Computer & Software Validation

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer System Validation—Definition and Requirements</td>
<td>1</td>
</tr>
<tr>
<td>Computer System Design</td>
<td>4</td>
</tr>
<tr>
<td>System Design and Control</td>
<td>8</td>
</tr>
<tr>
<td>The Nine Most Common Computer Validation Problems—Identify Frequent</td>
<td>13</td>
</tr>
<tr>
<td>Deficiencies to Accelerate Your Validation Projects</td>
<td></td>
</tr>
<tr>
<td>Accurately Identifying Your Requirements—Will Any Computer System be</td>
<td>17</td>
</tr>
<tr>
<td>Right for You?</td>
<td></td>
</tr>
<tr>
<td>Computer Systems Quality and Compliance vs. Software Validation</td>
<td>20</td>
</tr>
<tr>
<td>Computer Systems Change Control</td>
<td>23</td>
</tr>
<tr>
<td>How to “Right-Size” Computer System Validation Based on Criticality</td>
<td>27</td>
</tr>
<tr>
<td>and Complexity</td>
<td></td>
</tr>
<tr>
<td>Practical Use of Automated Tools in Computer System Compliance</td>
<td>31</td>
</tr>
<tr>
<td>Selecting and Partnering with a Vendor for a Qualified Software Product</td>
<td>34</td>
</tr>
<tr>
<td>Information Security—A Critical Business Function</td>
<td>38</td>
</tr>
<tr>
<td>Disaster Recovery and Business Continuity</td>
<td>44</td>
</tr>
<tr>
<td>System Definition: Defining the Intended Use for a System</td>
<td>51</td>
</tr>
<tr>
<td>Lessons Learned in a Non-Regulated Software Validation Project</td>
<td>66</td>
</tr>
<tr>
<td>Computer Validation in the New-Clear Age</td>
<td>78</td>
</tr>
</tbody>
</table>
Welcome to “Computer Validation Forum.” This column discusses topics and issues associated with computer validation in order to provide a useful resource for daily work applications. It provides information regarding regulatory requirements for the validation and qualification of computerized systems.

Computer systems are used widely in the daily work structure of all the life science industries. Technical considerations associated with computer systems and the validation and qualification required are broad and complex. Understanding the basic principles supporting computer systems is fundamental to daily operations. Control and compliance are the key integrators for all computer systems in the life science industries today.

Your questions, comments, and suggestions are required to fulfill the objective for this column. Please send your comments to column coordinator Sharon Strause at sastrause@aol.com or to journal coordinating editor Susan Haigney at shaigney@advanstar.com

KEY POINTS
The following key points are discussed in this article:
• The definition of computer system validation (CSV)
• Project management and the software development lifecycle (SDLC) are the starting points
• Requirements are the primary key to CSV
• Other points to consider include US Food and Drug Administration requirements, the overall quality process, validation, and documentation.

INTRODUCTION
This first installment of “Computer Validation Forum” introduces a series on the subject of computer system validation (CSV) by defining CSV, looking at the importance of project management, and specifying CSV requirements.

THE DEFINITION OF CSV
Computer system validation establishes documented evidence providing a high degree of assurance that a specific computerized process or operation will consistently produce a quality result meeting its predetermined specifications. Many will recognize this definition as an interpretation of the US Food and Drug Administration’s original process validation definition.

Components Of A Computer System
The components of a computer system include hardware, software, operating procedures, processes, and personnel. The Figure illustrates the areas required for consideration in the validation and qualification of computer systems.

IMPORTANCE OF PROJECT MANAGEMENT
A CSV project that meets budget, is implemented in a timely fashion, and meets all the regulatory requirements for the system must start with a formal project planning process and a system development lifecycle (SDLC). These programs require both an experienced project manager and a qualified validation manager. FDA has stated many times, “Those who fail to plan, plan to fail.” Planning is a critical factor for the entire CSV project. If your company does not have a project management tool, there are many on the market which can be utilized to keep track of multiple timelines, deadlines, personnel, critical meetings, and due dates.

There are many SDLC processes, which are used in validation—the waterfall model, the V-model, the “Onion” model. It doesn’t matter what SDLC process is used as long as it begins with the development of the project and ends with the ongoing maintenance of the system once implemented. It also includes the ultimate retirement of the system.

Once a project management team has been established they can begin requirements gathering.

REQUIREMENTS
Requirements will determine the scope of the project. The validation and/or qualification should be the first major deliverable for any computer system. Again referring to the Figure, requirements include the following:
• **Software.** How the software is to operate.
• **Hardware.** The hardware including the server
• **Controlling system.** The operating system on
the server and the database used to collect the data from the software.

- **Equipment.** Equipment is other computer systems or pieces of manufacturing equipment with which the software may interact.

- **Operating procedures and documentation.** These all have requirements that include people who will be doing the work of validation, people who will be trained to build the system, and people who will be trained to utilize the system once it is in place.

- **Controlled processes.** Established controlled processes and change control need to be reviewed or addressed to ensure that control is maintained throughout the life of the project and for the ongoing stability of the system once validation and qualification is complete.

- **Total computerized system.** Networks may be local or wide area, may utilize the web, may be within a corporate intranet or utilize the facilities of the Internet.

- **Operating environment.** Security will be addressed as both a part of the operating environment, the software and operating systems on the hardware, and all interfaced equipment.

Another way of determining the requirements is to ask the questions “who, what, why, where, and when.” Answering those questions will make the requirements gathering process easier and will help in determining the priorities of the system.

Once the system requirements have been gathered, the process of determining the regulatory requirements will begin (see Reference section). For what will the data developed on the system be utilized? Regulations need to focus on the purpose, use, and reporting of the data. There may be regulations outside of FDA that will be impacted by the data. For example, in an enterprise resource planning system, data will be subject to financial regulations, possible Environmental Protection Agency (EPA) regulations, possible Occupational Safety and Health Administration (OSHA) regulations, etc. Again requirements will help to determine the regulations required and ultimately the extent of the validation and qualification that will need to be done on the computer system.

Requirements gathering should take time, because it is the foundation of the overall project and the validation required. CSV can be as simple as an Excel spreadsheet or as complex as an enterprise resource system, thus the reason for the critical nature of realistic and testable requirements.

Once testable requirements have been established, the project can begin; validation can be established; risk evaluation can be started; and the goal of a validated and qualified system can be reached.

**POINTS TO CONSIDER**
Additional points should be considered in the validation and qualification of a CSV, including FDA requirements, quality process, validation checkpoints, and documentation.

**FDA Requirements**
FDA requirements regarding current good practices (CGXPs) are as follows:
- Hardware is considered to be equipment within the meaning of the CGXP regulations
- Software is regarded as records or standard operating procedures (SOPs) within the meaning of the CGXP regulations
- Software maintenance is considered revision or change control
- Record controls require programs to ensure accuracy and security of computer inputs, outputs, and data
- Record access requirements—available for inspection and subject to reproduction

**Quality Process**
The quality process needs to be in place and should include the following:
• SDLC methodology
• Project planning
• Personnel qualifications
• Documentation standards and procedures
• Methods for review and approval
• Design standards
• Programming standards
• Configuration management
• Testing standards and procedures
• Separation of development, test, and production environments (logical/physical)
• Move to production process
• Clearly defined responsibilities
• Involvement of customer/user, quality assurance professionals, and technology professionals
• Change management
• Change control
• Training process
• Process for continuous evaluation, incident monitoring, and error correction
• Processes and procedures for physical and logical security of system and data.

