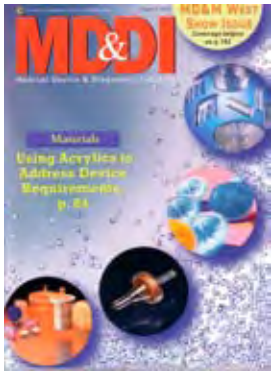
Using Design Controls to Reduce Time to Market

by David A. Vogel, Ph.D.
Intertech Engineering Associates, Inc.

as published in Medical Device & Diagnostic Industry, January, 2007



The secret to expediting your product to the market is to increase efficiency. Design Control requirements ensure good engineering practices, and thus increased efficiency and shorter development cycles.

FDA's quality system regulation (QSR) is often perceived by medical product developers as unnecessary overhead. Software developers, in particular, often see it as a nuisance that delays the completion of the development of the product. Actually, just the opposite may be true. Understanding and following the good engineering and project-management principles outlined in the QSR can help manufacturers shorten the development cycle. Not only is the time to market faster, but the end result should also be a more stable, more reliable, and higher-quality product. The difference in both timeliness and quality can be enormous.

Subpart C of the QSR focuses on design controls. For software, the design control requirements are further interpreted in FDA's "General Principles of Software Validation," often referred to as the GPSV.

An obvious question is, "How can following regulations help me get products out to the market faster?" The answer is simply that design control requirements add no more overhead than would be required by a good engineering process. A well-defined engineering process facilitates communication between stakeholders and development team members and leaves a documentation trail for those who will maintain the product in the future.

Design controls require that medical devices be developed using a formal, documented engineering process of the device manufacturer's choosing. Following a defined engineering process is far more efficient than defaulting to uncoordinated development.

COMPANY PROFILE

Intertech Engineering Associates, Inc.

Address: 100 Lowder Brook Avenue
Suite 2500
Westwood, MA 02090
www.inea.com - (781) 801-1100

Industry: (Electro)Medical Devices

Services: Assessments
Training
Consulting
Hands-on Engineering

Skills: Product Design
Risk Management
Requirements Engineering
Electronics Development
Software Development
Software Verification and Validation
Production/Quality System Software Validation

The QSR requires developers to plan before they act. A nonsoftware engineer once equated the discipline of software development to that of pouring concrete. One would never mix and pour concrete before developing a plan and building forms that would define the shape of the concrete. Both the plan and the forms are needed to guide and control the concrete so that when it hardens, it conforms to the desired shape. Without forms or molds, it would be difficult to shape and reshape poured concrete so that it would ultimately harden as desired. Once concrete has hardened, it is impossible to reshape it. (See the sidebar, "The Value of Design Controls: A Concrete Example.")

The same concepts apply to software. Software without a detailed set of requirements or specifications is like wet concrete without forms. It must be shaped and reshaped, designed and redesigned, until it hardens. If the end result doesn't meet the customer's needs, software, like concrete, becomes difficult to change late in the project. Patched software is much like patched concrete. It seldom looks as good as the original, and the structure is usually less sound.

Design controls for pouring concrete would require the following steps:

- Identify the user's needs.
- Develop a plan for achieving the desired result.
- Build and verify the forms before the concrete is poured.

None of these steps would be considered nonessential overhead for pouring concrete. Everyone would agree that following this process would get the project done rapidly, with a high-quality result. Unfortunately, the same cannot always be said for software development.

FDA established the design control requirements for good reasons. As the agency notes:

> Since early 1984, FDA has identified lack of design controls as one of the major causes of device recalls. The intrinisic quality of devices, including their safety and effectiveness, is established during the design phase.

This is especially true for medical device software, where, except for controlling what version of software is installed, the only control of quality takes place prior to the release of the software.

The design control requirements in the QSR are straightforward. They require the developer to prepare a plan, analyze patient and user risks, identify requirements, develop a design to realize the

THE VALUE OF DESIGN CONTROLS: A CONCRETE EXAMPLE

To understand what is wrong with the iterative method of designing medical products, one need only look at the analogy of pouring concrete. Imagine a beautifully constructed sidewalk that is allowed to harden before its first use. The contractor designs it in a perfectly straight line, but doesn't understand that the users will walk to different destinations and thus will require branches in the sidewalk. The contractor patches the sidewalk to address this need, but the patching introduces several new cracks and flaws that did not exist in the first implementation.

