The Struggle for "Perfect" Software: Compliance Considerations

Software development industry customers are beginning to be more specific about their expectations for software functionality and development.

by
Gloria Metrick, MBA
Owner
GeoMetrick Enterprises
and
Nadejda Segal, Ph.D.
Information Services
Senior Consultant
Apotex Inc.

As software continues to become more complex and widely used, software users worry more about finding ways to ensure high quality. One would first think of the software development industry as being a driving force behind finding ways to allay these worries. Instead of the software development industry acting alone, it appears to be the software users, along with their regulatory bodies, are also taking a great interest in this issue. Software development industry customers are beginning to be more specific about their expectations for software functionality and development. These users are being driven by the regulations to work together with the software development industry to achieve a "perfect" level of software quality. In cases where the vendor does not provide this quality, the customer must drive it there.

First of all, there are two aspects to the definition of "quality:"

❶ We think of high quality as having software with few or no errors. The process of reducing software errors in computer code is commonly known as debugging.
❷ Increased quality also indicates receiving extra functionality (i.e., getting more options and more choices), in that it provides us with more ways to map and control the work process.This is called development initiation.

There are two major choices for the customer in software development initiation and the debugging processes.These choices are "buy" versus "build," although there is an intermediate option where the customer purchases software and initiates some amount of development on their own. With these as the choices, let us discuss the first choice as the "buy" option, then the second as "build."

The "buy" choice indicates that the software developer (i.e., the software vendor), itself, initiates software development. A customer chooses the software and takes it "off-the-shelf" if it fits his expectations and plans. Buying the software off-the-shelf, the customer expects that the software quality in his environment must be achieved with his participation. As suggested by many of the regulations, the customer expects that the quality of the software that functions in his environment is his responsibility, not that of the software vendor.The customer then becomes a major power behind driving software quality up to the "perfect" level if the vendor has not already begun this process toward quality.The customer starts pitting the software against real-life examples, usually using examples from his own process model. He discovers software defects and weaknesses that, in the vendor's modeled system, are difficult, and sometimes even impossible, to observe.

In order to reduce the expenses for software support, as well as to attract more buyers, the vendor has a certain interest in reducing problems in the software; however, if the vendor does not show the appropriate level of interest in obtaining this quality, the customer will become a motivating influence toward this quality.This is because of the customer's ultimate responsibility for the system's performance, as dictated by his regulatory bodies, with the regulations being a response to society's demand for high quality. Even though the Food and Drug Administration (FDA) recently announced the withdrawal of some of its guidance documents related to Electronic Records and Electronic Signatures, and has shifted primary responsibility for implementing Part 11 to the Center for Drug Evaluation and Research (CDER), the regulation itself has not been withdrawn. The unusual step of withdrawing a guidance and shifting the implementation to CDER suggests the Agency is significantly rethinking its approach to Part 11 enforcement, but the regulation remains the same.

In the "build" choice, the customer orders the software from the developer, whether the developer is internal (the customer's Information Technology (IT) department) or external (using a consultant, or in partnership with a software vendor). In this case, both the customer and the developer work together from the very beginning of the software lifecycle. From the beginning of the software's inception, the process can become more focused on the creation of the software, than on its quality, just as this can be an issue with the "buy" choice.

In either choice, software developers can become more interested in the completion of development, and bringing the product to the customer. If this happens, though, the customer must focus a transition to the improvement of its quality. As an industry, software developers also tend to place a heavy emphasis on technology over usability.[1]

Regardless of the choice, a version of the software becomes selected to be purchased or built for the customer with the intention of installation for the selected end-users, and will be subject to FDA, European Union (EU), and/or Canadian Therapeutics Products Program (TPP) inspections, if it relates to a GxP process.