Validation
Validation checkpoints should be in place as part of the overall project management process. Consider the following:
• Evaluation, analysis, and rationale for system and its validation
• Validation strategy
• Business, system, and function requirements
• Detailed system design specifications
• Validation protocol
• Test plan
• Development testing and verification (structural, unit, integration, and system)
• Vendor and supplier evaluations
• Hardware and software qualification (installation qualification, operation qualification, performance qualification)
• Procedures
• Utilization
• Administration
• Maintenance
• Monitoring
• Change management
• Change control

• Installation plan and records
• Training plan, procedures, and evidence of training
• SOPs
• User acceptance
• Validation report
• Retention of critical documentation.

Documentation
Documented evidence should include the following:
• Validation plan
• Business and system function requirements
• System design specifications
• Validation protocol
• Test plans, scripts, results
• Documented development testing (i.e., unit, integration, system testing)
• Installation qualification
• Operation qualification
• Performance qualification
• Validation report
• Standard operating procedures
• Manuals (e.g., development, user, support)
• Change records
• Logs, operational records, audit results.

REFERENCES
21CFR210, Current Good Manufacturing Practice in Manufacturing, Processing, Packing, or Holding of Drugs: General.
21CFR211, Current Good Manufacturing Practice for Finished Pharmaceuticals.
21CFR820, Quality System Regulation.

ARTICLE ACRONYM LISTING
CGXP Current Good (ALL) Practices
CSV Computer System Validation
EPA US Environmental Protection Agency
FDA US Food and Drug Administration
GXP ALL Good Manufacturing Practices
IQ Installation Qualification
OQ Operational Qualification
OSHA Occupational Safety and Health Administration
PQ Production (Performance) Qualification
SDLC System Development Lifecycle

Originally published in the Spring 2009 issue of Journal of Validation Technology
Computer System Design

Robert Smith

“Computer Systems Quality and Compliance” discusses the quality and compliance aspects of computer systems and aims to be useful to practitioners in these areas. We intend this column to be a useful resource for daily work applications.

Reader comments, questions, and suggestions are needed to help us fulfill our objective for this column. Case studies illustrating computer systems quality and compliance issues by readers are most welcome. Please send your comments and suggestions to column coordinator Barbara Nollau at barbara.nollau@av.abbott.com or journal coordinating editor Susan Haigney at shaigney@advanstar.com.

KEY POINTS

The following key points are discussed in this article:

• Systems design is the process or art of defining the architecture, components, modules, interfaces, and data for a system
• System design should consider the entire system lifecycle to properly manage costs and compliance
• System changes, maintenance, and future expansion or other organizational changes should be part of system design
• The role of quality is often compromised in system design in favor of project cost and timing
• Security issues, both external and internal, are an important consideration
• System designers must consider the needs of the quality area in system design and must actively solicit their input
• Quality unit personnel, in turn, must carefully consider their needs, and clearly communicate these needs to system designers
• Do not under estimate the cost and time impact of even the smallest change.

INTRODUCTION

My six-year-old daughter is often fascinated by things that fascinate me. On the cover of a book that I had asked for her to bring to me was a picture of a kettle with the spout and the handle on the same side. She studied the picture for a moment and then reported carefully, “that is not a very good design!” I was delighted in her discernment. It was easy for her to understand the intended use and “know” that this will not work very well.

How often do we fail to have these insights when designing GXP computer systems? More often than we’d like to admit. Pressures mount to do more with less, hit timelines, show return on investment, and meet commitments. These are all admirable things, and senior managers should push system designers and project managers to contribute to the business by thoughtfully executing against those mandates. At the same time those very same project teams need to keep stakeholders informed about the technical debt they are accumulating. If teams are making decisions to sacrifice quality or maintainability in order to meet those demands, technical debt is incurred. The payment on technical debt, like personal debt, has a cost that can be felt for a long time. The recurring costs of technical debt are far greater than addressing the issue presently.

The more likely that changes to a system will occur, the more impor-
important it is to understand the long term cost of those changes. Elements of a system that are subject to higher velocities of changes are the best candidates for analysis. This column will explore some common tradeoffs that lead to technical debt.

**FIX ONE–BREAK TWO**

One small example that can lead to technical debt is hard coding a “variable” that, by its very name, we know will change over time, to save a few days development time. This might be a password, a common security mistake, or some configuration setting like the name of a database server. It is easy to hard code such a thing to save time, but because the likelihood of change is high, the cost of this shortcut is high. This is true for two reasons. One is that a validation process must be re-executed and the other is the risk that something else might get inadvertently changed or that there is some unintended consequence. This is commonly called the fix one–break two syndrome. In short, it is a change that leads to technical debt.

A password mistake is a perfect example. Good security requires frequent changing of passwords. If a password is hard coded, then a new version of the software (called a release) is required to update the password. For a validated system, this will result in an even larger cost the organization will pay over and over again. If the organization does not change the password to avoid this cost, it has traded good information security practice to pay the technical debt and also accepted a 21 CFR Part 11 compliance risk. Assuming the system has a reasonable life of five years, the technical debt per year of not making the password easy to change is either poor security and a compliance risk or the cost of two or more releases per year over five years. Besides the recurring costs of the releases, the organization will also assume the risks related to releasing and validating the application. Surely it would be more efficient to handle the password correctly in the first place. Pay now or pay a lot more later.

**MANAGEMENT OF CHANGE**

Understanding the concept of change velocity is important for any system, but even more so for validated systems. Specific strategies need to be in place for dealing with varying rates of change. What is the best way to manage these varying rates of change? What are the costs associated with the changes and how should an organization manage them? Vendor software may move at one speed. Internally developed customizations probably move at another rate until the system matures, but may accelerate if business processes change. Microsoft patches its operating system monthly, commercial application vendors might patch quarterly, and an analytical chemist might not change a calculation for years—it makes sense to separate these. Often in looking at production or deployment phase plans there is a one-size-fits-all approach. This often leads to something that is impractical or worse.

Upfront planning to develop specific strategies to handle different change velocities and understand the risks associated with these changes helps significantly to develop cost-effective plans that look at the system over time. Focusing on lifecycle cost planning will minimize the technical debt of the deployed application.

**Changing A Password**

For example, systems that have passwords that are used infrequently are going to result in passwords that expire or are forgotten by users. What is the strategy for managing this? Let the help desk do password resets manually by routing a ticket to the database administrator? That’s the most expensive solution. Write a tool so that the help desk can do it for them? This is the better approach. Add a self-service feature in the application? This is the best approach. Knowing what to do requires some planning and time up front. Imagine a 1000-user system and assume 30% will need one password reset a year. This is an optimistic estimate. Suppose each help desk call costs $50 by the time the security administrator changes the password and the system is in use for five years. The organization will spend at least $75,000 on tickets alone. This is more than it would cost to implement a self-service “I-forgot-my-password” feature. This model doesn’t even consider any impact to the business, such as inability to release a lot while an engineer is locked out, so the total technical debt could be much higher.

