Validating Off-the-Shelf Software: Throw Away Your Checklist

You're Responsible, No Matter What

In any business, the build or buy decision is extremely important – a balancing act involving time, development expense, need for maintenance, suitability for the company's specific tasks, and compatibility with the company's infrastructure.

Perhaps the most difficult case is software. Lots of very good software is available today – whole systems that handle manufacturing processes or document management, or components which can be incorporated into software under development. Any piece of software you can purchase rather than develop, means that much effort saved. The developer has already worked out a solution to the problem; you can just use it!

In many other industries, the story would stop there. The software is installed (or incorporated), you have a solution, and the developer is responsible if anything goes wrong.

However, in our industry (medical devices, pharma, biologicals), purchasing the software ***isn't*** the end of the story. All of quality system outlines - 21 CFR 210, 21 CFR 820, or ISO 13485 – say basically the same thing. You, the manufacturer who sells the medical product, are responsible for all aspects of its quality. You need to review all steps of production, whether carried out in-house or outsourced. Where your product needs to be sterilized or lyophilized, you need to know how the contractor performs these services. Sheet metal enclosures, glass vials, polymer support materials, active pharmaceutical ingredients – you're responsible for ensuring the quality, no matter what you purchase.

The software you buy is another case of the same question. That software may:

- Run a manufacturing line
- Handle corrective and preventive actions
- Organize and maintain your SOPs and work instructions
- Record employee training

Or, if you have an in-house group developing software as part of your product, the software you buy may be some readily available element:

- Calendar date picker
- Barcode print formatter
- 3D graphics library
- HL7 interface

Software you purchase, in FDA lingo, is off-the-shelf software ("OTS software", as used below). In IEC 62304, it is termed SOUP (software of unknown provenance), even though in some cases we may know quite about how it was designed and developed. "Ensuring the quality and suitability of the software" is simply another name for validation.

How, you may ask, ***DO*** we validate software that someone else developed and we simply bought?

The answer isn't complicated, but it can seem difficult at first. If you use a systematic approach, the steps become much clearer.

"Nothing is particularly hard if you divide it into small jobs." - Henry Ford

Third-Party Code is Still Your Code

The first question to ask is how the software will be used. At the highest level, the uses fall into the two cases mentioned:

(a) incorporated into software of your product, or
(b) installed and used as part of your manufacturing or quality process.

Consider the first (we'll talk about the second one below).

There's no harm in building existing code into a software product. You may be using a specific piece of equipment, that comes with drivers allowing it to communicate with a PC through a USB interface – and have no reasonable choice but to use the code which comes with the device. In some cases, the software you purchase may be the entire software for a device, such as a sample handling system or a diagnostic device.

Follow the FDA Guidance

For any code you build into your own software, the reference to consult is FDA's guidance "Off-The-Shelf Software Use in Medical Devices" (http://www.fda.gov/medicaldevices/deviceregulationandguidance/guidancedocuments/ucm073778.htm).

Figure 1, redrawn from that guidance, shows the process the agency expects you to follow. (Note: the term "level of concern" has been removed from this diagram. The off-the-shelf software guidance dates from 1999; a later document, the 2005 "Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices" defines "Level of Concern" differently, for a different purpose, so the term is removed to avoid confusion.)

Step 1: Does the product contain off-the-shelf software? If not, stop.

Step 2: Provide Basic Documentation for the OTS software. Section 2.1 of the guidance specifies content of Basic Documentation; this includes answers to the questions:
a) What is it? (title, version, and manufacturer address/location)
b) What Computer System Specifications does it require?
c) How will you assure appropriate actions are taken by the End User?
d) What does the OTS software do?
e) How do you know it works?
f) How will you keep track of (control) the OTS Software?

Step 3: Perform and document Hazard Analysis of the product and the OTS software. (section 2.2)

Step 4: If you can show that the software presents only a minor risk, you're done. Otherwise, continue.

Step 5: Institute the necessary hazard mitigations. (Section 2.3)

Step 6: Describe and justify any risks that remain after mitigation, that is, "residual risk". (Section 2.4)

Step 7: If you can show that the software presents a minor or moderate risk after mitigation, you're done.

