onto the disk location up to the point that the software is ready to use and includes: installation testing, operational qualifications (if needed) and account setups.

performance standards. This results in the customer losing out and, most probably, the company will ultimately pay the price in lower product adoption rates.

WHAT ARE THE CUSTOMER'S WANTS AND NEEDS DURING INSTALLATION?

It is said that for all endeavors there are only a handful of fundamentals that need to be mastered to perform that task well; similarly, customers that are installing software have a few wants and needs that, if satisfied during software installation will result in a pleasant and acceptable user experience. Wants and needs fall into MUST HAVES and MUST NOT HAVES of an installation process, the below is some suggested rules for judging whether the user has been properly incorporated into the programming calculus.

Must Haves	Must NOT Haves
1) Simple installation with automatic configuration steps, whenever possible: target installation time < 5 minutes	1) Too many configuration steps including MANY trivial configuration steps
2) No ambiguity on correct **installation**; it is either PASS or FAIL	2) Un-explained or un-actionable errors during installation
3) No ambiguity on correct **operation**, it is either PASS or FAIL	3) Doubts that the operational routines are both correct and working correctly
4) If a has problem occurred; clear, actionable next steps to rectify the situation	4) Obtuse messages requiring uncommon knowledge to initiate corrective action

USING SAS' AUTOMATION CAPABILITY TO BECOME MORE USER-CENTRIC

SAS is a perfect tool for verifying an application's installation and operational functionality, but as with anything good, it can take a little thought and discipline to make it work for you. A typical project can be illustrative.

In a real-world example, an application that consists of SAS: datasets, macros and simple AF screens; the code was to be; tested, packaged into a deployment archives, un-packaged as it would be at the user's site and then subjected to the Installation and Operational Qualification processes. SAS was used to deploy the application from the development area to both a testing and a 'packaging' area via a sequence of simple SAS statements; thus using SAS to maintain synchronized copies of the software. This had the additional advantage that the same process can be used to set up new deployment areas for, say, the application ported to both SAS v9.1.3 and v9.2 as well as for a spattering of Operating Systems, say Windows XP and Windows 7.

For each of the four versions of the software from above, there was to be at least two identical work areas: one area was to be left untouched during testing and was the source of the deployment package sent to the client; the second area was used to perform full testing of the application in a 'clean' environment.

Additionally, once testing was completed, the first area was to be 'packaged' into a full deployment package, unpacked into a new area and subjected to the full Installation and Operational test suites (IQs and OQs, respectively). For simple applications, maintaining synchronization between the various deployment areas via Windows' cut and paste functionality was possible but not recommended because of the concerns that simple errors could have undetected consequences; for applications with dozens of source files it was felt that the only safe path was in scripting or automation, through SAS if possible but, scripting none-the-less.

DEPLOYING THE APPLICATION

The tasks for populating each deployment area are:

1) Ensure a clean macro catalog by killing the macro catalog in each deployment area and copy the updated compiled macros back in;
2) Ensure a clean macro catalog by killing the AF catalog each deployment area and copy the updated compiled, non-editable, AF files back in;
3) Remove all SAS datasets from the deployment area and refresh the datasets with the currently required data; and
4) Refresh text files in the deployment areas by using SAS to first delete them in the deployment area then copy all required text files back in.

The code for Step #1, macro deployment is:

```
*=====================================================*;
* MACROS                                              *;
* ========                                            *;
*                                                     *;
* Process to follow for each catalog...               *;
* ====================================                *;
*  1) Kill the target catalog                         *;
*  2) Copy the master macro catalogs to the target    *;
*  3) Repair the catalog to ensure it is in the best  *;
*     condition for deployment                        *;
*=====================================================*;
  proc catalog cat=DployOne.sasmacr kill ;
  run ;
  quit ;
  proc catalog cat = macStore.sasmacr ;
    copy out= DployOne.sasmacr ;
  run ;
  quit ;
  proc datasets lib= DployOne ;
    repair sasmacr / mt = cat ;
  run ;
  quit ;
```

For Step #2, updating the User Interfaces, the code is:

```
*================================================*;
* USER INTERFACES                                *;
* ===============                                *;
*                                                *;
* Follow the steps for the catalogs but, prohibit *;
* editing and source code.                       *;
*================================================*;
  proc catalog cat= DployTwo.catTwo kill ;
  run ;
  quit ;
  proc catalog cat= DployTwo.catTwo ;
    copy out= AreaTwo.catTwo noedit nosource ;
  run;
  quit ;
  proc datasets lib= DployTwo ;
    repair catTwo / mt = cat ;
  run ;
  quit ;
```

HANDLING THE VOLUME OF WORK THROUGH AUTOMATION

When working with the above it became obvious, very quickly, that if a better system was not developed that it would be impossible to get the product out and, due to unexpected failures during the testing processes, more and more automation was introduced. In hindsight, it seems intuitively obvious that, for the above example, the level of testing associated with this deployment is not possible with limited resources, unless… unless scripting or automation was introduced into the equation. Scripting was first introduced for pushing the software into the 'deployment' areas but, the further we got into the process the more it became obvious that both the IQs and OQs would have to be automated as well since we did not have the resources necessary to run through the scripts for the four different scenarios.

Eventually, all deployment tasks, including the IQs and OQs were scripted. A Setup Configuration Utility, located the SAS configuration file (sasV9.cfg), copied a modified version into the applications directory then launched the remaining setup tasks. Interestingly, because of the challenges presented by the broad deployment outlined above, the user tasks were reduced from upwards of 70-100 to less than 10, actually, less than five. All setup tasks were fully automated and required no more than the batch submittal of a Setup Configuration Utility; Installation Qualifications, First Use and Operational Qualifications tasks were performed via SAS scripts and the user was able to start working with the application within the targeted five minutes. Failed IQ or OQ operations were identified by an obviously named file containing a simple description of the problem and what to do.

CONCLUSION

The four objectives identified above: installation in under five minutes, clear Installation and Operational Qualification successes or failures, if a failure occurred, a simple but effective directive to the user was provided that proved useful in satisfying the wants and needs of the customer.

The author can be contacted at:

Joe Perry
Perry & Associates Consulting
342 Spring Canyon Way
Oceanside, CA 92057
Phone: 760-822-7959
email: PerryAJoE@cox.net

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.