Seapine Software™

Agile in FDA-Regulated Environments

Why Agile?

Agile development methodologies improve the economics of product development by reducing costly and unnecessary project overhead. A major advantage that Agile has over Waterfall is the reduction in wasted time and effort that Waterfall developers spend designing or documenting functionality that is never implemented or that changes before implementation.

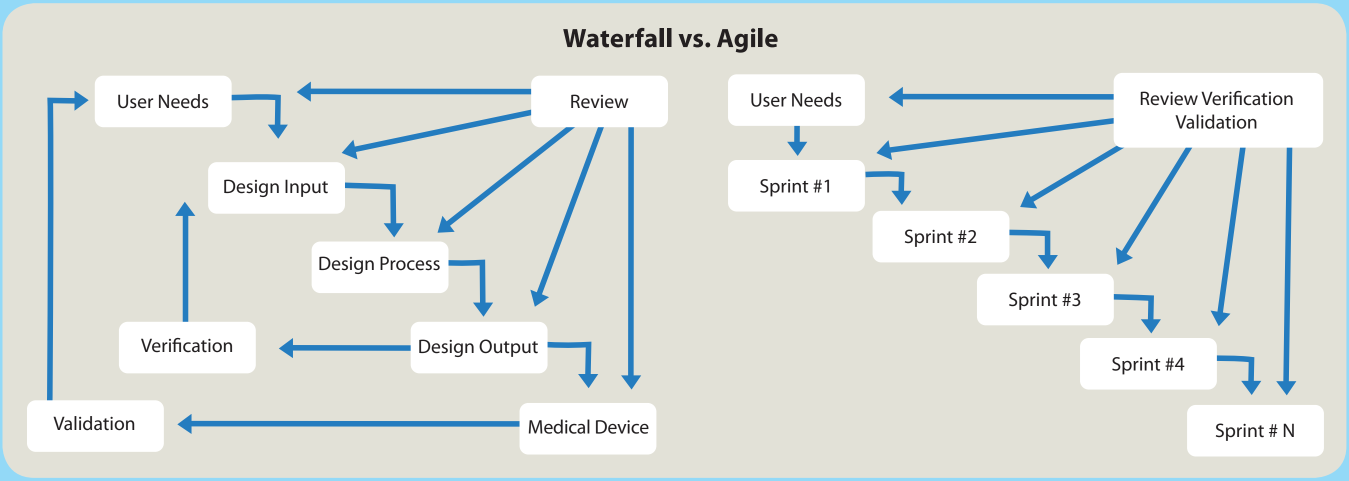
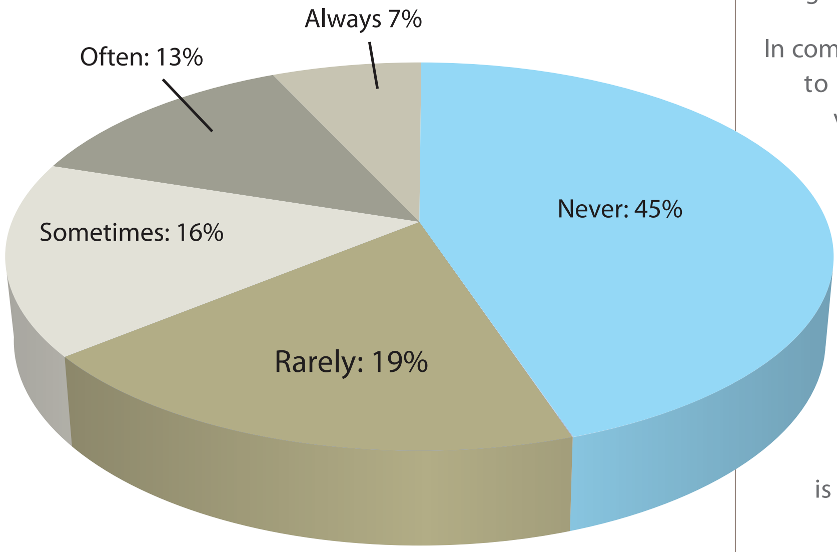


Figure: In Waterfall, an error in the Validation stage can kick the project back to square one. In Agile, Validation happens after each sprint.

