Three Huge Mistakes Companies Make Computerizing Weld Data

Nick Mossman, TWI NA, July 2018

You're looking to digitize some of your welding activities. You might have seen some of the reports that your colleague/competition can provide to their client. Or you might have heard some discussion about Industry 4.0 at some management meetings.

What's the next step? You have a look to see what the Internet can suggest, but the canned solutions look expensive. You have some initial discussions internally, perhaps with your IT Group, to see what they think. You have a few options here, including do it yourself or go outside.

As developers of welding software, we at TWI see the same mistakes made over and over. I wanted to write this article to expose the source of these mistakes and to help save you some heartache. Naturally we think that the TWI solutions are the most cost effective, but the advice in this article is sound and I'm sure that at least some of it will resonate with you.

Mistake #1 – Using Microsoft Office (Excel, Word, etc.) to manage welding data.

Let me start with clarifying that this may or may not be a mistake, depending on the scope of what you need to do. I'll actually get back to recommending this as a solution for small implementations, but only small. That hints at where the problem lies with MS Office. Most people know the tools and the functions, but many users under-estimate the scope of their welding data management and end up spending far too much time in development and data entry.

For example, at the start of a project you may think the tracking of welding data will be fairly simple. You just need a few columns for weld data, a couple for the welder, a few others for the NDE. All seems to be fine until you need to squeeze in more. Such as material data, oh wait, we sometimes have multiple materials, we need double the columns for that. Oh yeah, we also have two welders on some welds…double the welder columns. Wait, sometimes we have three or four welders. Shoot. Then we get over to NDE where we initially just think of VT and RT. We forgot to add columns for MT, oops. Oh wait, we also have PT occasionally… and PMI... and sometimes we do PAUT instead of RT. Forgot about Hardness too, where should we put that data ?

On and on this goes until you get to the point that you start to understand the concept of relational tables. There's a reason why experienced developers think ahead to find all of the "one to many" and "many to one" relationships, and how these sets of data need to work.

This is just the beginning of the problems too. All may seem nice and easy to start, and the users can just start punching in data as they wish. But the end goal of storing this data is so you can eventually create reports and track your progress. That should be easy enough, but users will naturally enter data in whatever fashion they wish. This makes accurate and meaningful reports impossible. For example, would you want to know the rejection rate for Billy Bob? Well, you're going to need to find a way to find a way to combine data for Billy Bob, Billybob, Billy, William Robert, and whatever other spellings/nicknames users wanted to enter. Trash in is trash out.

Then there's another problem. Billy welded on several welds with other welders. Some of those welds were rejected, sometimes Billy's fault and sometimes not. Your spreadsheet didn't record the data in a manner to deal with this. Countless issues will keep popping up which leads to constant redesigning and data fixing. You should also consider the possibility of the person responsible for this Excel development leaving the company, forcing you to reverse-engineer their design. It is likely to end up being far more expensive than you originally thought.

Mistake #2 – Let's just buy something cheap and easy.

First of all, obviously you get what you pay for. At TWI we've been developing software since 1985 and have seen many cheap competitors come and go. They will first provide an application that has most of the mistakes described above, but with patchwork solutions being added as they go. That's understandable and we certainly went through those growing pains as well. However, here are examples of some pitfalls with cheap software:

1. The company comprises just a person or two, who don't have enough time to handle support calls.
2. They are not profitable enough to stay in business, or they get purchased by a firm who decides the software isn't profitable.
3. Customizing the software to meet your needs is highly resisted or flat-out refused.
4. Their sales literature can grossly over-exaggerate their features since their reputation is not established or important.
5. When/if you get into a bind, they're aren't capable or willing to go above and beyond to get you out of trouble.
6. The scope of the software is fine for what your company does today, but can it expand to cover other areas like:
    a. Intense NDE and PWHT requirements of the power industry.
    b. Verification and validation processes in Nuclear.
    c. Lots/Progressive Examination of the Oil and Gas industry.
    d. PED/ISO requirements for Europe.
    e. Requirements from various AWS codes.
    f. Any other specialty work your company bids on.