Step 8: If, after mitigation, the software still presents major risk, provide special documentation. (Section 2.5)

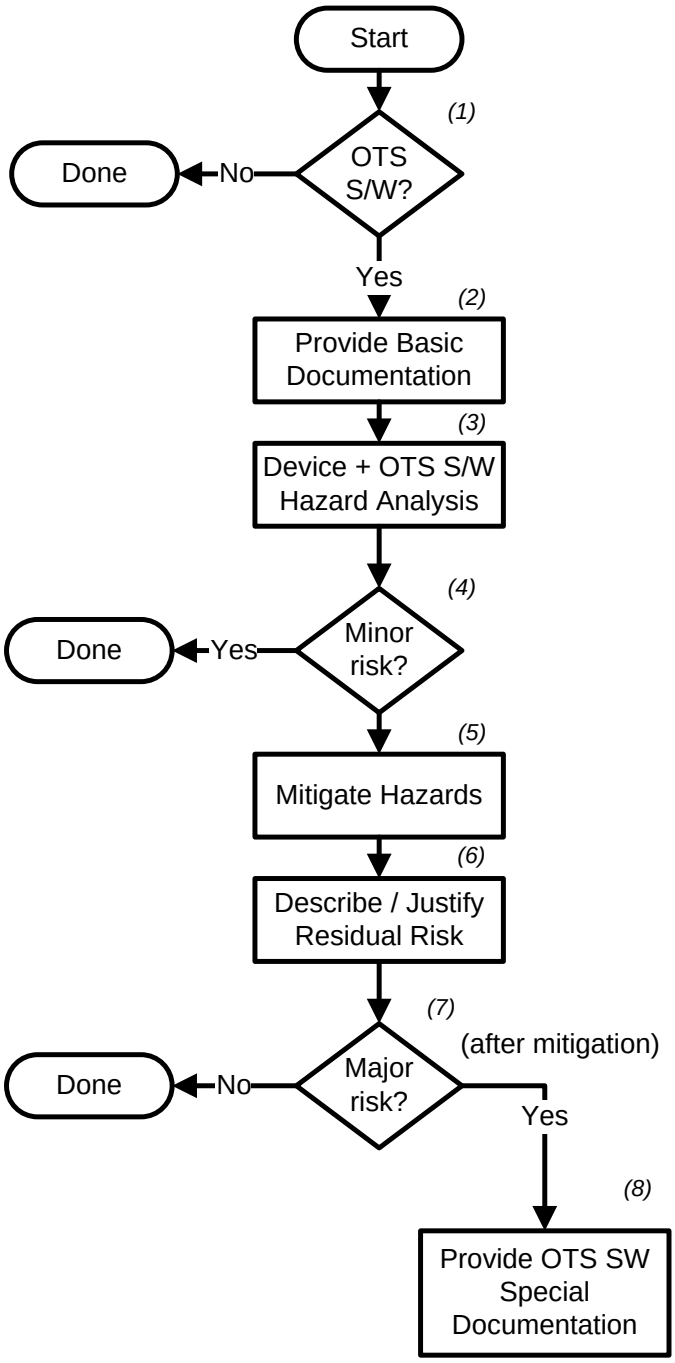


Figure 1. Validating OTS Software Incorporated in a Medical Device
(redrawn from the FDA guidance "Off-The-Shelf Software Use in Medical Devices")

This sequence may appear complicated, but in fact it's a focused version of the flow chart given in ISO 14971, and fits very well with overall device design and hazard analysis.

Hazard analysis is discussed below. The point here is to treat OTS code at least as carefully as you do your own code: ask the difficult questions, and make sure you have reasonable answers.

- Can you document precisely what you have, down to the version number?

- Do you know how you'll ensure version control of the OTS software?
- What function does the OTS software perform, and could anything about that function foul up so as to harm someone?
- Is the OTS software well established and widely used?
- Is the OTS software relatively stable, or has it undergone numerous recent updates?

The guidance defines several important terms – major vs. moderate vs. minor risk (called "Level of Concern" in the guidance), hazard, hazard analysis, hazard mitigation, serious injury, and safety. Understanding the agency's precise intended meaning of these terms, is crucial to understanding the entire guidance. Again, bear in mind that the term "Level of Concern" is also used in a different, later guidance, and should be replaced here with a different term (for example, "Risk Classification") to avoid confusion.

What Does the OTS Software Do and How Do You Know It Works?

This question can be answered a number of ways, and no single approach is the right way for all cases.

Bear in mind that the OTS software is part of your overall system. If it performs a unique and highly specialized function, you may be well advised to create a "test harness" to feed inputs and record outputs, so you can vary conditions across a wide range and determine whether any extremes will cause the OTS code to freeze, crash, or behave in any other unexpected way.

If the OTS software provides a common or relatively simple function, then testing the entire system with the OTS software included may be sufficient. A good justification for this approach would be to provide the third-party developer's list of known issues (assuming this list is available), and show that none of the conditions noted are either relevant or significant in your device. Make sure to document this justification!

Version Control and End-User Information

Sometimes, your biggest headache is not what the OTS software currently is and does, but how it could change, or what and end-user needs to know about it.