We assume you have a basic knowledge of FDA requirements, as well as any other regulations for your industry. If you need an overview of the FDA's regulations, we recommend Seapine's "**How to Have a Painless FDA Audit**" white paper.

A report by the Standish Group found that 45 percent of code developed using Waterfall is never actually used. Add in the 19 percent reported as "rarely" used, and that's over half of a team's programming time and effort wasted. Then, add in the time spent validating and documenting that code, and you begin to see why Agile is gaining in popularity.

Average Percent of Delivered Functionality Actually Used with Waterfall Development:



SOURCE: Chaos Report v3, Standish Group

Agile and the FDA

At its core, Agile is a set of concepts and beliefs that stress flexibility and shared responsibility over rigid rules and formal processes. Agile teams welcome change to product requirements throughout the development cycle, with all team members working together to deliver a high quality product.

In comparison, the FDA requires concrete documentation to prove that processes were followed, features validated, issues addressed, and risks mitigated. Regulations such as FDA's Quality System Regulation (QSR), and standards such as ISO 13485, provide manufacturers of finished medical devices with a framework of basic requirements to use in establishing a quality management system.

So, can Agile be used in an FDA-regulated environment? Yes. Despite the apparent differences, it is possible to successfully adopt Agile practices in regulated environments if the transformation is treated with insight and caution.

“Software requirements are typically stated in functional terms and are defined, refined, and updated as a development project progresses.”

Many industry professionals think that FDA regulations require Waterfall. Although untrue, this misconception often prevents companies working in regulated environments from even considering Agile practices.

The truth is the FDA doesn’t mandate a specific development methodology as long as you produce the required artifacts and can prove the device is safe and effective, meets all design requirements, and satisfies user needs. The FDA publication “**Design Control Guidance for Medical Device Manufacturers**” outlines this in the section on Concurrent Engineering.

In this paper, we explore the areas that Agile methodologies can be used to effectively increase software development agility while continuing to meet FDA regulations. While we look specifically at medical device development, the information we share can be applied to most regulated environments.

Need an overview of Agile, download our **Exploring Agile: The Seapine Agile Expedition** eBook.

Complexity Drivers

The issue of complexity is one area where the values of Agile and the FDA clearly align, with both striving to keep complexity to a minimum. The FDA realizes that greater complexity leads to greater risk. Agile reduces the amount of wasted project work, which subsequently reduces complexity and allows the team to focus on delivering a quality product.

Several factors make projects more complex:

- Size
- Location
- Requirement density and volatility
- Architecture

Agile methodologies address these complexity drivers in several ways. First, Agile involves less hand-over documentation because requirements that are not in the final product are not included and, since design is postponed until implementation, there are no design changes, which reduces project size. Second, triaging active requirements addresses the amount of functionality that must be delivered. Finally, a smaller and more straightforward architecture, supported by refactoring techniques, reduces the footprint of the system.

Requirements and Agile

Agile methodologies tend to be sparse on documentation because of the focus on working software in place of extensive documentation. The FDA, however, requires evidence of design control, or how the requirements set has evolved during the development of the product, risk mitigation, and adherence to a Quality Management System (QMS). This difference in managing product requirements would appear to be a hurdle to FDA compliance as a team adopts Agile practices.

Look further into FDA guidance though, and you’ll find this from the **Regulatory Information Guidance**:

“Software requirements are typically stated in functional terms and are defined, refined, and updated as a development project progresses.”

Because of the unpredictable nature of requirements, Agile is actually superior to Waterfall when it comes to managing requirements and their associated changes.

The typical Waterfall process involves a large amount of time developing requirements on the front-end of a project, which adds overhead in managing those requirements and their changes throughout the project lifecycle.

Agile design is done Just In Time (JIT), at the last possible moment before the requirement is implemented. Agile and JIT lead to less wasted time developing requirements that will not make it into the final product, and less overhead in managing requirements and associated change requests and design rework. This closely aligns with FDA guidance by reducing the amount of documentation to only those requirements and changes that are actually in the shipping product.

In addition to improved efficiency and FDA alignment, JIT design improves product quality. The best time to perform design is when the greatest possible amount of information is known, which is as late as possible in the project lifecycle. This allows for a better and less volatile design and better-informed decision making, which should produce fewer defects.