Some History of Software Development

The software profession is not what we would consider a new profession, but it is not a mature profession, either.[2] In the early days of computers, there was just "programming" (the act of writing programs) as opposed to "software development" (the process of managing the creation of software). Coding was not necessarily planned out to any great degree, and the activity was limited in what it could produce by the lesser capabilities of early computers and tools. Many years later, we have computers on our desktops that would have qualified as super computers years ago, as well as countless software development tools and methodologies to choose from. Despite that, there is still a great deal of plain "programming" continuing to occur.[3] That is to say, there are still pieces of software that come across our desktops that do not have the structure and reliability that we would desire. As late as 1993, the Institute of Electrical and Electronics Engineers (IEEE) Computer Society and the Association for Computing Machinery (ACM) created a steering committee to help "establish…software engineering as a profession."[4]

Over the years, the software engineering profession has become just that – a profession as any other profession where the goal is, ideally, to improve the professionalism of work done. The systems we create with code have grown to enormous proportions to meet, and in some cases, surpass the capabilities of our modern desktops. We then become concerned that this profession leads the way to promote structures that will heighten the professionalism of development by the creation of more reliable systems with higher quality. We also hope that this profession will be able to keep the development of software elevated, so that it is a matter of applying the proper procedures, as opposed to merely problem-solving (i.e., the difference between controlling the process, versus merely patching each problem that arises).

Current Issues

One of the unfortunate consequences of each new wave of technology is that when a new wave hits, it is sometimes used to justify the abandonment of the latest development processes, only to become apparent that there are few revolutions to the actual development process itself. It becomes clearer, then, that high-quality development seldom occurs without appropriate processes attached.

Documentation

One argument against controlling software processes suggests that the development process could overwhelm the actual development effort, causing development to become so bogged down by the actual process that progress stops or is, at the least, severely impeded.[2,3] This would be a valid argument, but for the fact that development standards are living documents that should illustrate practical procedures. They are not meant to prevent progress. The idea being that, instead of throwing them away, these documents must be updated as needed to reflect an appropriate process. Another factor to this is that many projects leave the documentation until the end of the project. If procedures are followed from the beginning, documentation can be managed, along with the actual programming or configuration. If documents are left for the end of the project, they will hold up the project.Then, documents become an afterthought, and no longer part of a managed process.

Others claim that these processes are no longer needed, as the newest wave of tools is self-documenting. In software development, there have been efforts to justify each new tool as self-documenting, including older programming tools.[2] None of these tools force a developer to create code that is legible or manageable. Indeed, the person that developed Assembler (a cryptic code that operates at a lower level than the tools that most modern projects are using) likely argued that it was self-documenting, as it was much easier to read than all those zeros and ones. Merely putting comments in code to indicate what the programmer has done is not enough. One of the most unfortunate outcomes of allowing software to emerge with nothing more than program comments is that you now have documentation only of what you have, not of what you were supposed to create.

One resource suggests that self-documenting code is a goal that we should strive for, in order to create better code, but one that will probably never be realized.[5]

A Lot of Work and is Not Much Fun

The most popular argument against using software development procedures is that they are a lot of work and are not much fun. This particular argument is entirely true. Most software developers are attracted to the profession, as it is full of problems to solve. Problem-solving is fun and documentation is work. Of course, the vote is tilted toward the fun part and against the work part.

Customization

Another wrinkle to our projects is that much of the software we work with (e.g., Laboratory Information Management System [LIMS], instrument software) is often neither custom software, nor do we install it directly “off-the-shelf,” as there are often modifications to make to them, such as macros. This has left these software installations in a type of never-never land, where it appears difficult to apply a typical software development methodology; yet, there is often some development that occurs. Many installations suffer either from over-management, which burdens the installation process (thus, threatening project completion), or from virtually no management (leaving the control of the process mainly unfulfilled).

Fast, Cheap and Good

As the market continues to demand faster shipping of new versions and “fixes,” the software industry continues to struggle to balance getting software out fast, and getting it out “good” (i.e., with a high quality). In the project management profession, there is the saying that a customer can ask for the project to be “fast, cheap and good,” but anyone can really accomplish two of the three at the same time. That certainly applies to software projects.

