UBM Canon

MEDICAL DEVICE AND DIAGNOSTIC INDUSTRY

JUNE 2011

MD+DI

NOBLE ULTRALIGHT Athermal Laser Machining
Light Speed: redefined
OUR ATHERMAL LASER PROCESS CUTS NITINOL, BIOABSORBABLE MATERIALS AND DEVELOPMENT TIME.
Norman Noble, Inc.
Micropreceision Medtech Manufacturing

CATHETERS AND STENTS
Testing the Mettle of Platinum in Stent Technology
page 36

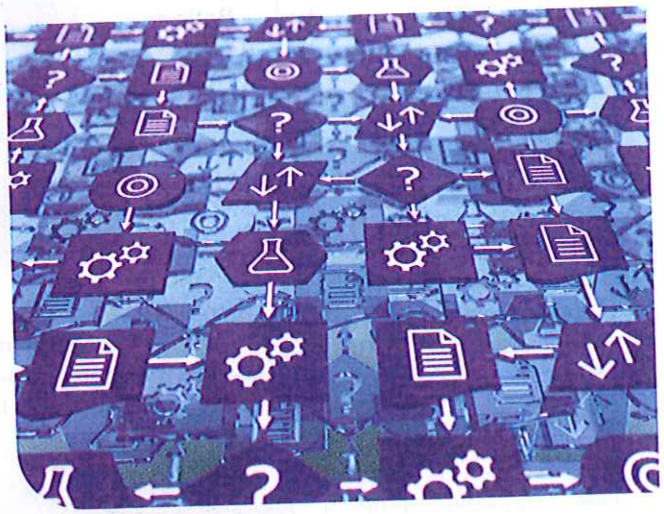
SOFTWARE
SOPs in Software Design: How to Make Them Matter
page 44

mddionline.com

Best Practices for Development of Software in Medical Devices

What is the difference between an SOP that is used and an SOP that gets locked in a drawer? The answer might be open-source project software.

MATTHEW RUPERT



Here's a dirty little secret: You know those standard operating procedures that took so long to create and that everyone was forced to read? Of course you do! And assuming employees do what they are asked, the documents were diligently read. That's the good news. The bad news is that nobody (including the author) remembers exactly what those procedures say.

Medical device software is audited and controlled by standards defined by 21 *CFR* parts 11 and 820. These long documents are oddly phrased and difficult to apply. In many companies, training a software designer on standard operating procedures (SOPs) that satisfy the *CFR* involves having the developer read the SOPs and then answer a few simple test questions about them. The tests are graded, signed, and placed in a permanent file.

A new developer's job likely involves reviewing use cases, writing test scripts, and running the scripts. When one task is finished, the next task is probably whatever the boss tells him to work on. Soon enough, the SOPs that were read during the first few weeks of employment are a distant memory.

Perhaps six months into the job, an internal audit of the project is held. The purpose of this audit is to perform activities similar to those that might occur during an FDA audit for a 510(k). The designer is called into something akin to an interrogation room.

The first question might be, "So, as you work, how do you know what to do?"

An answer from the new designer might be, "Uh, well, I guess I do whatever my boss tells me to do."

Wrong answer.

The auditor continues, "No, what I mean is, how do you know what you are supposed to create?"

Here, the squirming might begin. "Um, because my boss tells me what to do... I read the use case and requirements, and write the test."

The questioning continues for an uncomfortably long time, until finally the auditor gives up and tells the designer to return to his cubicle.

What Was the Problem?

So what went wrong during this audit? The most simple answer is that the designer wasn't prepared. The employee didn't understand the questions, but the overall impression is that the designer doesn't know how to do the job.

The real problem, of course, is the process. The stacks of procedure documents, the training— they all look good. Someone worked very hard to come up with everything needed to establish how projects must be designed and developed per FDA' requirements. However, a vita element is missing: application.

No one on the team is necessarily doing anything wrong. Th designer was just doing whateve the boss said to do. That's a goo thing, right? And the boss was doing wha ever her boss said to do. Also good, right?

Well, not really. No one is following tl SOPs, the very system that was written sp cifically to tell each person what to do. Th all know about the *CFR*, and that procedur had to be created to satisfy it. But the prac cal terms are missing.

