Software User Interface Requirements for Medical Devices

by David A. Vogel
Intertech Engineering Associates, Inc.

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

Communicating, developing, and managing software user interface requirements are difficult tasks. Fortunately, guidance documents can help device manufacturers.

Medical devices have to be designed so that people can use them easily and reliably. Human factors considerations have gained increased attention since the release of FDA's guidance document, *Medical Device Use-Safety: Incorporating Human Factors Engineering into Risk Management*, in July 2000. This document, and a 1996 FDA document, *Do It by Design - An Introduction to Human Factors in Medical Devices*, both give useful guidance-level information but are not recipes for successful engineering of human factors considerations. The documents also give many examples of the things that can go wrong with human factors in medical devices.

Human factors engineering must be considered in the industrial, mechanical, electrical, electronic, and software design of a medical device. This article only examines how human factors considerations should be handled during the gathering, maintenance, and validation of requirements for medical device user interfaces that are implemented in software. It won't provide tips on how to make devices easier to use, but it will lay out a methodology for incorporating, tracking, and documenting human factors throughout the development process.

For medical device software professionals, including developers and software validation engineers, it should be evident that there is a relationship between validation and human factors engineering (HFE). FDA defines *validation* in the *General Principles of Software Validation* (2002) as "confirmation by examination and provision of objective evidence that software specifications conform to user needs and

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

intended uses, and that the particular requirements implemented through software can be consistently fulfilled." Clearly, the human factors studies, tests, and design recommendations are focused on only one thing: user needs and intended uses. How those needs and intended uses are gathered, documented, and communicated are the focus of this article. Verification that those requirements can be "consistently fulfilled" also relies on having adequate requirements that are effectively communicated to the verification team. This too is addressed in this article.

A Brief Primer on Human Factors

Step one in human factors engineering is determining the humans for whom the device is designed. Who are the people who will actually interact with the device? The list might include patients, patient family members (home treatment), doctors, technicians, nurses, and service personnel. It may even include production personnel who might deal with human factors issues associated with the assembly, packaging, and transport of the device.

From a regulatory point of view, it might first appear that FDA is only concerned with human factors designs that promote safety and efficacy for the patient. However, care providers may be just as confused by a poorly conceived human factors design, which could lead to errors and unintentional actions and compromise safety and efficacy. The same logic applies to other potential device users.

Human Factors and Software

Human factors engineering as a discipline concerns itself with the study of how users employ and respond to a device. HFE leverages this information to optimize the user experience. HFE in medical devices is somewhat different from that in other industries in one important way: The optimum and preferred designs for a device are not necessarily the safest designs. HFE for medical devices needs to balance the need for safe and accurate use with the desire for efficiency, attractiveness, size, cost, and other user preferences.

Software designers need to understand human factors engineering considerations when planning the software for a medical device. Unfortunately, software engineers are not always the best at human factors engineering. To deal with this, device firms can develop specific user interface requirements.

Developing User Interface Requirements - If They Are Required

It is evident from the few competing considerations mentioned above that many stakeholders, including users, should have an input into the development of requirements for a device's user interface. But the right user interface design is not often obvious at the beginning of a project. Besides human factors, many needs must be considered. Lawyers may have opinions about intellectual property territories or protected trade names that must be avoided. Marketing personnel may have an opinion about what sells. Clinical specialists can provide opinions on terminologies and workflows that are common and well understood in the care environment. Users may be concerned about confusion if other devices have radically different interfaces. Human factors engineers can capture some, but not all, of this information.

For example, HFE professionals understand the users of a device well enough to optimally design the arrangement of information on a display so that it is intuitive to use. These engineers also have the expertise to design user navigation through the interface so that it seems natural and requires only minimal training and documentation.

However, these experts may not be aware of whether a competitor owns patents on certain aspects of a user interface design. They may not have insight into how prior generations of a device may affect next-generation designs. For these reasons, user interface design requires a delicate balance of legacy and improved elements. Too much change will confuse legacy customers who buy new devices. Too little change will create a perception of design stagnancy and can create market problems. No single person, whether a human factors engineer, software engineer, or product manager, is likely to have insight into all disciplines that will have influence on the new product's human factors design.

Many companies take on the human factors design with available resources; others contract with HFE experts to guide them through the process. It is also common for software developers to take on the task of user interface design and development alone. They often argue that the user interface is part of software design and should not be considered a set of requirements. The rationale most often given is that the developers do not want to be prevented from making needed late life cycle changes to the user interface. Embedding the user interface in requirements constrains the developers' ability to make changes quickly and often.