Software developers issue updates all the time, and version 2.2.4 may have additional features or behavior you didn't encounter in version 1.9.7. You may need to maintain a pristine installation copy of the OTS software you incorporated in the product.

Alternatively, if you choose to allow field updates of OTS software in your product, take control of this process. Subject the new OTS software to at least as much evaluation as the original, and make sure your customers only get what you have checked out!

More often than not, the OTS code is integrated into your device application, so that the end user will not need to install or configure it, and your system's User Manual also covers its use. If this is not the case, put yourself in the end user's shoes – what kind of information or instructions would you want, to be sure you can set up and use the software correctly?

Manage the Risks

In many cases, the OTS software clearly falls in a low risk classification. If so, document the reasons along with the Basic Documentation, and you're all set.

If the OTS software falls in a moderate or high risk classification, risk mitigations will be necessary. How do you mitigate possible software hazards? Some options include:

- Apply range and validity checks to user inputs, if these will prevent hazardous device actions
- Apply output checks to values determined by the OTS software (e.g. radiation intensity, therapy duration, drilling depth, or rotation / vibration speed) and prevent mechanical action or energy output outside safe limits
- Flag diagnostic results which fall outside stored limits

- Monitor all processes with a "watchdog" program which will detect whether any process has stopped for any reason

Don't forget that risk management applies to the whole system, not just the software – and that software risks may sometimes be best mitigated by physical designs such as device enclosures, limit switches, or circuit breakers!

What if it's still high risk?

If, after all risk mitigations are applied, the OTS software can still be classed as high risk, the FDA guidance requires you to provide special documentation which shows that:

a) The OTS software was developed via methods appropriate for its end use in a medical device,
b) Verification and validation of the OTS software were appropriate and adequate for its end use in a medical device, and
c) The OTS software can still be maintained even if the original developer no longer provides support.

Item (b) includes verification and validation both by the original developer, and by you, the device manufacturer/

Item (a) is typically satisfied by auditing the developer's quality methods and documentation. ***If you cannot arrange such an audit, have no other information about the developer's quality methods, and the OTS software classes as high risk after all mitigations, it would be wise to reconsider using this OTS software in your product.***

Process / Quality Software: Use Critical Thinking to Validate

Now consider the second case noted above – software which serves as part of the manufacturing process or quality system.

Pharmaceutical GMPs state that "Automatic, mechanical, or electronic equipment or other types of equipment, including computers, … shall be routinely calibrated, inspected, or checked according to a written program designed to assure proper performance." (21 CFR 211.68)

The medical device quality system regulation similarly specifies that "When computers or automated data processing systems are used as part of production or the quality system, the manufacturer shall validate computer software for its intended use according to an established protocol. All software changes shall be validated before approval and issuance. These validation activities and results shall be documented." (21 CFR 820.70(i)).

ISO 13485 section 7.5.2 (Validation of processes for production and service provision) states that "The organization shall establish documented procedures for the validation of the application of computer software (and changes to such software and/or its application) for production and service provision that affect the ability of the product to conform to specified requirements."

All of these quality standards agree: whether developed in-house or purchased, the software used for production floor operations – or for any other activities that affect product quality – must be validated. But how?

Is Validation Required?

AAMI Technical Information Report 36 "Validation of software for regulated processes" suggests asking a series of questions to determine whether the software is subject to validation:

1. Could the failure or latent flaws of the software affect the safety or quality of medical devices?
2. Does the software automate or execute an activity required by regulation (in particular, the requirements of the QSR)?

3. Does the software generate or manage data to be used in or support of a regulatory submission?
4. Does the software generate or manage records that are required by a regulation (e.g., device master record, device history record, design history file, or clinical trial records) or records that would be accessed in the future to provide evidence of the completion of an activity required by regulation?
5. Is the software used to execute or record an electronic signature required by regulation?

Even after answering these questions, there is no defined rule for deciding when an application needs to be validated. The decision is up to you. Generally the decision is clear – any piece of software which supports production or the quality system should be validated, as much to assure your quality as to satisfy the regulators. No matter how you decide, be sure to document your reasoning!

Prerequisites for Validation

Product design and manufacture involves people, processes, tools, and materials as well as software. In this context, software validation is only a part of the larger quality effort. Validating process software is intended to provide confidence – that the associated process will perform correctly and consistently.

Software validation only has value if other elements are already in place.