Applying SOPs

21 *CFR* 820 covers quality system regu tions (QSRs) for medical devices. It outlir current good manufacturing practices a explains the controls needed as part o quality system, but it doesn't provide ma concrete examples or specifics on how best apply them.

For the purposes of this article, we discuss SOPs only in terms of software velopment projects. It is necessary to h certain procedures laid out that explain l the QSRs are actually performed. To nd, there are two ways to write SOPs, each f which has its place.

The first approach is to tailor a number of rocedures specifically to the project. This ray, designers can handle changing tech-ologies and environments. This approach is ine for a small environment with a few con-urrent software projects that have similar nvironments, but it won't scale. As projects hange, it will be necessary to revisit all of he SOPs. The outputs of this approach can ecome a procedural and documentation ightmare.

The second approach is to create SOPs hat are agnostic in terms of environment nd technology. These procedures are rele-ant for more general use and for a longer pe-od of time. Such high-level SOPs, of course, re not pragmatic on their own. They need nother layer applied in a practical way.

In this second approach, there is no tech-ical constraint by the procedures. The de-elopment teams don't have to lean toward sing a single version control system, da-ibase, ticketing system, or tool simply be-ause it would be too difficult to refine the OPs. The SOPs do not hold users hostage i what they must use; they only define what esigners must do. An SOP that prevents fficient design by prohibiting the ability to volve with technologies and practices is a ad SOP.

Many good developers view *process* as dirty word. Too often, companies let pro-esses become so cumbersome that they on't serve any purpose other than to frus-ate those forced to work within their con-nes. In such a situation, the processes are idestepped, rendering them useless.

So how do you make a good process? How an an SOP be created to serve rather than estrict productivity and good design?

The answer is to let SOPs serve as guide-nes and create work instructions on a per-roject basis that explain the practical use f those guidelines. This is done (as com-rehensively as possible) during the proj-ct planning. The corporate-wide SOPs are pecified to the project, giving designers the pportunity to determine the best applica-ons. For a given SOP, designers must have ne or several work instructions explaining ow to implement the SOP and relevant ap-lications. For example, if a design control OP states, "Versioning control is used for urce code," the project plan (or configura-on management plan) states, "Subversion is used for source code version control. The repository for project X resides at..." The proj-ect plan has the detailed work instructions to help clarify and apply the procedures.

Remember the story about the awkward audit interview? Our hero had read the SOPs but didn't really know how they applied. This is because the SOPs provided a high level of design control, but the practical application lived only in the heads of those who hap-pened to be using the tools for the project. And even if, by luck, an SOP was followed, it was only because the approach happened to be in line with a general guidance and not with the specifics of any work instruction. Those SOPs amounted to little more than lip service paid to the *CFR*.

Work instructions point to the specifics of how and when to use project tools such as Redmine, Subversion, and Hudson (dis-cussed later in this article). Essentially, the SOPs can be thought of as the framework and the work instructions as the implementa-tion. These work instructions should enable designers to use the technology available to make processes applicable. It is important not to think of these procedures as a road-block to work—on the contrary, they should make the work more efficient. If they do not, it's a clear indication that something in the process is off.

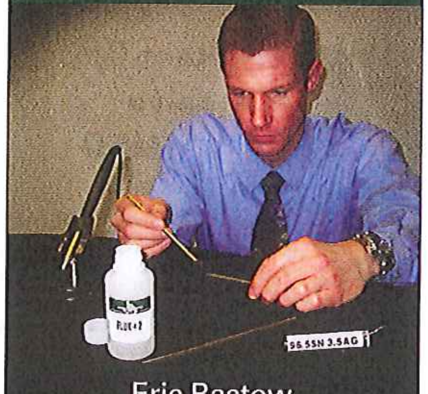
Open Source Is the Key

When it comes to handling design and de-velopment activities of a software project, it is a good idea to choose a plan that works well for any software project, not just one pursuing FDA clearance. The biggest needs of all software development are good com-munication and traceability. Choosing open-source software is often the best way to meet both needs.