**Changing A Storage System**

Another example is the case of an electronic record storage system. Let’s use some numbers to illustrate the point. To make the math easy, let’s assume that a basic validated system costs $1 million and has a 10-year life. The team reports that they need an extra $100,000 to address an archiving feature or the system will outgrow the storage system early in the system’s expected life. The extra money is deemed too expensive. The project was already spending every dime, so the decision is to address it later. Over time business needs change slightly as it becomes paperless, and in five years the system is critically low on storage. A new project is proposed to add the archiving feature. Because this is a validated system and now contains five years of electronic records, it will take a full release and sufficient testing to show that the records are archived correctly. Let’s say the team can do this for $500,000 and delivers it
robustly on time. But now the last five years of the system depreciation costs twice as much. Would the $100,000 in initial project costs have been worth saving $400,000? This is the kind of technical debt that needs to be managed thoughtfully at the beginning.

**SYSTEMS DESIGN**

Systems design is the process or art of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. Today, more than ever, system design must be cost effective. Today’s economic conditions require full lifecycle cost to be factored into decisions. It is not uncommon for the maintenance phase to prove more costly than the implementation phase. The maintenance phase is often not considered or analyzed but is a counter force of getting the cost out of the business. The proverb of the frog sitting in water with the temperature slowly going from cool to boiling is a good reminder. The frog doesn’t notice the heat because the rate of rise is slow, but in the end he is cooked. From the preceding examples we can clearly see that understanding and managing technical debt can have a profound impact on GXP computer systems and allow us to jump out while the water is cool.

Anytime we are asking the organization to pay more or take more time in the implementation phase, we have to articulate the value proposition. That proposition will be the benefit of addressing a lifecycle cost now vs. assuming the recurring cost and risk over time. Few teams are getting a blank check in today’s environment. How does a team explain the value proposition? Some points are obvious, some are not so obvious. Most decision makers want to be rational and make wise decisions for their organizations. In order to support fact based decision-making, teams must tally the technical debt and make sure that decision makers understand what they are buying on credit—sort of the fair disclosure doctrine of GXP system development costs. It must be expressed in business terms identifying clearly what the cost and the benefits are. Numbers and specific examples that support business decision-making are critical for influence. It cannot be expressed in technical “geek-speak” language.

**THE STOOL HAS FOUR LEGS**

Yet another type of technical debt is assuming that quality of a system is simply something that exists at some constant level. Often this happens when quality is assumed by taking it off the table with statements like “we never compromise on quality.” Traditional project management paradigms articulate that there are three legs (i.e., scope, resources, and time), but with a wink we all know there are really four—quality does not simply exist. Quality is often traded to make the other three. If teams and their sponsors agree right up front that quality is not a magic property that appears in a system, but is something that is designed in, then the stage is set for initial planning and subsequent discussions about trade-offs and tuning to ensure that all four variables have a place at the table. When quality is simply assumed, then bad things can happen and they usually show up in the form of technical debt.

In this author’s experience, most organizations have strong formal and informal mechanisms to ensure project costs do not exceed the plan. And for good reason, as the system development community has accumulated few headlines for on-time, on-budget, on-scope, and on-quality success. The technical teams need to do a better job of expressing the quality trade-offs in business terms and identifying risk factors that the business can understand. Telling a business leader we need more time to fine tune the user interface or make usability changes is hard to relate to a business impact. Stating that there are data that suggests one in five users makes errors that could result in erroneous filings to a governing body and here are the errors is something that can be processed in the business risk management and review framework.

Thus, in order to have a fact based dialog, decision makers need to be involved up front with competent system designers who understand both how to get things done and how to consider what the organization will pay over time. These “pay-me now or pay me later” time bombs are not just measures of technical acumen. They are also indicators of business savvy. Business leaders need to have trusted technical leaders that can help get the cost out of the business by not just excelling at technical execution, but also by understanding how to speak to the business.

If a team understands its customers, it can implement in a cost-effective way. For example, enabling users to add reports using validated features can avoid more costly-to-deliver and harder-to-get-scheduled IT releases. In this author’s experience, it is rare to see those trade-offs surface up front. Most senior business leaders would rather know they’ll get all the reports they asked for upfront in the validated system, but anything else will be another costly release. Most would like the chance to ask if there is a way to avoid those costly releases.

When designing for maintainability, the concept of change velocity comes up again. In this author’s experience, there are many tightly-coupled or interfaced systems that should be loosely coupled. Tight coupling occurs when one module or system relies on another module or system so strongly that a small change in one will require an implementation change in the other. The following is an example of tight system coupling: System A needs to view System B’s records. To make things fast, the B team sends the A team source code from their system. A implements B’s code and the organization is happy. Any time a user
of A needs a B record, they can get it. Later B adds another record type and users of A still need to see it. But now both A and B have to release anytime there is a change—Not good—Pay a lot later.

What is the correct solution? B could have implemented a service for A, “show me a record.” With a little thought something as simple as “show me the record-this-ID” could be implemented. Then A and B are loosely coupled so one system can be changed without the need to change another. The cost effective paradigm is to make tight coupling rare. It might cost a little more up front, but it will save a lot later.

This can pay back in more ways than one. Not only can an organization avoid extra release costs, it can also improve uptime, as now only one system needs be taken offline to make an upgrade.

PLANNING FOR THE FUTURE
Understanding how the user community is expected to change and probable impacts on electronic data can have a dramatic impact on lifecycle costs. Does the system need to support a business acquisition plan? If so, this could dramatically affect the user count and make one design appropriate or inappropriate by altering scalability needs. Will more than one geographic location be using the system? If so will data consolidation be required? Knowing the answer to questions like these may not only affect system architecture, vendor selection, and technology selection, it may also require the addition of a data warehouse to meet reporting needs. Often fixing things like these later becomes massively expensive when compared to enabling the system for scalability up front. Often, senior leaders will make different choices if they have the data and facts to allow good decision support. Skipping these steps frequently leads to unanticipated costs and can undermine the technical team’s credibility.

Security is often addressed as an afterthought. Sometimes teams work hard to get the system to work, then say, “let’s make it secure.” At this point it is too late. Security, like quality, needs to be designed in and requirements should be stated clearly up front. The requirements need to be clear and related to risks.

Often GXP systems are closed systems on internal networks and not subject to skilled, determined attackers. But insider threats are real and the most prevalent. These threats run the gamut from disgruntled employee sabotage to someone “correcting” their mistakes to avoid reprimand to misappropriation of intellectual property.

Some systems in the life sciences sector may also contain protected health information and may be subject to government regulation, most notably the Health Information Portability and Accountability Act. Understanding the risks, vulnerabilities, and countermeasures is important in system design, and it is the most cost effective as part of design as opposed to later. Often failure to plan for this creates expensive and time-consuming redaction programs.

IMPLICATIONS FOR COMPLIANCE
Compliance personnel should always be part of computer systems design activities—the fourth leg of the stool. They can provide valuable input regarding quality requirements that will minimize future costs and system downtime. When the quality area is overlooked, future changes to the system will surely be needed, and these future changes equate to additional costs, downtime, and potential problems affecting other systems. The quality area must also be mindful of the importance of their input. The quality area must carefully consider its needs and must clearly communicate these needs to the systems designers—do not underestimate the cost and time impact of even the smallest change.

CONCLUSION
Good software design is complex. These are just a few examples of how shorting the initial planning and implementation can result in significant downstream costs. Business owners of systems and budget decision makers should set clear expectations that while certain budget and schedule goals are in place, the expectation is that system designers provide solid information related to lifecycle costs. That information can be used to get to the best decisions related to managing technical debt and cost effectiveness.