- An overall quality system – a process which defines quality goals, ensures that product quality is measured, and that defects are investigated and action taken to correct or prevent them. In particular, specific aspects of the quality system with substantial impact include:
  - asset and infrastructure management (human and hardware)
  - change management (including configuration management), and
  - vendor management
- Clear understanding of the process in question – how the activity needs to be performed, and what steps or subprocesses involve risk of errors or defects

TIR 36 Provides a Framework

AAMI TIR 36 presents a well-reasoned approach to validating off-the-shelf process or quality software. The document is well worth studying – all necessary steps are considered, but since no two systems provide the same functions or present the same risks, following the report's approach is not simple. (Though TIR 36 was developed for medical device manufacturers, its concepts apply equally to pharmaceutical or biological product manufacturers.)

To quote from the report: "Over time, many practices have evolved into a checklist-mentality approach based on a compliance need. At times the checklist approach inadvertently causes activities to stray from value-added activities that appropriately substantiate that the software performs as intended. Straying occurs when a single solution is sought that is intended to satisfy a large number of stakeholders, each with a potentially different set of objectives and requirements."

The report stresses two key concerns about validating software for regulated processes:

1. Useful validation involves critical thinking – a "checklist mentality" is to be avoided. Every validation, for every software package, must be tailored to the features of the package and the types of risks it presents as it is used.
2. Validation is only part of a larger process of life cycle control, which begins when the software is first evaluated and lasts as long as the software is in use. Figure 2 depicts the overall life cycle controls.

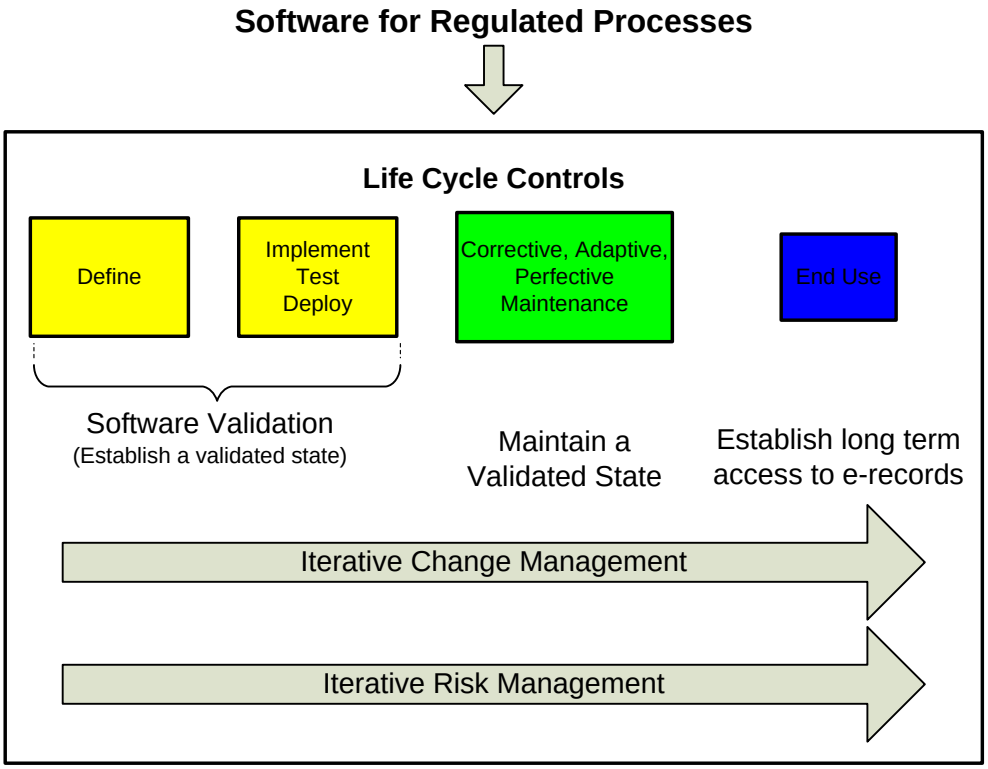


Figure 2 Life Cycle Controls
(redrawn from AAMI TIR 36)

The goal is to establish a validated state for the software, then to keep the software in that state until it is taken out of service.

Follow a General Sequence

The activities noted in Figure 2 suggest a general series of steps necessary for validating any software for regulated processes.

Phase 1 - Define

- The process, the requirements of the process, and the risks if the process goes awry.
- The intended use of the software in context of the process being automated
- Precisely what the software is expected to do.
- What portions of the process are **not** automated by this software, and how quality is assured in those portions.
- Process activities upstream and downstream of the one in question, which can be considered in evaluating risks from software failure.
- Boundaries between the software being validated and other software systems in use.
- The type and extent of validation to be carried out.

Phase 2 - Implement / Test / Deploy

TIR 36 stresses that critical thinking is essential for process software validation: no two cases are ever alike, and a number of activities from a so-called “toolbox” can be employed to build confidence that a given software application will work reliably for a given process.