Project designers want to know that use cases, business rules, requirements, docu-ments, code revisions, hazards, and tests can be traced backward and forward. Despite having some excellent open-source technol-ogy available (for free), many companies in-sist on manual documentation of such trac-ing. The result is a mess of documentation that is difficult (if not impossible) to main-tain and likely to contain errors. You're just begging an auditor to find a problem.

When used properly, software project management tools can make detailed trac-ing easy and downright fun. So why isn't ev-eryone using them? Some pervasive myths

From One Engineer To Another®



Eric Bastow
Senior Technical Support Engineer
ebastow@indium.com

- Solders and fluxes that work with a variety of materials, such as Nitinol
- Technical and application expertise
- Prototype to production quantities

Find out: indium.us/C204

- answers
- blogs
- tech papers
- one-on-one
- support
- live chat

scan code with mobile device

INDIUM CORPORATION®

ASIA • CHINA • EUROPE • USA

indium.com

©2011 Indium Corporation

and objections can stymie adoption of open-source tools.

Myth: Third-Party Tools are Difficult to Use Within the *CFR*. Some OEMs are under the impression that the *CFR* requires that all software and tools be validated. The *CFR* states that OEMs must show intended use of the software and show that the tools work the way the company thinks they should. *CFR* doesn't exist to encourage poorly engineered software. Nor is it meant to tie users hands from the best tools available. It exists to ensure the tools are used correctly and as intended.

Myth: Open-Source Software Cannot Be Trusted. Evidence to debunk this myth could fill several articles. Widely-used open-source software is often better supported than its closed-source counterparts. Check out the free software provided by GNU for proof (www.gnu.org).

Objection: Technologies Change. How Do I Know Subversion Will Be Used in 10 Years? You don't, and it may not be. But using legacy systems (Visual Sourcesafe, anyone?) is not an obstacle. The technology may change, but servers can be maintained and archived for as long as necessary.

Objection: We Don't Have Time for Such Overhead or IT Support. At the risk of sounding cliché, OEMs don't have time to ignore SOP application. A little forethought streamlines the process and mitigates risk. If a designer ever wonders how to update tracing long after the completion of a software requirement, document change, or test, the procedures have failed. And make no mistake, if there is no practical approach to applying procedures, such an occurrence will happen at some point in the project. Sure, there is some up-front cost to be considered, but it comes with greater efficiency throughout the project life cycle.

The Tools

There is no one-size-fits-all approach. The tools used must be evaluated with consideration to the environment, project, and corporate needs. Although all the tools discussed here work for a wide range of needs, there are many worthy alternatives (Trac, Git, Mercurial, etc.).

The tools discussed in this article are used as follows:

- Subversion enables version control of the entire process. It is available at www.subversion.tigris.org.
- Redmine allows ticketing and includes issue tracking system. Find it at www.redmine.org
- Hudson ensures continuous integration builds. Go to www.hudson-ci.org

These tools serve the needs of a software project, and are widely used and supported. Furthermore, they are free and have been written by some of the best software developers in the world. In the typical corporate environment, each tool can use LDAP authentication. Each tool can be run on Windows, Linux, Solaris, and MAC OSX.

Project Management

The earliest part of any software project is the planning phase. At this stage, the team holds meetings to discuss high level needs. Some presentations and documents are created. Project management plans have not been developed, but they should be thought about. Creation of the work instructions (SOP application) should begin at this stage.

The design history file (DHF) must contain all of the historical data that goes into a project, so even at this early stage it is necessary to decide on a version control system and create a repository. Even though no tracing is required yet, this information should be kept in the DHF. Because the earliest phases of the software project result in outputs that are to be included in the DHF, it is necessary to determine the version control tool and establish the version control repository early on.

Subversion

Subversion lends itself to integration with other tools used throughout the project. When used with issue tracking software, every problem can be linked directly with a set of items in the repository that are related to addressing and resolving that issue.