ABOUT THE AUTHOR
Robert Smith is an application technical lead responsible for quality systems software development at Abbott Vascular. Prior to this, he was Sr. Director, Engineering at Symantec Corporation, where he was responsible for developing enterprise client, host, and server based corporate security products as well as the Symantec and Norton Live Update offering. Robert has 25 years of software development experience including VC start-ups funded by The Mayfield Fund, Granite Capital, and Wasatch Venture Fund, and holds CISSP and PMP credentials. Robert can be reached at robert.smithii@av.abbott.com.

Barbara Nollau, column coordinator, is director of quality services at Abbott Vascular. She is responsible for validations, reliability engineering, supplier quality, microbiology, and document management at Abbott Vascular. Ms. Nollau has 25 years experience and increasing responsibility in pharmaceutical and medical device industries, spanning areas of manufacturing, quality assurance/compliance, and information services/information technology.

Ms. Nollau can be reached at barbara.nollau@av.abbott.com.
“Computer Systems Quality and Compliance” discusses the quality and compliance aspects of computer systems, and aims to be useful to practitioners in these areas. We intend this column to be a useful resource for daily work applications. Reader comments, questions, and suggestions are needed to help us fulfill our objective for this column. Case studies illustrating computer systems quality and compliance issues by readers are most welcome. Please send your comments and suggestions to column coordinator Barbara Nollau at barbara.nollau@av.abbott.com or journal coordinating editor Susan Haigney at shaigney@advanstar.com.

SUMMARY
An illustrative incident at a pharmaceutical company that is representative of actual events is discussed. This incident involves software control of a drug dispensing system in pharmaceutical manufacturing. An error in amount of drug weighed occurred. The investigation identified several problem areas. Lessons learned, areas of concern, questions to be asked, and actions to be taken are discussed.

INTRODUCTION
The following discusses an illustrative incident at Pharma154, a fictitious pharmaceutical company that makes the global commercial supply of Pinkoswill, a potent drug product. Because this drug product contains a potent active ingredient, weighing the correct amount of drug in the manufacturing process is critical.

Personnel involved in the incident include the following:
- Alex, vice president of regulatory affairs
- Bob, vice president of information technology
- Annie, software development manager
- Alicia, software contractor
- Sam, systems test lead
- Salli, system administrator
- Manufacturing engineers and operators.

While the incident, company, drug product, and personnel involved are contrived, the following is representative of actual events for which the US Food and Drug Administration has issued warning letters.

THE INCIDENT
“I need you here. Now!” exclaimed Alex, the VP of regulatory affairs at Pharma154.

“Alex, are you crazy? It’s Sunday. It’s 5:00 AM,” slurred Bob, Pharma154’s vice president of IT.

“Bob, listen, there are three reported hospitalizations tied to Pinkoswill. They are all in critical condition. Surveillance is coming in now, we think there may be others. We expect the FDA to be here Monday morning. This is serious,” Alex explained coolly.

Bob started to wake up, “What does this have to do with IT anyway?”

Alex said, “We are not sure. Something has gone wrong. The labs say the dosage in the suspected lots is almost four times spec. We have got to figure this out.”

“Look Alex, this is clearly some manufacturing problem. I have a life. If something points to IT, then call me. Otherwise, I have things to do. OK?”
said Bob.

"I thought you would want to be in on this. It's important. But, I have to admit, we do not have anything that points to IT. I'll call you if something changes," Alex managed to squeeze out before Bob hung up.

Within hours, CNN reported: "Massive Pinkoswill recall, FDA investigates. All patients should stop taking this medication immediately and see your physician."

**THE IT GROUP GETS TOGETHER**

On Monday morning, an emergency senior staff meeting was called in the Pharma154 boardroom.

Alex addressed the room, "We have ordered a world wide recall of Pinkoswill, not that we had much choice. The FDA would have had an order in our hands later today anyway, so we made the call to be proactive. The analytical labs have analyzed samples from the last three lots. About 15% of those lots have an overdosing of about 400%. We do not know why. We have chemists and engineers on the lines now and at our suppliers. We reviewed our sample data and the stored samples—they all check out. So we have some variation that we do not understand yet."

Bob, after listening to Alex’s explanation of the weekend’s events, was glad he did not waste his Sunday waiting around for manufacturing to figure out its problem. When he got back to his office he saw some serious faces. Sitting at his conference table were Annie, his star software development manager; Alicia, a software contracter; and Sam, his systems test lead.

Bob asked, "Why all the serious faces? This whole Pinkoswill thing is just some manufacturing problem. They have it sorted out, no one died, at least not yet. It is going to hurt for a quarter or two. Come on, we have lots to do."

"Well! Uhm. You might want to ask Alicia what her idea is," said Annie.

Bob was in no mood for this. "Let’s let manufacturing figure out their problems. We have our own problems to worry about. Last time I checked you had a couple of projects that should be keeping you pretty busy," grumbled Bob.

"I really think you should listen to her, Bob," said Sam.

"OK, let’s have it and be fast, I have a meeting in 10 minutes," snapped Bob. Then he said, "I'm sorry. It has been a rough couple of mornings."

**The Problem May Be In The Software**

Alicia reported that if the scales and controlling software failed in some way, it is possible that the active ingredients in the recipes could get over-spedied. "The filler is added to make the weight. This is a design flaw that I pointed out but we postponed correcting it."

Annie pointed out, "We postponed it because it can’t happen. There are two weight check and software controls. That's why there is nothing wrong with the design."

Alicia said timidly, "Can I add something?" When no one said anything she went on to explain. She was more than a little embarrassed. "When I first got here I couldn't get the software to interface with the scales correctly. The manufacturing engineers were very frustrated with the personnel change and could not believe that they had to get another software engineer up to speed. They told me to ‘figure it out.’ They were not very helpful."

Salli, a system administrator at Pharma154, had told Alicia that the last person that had the scale interface job got it working somehow. Salli said she made a back up of his hard disk before he left. She would restore the files for her and maybe something would help.

"I poked around at all the stuff from the backup. It took me a while but I found some stuff that seemed to work. It passed all the basic tests. So I copied that into our test environment,' Alicia recalled. "I was really concerned because we do not have any real version control. I even wrote a bug report on that. The manufacturing engineers closed it and were thrilled that I finally ‘figured it out.’ But I didn't! All I knew was that when I put that DLL in the directory, the tests passed. They signed off and I think that is what went into production," Alicia concluded.

Annie said, "We had better get Salli in here."

When asked what her role in the situation was, Salli offered in defense, "Look, I was just trying to help. All I did was give her the files, she put them into test and the manufacturing team signed off."

Sam asked, "Why don't my team and I go out on the line and do some testing."

"Fine," barked Bob, "but I want an answer tonight."

**VISIT TO THE LINE TO TEST THE SYSTEM**

Sam gathered his team and headed to the line. No one was happy out there. "We need something to put in the drug hoppers to test the scales. That stuff weighs nothing. They all looked at each other for a while. Sam saw a five-gallon water bottle by the cooler in the break room that he could see outside the manufacturing area through the observation window. Sam asked one of the manufacturing engineers if he could put that bottle in a pre- and post-mixing process hopper.

The engineer laughed at him, "That must weigh 100 times more than the compounds we mix."

“That’s the whole idea,” said Sam.

“Go ahead. It won’t break anything and we have to sanitize the whole line anyway,” stated the engineer.

Sam came back carrying the water. "This has to weigh 40 or 45 pounds," he grunted as he strained to set it into the hopper.