- Installation qualification / operational qualification:
  Basic checks on installing the software and its interfaces with other system will demonstrate that it is correctly installed, that it is functioning in the intended environment, and that data are flowing as designed.
- Vendor audit
  To assess the vendor's software quality system, for confidence that the software will be safe and usable.
- Testing
  As appropriate to the application's function and risks associated with failure. For any specific system, only some of these types may be useful. Various options (some which overlap with others) are described in TIR 36:
  - Use case testing
  - Interface testing
  - Vendor-supplied test suite
  - Software system testing
  - Use case testing
  - Normal case
  - Robustness testing / Stress testing
  - Output forcing testing
  - Combination of inputs testing
  - Beta testing
  - Performance testing
  - Final acceptance testing
- User procedure review
- Internal training for the application

Phase 3 - Maintain

Establishing that a piece of software will function reliably is only the first step. Data accumulates. Changes occur, whether we intend them or not. Operating system updates must be installed. Corrections and enhancements to our application are issued – some of them essential to security. Other software with interfaces to our application are updated, which may affect the application with which we're concerned.

Elements of the maintenance phase:

- Have a plan.
  Think about whether and when new versions of an infrastructure application will be considered and installed, and what other steps will be required.
- Monitor the system once it has been installed
  Record any defects found and analyze them periodically, assess whether actual use has changed from what was previously documented, audit data to assure that nothing has become corrupted.
- Follow backup / recovery procedures
- Follow other kinds of operational controls:
  access security, user rights administration, database administration, and business continuity planning (in the event the application or the network becomes unavailable and the application's function is crucial to ongoing work).
- Find out about any new version
  What are the changes? What are the known issues with the new version? If there is a database, will the database schema change? If so, what is the transition process and how will you check it?
- Determine whether upgrading to a new version will require revalidation.
  The answer is not always "yes" – it depends on whether new features, new intended uses, or new risks will be introduced when the new version is installed.

Phase 4 - Retire

Typically, process or quality software cannot simply be deleted. A series of decisions need to be made, and if the software generated and stored production data, the data must be kept. TIR 36 suggests answering these questions

- Is there software replacing the retired software?
- Can the data be migrated to the new software?
- Should the data be migrated to a portable format for long-term retention?
- What are the data retention requirements for the type of data?
- Will the data be stored on durable media?
    - If so, what are the storage instructions or procedures, and can the data be retrieved containing all associated data requirements?
    - What is the procedure for maintaining durable media and software that can read it?
    - Will an archived hardware platform be stored for using and retrieving the retired application?
    - How will stored hardware be maintained?
    - Would the retired software ever need to be accessed as part of a complaint or CAPA investigation?
    - Will the platform and application be needed to re-create a software program?

Let Risk Management Guide You

Risk assessment and management are essential to validating either general type of OTS software.

But how do we manage risk for software? After all, software is simply ones and zeros running around in the circuits of a computer, and can't harm anyone directly.

We start with ISO 14971, which outlines the overall risk management process for medical devices. Figure 3 shows the schematic given in the standard.