We must consider, then, how we can obtain consistently high-quality software.

On the curve, the optimum correlation between quality and expenses is represented by the area between point number two and point number three. Notice that as expenses continue to increase after point number three, quality does not continue to rise much per the added costs. If expenses are not adequate, the quality does not raise high enough (point 1) as shown in *Figure 1*.

Both the customer and the vendor must then consider how the high quality of software can be obtained, so that an affordable system is produced.

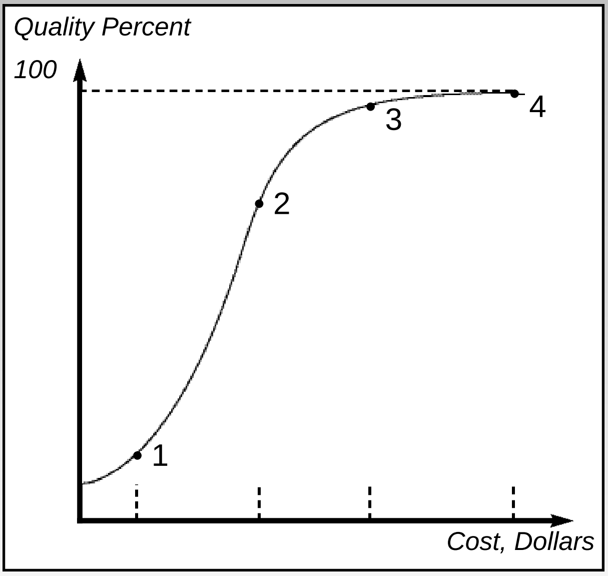
Understanding the Software Profession

Many Opinions

There are always new methodologies coming, with Scrum and Crystal Clear[8] being just a sampling of some new names, along with names you might recognize, such as Object-Oriented, Joint Application Development/Rapid Application Development (JAD/RAD) and Structured Analysis.[9]

Figure 1

Predicted Growth of a Complex System Quality Cost*



The predicted growth curve for complex systems quality expenses was estimated by acquiring published and calculated values on the software market through last three years.

There are multitudes of software societies and universities that publish on this topic. In fact, it is not a lack of opinions that is of issue, but that there are so many to choose from.

Being that there is much information on the topic of software quality, the problem may be that software development projects suffer either from a indecisiveness on what paths to try — that we become overwhelmed with the choices, and decide to try none of them.

The Right Criteria for "Software Development" or "Software Engineering" Professionals

Part of the struggle involves personnel, of course. Software professionals are often certified on tools, but not on the development process, itself. It is difficult to build a team with personnel that have no experience working with software development tools and methodologies. It is easy to ask a recruiter or software services vendor to send a person with skills in particular tools, but difficult to develop the right criteria for a person experienced in "software development" or "software engineering."

Not only is it difficult to come up with the appropriate criteria to ask for, few of us know to ask specifically for those skills. A question as simple as, "Generally describe the last methodology you used on a software development project," might give a new insight into the interviewees that pass our way. Without asking for those skills of the personnel on our projects, we cannot criticize them for lacking these skills.

If internal personnel, there is a chance that this information can be gathered by questioning their co-workers, project managers, and supervisors. For external personnel, the interview questions must be more rigorous.

Who is Driving Quality?

The question remains as to whether the enforcement of regulations influences the development of software.The answer is, "Yes." Despite the personal opinions of any developer, as well as the methodologies the developer does or does not adhere to, the developer must attempt to satisfy the checklist sent by the potential customer and the quality audit performed by the customer; otherwise, the software might not be purchased. In the customer's quality checklist, the developer must select the checkbox against almost each question regarding the available quality options required by the regulations. If the developer is not as interested in quality as the customer is, the customer might have a lot of influence in order to improve its products' quality, but it becomes a power struggle.The most powerful, organized, and persuasive customers will hold fast to get the type of development structure that they believe will produce the highest quality of software. These customers are up against powerful, organized, persuasive developers, who are trying to push their own mode of operation; hence the customer can become involved in a struggle for "perfect" software, both individually, and by trying to influence the software developer.