For example, developers often want control over the actual wording of the text that is static on a display. Or they may want control over the wording in alarms, alerts, or error messages. Oftentimes the reason for this is that developers want or need to adjust the text strings to fit available space. In addition, they want the freedom to define the text for error codes because these codes are needed during the design and development phases of the code. Anyone who has encountered an error message similar to "Error 0A2E-Page zero violation at x002567a8H" has experienced a portion of the user interface that was controlled by the software developer. It is loaded with information that is of value to a software developer, but it means little to the device user. It conveys no information on the situation's level of danger or what steps to take for correction.

Software designers need to understand human factors engineering when planning the software for a medical device.

A major reason for having design controls is to limit late life cycle changes made without the knowledge of the development process stakeholders. If one accepts the view of FDA that human factors, which include user interface design, are important to the safe operation of a medical device, then it is difficult to defend software developers' argument that user interface details should not be considered requirements.

Even from a business perspective, it makes good sense to treat the software user interface as a set of well-controlled requirements. User interface design is part science, part art, and part emotion. Getting consensus on a design is time-consuming and therefore expensive, regardless of whether the requirements are defined by outside consultants or in-house teams. Preserving the investment in the collection of those inputs and controlling changes that deviate from those opinions are necessary steps to protect the investment and to reduce the likelihood of having to rework a design that doesn't meet stakeholder expectations.

Communicating and Documenting User Interface Requirements

Regardless of how the requirements are gathered, the team members must communicate with each other to understand user interface design proposals and inputs from the stakeholders in the process. The medium and language used to communicate these requirements can be problematic.

Development and verification test engineers want to see detailed requirements. Developers want the details so they don't need to fill in requirements as part of the design and implementation process, guessing what the various stakeholders want. Verification testers want the details that will help them design test protocols without guessing about how the features of the user interface will be implemented.

The question of how much detail to put in requirements is a source of much debate. However, the following items, as a minimum, should be considered for specifying a user interface:

- Size, placement, and color scheme of the display or subdisplay area.
- Font, font size, and font color.
- Exact text to be displayed and any formatting that is critical.
- Navigation to and from the display element.
- Details of any user inputs that are activated or deactivated in that display mode.

- Details of any animation or live (updated in real time) data values on the display (e.g., "The heart rate display shall be updated once per second to show the instantaneous heart rate as of the most recent interpulse interval" or "The pump shall indicate that the timed delivery is active by displaying a stop-watch icon whose hand rotates 90 degrees every second").
- Any underlying functionality that may cause changes to the display mode.

In contrast to the needs of the technical stakeholders, nontechnical users of the requirements need fewer details. They prefer high-level views of the display designs and navigation through the user interface to see how it is affected by alarms, alerts, notifications, and help screens. They need to visualize the look and feel of the interface, and that is hard for anyone, technical or nontechnical, to do from hundreds of pages of text descriptions.

Consequences of Miscommunication

The inability to agree on communication and documentation levels for user interface requirements is often what tempts developers to simply implement the interface as a means of communicating the interface proposal. As undesirable as this is, it is very common. And there are significant consequences of such actions.

Using operational software to communicate with other stakeholders is probably the least efficient means of doing so. Implementing the interface software is time-consuming and ties up the software developers.

Software prototypes of user interfaces are likely to have flaws. Reviewers may concentrate on the imperfections of the prototype implementation rather than focus on the overall design.

Pushing forward with interface implementation forces the implementer to make decisions too quickly or to skip over areas that are not defined. On-the-fly decisions may not be properly documented and the design may be incomplete. Once the entire stakeholder team is exposed to the prototype design, the design may require changes that are expensive and difficult to implement. There might also be resistance from the software engineers because the code is already written.

Latent defects can be introduced in the code if changes are made too frequently and too quickly. Rapid prototyping software is rarely intended to be used in the final product, but all too often that intent is not honored. Complex software can be designed with many dependencies and interdependencies. Often times these complexities are not well documented. This is especially true when software is designed quickly for prototyping purposes. As the prototypes are evaluated, there may be a tendency to react to criticisms without thinking through the effect of each change, and this can lead to unintentional defects in other areas.

The question of how much detail to put in interface requirements is a source debate

Inadequately documented prototype code does not give the verification test team needed inputs for its test designs. The team is left only with the software itself as a self-documenting requirements specification. Any test activity under these conditions is of little value. The tests will only verify that the software does what it does, not that the software does what was specified or desired by the team.