A landscaper eventually looks at the sidewalk and suggests that some gentle curves in the sidewalk would make the garden aesthetically more pleasing. The contractor jackhammers out half the sidewalk, introduces curves (where convenient for him to implement), and asks the landscaper to sign off on the changes. The landscaper doesn't like where the curves were placed and recommends moving them about 20 feet farther down the sidewalk. The contractor reluctantly obliges, but in the process destroys more of the lawn and introduces more flaws to the patched concrete.

Ultimately, the customer is not happy because the sidewalk has cracks and bumps where the patches don't join perfectly. The lawn is destroyed, the project took too long, and it all cost more than the contractor's original estimate. To top it off, the customer wanted a stamped surface on the concrete, not a smooth finish. However, since the contractor did not ask about the finish, the customer didn't get what he wanted.

When presented with the complaints, the contractor replies that if the customer had simply kept the original sidewalk, there would have been no problems. So is the result the customer's fault? Is it the workman's or the landscaper's fault? Or was it caused by failing to control the quality of the design and development process?

requirements, and implement the design. These steps must be done in a controlled way with appropriate reviews, approvals, validation, and management of the configuration and versioning of the software. In other words, the development process needs appropriate cross-checks. This approach provides a logical framework, yet is not too demanding in the details. The regulation allows a medical device manufacturer to achieve these objectives through any corporate process, following any life-cycle model.

Why Device Manufacturers Should Want Design Controls

The design control requirements read like a good textbook on engineering project management. There are plenty of reasons medical device manufacturers should want to implement design controls regardless of the regulations.

Controlling the Source of Design Inputs. When designing a new product, all stakeholder needs must be considered. That includes the needs of the users (patients, clinicians, and others), marketing (what's needed to be sure it sells), manufacturing, service, training, and legal factors, as well as any important design elements that must be implemented to ensure the safe and effective operation of the device. Of course, FDA is really only concerned with user needs as they relate to the safe and effective operation of the device, but it's clear that the advantages of a controlled design process reach far beyond the regulatory requirements.

A controlled design process identifies as many needs and requirements as possible early in the development life cycle and allows for additional requirements or needs to be discovered as the development process progresses. The reviews and approvals act as a cross-check that all stakeholder needs have been considered and captured, and that they are traceable to detailed design documents and verification tests ensuring that they have been implemented correctly.

Out-of-control design processes often allow programmers to start writing software on day one of the project. At such an early stage, there is no assurance that the programmer has addressed or is even aware of the needs of all the product stakeholders. There are no documented requirements to verify the device against to ensure that user needs have been addressed properly, if at all. (See the sidebar, "Developing Embedded Software.")

The out-of-control design process usually follows an iterative cycle of implementation, followed by exercising and critiquing the software, followed by some less-than-enthusiastic reimplementation (patching) to address the criticisms of the prior iteration. The programmers may have implemented a beautifully engineered software design, but unless they perfectly identified and implemented the needs of all the product stakeholders, they may have beautifully engineered the wrong product.

Promoting Communication with the Project Team. Capturing design inputs is an example of using the design control requirements early in the process to improve the quality and efficiency of medical device development. A prime mechanism for improving the process is to promote the communication of the design inputs to the development team through documentation.

Improving communications through-out the product development (and validation) life cycle can only help make the process more efficient. Yet there is a perception in the industry that documentation adds to the development overhead. A number of factors account for this (false) impression.

The team isn't really following the process. In other words, if programmers start writing the software before the preliminary requirements and design documents are completed, they see no value in retrospectively documenting requirements and designs, and, in fact, they are right. The value (other than regulatory value) is in using the documentation to make the development process more efficient. If the software has evolved iteratively, there is little value in retroactively documenting the process. The residual value is to comply with regulations and to support maintenance of the product. Unfortunately, because

the development team views this as an overhead task, they are not likely to give it the thorough treatment that would be necessary to fulfill even these needs.

Times have changed. Perhaps there was a time 30 years ago that a single programmer could produce reasonable software and maintain it without a formal manage this in his head without documentation simply because the applications were so limited, and a development team often was composed of a single software guru.

Those days are behind us. Modern medical device software often is many thousands or even millions

DEVELOPING EMBEDDED SOFTWARE

The following example examines a project to develop software for an embedded medical device.

A product was to be developed on a custom hardware platform. The medical device company had a team of software developers (Team A), but not enough resources to produce the product in the scheduled time. The company sought outside resources and found them in a contract medical device developer. The contract developer assigned another team of developers (Team B), which was similar in size and qualifications to Team A.