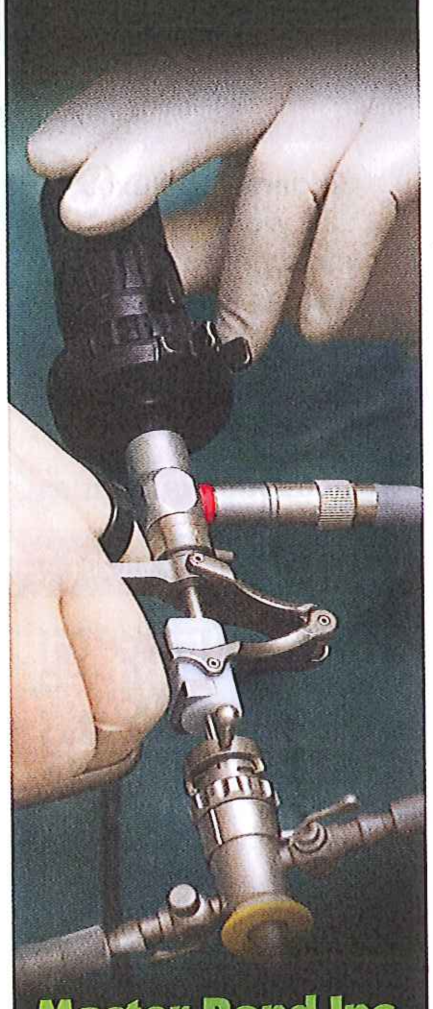
It may be best to use a single version control system and repository for all of the material that goes into a project. Material such as project management plans, documents, presentations, code, test data, and results should all go into the same repository. If documents are stored in one place and software code is stored in another (or in a different version control system altogether), project traceability could be lost.

Subversion is better than many of its predecessors because it uses changesets. A *changeset* provides a snapshot in time of

holding it all together

Heat Resistant Epoxy Adhesive EP46HT-2MED

- Tg over 220°C
- USP Class VI approved
- Sterilization resistant



Master Bond Inc.
Adhesives, Sealants & Coatings

154 Hobart Street
Hackensack, NJ 07601 USA
+1.201.343.8983
main@masterbond.com

www.masterbond.com

the entire project repository. When documents, presentations, or source code are changed and committed to the repository, a new changeset number is created. Users can check out all items in the repository tree of that changeset and designers can pull everything relevant to the project at a specific point of change. There is no need to tag or label the repository to revisit a particular instance in time. Every single commit to the repository effectively results in a tag.

This is not to say that tagging is no longer useful. All software releases, including internal releases, should be tagged. The work instructions tell project engineers when and how to perform tagging.

Subversion also allows for the control and history of directories and files (including name changes). The most commonly used predecessor to Subversion, CVS, did not maintain a version history of a file or directory if it was renamed.

When software is released, it is typically given some kind of version number (e.g., 1.0). This is good, but it doesn't convey the specifics of what went into that build. It's a good idea to include the Subversion changeset number somewhere in the release. A good way to designate the version is to use a build.xml (or build.prop) file that includes the version number of the release, the Subversion changeset number, and the date of the build. The build scripts generate the last two values.

As far as actually using Subversion, within Linux/Unix, all commands are available from the command line. When working in Windows, TortoiseSVN seems to perform well. It integrates with Windows File Explorer, showing icons that indicate the status of any versioned file. It also provides an interface for viewing file differences (even differences in the Word documents) and repository history.

Ticketing and Issue Tracking

It's time to forget the notion that ticketing systems are bug trackers. One of the previously most popular open-source tools, Bugzilla, even used the word *bug* in its name. But issue tracking is not only for software defects. It can be used to address documentation needs, capture software requirements, and handle software defect reporting.

Further, it might be best to get away from using standard documents for the capture of software use cases, requirements, and hazards. By capturing everything related to a software project in our issue tracking tool, designers can leverage tools such as Redmine to enhance team collaboration and project tracing.

Redmine

Redmine has wiki power. Documents include project management details, work instructions, use cases, requirements, and so on. If management is comfortable with it, such information can be placed into the wiki.

All developer setup, lessons learned, and other informal notes can be placed in the wiki, which allows developers to educate other developers. A wiki page can help all developers do their jobs better.

Another feature of Redmine is that users can link to tickets (issues), projects, subprojects, and Subversion changesets. Work instructions should explain that no ticket can be closed without a link to a Subversion changeset (unless the ticket is rejected).