Over the years, as per the customer's requirements, the developer has been required to extend the software's functionality to meet the most modern, extra requirements, such as security and electronic audit trails in order to fully replace paper. It does mean that the software covers more required functions, but also how these functions perform is an issue.

So, software becomes bigger, both by the addition of more required functions, as well as the included safeguards to prevent user error in the "core" functions of the system (i.e., each function becomes more complex as it attempts to prevent the user from using it in any unintended manner). It becomes more expensive to create, test, and maintain. If it is the customer's software, the entire burden rests on the customer. For purchased software, the expenses are maintained by the software vendor, but passed onto the customer in the purchase and maintenance price, as well as the other expenses of maintenance that will rise for the customer (e.g., revalidation becomes more cumbersome as the system gets more complex). Someday, the situation could reach the point when companies will not be able to afford more for software perfection. If that were to happen, the companies would quit paperless technology, returning to the traditional paper technology, and some probably already have gone back to paper systems from paperless systems that became so costly, they could never be finished and put into production.

On one hand, this means that regulations have to balance between the real and relatively achievable requirements. Both types of industries can find this equilibrium involved in the software quality improvement, if these industries work together. On the other hand, the requirements cannot be ignored if these are human health and safety matters. Here the software must strive to be perfect with no compromises, and this is the responsibility of the customer to obtain said quality.This will not be achieved between the customer's industry and the software development industry, generally, but between the individual customer and the developer of their own system, although heavily influenced by the opinions the customer sees within its industry. Even within purchased systems, customers do not necessarily work directly with the vendor on all changes, but typically with a selected person or group of individuals either sent by the vendor, or brought into the project directly by the customer.This can cause a variance in quality coming in a vendor's different customer projects, as each person sent by the vendor could potentially have different training and dissimilar experiences. Each vendor addresses or does not address this issue of consistency in their own manner.

Regulatory Requirements

Regulatory requirements for software validation were analyzed by the FDA and other regulatory bodies and published several times beginning in 1996.[10] Later, software validation principles were published and took effect.These include all software lifecycle activities and references to software development methodologies. The regulatory standards requirements being established for every stage in the software lifecycle and for development methodology implementation are discussed in this article.

Regulatory Requirements for Methodology

These published requirements include all software lifecycle activities and references to software development methodologies. Regarding the methodology that can be followed in each particular case, there are no recommendations, but there is a concern of implementing a chosen methodology accurately during the entire software lifecycle.[11]

Now it is a regulatory requirement to formalize the development process to expedite this control. For years, developers successfully avoided restrictions and monitoring, counting the development process as highly creative, and not able to survive under controls and limitations. The development process has been recognized as a creative one. The reactions of the developers to these controls are that they feel restricted and anchored, being allowed no creativity. Creativity is not gone, though, as it is can coincide with their development skills, working knowledge, and expertise. Software development is not hindered by these controls, when the developer is highly skilled. Furthermore, choosing the proper methodology for projects helps developers to organize the process, and be more productive. Regulation documents describe some productive methodologies that can be selected for development projects. To name a few, the waterfall, spiral, rapid prototyping, incremental development, software lifecycle models are defined in the FDA's *Glossary of Computerized System and Software Development Terminology*, dated August, 1995. Many lifecycle models are described in various references listed in Appendix A of *General Principles of Software Validation*, 2002. Due to the fact that some regulations could be interpreted broadly, a variety of concerns has been raised.The agency "believe that some of those broad interpretations could lead to unnecessary controls and costs and could discourage innovation and technological advances without providing added benefit to the public health."[13] As a result of significant discussions, the agency has proposed "a risk-based approach," and intends to interpret the scope of Part 11 narrowly during the discussion period. But under the narrow interpretation of the scope of Part 11, records must still be maintained or submitted in accordance with the underlying predicate rules.