After several meetings, the two teams agreed that the best aproach was to divide the software into two relatively independent pieces, the application code and the platform code. Team A was assigned the application code because it presumably had a better understanding of user needs and was closer to company resources to develop the application interface with marketing and clinical specialists. Team B was assigned the platform layer that became a custom embedded operating system replacement that interacted with the hardware, which was still in development. In the end, it worked out that the two groups each had roughly 20,000 lines of code to write.

Team B spent roughly 4 weeks discovering, negotiating, documenting, and reviewing requirements. There were requirements for interacting with the custom hardware and requirements for interacting with the application layer of software being developed by the other team. Once the requirements had been nailed down, Team B spent 2 1/2 months detailing the software design down to the function level and 4 additional months implementing it. Independent testing took about 3 weeks. The entire project took just over 7 months to document, implement, and test.

Team A did not follow a controlled design process and skipped both the requirements and design documentation, thereby saving 3 1/2 months at the start. The software engineers (programmers) started writing software immediately. Without clear requirements or design guidance, Team A iterated for 11 1/2 months to write the code. Testing took more than 3 months to complete. Team A even had the benefit of the requirements Team B wrote for interfacing the platform layer. Furthermore, as it became more apparent that Team A was slipping off schedule, more of the software functionality was shifted to the platform for Team B to implement.

Team B, which was also hampered by debugging the software on new custom hardware, finished the platform software and handed it over to Team A with a complete set of documentation, test protocols, and test results. Team A worked more than 7 months longer to finish its software -- almost twice as long as Team B.

Team B produced very solid software. Only three defects were found in Team B's platform software during the integration process over the next 7 months. Team A's software was troubled with hundreds of defects that surfaced over their 3-month test period.

What was the reason for the dramatic disparity? The two teams had similar years of experience and were similarly talented. Implementation for Team B took place in a relatively straightforward process. Pages from the design document were simply converted into code. Team B was able to add one more engineer to the project during implementation. This was easily accomplished because the implementation required little knowledge of the overall requirements or design to translate each individual function's design into code.

To be fair, Team A was pressured by company management to begin showing progress by writing code very early in the life cycle. Spending time on documentation was seen as extraneous overhead that would only add to the schedule. Team B had the advantage of not being local and not being subject to the daily oversight and pressures of the schedule-conscious management.

Team A iterated many times: writing software, showing it around to get opinions, then rewriting code to respond to the criticisms. Many of Team A's defects had to do with lack of coordination among the team's developers. Time could have been saved by communicating the intended operation of the software through specifications rather than through prototype throwaway code.

As schedule pressure mounted, Team A was unable to bring on additional resources to help. However, bringing in new resources that late in the project would actually have slowed the team down. Interactions with the new resources would have had to be through verbal communications from memory or from reverse-engineering existing software.

In the end, the stark difference in Team A's and Team B's performance and results didn't result from a disparity of talent or dedication. It came down to planning, communication, and using a well-defined development process -- exactly what the design control requirements of the QSR suggest.

commonly comprise internal team resources and outsourced resources -- both of which could be located anywhere in the world. Multi-time-zone development teams are quite common.

Software development evolved from a one-person task to two- and three-person teams that operated by sharing details over the cubicle wall. Now, with the size and complexity of modern medical software, the development process is uncoordinated and chaotic without formal, controlled documentation of the requirements and designs. Unfortunately, many software professionals have been in the industry long enough to remember the days when software could be developed by two-person teams. It is not unusual for these professionals to think the development and management techniques of 20-30 years ago should still apply today.

Thinking ahead isn't easy. Engineering products is not easy. Gathering, organizing, documenting, reviewing, negotiating, and revising requirements and designs is a difficult and imperfect activity that requires both technical and social skills. Early life-cycle phases require a lot of creativity and imagination to design novel ways to solve a problem and to anticipate what might go wrong. There are frequently differences of opinion among team members. These differences require negotiation and compromise.

Programmers can become quite expert in the complex web of computer languages, tools, operating systems, and off-the-shelf components that are used to create medical device software. Clearly, these skill sets are fundamental to a company's ability to develop and produce software-controlled medical devices. But it must be clearly understood that the skills that are required for programmers and software engineers are not the only skills needed for developing products.

Increasing the Efficiency of the Product Development Process. Increasing efficiency is the key to getting products out the door faster. Gathering requirements, communicating through written documentation, and verification of testing activities are three areas that are critical to increased efficiency.

Gathering requirements early cuts down on the number of implementation iterations. By starting the implementation process before the requirements and designs are complete, people get a false sense that the schedule is being advanced. There is no reason that independent sections of funtionality that are well-defined and designed can't be implemented while other functionality stabilizes. Too often, however, development teams turn the programmers loose prematurely in hopes that they will figure out the right solution in coding language that the team couldn't agree to in English. It is much faster to turn around natural-language descriptions of product requirements than computer code.