In an Agile process, you start with an outline of what the product needs to be able to do. From that outline of requirements, user stories are created, breaking down the requirements into more manageable components and sub-components.

Agile user stories describe the functionality from a customer perspective--something they need to do. User stories are implemented during the sprints, and are written in this format:

As a <type of user>, I want to <goal> so that <reason>.

User stories should follow the **INVEST** principal:

Independent

Negotiable

Valuable

Estimable

Small

Testable

They should also contain the three **C**s:

Card (the user story)

Conversation (what is expected)

Confirmation (the acceptance criteria)

The relationship between the original requirement and the derived user story must be documented and maintained for traceability.

Risk Management Techniques

Risk and requirements go hand-in-hand for medical device manufacturers, and risk management is a key activity in regulated environments. Risk management in an Agile environment is a little different than it is in Waterfall, although the same steps are still followed.

According to the FDA, the major steps for an acceptable risk management model include:

1. Risk analysis
2. Risk evaluation
3. Risk control
4. Evaluation of residual risk
5. Post-release surveillance

When adopting Agile in a regulated environment, the risk management steps most affected are risk analysis, control, and evaluation. You should perform these steps in each sprint, and maintain a backlog of requirements with associated risk scores for each. This score is then used in evaluating and selecting requirements for the next sprint.

Verification and Validation

One of Agile's strengths is that it is people-oriented instead of process-oriented. As discussed, the FDA requires documentation that shows you have a good quality- and safety-oriented process in place. Meeting those expectations involves the same basic practices, whether the product development lifecycle is called Waterfall or Agile. The advantage of Agile is that it generates less process and documentation overhead, because you're only designing, building, and testing features that will be in the final product. You're also designing, building, and testing at the last possible moment, so there is less rework when things change.

The first step in any validation and verification process is to establish how the software will be validated, including the procedures you will use for validation and verification. Where your design and development plans specify what will be validated, your procedures should identify validation actions or sequences of actions.

After you have defined your validation procedures, you must document that the process was followed. This involves

NOTE: Documentation can be viewed from two perspectives: project documentation (describes how you are creating the product) and product documentation (describes how the end result works). An Agile process generally creates project documentation, but the FDA expects product documentation. Your development tools should make the generation of product documentation as seamless as possible.

the same deliverables that Waterfall would produce: work plans, requirement specifications, design documents, test cases, test plans, and test results. You must provide objective evidence that the requirements are satisfied. Agile is not an excuse to eliminate documentation. However, in Agile, these documents can be modified throughout the project.

Software Validation Planning

Established design and development plans should include a specific outline for how the software validation process will be controlled and executed. The software validation plan defines what the software validation effort will accomplish. All validation planning is documented in the plan, with details of the plan evolving throughout the project just as requirements may evolve. The validation plan is a significant tool in your quality management system.

The software validation plan usually describes the scope, approach, resources, and schedules of the development process, as well as the types and extent of activities, tasks, and work items. In regulated environments, product risk management is a crucial piece of the software validation plan.

The plan also identifies required tasks and procedures based on the software lifecycle model in use. Specific tasks for each activity are usually defined in the plan, including procedures, inputs, outputs, acceptance criteria, and required personnel and resources.

Agile software validation plans and associated tasks should describe your plans for creating user stories from requirements, developing user acceptance criteria, and selecting user stories for iterations. The FDA expects that your efforts will be focused on safety-related functionality and architecture.

Agile plans should also define the length of iterations, as well as the development practices that will be used (e.g., test-driven development, continuous integration, pair programming, feature-driven programming). Your validation plan should also describe your plans for refactoring, stand-up meetings, sprint reviews, retrospectives, and the charts you will use to visualize progress (e.g., burn down, burn up, velocity).

Independence in Review

The FDA expects independence when reviewing, verifying, and validating the product. This can be a potential pitfall in an Agile environment, depending on which Agile methodology you use. In an ideal Agile world, everyone does everything; developers would also do the testing, for example.

The FDA, however, requires that testing be performed by groups outside of development that are not directly responsible for the item being reviewed. Your company needs to define a strategy that ensures requirements with the highest patient risk are independently reviewed.