They all stood back. The scale read 46.75 pounds. “Good guess!” they cheered. Sam went to the software; it said the weight in the hopper was 41.25 pounds.
They all wondered how that could be. Annie said, “I remember some problem a long time ago about boot order and the USB interface to the scale.” They decided to reboot everything. They turned off the computer system and the USB hubs. Some one said, “Let’s turn off everything.” They did that too. Sam wondered aloud if there was some protocol for restarting.

One of the manufacturing engineers on the other lines offered to help. He told them the order in which to turn everything back on. They did and now the software read 46.75 pounds just like the scale. Sam, said, “This is not good.”

“Why?” asked Annie, “Everything is working fine now.”

Sam said, “Let’s just try a few things. What is this other USB cable for?”

The manufacturing engineer informed them that it controlled the hopper shape knife gate valve. They all laughed. “The what?” sang the software team almost in unison. The engineer explained, “It controls how much of each ingredient goes into the mixer. It opens until the right weight is in the mixer and then closes.”

Alicia spoke up, “I wrote the code for that. The valve is closed. I send a command to open it, then when the weight rises above the spec, I send the close command.”

“What happens if it stays open?” asked Sam. The manufacturing engineer explained that would ruin the batch and the incorrect mix would be caught at the post-mixing weight station.

Sam pulled the bottle out of the pre-mix hopper and put it in the post-mix hopper. It weighed 46.75 pounds on the scale and the software. They all agreed that made sense.

Sam asked the engineer, if he could unplug and re-plug the cables. “Sure,” he told them, “the techs do that sometimes if the valves need maintenance.” So Sam unplugged the USB-controlled hopper shape knife gate valve and plugged it back in. The room was very, very quiet.

The software displayed a strange error message. Salli commented, “That’s odd. It says ‘Unit test parameters exceeded, using default test values. Click OK to continue.’ That’s not any error message I have ever seen before. The wording makes it seem like some default or testing mode.”

The engineer said, “We’ve seen that a few times after valve maintenance, but we usually reboot everything.”

Sam clicked the OK button. The scale went blank and then the software and the scale both reported 41.25 pounds. You could hear a pin drop.

Annie asked, “What goes in the mixer first?”

The engineer replied, “The active ingredient. We don’t want to add anything else unless that weight is accurate. It cuts down on scrap. That stuff costs like a thousand times more than everything else that goes in. We got a process validated to reclaim it a few years back.”

“So if the scale was doing what we see now, the valve would let in a lot of the drug?” asked Annie.

“Yes,” the engineer replied, “That’s why we weigh it a second time. Only the exact recipe will produce the correct post-mix weight. We have that down to a science.”

Alicia was the first to see it. The scale error is constant. Both scales were off by exactly the same amount.

And though they all thought it, Annie was the first to say it, “We have a serious problem. A real serious problem! We have got to tell Bob.”

The team informed Bob of the situation who then contacted the VP of regulatory affairs.

“Alex, this is Bob. We have a problem. My team found a situation. It appears that if there is some maintenance performed on the line, a real problem can occur. I am no chemist but I think something like five pounds of extra drug might give some people a real bad day.”

As would be expected, FDA investigated the Pharmal54 situation. The FDA-483 the company received from FDA was not kind. A warning letter was expected to follow. The possible fines assessed could be astronomical. The lawsuits the company may incur will probably be worse.

INVESTIGATION

During the corrective action and preventive action (CAPA) investigation, the following items were documented by outside investigators:

- Software developers were not practicing version control. Software and associated source code files were not kept in a repository. This is in stark conflict with the International Society for Pharmaceutical Engineering (ISPE)’s GAMP 5: A Risk-Based Approach to Compliant GxP Computerized Systems. This lack of appropriate software version control was a direct contributor to the event.
- The company lacked a formal procedure for deploying baselines from a controlled repository. This allowed the personnel to retrieve software from a backup that was not controlled or cataloged and then allowed the use of the software in a production system.
- The lack of a software version control tool and corresponding processes allowed a unit test Dynamic Link Library (DLL), which is a way to deploy software so it can be used by other software, to be used in production. The unit test scale interface DLL was written in such a way that it provided its expected values if the scale encountered an error.

The investigators interviewed the former software developer. He reported that the manufacturing engineers and he were in dispute regarding the reliability of the scale firmware (firmware is software that has been committed to a chip in hardware). He believed the scale firmware was not in control. He reported his concerns...
and was told to work around the problem. He created code that simply ignored a malfunctioning scale and supplied the parent program with historical successful values. This allowed the system development to proceed without dependency on the scale. Evidence was found in various bug reports that this software engineer reported these problems. It appears, in part, that his release from the project was due to his reporting of poor controls.

The scale firmware was also not version-controlled. This allowed scales on the new line to have old firmware put into production. This firmware had a defect that in certain conditions, like the ones triggered by hopper shape knife gate valve maintenance, caused the scale to recalibrate. The original developer attached the new firmware to the bug report, but that report was closed after his departure. Due to the lack of version control and formal procedures to control the validation and deployment process, incorrect and unsuitable versions were deployed.

The result of the inadequate software version control, deployment practices and hardware/firmware version control allowed approximately five pounds of the Pinkoswill active ingredient to be added to the three affected lots. Company chemists and lab personnel acknowledge that this is, at minimum, a serious overdose risk to patients.

The three lots were able to escape into the supply chain due to the lot-sampling plan being incorrectly constructed because of a side effect of the test software. When in testing mode, the problem DLL did not send lot information to be included in the lot sampling plan. Although the lot sampling plan was a validated approach and relied on a risk-based analysis, that analysis did not identify any configuration management risks or failure of the scale system to properly function. The failure to identify and manage risks associated with configuration management fails to comply with the regulations.

The investigators noted that, per the regulations, the company had an obligation to prevent mix-ups. The lack of management controls and adherence to basic controls around software and firmware versioning fell below minimum standards for industry.

LESSONS LEARNED AND AREAS OF CONCERN

In most life-sciences organizations, management comes from scientific, sales, finance, or other non-software or system development backgrounds. As a result these organizations often do not have adequate system development controls in place. There are also many times when organizations do not see themselves as needing to practice software and system development at anything more than “it seems to work.” Where does your organization fall?

Software and systems have become pervasive in organizations from controlling quality systems to production lines to devices instrumental in patient care. Failures in systems and associated controls can and do lead to patient risks. Does your company have adequate tools, controls, and management review?

Systems today are very complex. Much of the software and systems are assembled by contractors that often leave when the project ends. Is there a change control record? Is there a version history with accounting of all the changes? This is extremely important. It is important to know when changes are made and why. In the story presented in this article, Alicia was “given” a piece of software. She did not know where it came from, who wrote it, why, or when. It was test software but only the departed contractor knew that. Alicia had no knowledge of the bug in the scale firmware, and due to pressure, the “working” system was released with a test software component that simply reported to the parent program a weight it was programmed to return if the scale firmware had an error.

Software Version Control

If the Pharma154 Company had software version control and was using it properly, this scenario would have been prevented. Software version control provides key benefits that comply with good automated manufacturing practices (GAMP). These include the following:

- Frequent check-in and checkout (daily) of work. This provides clear visibility and accounting around who made changes and when
- Good process ties check-ins to a stimulus (i.e., requirement, work instruction, bug, or task)
- Labeling (i.e., production version, test version, development version)
- A central and controlled repository where all software or firmware is stored.

