Angelica Rodriguez

Ben Paul

Corey Alexander

Lindsey Watterson

Biomedical Engineering Domain Track

In the software side of biomedical engineering, there are five main quality factors. The software used to analyze data, and to gather it, must have a high level of correctness. All calculations and data storage made must be extremely accurate, because simple errors could potentially cause medical issues. The efficiency of the software is also important. Many times medical tests need to be completed as quickly as possible, so treatments or medications can be administered. The software also needs to be secure. Medical devices that rely on software must be secure in order to protect patient information and in some cases the health of the patient. The importance of this is evident in light of recent occurrences of hackers gaining access to patient heart monitors. In addition to these first three qualities, the software must also be reliable. These three previous factors do not matter if the equipment cannot perform its tasks repeatedly due to low reliability. Biomedical software must also have high usability. Many of the people using biomedical devices have only a basic knowledge of how software works, if any at all. So the simpler and easier it is for them to use the better the equipment can serve its purpose.

All software developers in the biomedical engineering field must adhere to the rules set out in Part 11 of the FDA’s Title 21. Although it doesn’t state specific testing procedures to be followed, section 11.10 of this document states the following criteria must be met by the procedures chosen by the developer:

“(a) Validation of systems to ensure accuracy, reliability, consistent intended performance, and the ability to discern invalid or altered records.

(b) The ability to generate accurate and complete copies of records in both human readable and electronic form suitable for inspection, review, and copying by the agency. Persons should contact the agency if there are any questions regarding the ability of the agency to perform such review and copying of the electronic records.

(c) Protection of records to enable their accurate and ready retrieval throughout the records retention period.

(d) Limiting system access to authorized individuals.

(e) Use of secure, computer-generated, time-stamped audit trails to independently record the date and time of operator entries and actions that create, modify, or delete electronic records. Record changes shall not obscure previously recorded information. Such audit trail documentation shall be retained for a period at least as long as that required for the subject electronic records and shall be available for agency review and copying.

(f) Use of operational system checks to enforce permitted sequencing of steps and events, as appropriate.

(g) Use of authority checks to ensure that only authorized individuals can use the system, electronically sign a record, access the operation or computer system input or output device, alter a record, or perform the operation at hand.

(h) Use of device (e.g., terminal) checks to determine, as appropriate, the validity of the source of data input or operational instruction.

(i) Determination that persons who develop, maintain, or use electronic record/electronic signature systems have the education, training, and experience to perform their assigned tasks.

(j) The establishment of, and adherence to, written policies that hold individuals accountable and responsible for actions initiated under their electronic signatures, in order to deter record and signature falsification.

(k) Use of appropriate controls over systems documentation including:

(1) Adequate controls over the distribution of, access to, and use of documentation for system operation and maintenance.

(2) Revision and change control procedures to maintain an audit trail that documents time-sequenced development and modification of systems documentation.”  
  
Link to this documentation:

<http://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfcfr/CFRSearch.cfm?CFRPart=11&showFR=1&subpartNode=21:1.0.1.1.7.2>

To summarize some of the points of this standard, the programmer is free to do testing and development however they would like as long as they adhere to some rules and guidelines:

* It is necessary to have the ability to create backups of any data that is stored. These backups must have both a human readable version and a version that can be inputted back into the program if need be.
* It is necessary to have access control so that only authorized people can access the system. Also the program must keep logs of who accessed the systems, what they did and when they did it. And all changed must be reversible so the old information can be retrieved if needed.

Some of the other rules outlined in this document pertain to the operators of the system rather than the developers.

**Specific Examples Where Certain Criteria are Necessary**

Surgical Machines

* Accuracy: 1mm difference can make the difference between saving a life or losing it

Prosthetics and implant devices

* Reliability: needs to do the same time every time
* Security
  + Hackers can gain access to patient heart monitors
  + <http://www.theregister.co.uk/2008/03/12/heart_monitor_hacking/>

Other important factors

* Easy to keep up to date, fast and efficient

**What do you suggest?**

There are two main branches of biomedical software, both of which have their own testing methodologies.

The first is database systems used to store patient records and files. These systems are extremely prevalent and almost every hospital and doctors office in the country has one. One aspect of these systems that seems to be universal is that they aren’t user friendly. For this reason UX and usability tests for the frontends would be a great improvement. This would reduce the amount of time that new nurses or doctors waste learning how to operate the databases. The other thing these databases need is stress testing. It is necessary to ensure the system can support a large number of users and does not crash, since multiple doctors and medical personnel can be accessing the system at the same time. For example, a worst case scenario would be a doctor needs to look up information on a critical patient and can’t get to the information because the database has crashed under the user load. Stress testing could help avoid a situation like this, and make sure that the systems can handle the amounts of users that will be accessing them.