Mistake #3 "Let's develop something on our own!".

This is by far the biggest and most expensive mistake we ever see. While it isn't all that common we've heard of some colossal development costs going into 7 figures, and sometimes well beyond.

First you start with all of the problems above in Mistake #1. But it goes well beyond this because of two huge factors. The first is if you're doing it yourself, as an engineer who knows a bit about software, you will be using a tool that isn't as easy to use as MS Excel, so you'll be fighting the limitations of the tools all the way. We know this from our own experience.

Secondly, you're a bit more serious about it than scenario #1 and you'll want to do a bit more design before you start development. You're IT Group should insist on this anyway. The trouble is, like scenario #1, you'll make the same under-estimates of the requirements. You'll need to fail a few times before you can succeed. We at TWI certainly understand this as we've been down this road many time. You could prove it for yourself but here's an example (or warning) scenario:

The idea is fairly simple. Let's just make a user-friendly app for tracking welding activities, and perhaps something simple for welder certs and WPSs. How hard can it be, right? You work up some designs, talk it over with your IT staff and come up with a budget. It appears that it should only take a few weeks of your times, and maybe a couple months burden of some IT personnel. Conveniently, costs can be somewhat hidden in other department's budgets, so perhaps that will allow your project to fly under the radar.

All goes well at first and you quickly get a rudimentary 'proof of concept' put together. It doesn't really work yet, but it's enough to convince you and others that you're on the right track. You've already burned through half your budget however and you realize you're going to need to get a bit more serious about how much this will cost. IT might actually want some kind of project/job number to bill this under and it begins to be a bit less than 'under the radar' as you initially hoped. No problem, let's press on.

A few more weeks go by, communication with your developers has been a little frustrating, but they finally have something for you to present. It doesn't quite go as well as you hope. User comments come back with "this is fine, but we need it to also do such and such"…"we need it to look like the system that we're already using"…"we need it to be web-based instead"…"we need it to also work offline"…"we need"…"we need"…"we need". You try to push back for a while, try to get everyone to conform to your vision, but eventually you'll have to acquiesce to keep other people on board. It's time to scrap the first idea and take this to a higher level. Of course at this point you'll also need to bump the budget up once again.

The next edition is quite a bit more thorough and takes far more design and development. You're under increasing pressure to get something delivered and this project has consumed most of your professional time, and almost all of your personal time too. You must get this released however so you deploy an edition you hope will suffice. It should work but IT encounters permission and installation issues (hopelessly bang your head against the wall for a couple days). When they finally get past those hurdles the users experience glitches and crashes at every other turn. Everyone grumbles how this is a piece of trash even though you assure them it should otherwise work. Your budget is an absolute joke at this point but you still need to push on. You work to convince users to utilize awkward work-arounds and do your best to get your developers to keep swatting down an endless swarm of bugs

A few more weeks pass until you get to the crushing blow…the users shut down your program and returned to their trusty old spreadsheet.