Computer Systems Do Not Always Work

Companies today need to recognize that computer systems range from your Smartphone to lab equipment to manufacturing control systems. As these devices have become pervasive, there is a tendency to just assume they work and work together. In many cases, they do not. Bugs exist, incompatibilities exist, and often the formal structure of good version control and software/system best practices is not in place on internal projects. Some organizations confuse software-development-life cycles (SDLC) for software development best practices. However, most SDLCs are focused on an artifact trail to satisfy regulation rather than on ensuring best or essential practices are in place. Organizations need both sound SDLC that ensures key steps and artifacts are executed appropriately and methods and procedures to ensure essential practices are in place and practiced.

This can be particularly true when non-software and system development professionals are running projects. Today, there are many tool kits from leading vendors that allow users with no formal training in
system development to create powerful and complex systems—others in the organization then usually delight because “it works.” However, there are real risks in life sciences if those systems get used for quality or manufacturing purposes as bugs, version problems, or validation leakages (i.e., intended or actual use cases that do not traverse the full validation cycle but end up in use) may affect safety or efficacy of processes, devices, or drugs.

CONCERNS AND ACTIONS
Companies should take a good look at the software and firmware systems they have in place and what the associated regulations are in regards to those systems.

You Should Be Concerned If…
If your team does not have a software or firmware configuration management (SCM) system, that they use everyday, you should be concerned. If your team does not have a defect management system that they use everyday, you should be concerned. If your team does not have a formal way to label the version set that represents specific and frequent points of time you should be concerned. Often, it is the complex interaction of many pieces that results in an issue.

If you have doubts, get an outside assessment of your firm’s level of practice. Make sure the level of practice, the related risks, and the impact on functional areas are analyzed and understood.

System Design And Control Is Not Optional
Although the specific incident described herein is hypothetical, it is representative of real life. There have been FDA warning letters issued for the lack of these very controls and processes. These are essential and foundational processes that every organization needs to make sure are in place and functioning to stay out of the headlines and away from 483s, warning letters, and recalls.

ARTICLE ACRONYM LISTING

DLL Dynamic Link Library
FDA US Food and Drug Administration
GAMP Good Automated Manufacturing Practice
IT Information Technology
SDLC Software-Development-Lifecycles

ABOUT THE AUTHOR
Robert H. Smith is an application technical lead responsible for quality systems software development at Abbott Vascular. Prior to this, he was Sr. Director, Engineering at Symantec Corporation, where he was responsible for developing enterprise client, host, and server-based corporate security products as well as the Symantec and Norton Live Update offering. Robert has 25 years of software development experience including VC start-ups funded by The Mayfield Fund, Granite Capital and Wasatch Venture Fund, and holds CISSP and PMP credentials. Robert can be reached at robert.smithii@av.abbott.com.

ABOUT THE COLUMN COORDINATOR
Barbara Nollau is a director of supplier and alliances quality at Abbott Vascular. She has 26 years of experience and increasing responsibility in the pharmaceutical and medical device industries, spanning the areas of manufacturing, quality assurance and compliance, validation, and information technology. Ms. Nollau can be reached by e-mail at barbara.nollau@av.abbott.com.

Originally published in the Summer 2010 issue of Journal of GXP Compliance
The Nine Most Common Computer Validation Problems—
Identify Frequent Deficiencies to Accelerate Your Validation Projects

Frank Houston

“Computer Validation Forum” discusses topics and issues associated with computer validation in order to provide useful resources for daily work applications. This column provides readers information regarding regulatory requirements for the validation and qualification of computerized systems.

Your questions, comments, and suggestions are required to fulfill the objective for this column. Case studies submitted by readers are welcome. Please send your comments to column coordinator Sharon Strause at sastrause@aol.com or to coordinating editor Susan Haigney at shaigney@advanstar.com

INTRODUCTION
What validation problems are you likely to see over and over? When tackling complex validation challenges, you’ll save time, money, and headaches when you know the most common problems and where to find them.

The following analysis is based on validation work performed for a large US Food and Drug Administration-regulated company. The goal was to bring the company’s software validation evidence up to the level of FDA’s current expectations as well as those of the client’s own independent auditor.

Our efforts yielded 1,720 observations. As part of a “lessons learned” review, these observations were grouped into 22 different categories. The documents that most frequently contained the observations were identified. The results, in the author’s experience, are typical of the problems most companies face.

APPLYING PARETO ANALYSIS TO COMMON VALIDATION PROBLEMS
Through Pareto analysis of the categories of problems, it was discovered that about 80% of the observations were clustered around nine types of deficiencies as plotted on Figure 1. This case was an exception to the 80/20 rule, in that the top nine problem areas represented about 41% of the categories.

The following were the most frequent deficiencies found:

- **Missing information.** Documents or records omitted fundamental information or content that should have been included.

- **Inconsistency.** Documents contained statements inconsistent with other statements about the same topic in the same document or in the same validation package. What’s more, no explanation or reason was given for the difference. Jargon, varying terminology, and contradictions in logic frequently caused these kinds of inconsistencies.

- **Lack of needed detail.** This deficiency applied most likely to requirements documents. The requirements in the validation package did not adequately describe the characteristics of data, user interactions with business processes, or key processes internal to the software.

- **Traceability.** We found three frequent traceability problems:
  - The traceability matrix did not account for a traceable specification or an observation step in a test script
  - The trace was broken. Either a requirement was barren (lacked decedents or a test) or one of the detailed requirements or test results was an orphan (lacked a parent somewhere in the requirement tree).
  - The traceability matrix was incomplete. Requirement details were not explicitly numbered and traced to associated test steps. Requirements were not traced at a detailed level, so the reviewer need-
ed to infer the detailed links between specifications and steps in a test script.

- **Vague wording.** Documents used generalities such as “in accordance to an approved procedure,” or “applicable regulatory requirements,” or “all associated GXP and business processes.” In addition, documents used vague words such as “may,” “possibly,” “more or less,” and “approximately.”

- **Unverifiable test results.** Expected results were not described sufficiently so that an independent reviewer could compare and verify actual results. The IEEE Standard for Software Test Documentation, Std. 829.1988, Clause 6.2.4 (1) states, “...provide the exact value (with tolerances where appropriate) for each required output or feature.” For executed scripts, actual results were not recorded or captured in a way that allowed an independent reviewer to compare them to expected results. For example, “OK” was noted in the actual-result column with no reference to a screen shot.

- **Good documentation practice (GDP).** The following three frequent good documentation practice problems:
  - Hand-recorded data and testing evidence, such as test results, were presented in a way that could cause doubts about their authenticity (e.g., cross-outs without initials, date, and reason)
  - Data that confirmed a specific requirement was hard to find in the evidence provided (e.g., a busy screen shot crammed with data)
  - Handwritten corrections were made that changed the sense of a requirement or an expected test result, but no discrepancy report or change request was filed (e.g., changing an expected result from indicator “Off” to “On”). In GDP, hand corrections are allowed without additional documentation only for obvious typographical errors, such as dropped or transposed letters (e.g., correcting “th” or “teh” to “the”).

- **Incomplete testing.** Test scripts did not fully or adequately test the associated requirement.

- **Ambiguity.** Text could be interpreted more than one way, so it did not establish a single, unique requirement. The words “either” and “or” in a requirement are strong clues the text is ambiguous.

**ADDITIONAL OBSERVATION CATEGORIES**

Beyond these top nine categories, 13 other categories of observations were identified. These category definitions may seem to be somewhat subjective, but for this sort of analysis the objectivity of the definitions was less important than consistency in classifying the observations. For this reason, all the classifications were reviewed several times before locking in the data for the lessons-learned pivot tables. Even so, it was noted that between the “Ambiguous” and “Vague Wording” classifications, many observations could have fit in either one.