Redmine can be configured to search for keywords in the Subversion changeset commit. For example, if a user is checking in several files that address issue #501, they might include a comment such as this: "Corrected such and such. This fixes #501."

The tool can be configured to look for the word "fixes." Redmine uses the trigger word as a flag to close the ticket and link to the changeset that was created when the user did that commit. Likewise, when viewing the Subversion history, "#501" attached to the changeset will show up as a link to the ticket. The tracing works both ways.

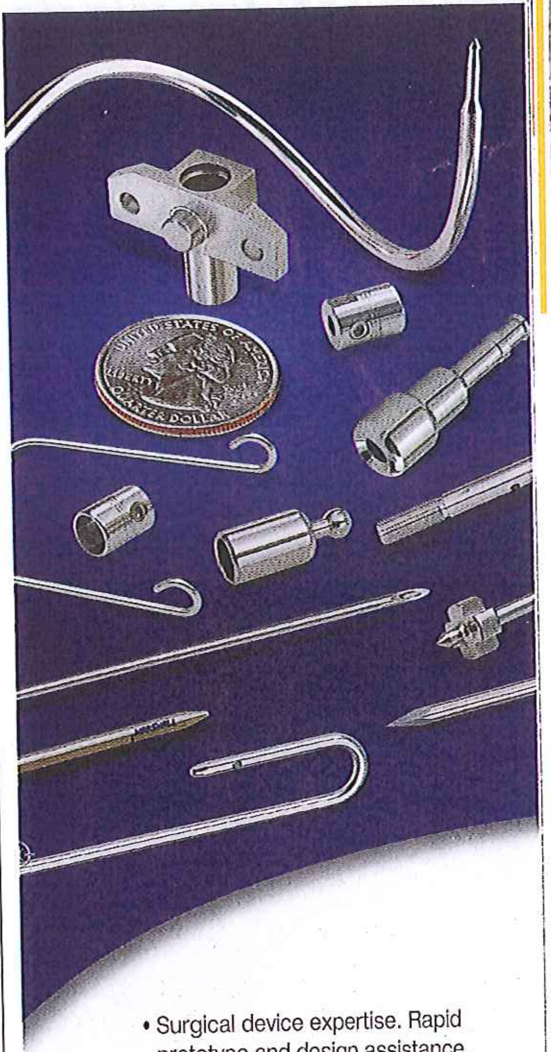
Redmine handles multiple projects and can be used company-wide for development. Each project can be tied to a different Subversion repository. Additionally, a single project can have multiple subprojects. Redmine gives the flexibility to use subprojects for sprints and specific branched versions, as follows:

- Hudson integration. Integrated users don't have to leave the wiki to see how the continuous integration (CI) builds look. A specific CI can be built from any page within the wiki or ticketing system.
- Full configurability. Everything can be configured, including the flow of tickets.

Issue Tracking Systems

It is not enough to leave functional requirements in the specification document. Doing so does not provide sufficient trac-

Just What The Doctor Ordered.



- Surgical device expertise. Rapid prototype and design assistance.
- Custom tips for trocar, bullet, barbed, pencil and more.
- Robotic Swiss Machine cell for 20,000 to 500,000 components.
- Proprietary 2D & 3D bending for wire and tubing. Unique contouring for introducers, tunnelers and disposable instruments.
- Over-molding for handles, and complete one-stop outsourcing.

Phone 1-800-321-6727
email rfq@marshallmfg.com
www.marshallmfg.com

MARSHALL

ing, nor does it provide a clear path from idea to functional code. Instead, designers should institute the following steps:

- All requirements and software design items are entered as tickets. For now they are simply high-level (or parent) tickets with no subtickets (children).
- The development team breaks down each parent ticket into child tickets as necessary. The system should be designed so that the parent ticket (the requirement itself) cannot be closed until all child tickets are completed.
- Hazards (as in hazard and risk analysis) are mitigated by a combination of documentation, requirements, and tests. Leverage the ticketing system to capture hazards and provide tracing. This does not remove the need for a traceability matrix, but it does enhance the ability to create and maintain it. (The Redmine wiki could be applied for use cases, requirements, hazard analysis, software design documents, and traceability matrices, thereby allowing for linking within. Such a concept may be a hard sell to management).
- Not all requirements are functional code requirements; many are documentation or quality requirements. These should be captured in the same ticketing system. Use the system to label the type of a ticket for different categories.