Communicating through written documentation increases the efficiency of the development process. Detailed needs and requirements allow for detailed designs. Detailed designs allow for more parallelism in the implementation process because coordination of activities is partially accomplished through documentation.

Early verification testing activities add efficiency and may cut time from the development schedule. Defining requirements early in the development process allows the verification test team to begin development of test protocols in parallel with software development. Early testing of prototype software reveals problems that can be solved early in the implementation process. The parallel implementation and verification activities also result in less testing time at the end of the project when the software is nearly finished.

Do Design Controls Stifle Creativity?

The requirements in the QSR aren't very onerous. They are actually quite logical. Most texts on software engineering and the management of software development projects, though they may differ in the details, agree with the principles of the design control requirements.

A common objection to design control requirements is that they stifle the creativity of the programmers and software engineers. On the contrary, these

requirements do not apply to the research phases of a product's life cycle, only to the development phases. It is important to remember, though, that the research phase does present some opportunities for rapid prototyping and experimenting with product ideas prior to the formal product development process, which would fall under the regulatory requirements.

Creativity should be emphasized early in the development process. Creativity related to what the product will do and how it will interact with users should be addressed in requirements documents that all interested parties have a chance to review and comment on. Creativity related to how the software and data will be structured to meet those requirements should be handled in the design phase and documented in design documents that can be reviewed by the other members of the product development team.

Creativity related to product design should be discouraged during the implementation process. Any creative changes during this phase will be largely hidden from the rest of the development team and from the requirements stakeholders. The implementation phase should be devoted to the implementation of the creative product and software designs of earlier phases. Any creativity in the implementation phase should be limited to creative ways to use the programming language to implement the documented designs. Even then, too much creativity can lead to software that is difficult to understand and maintain.

Conclusion

Although the design control requirements in the QSR are often perceived by medical product developers as unnecessary, this is simply untrue. It is especially important to implement design controls for the development of medical software. Good engineering and project management principles can actually shorten the product development cycle. Moreover, the final product will be more stable, more reliable, and higher quality.

Acknowledgement

David Vogel thanks Eric Smith of Wiklund Research and Development for introducing him to the comparison of software to concrete.

ABOUT THE AUTHOR:

David Vogel is the founder and president of Intertech Engineering Associates, Inc.

Dr. Vogel was a participant in a joint AAMI/FDA workgroup to develop a standard for Critical Device Software Validation which was subsequently included in the IEC 62304 Software Lifecycle Standard. He was also a participant on the joint AAMI/FDA workgroup to develop a Technical Information Report (TIR) for Medical Device Software Risk Management. Currently, Dr. Vogel is a member of the AAMI/FDA workgroup developing a TIR on Quality System Software Validation.

A frequent lecturer for workshops and seminars on topics related to medical device development and validation, Dr. Vogel also is the author of numerous publications and holds several patents.

Dr. Vogel received a bachelor's degree in electrical engineering from Massachusetts Institute of Technology. He earned a master's degree in biomedical engineering, a master's degree in electrical and computer engineering, and a doctorate in biomedical engineering from the University of Michigan.

Intertech Service Offerings:

Risk Analysis and Management
Software Design and Development
Electronic Design and Development
Requirements Development and Management
Documentation and Traceability
Verification and Validation
Evaluations, Reviews, Inspections
Planning
Project Management
Compliance Consulting and Training
Manufacturing and Quality System Software Validation

Leverage INTERTECH's expertise to:

Reduce Project Risk
Shorten Time to Market
Cut Development and Test Cost
Assure Quality Products

ABOUT INTERTECH:

Intertech Engineering Associates has been helping medical device manufacturers bring their products to market since 1982. Through a distinct top-down service model, Intertech offers high-level consulting and hands-on engineering. By balancing technical expertise and practical business experience, we support clients through all phases of product development. While we do make your job easier, Intertech exists not to replace but to partner with clients to help balance the concerns of quality, time and cost.

With considerable experience in FDA regulatory compliance, our time-tested development process can anticipate and solve problems inexpensively on the planning board rather than through costly solutions later in the development, test, or post-deployment phases. By using deliberate processes, Intertech ensures an improvement in quality and can build client expertise.

Call us today for more information or a free consultation at 781.801.1100

INTERTECH Engineering Associates, Inc.
100 Lowder Brook Drive Suite 2500 Westwood, MA 02090 USA

www.inea.com - info@inea.com - Tel: (781) 801-1100 - Fax: (781) 801-1108