The software may be written to test user interface preferences rather than for the underlying functionality of the device. This can cause unanticipated rework later in the implementation phase because the functionality is shoehorned in to fit the user interface design.

And it should not be overlooked that this method does not satisfy the regulatory guidelines of the sequence of activities that includes specify, design, implement, and test.

Developing User Interface Requirements That Work

Systematic design and development of software-implemented user interfaces for medical devices is difficult. There is no single medium for adequately communicating the requirements or the design to the broad spectrum of device user stakeholders.

One approach that can be successful is to use different tools and media for different stages of the interface definition. Very-early-stage work in which product-level requirements are verbalized by the team can simply be captured in text documents or requirements management tools. Later, some sort of visual medium or technique is needed to communicate look and feel. Storyboarding on paper, whiteboards, or slide presentations can work for early concepts and may be adequate for simpler interfaces. Later-stage, more ambitious interfaces might require more-sophisticated tools or media. Professional graphics software like Adobe Illustrator can be used to show both high-quality mockups of screen designs and navigation pathways through the interface. These storyboards or flowcharts can become very large for more complex user interfaces. However, they are quite effective as a communications tool that almost everyone can understand.

Rapid prototyping tools are also effective for communicating look and feel. Such tools are quite effective in quickly creating on-screen representations of concepts that are interactive with the reviewers. Unfortunately, these tools often lack a complete documentation trail. Storyboards and flowcharts are much easier to work with in later phases of this process.

Note that rapid prototyping software tools do not suffer from some of the problems related to rapidly developing prototype code that would be used in the device itself. For example, most rapid prototyping tools use proprietary languages, tables, or scripts to create the interactive user interface software. The so-called software is not encumbered with much of the underlying functionality, so changes made to the user interface are less likely to have unexpected consequences elsewhere in the system. Furthermore, even if frequent changes do introduce latent defects, they are less likely to be transferred to the final device because the language used to develop the rapid prototype generally is not used for the actual device.

Regardless of which tool or medium is chosen for the graphic description, the method should be

- Understandable by the lowest technical common denominator on the team.
- Flexible for quick changes based on review comments.
- Representative of what the users will actually see on the hardware display. (Don't ask the team to use their imaginations any more than necessary.)

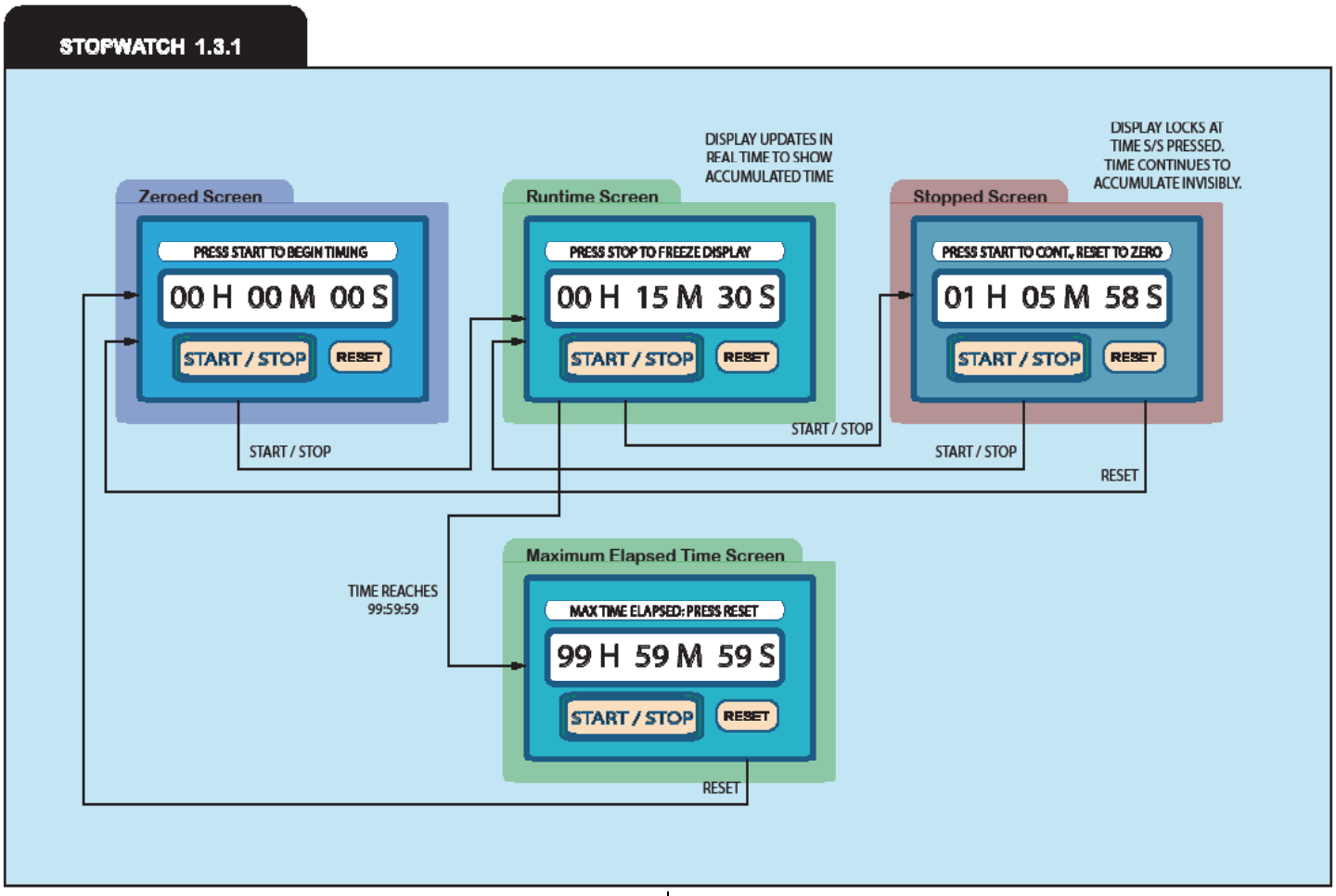
Reviews at this phase should be centered on the graphic tool or medium chosen for the graphic description. Criticisms and comments can be documented in text, but discussions about the operation of the user interface should always be visual and in the context of the graphic tool.