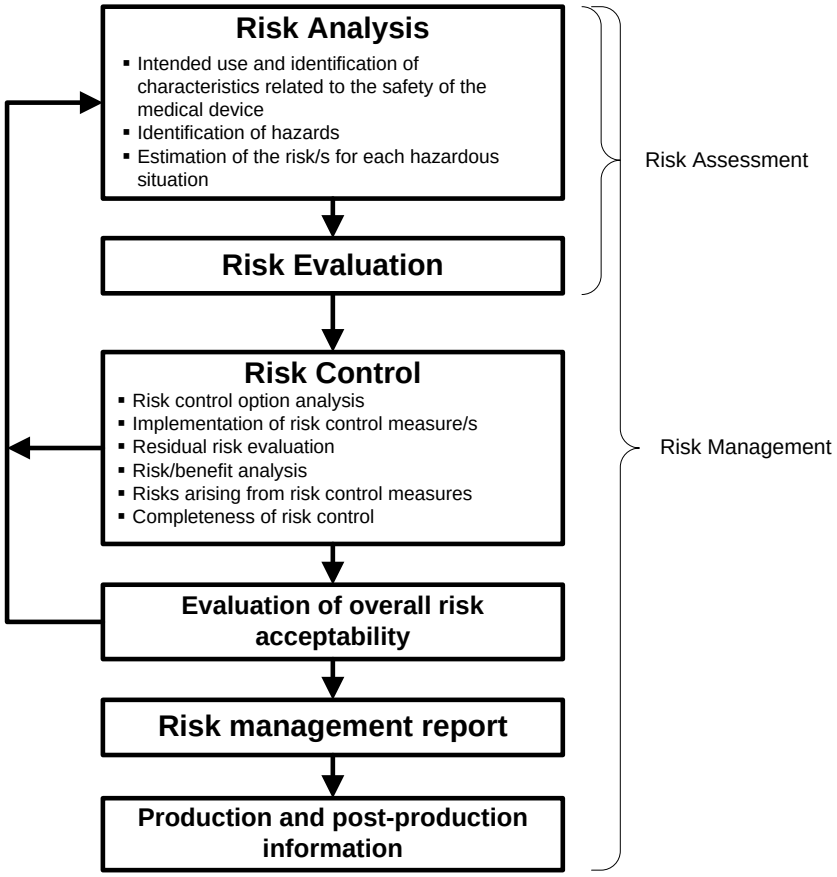


Figure 3 Schematic Representation of the Risk Management Process *(redrawn from ISO 14971)*

The standard addresses risk management for medical devices in general, and gives relatively little guidance on software-related risk. For more direct information about software risk, IEC Technical Information Report 80002-1 "Medical device software - Part 1: Guidance on the application of ISO 14971 to medical device software" is an excellent resource.

Recognizing Hazards

We need to understand what hazards are before we can evaluate or mitigate them. For our purposes in validation, we are not considering risks to the development schedule or economic risks to the company (though it could help to be aware of those). ISO 14971 gives these definitions:

> risk: combination of the probability of occurrence of harm and the severity of that harm
>
> harm: physical injury or damage to the health of people, or damage to property or the environment
>
> hazard: potential source of harm

These definitions fit the context of ISO 14971, namely the design and manufacture of medical devices. Evaluating risk is, in effect, asking what could go wrong with the device such that someone would be hurt in some way. In more general terms, "someone getting hurt" is "something bad happening" – and for process software, we need to apply the more general concept, so that "harm" includes incorrect release of product which should not be released.

Consider these as examples of types of harm:

Software in a medical device:

- Injury to patient, caregiver, or bystander

Software for a manufacturing or quality process:

- Harm to the medical device (or other medical product)
- Harm to the manufacturing process
- Harm to regulatory compliance
- Harm to manufacturing personnel or the environment

Evaluating Risks

A number of formal methods can be employed to assess risk in a product design or in a tool used for product support – no matter how we do the assessment, we need to use both imagination and reason.

The two most common questions are:

- What bad thing could occur, and why might it happen?
- What could go wrong in using component XYZ (and what would result)?

The first question is the starting point for Fault Tree Analysis, which works backward from the undesired outcome to identify conditions, combinations of conditions, and chains of conditions which could give that result.

The second question, applied methodically to each component within a design, provides the beginning of a Failure Mode and Effects Analysis (FMEA) or Failure Mode Effects and Criticality Analysis (FMECA).

Each of these approaches has value, and each can be overused. Each technique is discussed in a variety of articles and guidance documents; this article will not attempt to describe how to perform these analyses. The crucial element in either method, and in many other risk assessment methods, is *the ability to imagine possible failures*.

Mechanical or electrical failures are tangible and often predictable. What do we mean by a software "failure?" After all, software works the same way every time, doesn't it?

Not necessarily.

In software, whether embedded in a device or supporting a manufacturing or quality process, the concept of "failure" is not the item breaking, but instead not doing what it is supposed to do. A failure might constitute:

- A genuine defect (logic error, or bug) not previously found and corrected. This could involve a combination of conditions not anticipated during design.
- Accepting and using nonsense input without any validity check
- Presenting a confusing interface to the user, which leads to wrong inputs
- Allowing "back door" access to information (i.e. permitting either falsification by users, or unauthorized access by hackers)

At every step in use of the device or the system, we need to ask, over and over, "What could go wrong here?"

A raw list of failure possibilities, however, is not sufficient. We cannot act on every one we come up with – we would never release product if we tried that approach. Instead, we need to rank our possible failures.

For most other types of engineering, we get that rank by combining probability that the failure will occur and severity of the result. A semi-quantitative scale can be used to rank failure severity, for example:

Value	Definition
1	**Nuisance**: Failure that requires an operator to repeat an input or action, but does not result in loss of data that has been entered already. Unclear message or misspelled label with no effect on program operation.
2	**Moderate**: If the software fails or the user makes an error: • Potential risk to record integrity (loss or corruption of quality or manufacturing data), or could give unexpected / confusing output at some step of a process, but which can be avoided with a known workaround. • Potential risk of inability to demonstrate regulatory compliance, but which can be avoided with a known workaround.
3	**High**: If the software fails or the user makes an error: • Potential risk to product safety, which could result in non-serious injury, or environmental harm, on use of the product. • Potential risk of unrecoverable loss of product quality data, with no known workaround. • Potential risk of manufacturing incorrect product (for example, installing the incorrect software version), which will be detected and scrapped in QC testing or manual inspection. • Potential risk of inability to demonstrate regulatory compliance, with no known workaround.
4	**Unacceptable**: If the software fails or the user makes an error: • Potential risk to product safety, which could result in serious injury or death on use of the product. • Potential risk of releasing product which should not be released.