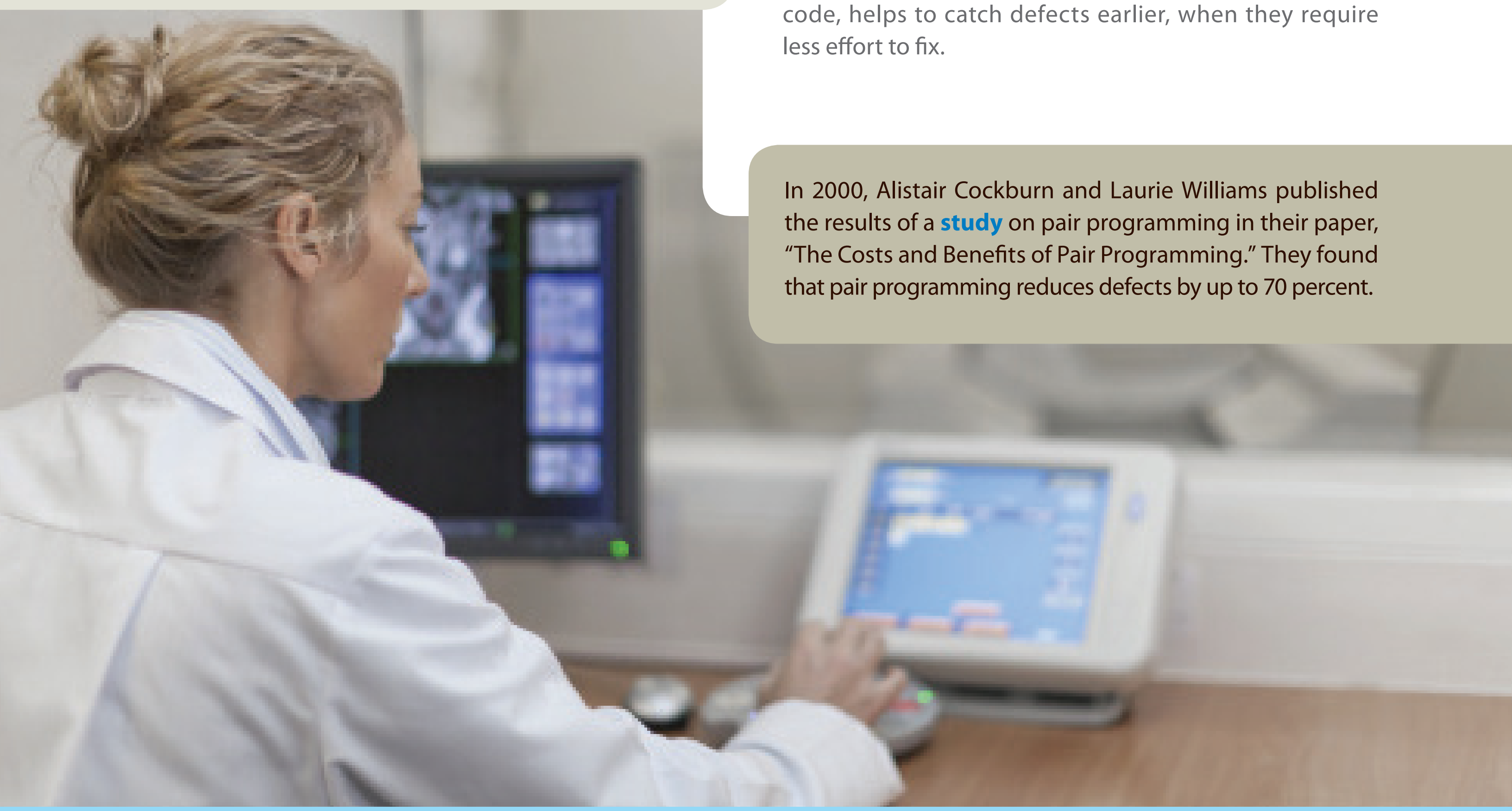
Validation After Change

In a regulated environment, the software must be validated after every change to ensure it is still fit for use and accomplishes its intended purpose. Automated regression testing and real-time traceability are critical components for validating changes efficiently throughout the product development cycle. They are also critical components for successful Agile adoption.

Automated testing pays off quickly when a change needs to be validated. If code was developed using test-driven development, there should already be a number of tests in place to verify that existing functionality continues to work as expected. Continuous integration and automated smoke tests also help with validation after a change because the failure is immediately apparent if changes break other code.