Requirements for Terminology

A glossary is an important part of any software validation documentation. Commonly used terminology is not finalized in the area of software validation and development. Due to the fact that some definitions found in the regulatory documentation can be confusing when compared to commonly used terminology in the software industry, it is recommended that a glossary of terms be created for each project.The customer side is mostly interested in such a document being created and referenced during the project and inspection. This facilitates the communication process between the customer and the vendor or inspector, and can be based on the *Glossary of Computerized System and Software Development Terminology*.[11] Some examples of common misinterpreted terms are shown in FDA's *General Principles of Software Validation*,[12] and relate to requirements, specification, verification, and validation. Regulatory bodies perform inspections based on the project's documented definitions of terms.

As suggested by many of the regulations, the customer expects that the quality of the software that functions in his environment is his responsibility, not that of the software vendor.

Regulatory Requirements for Quality Planning, Requirements, and Design

Regulatory documents consider that "ensur[ing] accuracy, reliability, consistent intended performance…"[13] is part of the software development task, itself.This is the reason behind creating structured programming and standardized development in the first place.

Development project activities, such as quality planning, requirements, and design all are described in detail in the regulatory documents, and are requested to be executed, even though they are generally advised to do so by customers. The design, based on well-documented requirements, is also requested to exhibit a high degree of visibility. The relationship between the different levels in the design and traces to requirement documentation is recommended to be clear. Changes to the design in all design documents are prescribed to be easy to implement. So customers are forced to be aware of software design quality, and evaluate design as complete, correct, consistent, unambiguous, feasible, and maintainable. "Software design evaluations may include analyses of control flow, data flow, complexity, timing, sizing, memory allocation, criticality analysis, and many other aspects of the design. A traceability analysis should be conducted to verify that the software design implements all of the software requirements."

Regulatory Requirements for Coding Evaluation

For "off-the-shelf" software, the vendor does not typically release the source code for the customer. But as per regulations, not only the developer, but also the customer is expected to perform code and coding standards documentation review in order to ensure the coding standards are followed, and that code has been thoroughly reviewed, and errors were examined by the developer's side.This gives the customer some degree of assurance that errors were detected and minimized before code execution. The customer's evaluation of source code is often implemented as code inspections and code walkthroughs during vendor audits. A source code traceability analysis is also one of the vendor's audit elements.These verifications are performed not only by the developer, but also by the customer, and illustrate the fact that code is linked to established specifications and established test procedures.

Regulatory Requirements for Developer and User Testing

Software testing strategies and plans are recommended to be written for each project in order to illustrate each particular task that is to be conducted at each stage of development or user testing. These plans include justification of the level of effort and risks represented by their corresponding completion criteria. The testing process is based "on complexity, criticality, reliability, and/or safety requirements."[12] The regulation references well-known software engineering literature, and describes categories of software and software testing that occur in software engineering literature. Regulatory documents also mention different types of testing, including references to the traditional IT terms of "white box" and "black box" testing,[13] as well as, the idea of "change control" and regression testing.[13] These "tools" are all part of the long-standing software development process – a process that the FDA appears to be finding a need to spell-out more specifically.

Regulatory Requirements for Risk Analyses

The amount of structural testing coverage, loop, path, data flow, functional coverage, etc. is recommended to be investigated and documented. The complexity of the software and its criticality determine the degree to which testing needs to be extended as part of the validation effort in order to produce a reliable product. Note, too, that the regulatory suggestion is that more functional testing might be required of a system where the "source code or development documentation is not serviceable e.g., for most Commercial Off-The-Shelf (COTS) software, and for some contracted software.[13]

External Control

The next question to ask is, "How can you control the software that comes across your desktop?" Since much of the software we use is purchased software, sometimes with our own modifications, many of the things we must do to promote software quality have nothing to do with our own software development procedures.