The next stage of the user interface development is to detail all the requirements for the interface and its underlying functionality in a text document or requirements management tool. This step involves refining the look-and-feel concept that the stakeholder team agreed on.

The text description needs a high level of detail for input to the code implementation activity and the test design activity. No detail should be left undocumented. If a detail is not documented, the team has to assume it will not be implemented or tested. The software developers and test developers should be able to design in parallel based on the details provided at this stage.

As an example, consider a screen-based user interface design for a medical instrument. For every screen that was defined in the storyboard or flowchart, the requirements document should specify the following:

- The contents of the display (as taken from the graphic medium).
- The events that would result in the screen being displayed (represented in the graphic medium).



- The events that would result in leaving the display (represented in the graphic medium).
- The user inputs that are armed and disarmed while in the display. (These should be cross-checked against the graphic description for thoroughness. For example, is every armed input represented by an action if the input is activated? Is every input possibility that is not represented in the graphic model disarmed?)
- The underlying functionalty while the screen is displayed.

Figure 1 shows an example of a graphic description for a simple stopwatch function that is implemented in software with a graphical user interface. Each screen is represented on the chart to match as closely as possible the display as it will be seen on the device being designed. Each screen is named or labeled to help create a language that can be used to discuss the design.

The black arrows represent navigation from display to display. The action that initiates the transition to or from a display is annotated on the display itself. Functional requirements associated with a display are annotated in the margins.

Figure 1 is a very simple example. Real-world designs are obviously much larger and more complex. Large application presentations require significant organization to facilitate understanding and to avoid a rat's nest of interconnecting lines. Large applications also need large-format printing to create charts that can be reviewed in a group setting. It is not unusual for these displays to run in multiple panels of 3- to 4-ft lengths.

The important point to note in Figure 1 is that the language used to describe the proposed user interface is universally visual. Anyone can look at the pictures, follow the lines, and understand the designer's intent.

The large-format panels make it easy to jot notes, make changes, or list questions in real time during reviews. The universality of the visual medium makes it easy for all stakeholders to understand and participate in the design process.

The text document can be organized using an outline template to ensure consistency of detail from screen to screen. This document is neither entertaining reading

It should be a goal to communicate with the nontechnical team members using the graphic description.

nor is it a good way to get the big picture of how the interface will react to the user (that is the job of the graphic description of the prior stage). However, the format is invaluable for forcing engineers to think through every aspect of the interface operation before code or tests are written. Such critical thinking usually leads to additional questions and often brings to light missing or flawed information from the previous steps.