Tickets are created, closed, and modified throughout the project design and development phases. The project plan (created before any code has been written) explains the order in which tickets need to be done, focusing on the highest-level tickets. It might be best to use some sort of iterative approach (and allow the development team to use subiterations, or sprints).

Any activity that results in a design output should first be captured as a ticket. The goal here is not to describe the process, rather, it is to illustrate the fact that a ticketing system can and should be used for all project activities.

Hudson and CI Builds

All software projects should have CI builds. A build can be traced to a Subversion changeset, and therefore to the ticketing system. The CI environment, at least with regard to the ongoing development of code, gives a single point of overview for all other activities. Changesets, tickets, build status, and test coverage are visible. With the proper add-ons, users can gain insight into the quality of the code being developed.

The ticketing system must be used wisely. With Redmine or other systems, designers capture elements of software requirements and software design as parents. These parent tickets have one or more subtickets, which themselves can have subtickets.

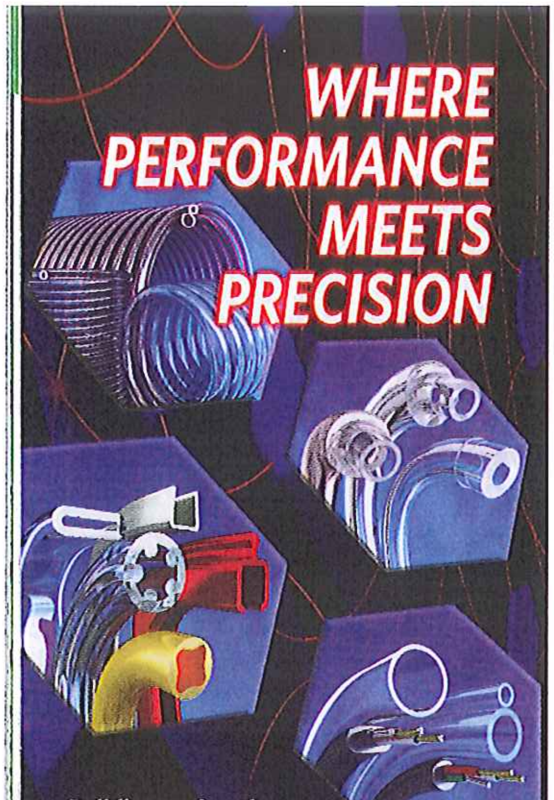
At its most basic level, Hudson runs whatever script it is told to. Hudson enables users to log the outcome, keep build artifacts, run third-party evaluation tools, and report on results. With Subversion integration, Hudson can display the changeset relevant to a particular build. It can be configured to generate a build at whatever interval is wanted (e.g., nightly, hourly, or whenever there is a code commit).

Any time a code commit of significance is done, it is good practice to check the CI build for success. If the build is broken, the user can immediately start correcting the problem (and if it cannot be corrected quickly, the user can roll the changeset out so that the CI build continues to work until the issue is fixed). Hudson can be configured to e-mail team members on build results (e.g., if a build breaks).

The build artifacts of a project, in general, should not be a part of the repository. Build artifacts, such as test results, compiled libraries, and executables belong in the CI build tool, where they can be viewed and used by anyone on the team.

Too often, a released build is created locally on a developer's machine. This is a serious problem, because there is no way of knowing what files were actually used to create that build. Was there a configuration change? Was a bug introduced? Is it an incorrect file version? Although developers have good reason to generate and test build locally, a formal build used for testing (or, more importantly, release) must never be created locally. Never.

Build artifacts are not checked into the source control repository for a number of reasons, but the most important one is that developers never want to make assumptions about the environment in which those items were built. The build artifacts should instead remain in the CI environment, where the conditions within which the build was generated are understood.