The following additional categories of deficiencies (i.e., ones that did not rise to the level of our most common findings but were still worth noting) were identified:

- **Compound requirement.** Requirements that were not unique; that is, the requirement statement actually stipulated two or more system characteristics. (When the predicate of a requirement sentence contains “and” or a series of commas, or when the requirement is presented as a compound sentence or series of bullets, it’s probably a compound requirement. This deficiency was often coupled with traceability problems.)
Figure 2: Top document types.

- **For your information.** Here comments on the potential to improve a document or process were included. The issue that generated the comment may or may not have had an impact on a determination of “substantial compliance.” Remarks on particularly good examples of documentation or development practice were also included.

- **Incomplete requirements.** Findings in this category fell into the following four subcategories:
  - The requirement in question implied another requirement, possibly complementary, that needed to be explicit to ensure verification
  - Regulatory impact analysis and risk assessment indicated a need for requirements that were missing from the user requirement specification (URS)
  - Requirements in a software requirements specification (SRS), a software design specification (SDS), or a configuration specification (CS) were not sufficient to address the associated URS item. This deficiency was often associated with a broken trace
  - System and business process analyses indicated the software had functionality that was used but had not been described in the URS

- **Rationale.** Statements or assertions were made without supporting rationale or justification. Or, the rationale or justification for a particular statement or assertion was not persuasive.

- **Lack of acceptance criteria.** Test and validation plans did not establish objective criteria based on the outcomes of various tasks in the validation process, such as vendor audit, testing, and problem resolution. The plans did not include criteria for assessing the seriousness of deviations as a basis for the overall evaluation and acceptance or rejection of the test and validation results.

- **Lack of process for resolving deviations.** A plan, protocol, or script lacked a process for resolving deviations (e.g., failure to meet expected test results, discovery of unanticipated behavior, or deviations from GDPs).

- **Questionable statement.** A statement appeared to be inaccurate or incorrect.

- **Redundant requirement.** The same requirement appeared more than once in a specification document.

- **Topical inconsistency.** The text within a topic pertained to a different topic.

- **Typo.** Typographical errors were observed.

- **Unsupported deviation.** The summary document omitted reporting on differences between planned activities and those that were actually carried out.

- **Not testable requirement.** The requirement was not presented in objective, observable, or measurable terms. In other words, the requirement did not describe a system response or characteristic that a reasonable person could sense or measure.

- **Violation.** The text set up or highlighted a violation of procedures or regulations.

These categories should be considered nothing more than suggestions or starting points to create a list of observations. As experience is gained, the list may need to be revised to cull out some categories and/or identify new ones.
Identifying the Most Vulnerable Documents and Records

Taking the next step to document the lessons learned from this project, the documents and records where the most frequent deficiencies were found were categorized. It was discovered that about 85% of findings were concentrated in six key documentation areas, as shown in Figure 2.

The following were the top types of flawed documentation:

- Specifications (including user requirements)
- Test scripts
- Validation plans
- Test plans
- Trace matrix
- Test results.

Although the exact order of problem areas may differ in any individual organization, it’s likely these same six documentation areas will float to the top. From the author’s experience, specification documents are usually the biggest pitfall for most companies.

FEWER VALIDATION PROBLEMS AND INSPECTION SUCCESS GO HAND-IN-HAND

After auditing many companies, large and small, and participating in countless remediation projects, it was found that the results described in this article are typical of companies worldwide.

More importantly, the author has seen first-hand that companies who reduce the frequency of these problems with focused remediation efforts are much more likely to weather future FDA inspections. It can be reasonably assumed the same would be true if the frequency of such problems were low in the first place.

It is recommended that companies use these results and definitions to assess their own validation projects, or devise their own categories and charts to pinpoint the company’s most common problems. Either way, you’ll have a major headstart in better allocating validation resources and making needed improvements quickly.

REFERENCES


ARTICLE ACRONYM LISTING

- CS: Configuration Specification
- FDA: US Food and Drug Administration
- GDP: Good Documentation Practice
- SDS: Software Design Specification
- URS: User Requirement Specification

Originally published in the Summer 2009 issue of Journal of Validation Technology
Accurately Identifying Your Requirements—Will Any Computer System be Right for You?

Janis V. Olson

“Computer Validation Forum” discusses topics and issues associated with computer validation in order to provide useful resources for daily work applications. This column presents information regarding regulatory requirements for the validation and qualification of computerized systems.

Your questions, comments, and suggestions are required to fulfill the objective for this column. Please send your comments to column coordinator Sharon Strause at sastrause@aol.com or to journal coordinating editor Susan Haigney at shaigney@advanstar.com

KEY POINTS
The following key points are discussed in this article:

• A clear statement of requirements is fundamental to determining what you want and what you need
• Write your requirements so they are unambiguous, complete, consistent, and testable
• The quality of your computerized system will be a direct result of getting quality requirements written
• All system users should have input into defining the requirements
• Map the current process or processes the computerized system is designed to replace. Incorporate any regulatory, statutory, and/or standards requirements.
• Optimize the process or processes you want to use
• Write your intended uses and requirements for the system in terms of how you will be able to test that the requirements are satisfied
• Write requirements for how the system should not work
• Review all requirements with all levels of users.

INTRODUCTION
Requirements are the foundation for determining what you want and what you need. People, in general, do not write down their needs, wants, and intended uses of the things they buy. Some do extensive research by going shopping, reading information, or searching the Internet. Others buy the first thing that appears to meet their needs. Others buy what everyone else seems to have bought, thinking that if it meets other people’s needs, it will satisfy them. Often, different people have different requirements and understanding of what is really needed. The only way to resolve the conflict when purchasing computer systems for regulated industries is through written requirements.

Writing requirements can be very difficult. Vague statements of goals and needs are often expressed. Statements like “user friendly,” “easy to use,” and “intuitive to the user” are often seen but rarely defined. Requirements must be written so they are unambiguous, complete, consistent, and testable.

DETERMINING THE REQUIREMENTS
The quality of your computerized system will be a direct result of getting quality requirements written. I have not used “user requirements” because those are only one part of all the requirements you need to document. Requirements should specify what the user and business need, not the abilities of the various products available.

For more Author information, go to gxpandjvt.com/bios

ABOUT THE AUTHOR
Janis V. Olson (Halvorsen) is Senior Validation Consultant at EduQuest, Inc., a global team of FDA compliance experts. Sharon Strause, the column coordinator, is a Senior Consultant with EduQuest, Inc. Sharon may be reached at sastrause@aol.com.
This is the only way to assure the system chosen meets your real needs. Too often, I have seen companies buy a software package or tool to automate one of their critical systems only to find during installation and testing that the system does not meet their needs and does not have a critical (to them) capability. For example, I saw a company try to add, at great expense, the capability of a complaint system before their requirements had been established. A year later, the company gave up and bought a different software package just to handle complaints—now that they understood their needs and processes. The total cost of ownership is affected by your ability to identify, right at the beginning, the product that meets the needs of your business and the users in the business.

The following are some steps to get you started in determining the requirements needed.

**Have All The Users Of The System Represented**

Users are defined as the people who will interact with the system. Users include those who input, change, and review data (i.e., users); receive reports from the system (i.e., users, managers); maintain the system (i.e., information technology department [IT]); manage and change the system (i.e., IT or super users); business owners; etc. Have focused meetings with users to understand their needs and how they see the system operating. Do not have meetings that only include one type of user. Cross functional meetings are needed to assure that conflicting requirements are identified. Get the users to be specific about their needs and wants. Write down what is said and what the system is required to do.