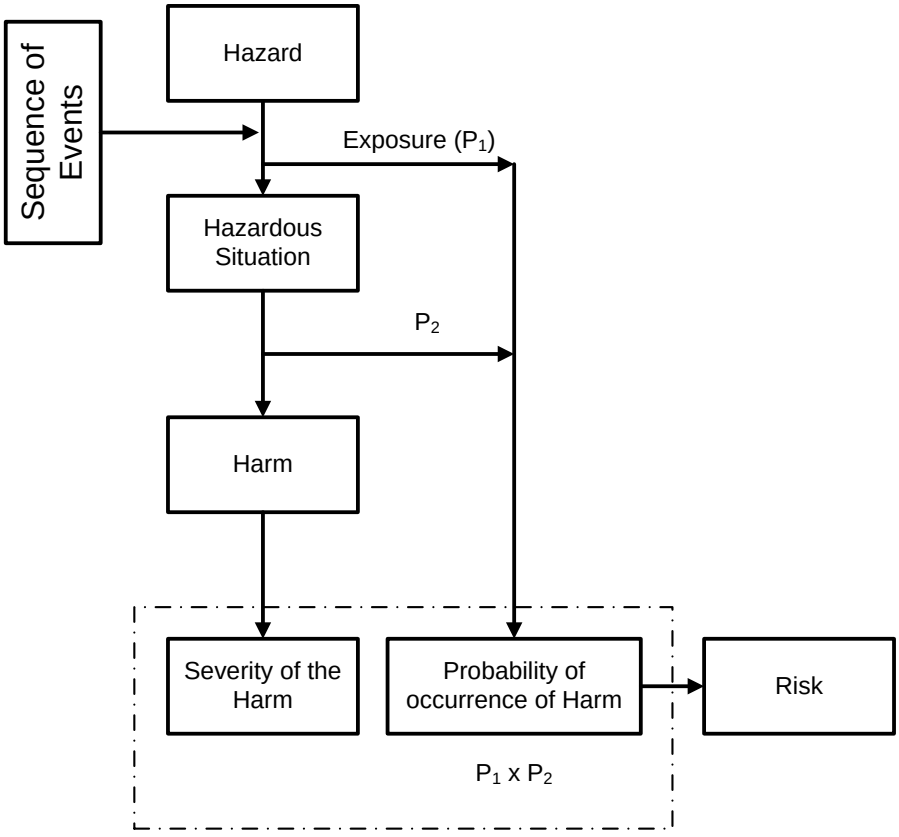
Probability of the failure occurring is a different matter. How do we estimate the probability that a specific error exists in a given piece of software? Such probability estimates are often little more than wild guesses. According to IEC TIR 80002-1:

> Since it is very difficult to estimate the probability of software anomalies that could contribute to hazardous situations, and since software does not fail randomly in use due to wear and tear, the focus of software aspects of risk analysis should be on identification of potential software functionality and anomalies that could result in hazardous situations – not on estimating probability. Risks arising from software anomalies need most often to be evaluated on the severity of the harm alone.

Probability of occurrence is not useful for software – if we can imagine a software error, we have to assume that it will occur at some point.

However, especially for process software (which we can't redesign to mitigate risks), we need a ranking factor which we can affect with our mitigations. If we can't modify the software, we can't affect the severity when the error occurs.

ISO 14971 defines *probability of occurrence* as P_1, and *probability of harm if the error were to occur* as P_2. By this approach, an error's impact is the product $P_1 \times P_2 \times S$, where P_1 is always 1 for software. (Figure 4 illustrates this relationship.) Mitigation may not change the severity of the error, but it ***can*** reduce the probability of harm when the error occurs, i.e. P_2.



P_1 is the probability of a hazardous situation occurring
P_2 is the probability of a hazardous situation leading to a harm

Figure 4 Relationship of P_1 and P_2 for Assessing Risk
(redrawn from ISO 14971)

Mitigating Risks

Clearly, the purpose of conducting risk analysis is to reduce risks as far as possible – to make products safe.