Building on decades of global innovation - and with ISO Certified facilities - Natvar is perfectly positioned to deliver precision custom polymer extrusions for a range of tubing and device applications in the medical marketplace.

- Medical PVC, PE, PP, PU & Nylons
- Co-Extruded
- Bubble/Bump
- Thermal Bond & Coiled
- Micro-bore
- Wire Co-Extrusion
- Multi-Lumen
- Profiles

CERTIFIED
ISO 9001:2008
Certified Facilities

City of Industry, CA 909-594-3660
Clayton, NC 919-553-4151

A LINK TO LIFE
NATVAR
TEKNIPLEX
Global strength. Superior solutions.

Email: natvarsales@tekni-plex.com
www.natvar.com

Also, because these builds remain easly accessible and labeled in the CI build environment, any team member can access any given build. It may become necessary to use a specific build to recreate an ssue that has been released for internal or external use. If the label of the build and repository changeset number are known, users can pull the correct build from the CI build server to recreate the necessary conditions.

Run Unit Tests Over and Over. Developers should do whatever they can to keep the CI build from breaking. Common reasons for breaks include the following:

- Forgetting to add a necessary library or new file.
- Forgetting to commit a configuration change.
- Accidentally including a change in the changeset.
- Having the build work locally but not on the CI build server (the CI build server should mimic the production environment as much as possible).
- Having unit tests work locally but not on the CI build server because of some environmental difference.

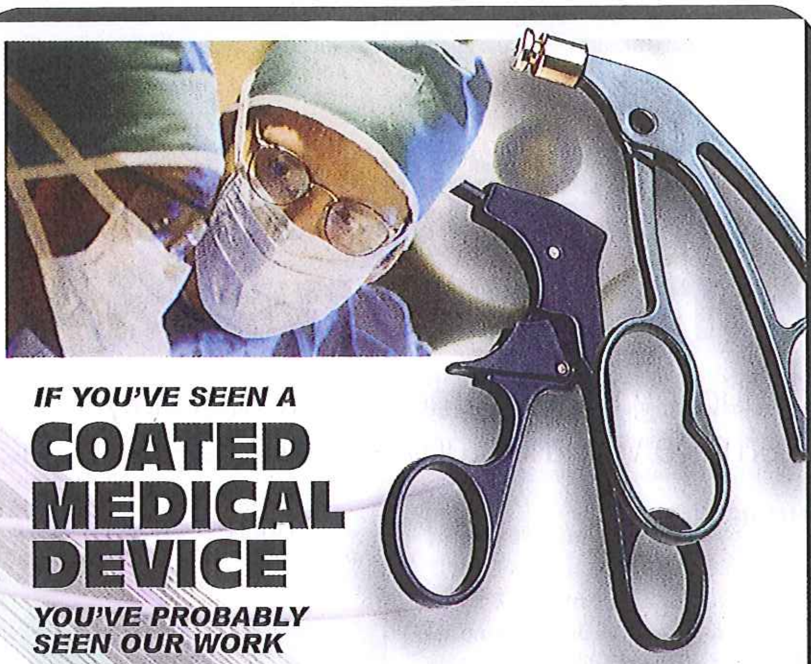
The most difficult software defects to fix (much less find) are the ones that do not happen consistently. Database locking problems, memory problems, and race conditions can cause inconsistent but serious defects. To fix such problems, it's a good idea to have unit tests go above and beyond what is traditionally expected and implement automating functional testing.

Conclusion

Revisiting the story shared at the introduction of this article, if the designer had been armed with work instructions to make the quality system an integral part of the daily process, he would have been much more prepared to respond to the auditor's interrogation.

Poorly executed SOPs can be annoying and a barrier to productivity. However, procedures that are appropriately tied to daily activities and allow users to freely choose the best tools can result in highly effective development of medical device software.

Matthew Rupert is a software architect II for Talecris Biotherapeutics in Raleigh, NC. M



THE LEADER IN MEDICAL, INDUSTRIAL & AEROSPACE COATINGS

Boyd Coatings Research Co., Inc.

1-800-258-0110

51 Parmenter Rd., Hudson, MA 01749 • Tel: 978-562-7561 • Fax: 978-562-9622
eMail: info@boydcoatings.com • web: www.boydcoatings.com/MDDI

Our Thanks to MDEA for the Distinguished Award!