**Map Your Current Process Or Processes**

No matter what the current process is, you must understand the flow and interactions both within the system and the interfaces to the system. The current processes may be manual, automated, or a combination of both. Use multiple layers of process mapping to show what is currently done. Include who does what, when, and how, including decision, review, and approval points. Include what is received and what is sent to other processes that are not in the scope of the new computerized system. Understand where the data come from, where the data are processed, and where the results go. Map not only the usual processes but also the exceptions to the current processes when problems arise. As a result, you may discover additional business requirements the new system will need to meet.

**Determine Any Regulatory, Statutory, Or Standards Requirements**

Write any regulatory, statutory, or standards requirements down individually and not by just referencing other documents or standards. These requirements must be stated in the way that you want them implemented in the computer system. For example, stating that a system must meet 21 CFR Part 11, Electronic Records: Electronic Signature (ERES) regulations is not specific enough to assure that the system will meet these requirements. You must be specific. For example, some of the requirements for ERES compliance include the following (see reference):

- Each user will have their own user name and password
- The user’s login user name and password will be the same as his electronic signature user name and password
- Identification of the individual doing work is from their login
- The computer system will check each user at login to determine the operation that can be done and the files that can be can accessed
- All signatures require the user to enter both the user name and password when the user signs a review or approval of an operation. The login process is not linked to the signature.

**Determine The Process Or Processes**

What are the efficiencies the new computerized system will be able to provide? If the current system is manual, the process identifies the person doing an operation by his name, number, initial, stamp, etc. that he must write or place on the paper and date. The computer can identify the person based on his login and can apply the date and time the operations are done. The computer can forward information (e.g., data, documents, requests for action, etc.) to the next person to review or approve without the user having to cause this to happen. The computer can also put data in several places, pre-populate fields with standard information, provide instructions to the user when required, etc. Any redundant operations in the current system may be eliminated by the computer if the process is designed correctly. Additionally, this is the time to optimize your process. One company developed a system to automate its documentation and tracking of corrective actions and preventive actions (CAPA) and had implemented over 60 electronic signatures from opening to closing of a single CAPA report. Needless to say, users of the system were extremely dissatisfied and said there was more work using the automated system than doing the same operations on paper.

**Write Your Intended Uses And Requirements**

Write your intended uses and requirements for the system in terms of how you will be able to test that the requirements are satisfied. Develop scenarios for how the system will be used. These scenarios can be used as part of the performance qualification of the comput-
erized system. Scenarios are often easier for users to review to assure all of their needs are being met by the system. They will help you identify standard operating procedures that will need to be rewritten or written prior to performance qualification.

Requirements For How The System Should Not Work
Write requirements for how the system should not work. Ask the “What if?” question as many times as needed. Conduct a risk analysis for the system and identify mitigations for those risks. Mitigations for the risk identified become requirements of the system. The goal is to assure that the system will fail in a safe manner. Define a safe manner. Safe could mean that the data are not corrupted; that the data are checked for consistency prior to being accepted; the user receives a warning message and instructions on what to do next; the system flags the fields that have not been completed and are mandatory; etc. Again, develop scenarios for how the system will not behave and assure the scenarios are testable.

Review All The Requirements
The reviews should take place on multiple levels. The requirements must be reviewed to assure they are unambiguous, complete, consistent, and testable. Unambiguous requirements are interpreted the same way by each person that reviews them. One company had requirements that appeared, on first reading, to be well written and unambiguous. However, the following were misinterpreted by the system developer:
- Users will have user names and passwords to operate the system
- Users will be operators, supervisors, or quality personnel.

The resulting system was designed so there were only three user names and passwords the system would accept, one for each type of user, not one for each user. Unfortunately, this was discovered during operational qualification and did not meet the intended needs of the company because it was planning on using electronic records. The company had to continue to use its manual batch history records. Complete requirements cover all aspects of what the system will and will not do. The design of the system will determine what is done by hardware, software, and people following procedures. All the users should review the requirements to assure that all of them have been covered in the requirements document. Consistent requirements do not conflict with one another. For example, one requirement stated that the user will enter the date when the complaintant reported an issue. A second requirement stated that the computer will pre-populate the report date of the complaint with the date the complaint was entered in the system. The two requirements are inconsistent with one another. Neither in itself is wrong, but taken together, the two requirements cannot be fulfilled at the same time, and one must be changed. Testable requirements can be tested singularly and together to determine if they are met. For example, stating that the user will enter complaintant information into the system without defining the type of information is not testable. As long as any information is entered, no matter what it is, the test would pass, even if there is not enough information to respond to the complaintant. Generally ambiguous requirements are not testable.

SUMMARY
Because the quality of a company’s computer system can directly depend upon the quality of the established user requirements, it is important to be as specific as possible when creating a list of written requirements. Requirements should include all user needs and regulatory and standards requirements. Written requirements should be clear, complete, consistent, and testable. Establishing these requirements before a system is purchased can save a company money in the long run.

REFERENCE
FDA, Title 21 Food And Drugs, Chapter I—Food And Drug Administration, Department of Health And Human Services, Subchapter A—General, Part 11 Electronic Records; Electronic Signatures, April 1, 2009. JVT

Originally published in the Winter 2010 issue of Journal of Validation Technology
Welcome to “Computer Systems Quality and Compliance.”

This column discusses the quality and compliance aspects of computer systems and aims to be useful to practitioners in these areas. We intend this column to be a useful resource for daily work applications.

Quality and compliance considerations associated with computer systems are relevant across the life sciences industries. Understanding the requirements and best practice regarding computer systems is fundamental because much (if not all) of our data and records are electronically created and maintained, and so many of our daily operations are automated. Computer systems have rapidly evolved, and industry and regulatory guidance regarding their use has evolved as well.

This column addresses computer systems quality and compliance with real life scenarios and challenges in mind. It is our intent to present these topics clearly and in a meaningful way so that our readers will have a basic understanding of principles, and then be able to apply these principles in their daily work applications.

Reader comments and suggestions are needed to help us fulfill our objective for this column. Suggestions for future discussion topics or questions to be addressed are requested. Case studies illustrating computer systems quality and compliance issues by readers are also most welcome. We need your help to make “Computer Systems Quality and Compliance” a useful resource. Please send your comments and suggestions to column coordinator Barbara Nollau at barbara.nollau@av.abbott.com or journal coordinating editor Susan Haigney at shaigney@advanstar.com.

SUMMARY
The following are key points that should be considered in computer systems quality and compliance:

• An evolution has occurred regarding thinking and terminology from software validation to computer systems quality and compliance
• Computer systems include software, hardware, operating system, technical infrastructure, use and maintenance processes, and the people who use the systems
• Computer system quality and compliance includes all the activities associated with acquiring or developing and deploying a system and then maintaining it until eventual retirement
• A true quality system builds quality in because it is the right thing to do, not because we are obligated to do so—because obligation typically doesn’t foster the same level of commitment
• Computer quality and compliance best practice is to apply quality principles and practices with respect to all the elements of the computing environment across all phases of the system life cycle
• When systems or technology services are purchased from outside vendors, the client company must gain assurance that the supplier has built quality into the product they are selling
• Building quality into the system results in systems that are reliable and compliant.