When using an Agile methodology, real-time traceability is the key to using traceability and impact analysis to drive your validation process. Because Agile intrinsically decreases the amount of documentation generated by any project, it is important to make sure your application lifecycle management tools automatically generate traceability documentation.

Impact analysis is necessary to determine the scope of a change and identify any impacted requirements. As requirements change and new code is written, everything linked to those artifacts must be re-validated; this is especially important if the change is connected to a safety-related requirement. Real-time traceability means that test cases, defects, and other project artifacts associated with those changed requirements can immediately be identified and flagged as needing attention. This streamlines the process of managing change, improving the team's agility and boosting product quality.



Defect Prevention

No one wants to release defective code but, when it comes to medical devices, defect prevention is critical. Medical device developers must have an extremely low fault tolerance due to the potential danger to people's lives if there are bugs in the final product. Fortunately, Agile promotes quality and reduces defects.

Coding

Sometimes requirements can be ambiguous because English is inherently imprecise. Code, on the other hand, is far less ambiguous than written requirements. For this reason, code can function as a communication tool, with unit tests demonstrating what the correct behavior of the code should be.

There are three Agile practices that are particularly effective in reducing the number of bugs: pair programming, test-driven development, and code refactoring.

Pair programming involves two developers working in tandem, literally side by side. One developer acts as the "driver," coding, typing, and focusing on the task at hand. The other developer serves as the "observer," watching and reviewing the driver's work while considering alternative approaches.

When pair programming is used, the paired developers often discover more solutions to coding issues than they would have working separately. There is also peer pressure to follow standards and not take shortcuts. This type of static testing, with two sets of eyes reviewing code, helps to catch defects earlier, when they require less effort to fix.

In 2000, Alistair Cockburn and Laurie Williams published the results of a **study** on pair programming in their paper, "The Costs and Benefits of Pair Programming." They found that pair programming reduces defects by up to 70 percent.

The FDA also sees the value in this kind of static testing, and has begun pushing for more of it in the Validation and Verification process. To help pair programming meet FDA expectations, you should introduce some form of documentation into the process. Typically, observers compile this documentation as they're watching and reviewing the drivers' work. This adds a bit of overhead to pair programming, but in the end you'll be better able to prove FDA compliance in the event of an audit.

Test-driven development (TDD) is a disciplined coding practice that has several benefits for improving quality. For example, TDD often results in a better design because of frequent refactoring and built-in unit tests for regression testing. TDD also results in a smaller footprint of code because there is less dead, untested code, and all code is unit-testable code.

The TDD process consists of four phases:

1. Write a test that fails.
2. Change the code so all tests pass.
3. Verify that all tests pass.
4. Refactor as needed, to get clean consistent design.

In a 2008 **study**, "Realizing Quality Improvement through Test Driven Development: Results and Experiences of Four Industrial Teams," case studies were conducted with three development teams at Microsoft and one at IBM, all of which had adopted TDD. The results were remarkable: the IBM team saw a 40 percent drop in defect density, while the Microsoft teams saw drops in defect density from 60 to 90 percent.

To learn more about TDD, check out Seapine's **TDD 101 Webinar series**.

Refactoring is the process of simplifying code by improving the design, without changing its functionality. It keeps code from becoming brittle, stale legacy code. It also leads to a better design because you continuously ponder how to improve the code instead of just figuring out how to make it work.

When should you refactor? First, it is only safe to refactor if you have unit tests to verify accepted behavior. Then it's a matter of looking for "**code smells**," or symptoms that indicate a deeper problem. This may mean that something is wrong with the code, so code smells should always be investigated.

Some common code smells include:

- Duplicated code
- Long method
- Large class
- Feature envy
- Lazy class/Freeloader

Be sure to refactor in small, controlled steps and make refactoring a constant part of your development cycle. After every refactor, you will need to validate the code, which is where unit tests can improve efficiency and help developers identify defects right away. Tests can be refactored too, but use caution because you cannot validate that tests have been refactored correctly.

Testing

Testing is performed as part of each iteration throughout an Agile project, instead of in its own phase at the end of the project. Again, the idea is that the earlier defects are found, the easier they are to fix.