The Iterative Process

For iterative refinement of requirements, engineers must go back to the stakeholder team for additional information or to present proposed solutions to problems discovered in detailing the text description. As the refinements are made to the text description, it is important to make the corresponding changes to the graphic description. As long as the two descriptions are kept synchronized, it may not be necessary for the nontechnical team members to review the detailed text description. Removing those extra steps helps streamline the review and approval process. It should be a goal to communicate with the non-technical team members using the graphic description.

It is also valuable to note that the graphic description is important beyond the initial design phases of the device. The graphic description is useful for training purposes for non-development-team personnel and is the best high-level view of the device operation for future development teams that may be called on to maintain the device.

Traceability

To recap, the requirements phase described here for user interface requirements has gone through three stages: the system-level interface requirements, the graphic description, and the text description.

It is important not to lose information from a prior stage as details are further refined. This means that some level of traceability should be in place to manage the requirements. There are challenges in tracing requirements from text to graphics and back to text as the requirements evolve through the three stages. Careful planning and thought about how traceability is to be implemented will affect the organization of the documents themselves.

Implementing a standard for uniquely labeling each screen, navigation path, or event can facilitate traceability to and from the graphic description. Hierarchical organization of the graphic description and the corresponding text descriptions will make the traceability more understandable, thereby making it a useful management tool, not just a regulatory nuisance.

Figure 2 shows and example of how this might work. Again, a sample of a simplified stopwatch implemented in a graphical user interface software system is used for illustration. The top band is the graphic medium description. This is similar to the example used in Figure 1, but this time the navigation arrows are labeled (e.g., 01, 02, etc.). The labels for what action initiates the navigation used in Figure 1 have been removed to simplify the information. Each display or screen is uniquely labeled in the tab for that screen. The annotations above some of the screens often become functional requirements related to that screen. They, too, are given numbered labels. This provides the ingredients for uniquely identifying required elements of the graphic description.

The Software Requirements box in Figure 2 represents the software requirements that could be derived from the graphic description. For each uniquely labeled screen (e.g., Zeroed Screen), there should be some

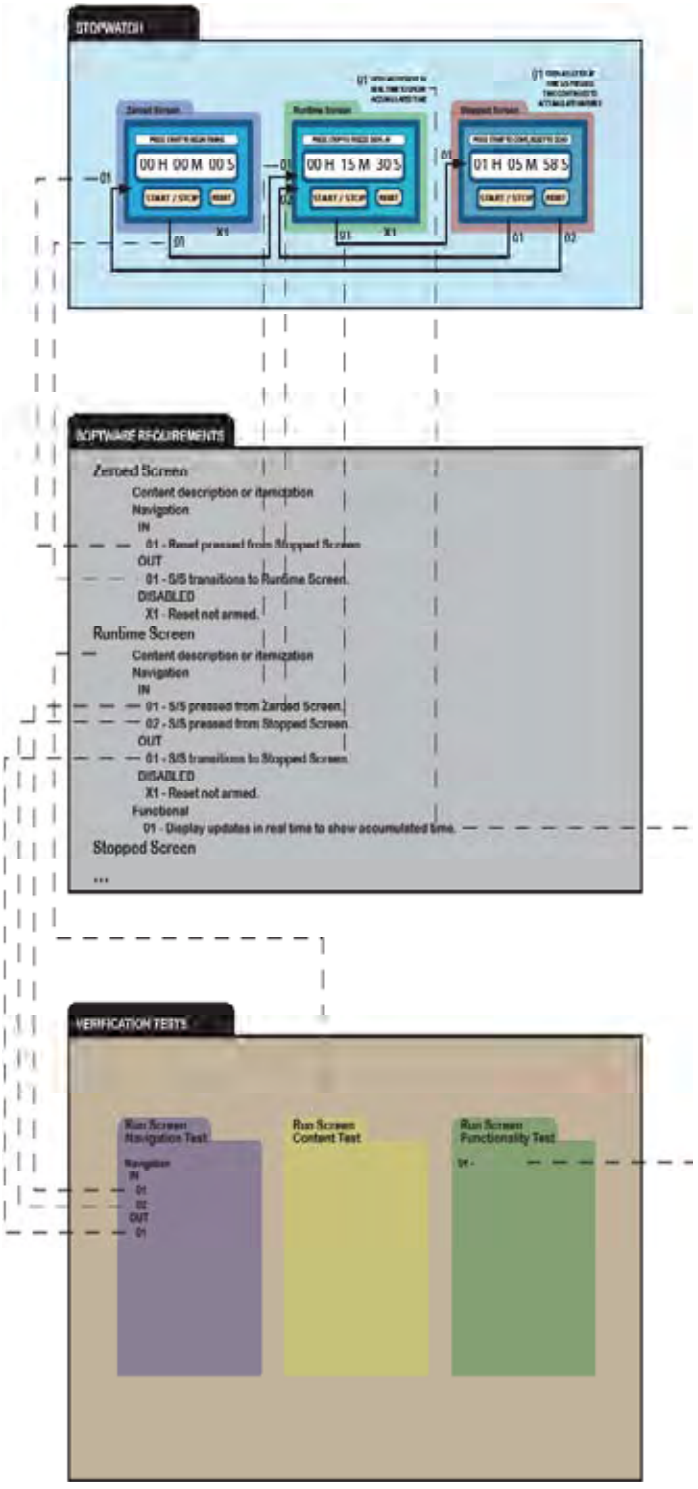
itemization of the contents of the display that ultimately should be verified. Such contents might include any buttons or other user controls, static displays, dynamic displays, banners, positions, colors, or fonts. Navigation to and from the display is detailed using the unique labels described above. It should be noted that detailing "to and from" results in redundant requirements. However, it makes the requirements easier to use and provides a cross-check in the requirements development and review processes.