At one time, there was the impression that locating a software vendor that was ISO-certified was an indicator of high-quality, and have since realized that the issue is more complicated than that. Then there was a trend toward requiring everyone that worked on our projects to be "certified" on particular tools. This, too, turned out not to be a simple indicator of quality.

Currently, those customers that purchase software often "audit" a vendor to verify that their software has been developed in a controlled and appropriate manner.What if the software requires modification to operate on the customer's site via macros or changes through other software development tools? Keep in mind that the software must always stay under control, which might mean that bringing on an external person to help in the process will require more rigorous exploration, such as in-depth interviews, an audit of the software services company or group sending the person out, or a combination of the two.Then it is up to the customer to determine whether they need personnel that know a particular software program or tool, and whether these people know said program or tool well-enough to be able to work with it in an appropriate manner. Additionally, it would be up to the customer to verify that these individuals had other skills that might be deemed necessary, such as software development skills. Remember, as well, that a software vendor's software development methodologies typically apply only to the creation of the software, not in its customization for a specific customer's use. Also consider that the personnel sent to modify the software are not usually trained to the same extent that those creating the software would be, nor using the same, if any, software methodologies.

It is important not to be coerced into relaxing your own procedures, unless you have solid reasons to know that the person you are talking to has the experience behind what they are saying. As always, outsiders are not responsible to the FDA for your system. At the same time, be open to discussion on what is reasonable.The idea that the safest route is to check "everything" is the road to another problem – a project that is never completed, costing the company a great deal of money for a system that will never be ready. Additionally, if the burden to validate a system is so great that it appears insurmountable, it is possible the project might never begin – a situation that is not uncommon.

Internal Control

Once your company begins to develop its own software or modify purchased software, it is important to create your own controls.

Steps Toward Software Quality:

- Regardless of the approach, it pays to retain software professionals who have participated in projects with a high-quality outcome. Ask their advice. They should be a good resource for ideas.
- Select the approach for the project ahead of time. Stick with it, but if it is failing, make a planned shift to a more appropriate approach.
- Start to create a project glossary, as well as a naming convention, as early as possible in order to communicate with all participants in the same language.
- Create process documents ahead of when they are needed (e.g., the requirement process, the programming process, how they all

fit together). Start far enough ahead that they can be reviewed by the team and adjusted.

- Create software quality documents ahead of when they are needed (e.g., requirement document standard, programming standards). Start far enough ahead that they can be reviewed by the team and adjusted.
- Reviews: Every step needs independent reviews. Everyone makes mistakes and omissions. No document should move forward without at least one other person reviewing it and signing-off, but too many reviewers can be as bad as too few. The reviewer(s) must sign-off on the fact that they thoroughly reviewed the document, and believe it to be complete and accurate. Nobody should be signing-off as a reviewer for their own document. In the theme that adding "eyes" to the process helps locate problems early on, try to split pieces of the process from each other. For example, in the extreme of the programming process, the person that writes the program design would not be the same person that writes the program. The person that writes the test plan would not be the person that wrote the program. This set of divisions aids the process of locating problems, but practicality maintains that this type of structure is too severe for smaller projects.
- Responsibility: Every project level needs sign-offs. The people creating and reviewing a document should sign-off that they truly believe it to be complete and accurate.
- Review the above steps in the most critical way. Verify that the project structure is not too lax, as that does not promote the best software quality. Also verify that the structure is not overwhelming, as software quality can fail as team members try to cut corners on structures that are too burdensome, as well.
- Buy-in and support: The project manager and/or team managers must support the strategy chosen, stand behind it, and make it clear that it is expected. The team members must be taught to understand that this is their process. They created it and must follow it. Management should not create the process for them, but should constantly reaffirm its importance.
- Process Failure: When the process fails, do not abandon it, but adjust it as often as needed, albeit not in a haphazard manner. Document its adjustments and distribute this documentation to all team members. Communication is an important part of the process.
- Throughout the process, do not allow documentation activities to fall behind.These require early planning and fulfillment in order to be effective and efficient. The dangers are either that it will hold up the project finish or, worse, that it will never be done.