User acceptance testing (UAT) seeks to verify that working software fulfills customer requirements. UAT acts as a final verification of the required business function and proper functioning of the system, emulating real-world usage conditions on behalf of the customer. If the software works as intended and without issues during normal use, you can reasonably extrapolate the same level of stability in production.

Unit tests attempt to verify that a specific piece of code behaves correctly, and that nothing was broken by the latest changes. As you build up unit tests over time, they become a built-in set of regression tests at the code level that help you find issues in existing code before changes get to the testing phase.

System tests verify the requirements at the highest level.

Integration tests exercise internal interfaces that are inaccessible by system tests, and reduce the complexity and effort of testing a large system.

Facilitating Agile with Software Tools

Because Agile generates less documentation, it can be challenging to maintain the required records and traceability matrices needed to satisfy FDA regulations. Fortunately, software tools can help with this and, in some cases, can automatically generate the required documentation.

Some integrated software tools are flexible and configurable enough to create a framework to facilitate Agile development and good quality processes. These tools make tracking and linking requirements, tests, defects and risk easier and less time-consuming.

The best tools allow you to perform issue tracking, requirements management, and testing in the same interface with a shared data and security model. These tools facilitate and automate traceability, as well as lessen your reliance on manual updates when something changes. As artifacts transition through their lifecycles, reusable data and content minimizes human errors and ensures greater visibility.

Requirements and Risk Management

A software tool can help track changing requirements, risk artifacts, and other associated work items. A good tool will automatically track requirement changes, track requirement to user story evolution, calculate risk values, and generate traceability matrices to record risk control measures, requirements, architecture and design elements, and verification and validation. With this kind of tool in place, your team can focus on building high-quality software and the tool will automatically provide the documentation when it is needed.

Test Planning

Testing is the critical activity that ensures requirements have been implemented correctly, but it's just as important to prove to the FDA that you've run the tests that verify and validate each requirement. An integrated toolset will manage your test cases and test results, automatically maintaining and reporting on the links between requirements and tests, along with comprehensive historical proof. Teams developing higher risk products require strong object evidence, which your tool should easily enforce and provide.

Issue and Task Tracking

Although Agile can help reduce the amount of defects, it won't completely eliminate them. You will still need a way to track issues and other tasks. Your software tool should be flexible enough to support your Agile process, as well as provide traceability from defects back to tests and requirements.

Automated Testing

Because testing is performed throughout an Agile project, it's not feasible to do a full manual regression test at the end of each iteration. A good automated testing tool will alleviate this pain by performing regression testing automatically, and improve test coverage in each iteration by allowing you to add automated testing for the new functionality.

Continuous Integration

Continuous integration involves integrating all developer changes early and often. Ideally, the continuous integration tool will compile the code and run all unit tests for each committed change. This way, integration issues and bugs are found earlier in the process, when they are less expensive to fix.

Elements of an Ideal Solution

When selecting your tools, you should consider the following capabilities:

Automation

A good tool uses configurable notifications and escalations, based on your business rules, to ensure items don't fall through the cracks. It reduces copying and pasting errors, minimizes the need to manually updating other systems or spreadsheets, and ensures the right people are being notified at the right time.

Security

No two businesses are the same, so a highly configurable security model to manage user permissions is necessary. The more complex your products and dispersed your teams become in the future, the more you'll need a security model that grows with your business. If you outsource or work with third-party suppliers, you need the added assurance that your intellectual capital is protected.