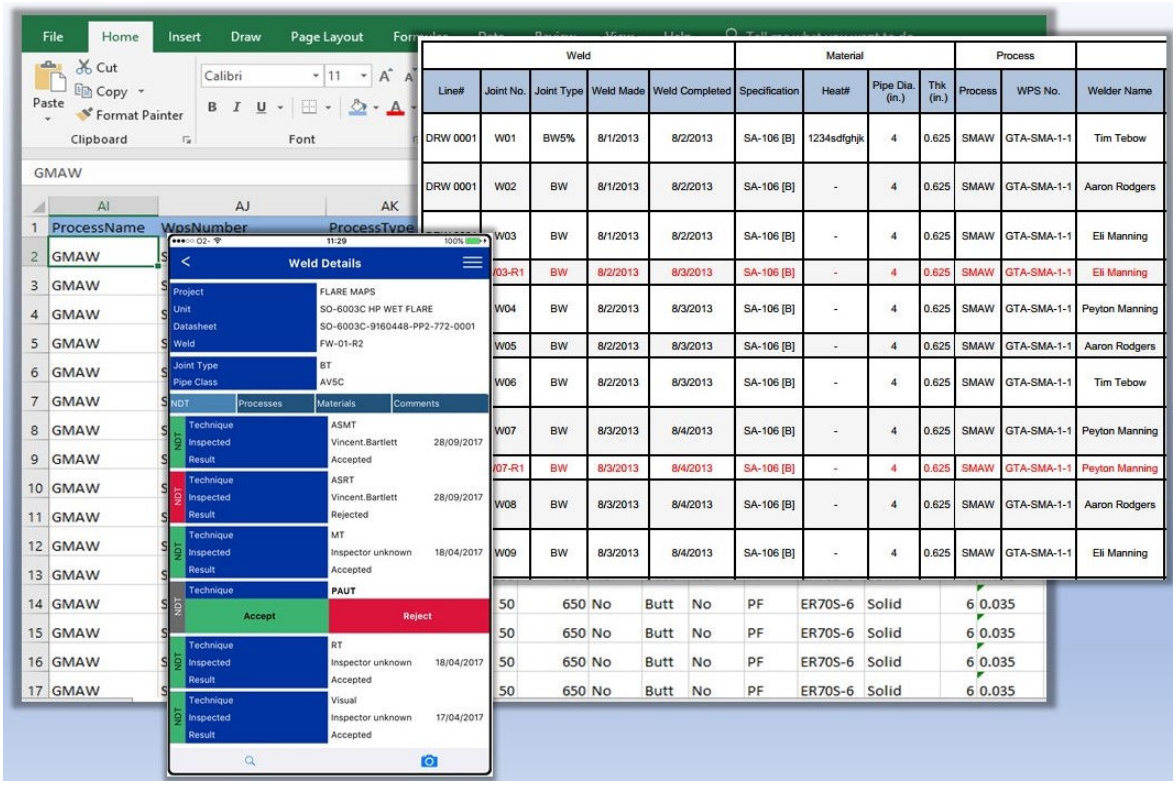
You're too far down this rabbit hole to stop now, however. Back to the drawing board you go to work things out. You realize at this point you need to consider a massive third design change, but

there's no chance – you'd need to treble your already increased budget. You're stuck patching on whatever fixes and work-arounds you can muster, tack on a couple extra features and then try to send it out to the users once again.

This goes a bit better but they're still grumbling. It's less glitchy, but now they complain it's pathetically slow. On top of that they find the labor-hours required to put data into this system is worse than their simple spreadsheet. Complaints grow, meetings are called, your reputation is suffering, and you begin to wonder why you didn't bother to read this article (*forgive the author's blatant self-promotion*).

Getting back to what I promised under Mistake #1, in comparison to #2 and #3 you can see that it's certainly the lesser of the evils. It's a reasonable start however and will teach you quite a bit about what you really need. Should you require a more robust system however I suggest you consult highly experienced welding software developers. I can't put enough emphasis on 'welding' here as well. Developers who understand the vast complexities of our welding and fabrication code rules, and the whacko twists and turns our industry takes, are able to understanding your requirements with minimal explanation. I won't try to do a sales pitch on you here, but I will write a follow-up article covering the options for creating Welding Data Management Software.

For now, just keep in mind, if you're thinking of taking on a project like this, you're getting into a far more expensive development than you can imagine. If you disregard the caution written here, hopefully there isn't evidence this was made available to you !!



About the author:

Nick Mossman is a consultant for TWI Ltd., a worldwide independent research and technology organisation, with expertise in applying engineering, materials, and joining technologies across all industry sectors.