User inputs that are disabled are just as important to detail as those that are enabled. In this example, they are identified with an X prefix. The lines that drop down in the figure from the graphic description to the software requirements represent the logical linking between graphic description and requirement. Note that the unique labeling implicitly creates the linkage without the use of any tool that provides links between the graphics tool and the text tool.

The Verification Tests box of Figure 2 shows in a sparse form how the verification tests for the user interface might be organized. In this example, the navigational requirements are verified for a screen in one test procedure, the contents are verified in a second procedure, and any functionality associated with the screen is verified in a third procedure. Again, the unique labeling provides implicit traceability. Alternatively, requirements management software tools such as Telelogic's Doors or Rational's Requisite Pro can be used to manage traceability.

Verification Testing

The methodical nature of the detailed requirements of the text description leads to methodical development of verification tests. The process promotes consistency of detail in requirements, design, implementation, and tests. This type of process also lends itself to using templates for each section of the text description, design, or test, thus making the development, review, and approval processes that much more efficient. In reviewing the text description, attention can be focused on the details, because the look and feel were reviewed in the graphic description.



Validation

Validation comprises many activities, including the verification testing just described. According to FDA's *General Principles of Software Validation*,

verification testing provides objective evidence that the "requirements implemented through software can be consistently fulfilled." However, it is quite possible to perfectly implement requirements in software but have a product that fails to meet user needs. For this reason, manufacturers often design a separate protocol of validation tests to provide "objective evidence that software specifications conform to user needs and intended uses."

Validation testing, or usability testing, for user interfaces is a topic complex enough for a number of articles. Suffice it to say that this type of testing should definitely include actual intended users using the device for all known intended uses, in all known intended use environments. For simple user interfaces or for devices used in noncritical applications, validation testing can be designed by a collaboration of those who understand the technical details of the device and those who best understand the intended uses, users, and use environments. Complex or highly critical applications would benefit from testing by organizations of professionals who specialize in user and usability testing.

Conclusion

Development of safe software user interface requirements depends on many human factors considerations that cannot be adequately addressed by software engineers alone. A team of stakeholders of varied interests in necessary to get a balanced input. Communicating with such a broad spectrum of technical capabilities is challenging. A method has been proposed here for facilitating that communication, preserving the agreements that result, and tracing them to the final product.

Because developing cutting-edge medical devices is so complex, it can be easy to overlook the fact that the devices are being designed for users.

Those users' understanding of devices will come from very limited or nenexistent training. They will often be using similar devices with dissimilar user interfaces. Poorly designed user interfaces can confuse nurses and medical technicians who use the devices to provide care.

Besides decreasing productivity, bad design has the potential to harm patients by producing anything from poor diagnoses to injury due to unintentional misuse of a device. Addressing human factors throughout the entire development process ensures that the user interface is as simple and foolproof as possible.

Acknowledgement

The author recognizes and appreciates the efforts of Patrick Wu in helping with the graphics for this article.

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