Once we've identified and assessed potential errors, we need to take the next step by answering one of two questions:

- How can we prevent this error from occurring?
- What can we do so that if this error occurs, its impact will be minimized?

ISO 14971 boils risk mitigation down to three possibilities (in preference order):

- inherent safety by design
  - What about the software, or the process which uses the software, can we modify so that the error cannot occur, or the device or process will react to an error to maintain safety?
  - How can we simplify our design or unify the architecture so that we prevent sequences of events that would result in a hazardous situation?
  - How can we organize and simplify the user interface to avoid the chance of use errors?
  - What programming rules can we employ – use of static rather than dynamic memory allocation, restricting certain error-prone structures – which will avoid software anomalies?
  - Can we check for and reject out-of-range user inputs?
- protective measures in the medical device itself or in the manufacturing process

    - How can we have the system react to an error to maintain safety?
    - If the patient's safety depends on the device operating continuously, can we employ fault-tolerant architecture, for example a watchdog task which transfers processing to second, simpler device such as an FPGA if the main program fails?
    - If the safe response is to shut down the system, can we include independent shutdown hardware (such as a circuit breaker)?
    - Can we check that outputs, or sample results for a diagnostic device, are within specified limits and flag or suppress those which would be unsafe?
    - What can we do to notify the user (caregiver or process operator) if the error occurs?
    - What steps can we include in our process (such as inspection or testing) to detect the error and minimize the chance that it will affect released product?
    - Can we use physical means (door locks with restricted entry codes) to limit system access to properly trained and authorized users?
    - For a production or quality process where data loss is a risk, can we back up the system periodically?
- information for safety
    - What can we tell the user or operator, in written instructions or in on-screen prompts, to tell him/her what not to input, select, or perform that would create a hazardous situation? Can we require explicit training for users of the device?

Of course, where we have implemented risk control measures, we must always verify that they achieve the intended purpose!

Evaluating Residual Risks

Do some risks remain either because they were impossible to avoid, or because mitigating them completely would require too much expense or make the system unnecessarily complex?

Have any of the mitigations we've put in place introduced new risks?

In essence, once we have modified our design or process with risk mitigations, we need to run through the risk evaluation again to determine where we stand in the revised design or process.

Managing Risks Long-Term

Risk assessment performed only once is close to useless. Customers encounter errors that were never anticipated in-house. Product software changes. Third-party software components are updated. Process software is patched or modified – and in most cases, data accumulates – as the system is used routinely.

Maintaining a safe state is a key element of maintaining a validated state. Elements of long-term risk management will include:

- Monitoring safety: keeping track of safety related issues once a product is released, or a manufacturing process / quality process "goes live."
- Managing change: examining changes for possible safety impact, ensuring that only authorized changes are implemented, recording all changes
- Maintaining the system: for process software, ensuring that the data repository is backed up regularly, checking proper condition of the platform where the application is running, periodically auditing key elements such as the authorized user list

Every new piece of information feeds into risk evaluation and management – so the process needs to repeat on a regular basis.

Adjust Your Validation to Your Use – and to the Risk

Purchasing software certainly saves development time, but you are still responsible for assuring that the software serves its purpose.

Whether the OTS software is a component for your product or part of your process infrastructure, the validation process need not be complicated. The questions to be asked, and the concerns to be addressed, are always the same – but validation is never a "one size fits all", cookie-cutter activity.

Every item of process software, every off-the-shelf component built into a software product, requires a unique approach.

- Know what you need the software to do.
- Know how the software will be used – and by whom – in your product, or in your process.
- Know how you will demonstrate that the software fulfills your needs.
- Understand the risks associated with the software, and how you minimize them.
- Establish how you will keep control of the software version.

Above all other concerns, let safety of the product guide your validation choices.

References

21 CFR Part 210, *Current Good Manufacturing Practice In Manufacturing, Processing, Packing, Or Holding Of Drugs; General*

21 CFR Part 820, *Quality System Regulation*

ISO 13485, *Medical devices — Quality management systems — Requirements for regulatory purposes*, Second edition 2003-07-15

ANSI/AAMI/IEC 62304: 2006, *Medical Device Software – Software life cycle processes*

FDA, Guidance: *Off-The-Shelf Software Use in Medical Devices*, September 9, 1999.

FDA, *Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices*, May 11, 2005.

ISO 14971, *Medical devices — Application of risk management to medical devices*, Second edition 2007-03-01.

AAMI TIR36:2007, *Validation of software for regulated processes*, 13 December 2007.

ANSI/AAMI/IEC TIR80002-1:2009, *Medical device software - Part 1: Guidance on the application of ISO 14971 to medical device software*, 26 October 2009.