Software quality is a great deal of work. Is it worth it? Where it regards our health and safety, the answer is, "Yes!" ❏

About the Author

Nadejda Segal, Ph.D., is currently an Information Services Senior Consultant for Apotex Inc., a generic drug manufacturing company. She provides computer systems validation, quality system assessments, audits, quality assurance, and training services. She has a total of more than 20 years of combined experience in validation, research, and teaching. Her areas of expertise include software validation of systems falling under 21 CFR Part 11, and complying with GxPs and biochemistry. She can be contacted by e-mail at nsegal@apotex.ca.

Gloria Metrick, MBA is the Owner of GeoMetrick Enterprises, a consulting company that helps people understand and manage their laboratory data. Her areas of expertise focus on preparation for and implementation of LIMS, as well as software development process and coding standards. She has been working with LIMS since 1987. She may reached by phone at 517-347-1181, or by e-mail at gloria@GeoMetrick.com.

References

1. Norman, D. A.The Invisible Computer. The MIT Press. 1988.
2. Constantine, L.L.The PeopleWare Papers: Notes on the Human Side of Software. Prentice-Hall. 2001.
3. The New Methodology:The New Methodology http://www.martinfowler.com/articles/newMethodology.html
4. Software Engineering as a Profession: History of the Joint IEEE Computer Society and ACM Steering Committee for the Establishment of Software Engineering as a Profession. http://www.computer.org/tab/seprof/history.htm
5. The Principle of Self-Documenting Code:The "Principled Programming" Project. http://www.developerdotstar.com/principle6.htm
6. Scrum Methodology: Scrum Methodology. http://c2.com/cgi/wiki?ScrumMethodology.
7. Extreme Programming – A Gentle Introduction:

Extreme Programming – A Gentle Introduction. http://www.extremeprogramming.org/
8. Crystal Clear Methodology: Crystal Clear Methodology. http://c2.com/cgi/wiki?CrystalClearMethodology
9. Pierson Applications Development: Training, Mentoring and Methodology. http://www.embeddedtechnology.com/ecommcenters/pierson.html/?ATC~C=009+S=001+R=001+L=a
10. Title 21 Code of Federal Regulations (CFR) Part 820, and 61 Federal Register (FR) 52602. *Federal Register.* October 7, 1996 respectively.
11. FDA. *Glossary of Computerized System and Software Development Terminology. Division of field investigations, Office of Regional Operations, Office of regulatory Affairs*. August 1995.
12. FDA. *General Principles of Software Validation; Final Guidance for Industry and FDA Staff*. Document issued on: January 11, 2002.
13. FDA. *Guidance for Industry 21 CFR Part 11; Electronic Records; Electronic Signatures Scope and Application*. February 2003.

Suggested Reading

- FDA.Department of Health and Human Services, Part II. Final Rule. *Federal Register*. Vol. 62, No. 54. 1997.
- Guidance for Industry 21 CFR Part 11; Electronic Records; Electronic Signatures. *Glossary of Terms.* http://www.fda.gov/ora/compliance_ref/part11.htm.

Article Acronym Listing

ACM:	Association for Computing Machinery
CDER:	Center for Drug Evaluation and Research
CFR:	Code of Federal Regulations
COTS:	Commercial Off-The-Shelf
EU:	European Union
FDA:	Food and Drug Administration
IEEE:	Institute of Electrical and Electronics Engineers
IT:	Information Technology
JAD:	Joint Application Development
LIMS:	Laboratory Information Management System
RAD:	Rapid Application Development
TPP:	Therapeutics Products Program