Traceability

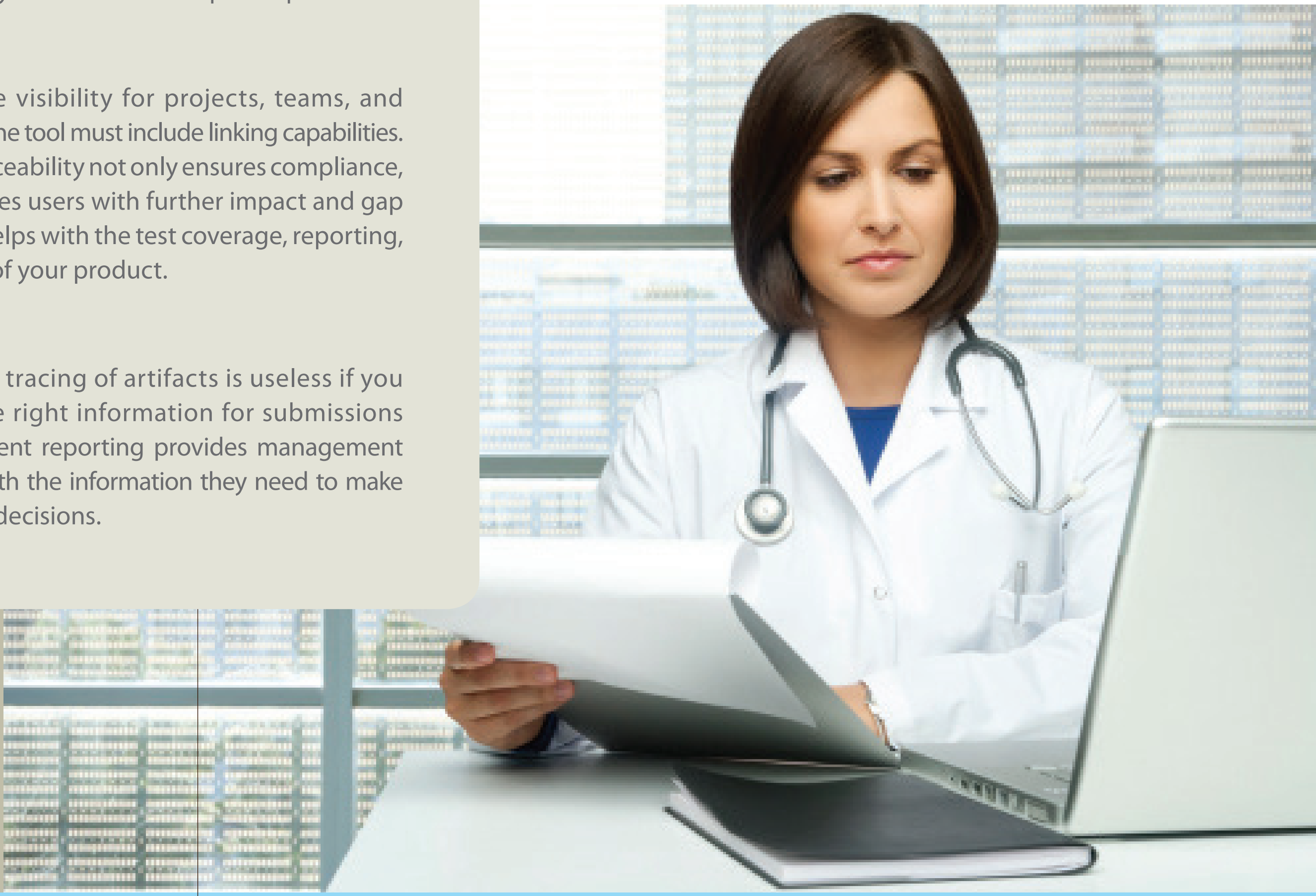
To help ensure visibility for projects, teams, and business units, the tool must include linking capabilities. End-to-end traceability not only ensures compliance, but also provides users with further impact and gap analysis. This helps with the test coverage, reporting, and readiness of your product.

Reporting

Managing and tracing of artifacts is useless if you cannot get the right information for submissions or audits. Efficient reporting provides management and auditors with the information they need to make well-informed decisions.

Conclusion

Because of its emphasis on working software and incremental development cycles, which reduce requirements and software changes, Agile can be an excellent way for companies in regulated industries to reduce risk while bringing high-quality products to market faster than their competitors. While the reduced amount of documentation may seem problematic from a traceability standpoint, integrated software tools can make up for this by automatically tracking changes, linking artifacts, and generating the reports needed to meet the requirements of the FDA and other regulators.



About Seapine for Life Sciences

Founded in 1995, Seapine Software is based in Mason, Ohio, with sales and support offices located in Europe, Asia-Pacific, and Africa. Hundreds of leading medical device, pharmaceutical, biotechnology, and clinical research organizations rely on Seapine to streamline their core development processes, drive innovation, and gain a competitive edge.

Learn more at **life-sciences.seapine.com.**