The other branch of software is in software that will either go into prosthetics and devices the user has surgically implanted, as well as software that controls surgical equipment. These software applications should be tested with a derivative of boundary value testing. These types of software applications need to be tested to make sure they perform with all possible inputs, and act the correct way. This means that robust worst case testing would be the best option to be certain that there systems can operate in all the conditions they will be given in the real world.

**Are there any regulations/standards?**

Yes there are. The standards are set by the FDA. The regulations governing software are outlined in Part 11 of the FDA’s Title 21, which we discussed and copied above.

**Examples of Success/Failure.**

Implants and Surgical Equipment:

The most crude and basic way to measure success of software in implants and surgical equipment is whether the health of the patient improves with use of the technology. For example, a patient who receives surgery using robotic equipment controlled remotely by a doctor, the software used in the robotic equipment can be considered successful if it allows the doctor to do his job and the surgery is a success. However, this doesn’t mean that if the surgery isn’t a success the software has failed, a multitude of other factors could have been the cause. On the other hand though, if a patient is implanted with an insulin pump, to automatically measure and control the amount of insulin in the patient’s body, and the pump fails to recognize the amount of insulin in the body and continues to pump insulin into the blood stream the patient’s health is almost guaranteed to decline. This would be an example of a software failure (assuming the mechanical portion of the implant functioned correctly).

Not all failures need to be as deadly, or impact the health of the patient, as much as the previous example. For instance a health monitor may fail, and give results that are incorrect, such as a patient flat lining when they obviously have not. As long as the doctors are competent, they will realize that the heart rate monitor is incorrect with no detrimental effects to the patients’ health.

Database and Hospital Systems:

These systems have easier to identify successes and failures. The systems and databases that doctors and hospitals use daily, are considered to be succeeding if they are storing the data correctly, returning the correct data, and operating within a specified speed. In contrast they are failing if they break any of these conditions. For example, if a doctor requests Patient X’s exam results from the computer and receives Patient Y’s results, this is a failure. To continue this example, the system also has failed if Patient X’s exam results didn’t successfully store and therefor can’t be retrieved when the doctor requests them. And lastly, if the system takes 30 minutes just to retrieve Patient X’s records, then this is a failure as it shouldn’t take a database 30 minutes to retrieve the information. All of these scenarios could lead to the doctor either not being able to treat the patient when it is needed, or treating the patient incorrectly.

As in many other industries, software success is inversely correlated to how often you heard about the software. If there are no problems, and the code is working as expected, then nobody talks about it. Most of the time you hear about bio-medical software, it is when there has been a failure.

**What Metrics are used?**

Currently, the metrics measured for testing is left up to the individual developers and companies, and isn’t regulated by the FDA. However, the two metrics that seem to be most common are McCabe Cyclomatic Complexity and Lines of Code written.

Cyclomatic complexity is measured as a way to tell how many different paths the program can take. This is useful in the bio-medical industry, as there should be as few different paths as possible so that the number of places a bug can occur is reduced. For example, in surgical implant technology, under different extreme conditions the device may function very differently in order to keep the patient healthy. This is as expected, but problems arise in testing if all of these different extreme conditions have their own unique branch that the program can proceed down. If makes testing easier, and the program more robust, if when the implant is in certain conditions, it branches off to do tasks specific to those conditions. But when it has completed these, it re-joins the main program flow. This is why McCabe Cyclomatic Complexity is a metric that is currently used, it helps the developers make sure their code doesn’t have too many different branches, where hard to find bugs may exist.

**What Metrics do you recommend?**

* Average Number of Comment Lines per Method
  + This metric may be useful to track in this domain, mostly for the fact that the people who use the code have a vested interest that the code works as advertised, but also most likely don’t have any coding knowledge. Biologists and Doctors may want to be able to look over the code and see if what is needed is being done, since it is their patients whose health may be determined by the software working correctly. Adequate (or excessive) commenting will help these non-programmers better understand the software, and have more peace of mind about its quality.
* Average Method Size
  + Smaller methods are generally agreed to be easier to debug and maintain. In the case of surgical devices and implants, finding all possible bugs is extremely important. Because of this, measuring the average method size will help developers make sure they are keeping the code in manageable chunks to it can be properly debugged later.

**What are common problems associated with quality assurance in the domain?**

One of the biggest problems in quality assurance in the bio-medical field, in particular surgical equipment and implants, is the inability to test the software under real world conditions. Extensive testing can be done in labs and such, but this can’t guarantee how the software will react under the conditions it will face in the real world. It is extremely difficult to find patients who are willing to have surgery performed on them, or implants implanted, for the purpose of testing if the software works correctly or not. Not to mention the massive amount of regulations in regards to testing medical procedures on humans. The software used for these things is tested extensively in labs, under conditions as close to what will happen when used on people as possible, but it is nearly impossible to test under all the conditions that the software may experience in the real world.