Xeridiem™ Medical Devices:
the 2011 winner in a highly competitive category

Surgical Equipment, Instruments, and Supplies

Medical Design Excellence Awards® 2011 WINNER

This international award was for a novel chest tube that minimizes clogging during thoracic surgery. We orchestrated the design and development for manufacturability and managed the supply chain, including order fulfillment.

The MDEA recognition for this device follows an EACTS award to Dr. Ed Boyle of Clear Catheter. Congratulations to the Xeridiem team and the innovator Dr. Boyle,who put his trust in us and Carbon Design Group.

Xeridiem delivers exactly what each client requires to bring a complex, single-use, minimally invasive medical device to market, on time, on budget. These are our promises.

xeridiem™ MEDICAL DEVICES

Your Vision. Designed. Delivered.

Xeridiem Medical Devices, founded as MRI 25 years ago, is backed by the technical and financial strength of our multinational parent company, Fenner plc.

4700 South Overland Drive · Tucson, AZ 85714 USA · 1-520-882-7794 · www.xeridiem.com
ISO 13485:2003-certified · FDA-registered

Beverly Lorell, MD is senior medical and policy advisor for the FDA and life sciences practice group in King & Spalding's Washington, DC office. Lorell, a cardiologist, was previously professor of medicine at Harvard University and served as vice president and global chief medical and technology officer at Guidant Corp. (Indianapolis). E-mail her at blorell@kslaw.com.

Pamela Furman Forrest is a partner in the Washington, DC office of King & Spalding's FDA and life sciences practice group. Her practice focuses on FDA medical device matters. Forrest holds a bachelor's degree from Yale University and a law degree from Stanford University. She can be reached at pforrest@kslaw.com.

Brian Woodward is general manager of the medical products business at Johnson Matthey (London), based in the company's San Diego office. He has been involved in the electronic materials and platinum fabrication business for more than 25 years. Woodward holds bachelor's and MBA degrees in business and management and has focused on ue-added component supply to the global medical device industry. Contact ı at woodwbk@jmusa.com.

Alison Cowley has worked in the market research department at Johnson Matthey since 1990 and currently holds the post of principal analyst. She is the company's specialist on mining and supplies of the platinum group metals (PGMs). Cowley conducts research into demand for PGMs in a number of industrial markets, including the biomedical and aerospace sectors.

Matthew Rupert is a software architect II with Talecris Biotherapeutics (Raleigh, NC). He has been designing and developing software for 15 years. Rupert graduated with a bachelor's degree in computer science from Ball State University in 1998. He has worked in a range of software development and lead roles, spending more than seven years on medical device software. Rupert lives in Wake Forest, NC. Reach him at matthew.t.rupert@gmail.com.

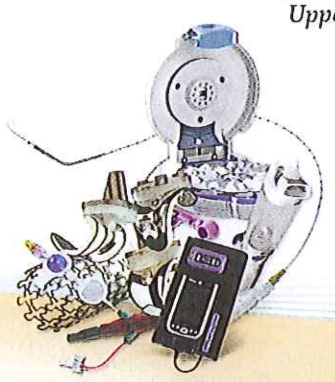
John Rokus is vice president of continuous improvement at Micro Power Electronics (Beaverton, OR). Rokus, a certified Six Sigma black belt, trains and coaches Micro Power employees in lean deployment and applying Six Sigma problem-solving methodologies. He earned his bachelor's degree in industrial and manufacturing engineering from Oregon State University (Corvallis, OR). E-mail him at jrokus@micro-power.com. M

UBM
Canon

Attend San Diego's Very Own Medical Device Conference and Supplier Showcase

MEDevice forum • San Diego

SEPTEMBER 13–14, 2011
San Diego Convention Center
Upper Level Ballroom
San Diego, CA



Medical Design & Manufacturing® MD&M

Brought to you by the producers of the world's leading medical device conferences and expositions

Official Media Partners:
Qmed MDDI MPMN

Register online today!
MEDeviceForum.com
Promo Code: AA

19471_SD_MED11