PARASOFT®

IEC 62304 SOFTWARE COMPLIANCE IN THE MEDICAL INDUSTRY

Table of Contents

Overview

MEDICAL DEVICE INDUSTRY OUTLOOK

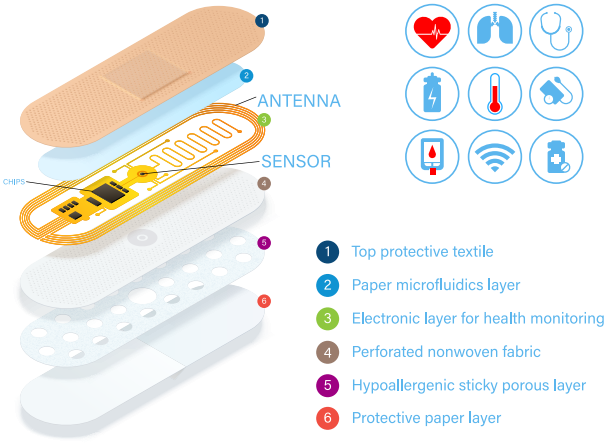
The medical device industry is going through a highly innovative period with unparalleled growth and improvements in healthcare due to technology advancements. Technologies like 5G offer massive connection power at speeds required to assist this transformation.

All this is being fueled by modern living and the need to:

- Track past health records.
- Obtain information in realtime.
- Find better ways to identify, diagnose, and treat diseases remotely or in person and in more efficient and cost-effective ways.

The medical industry is accomplishing life-changing transformations, including medical strides in the following areas.

- Robotic surgical systems.
- 3D printing.
- Nanotechnology-based devices.
- Smart implants with sensors that communicate with your cell phone or computer and transmit data that allows doctors to monitor the healing process and influence recovery so as not to have to operate again.



CONSUMER WEARABLES

The transformation is also evident from devices such as wearable technology that monitor your health vitals. Grade I and II wearables such as wrist watches, rings, and clothing monitor biometrics like O2 saturation, ECG, distress, hydration, psychological state and more.

Grade III comes in the form of implantables, epidermal patches, or ingestibles that can measure biometric values including glucose, cortisol, drug adherence, tumor response, and much more.

INTERNET OF MEDICAL THINGS (IOMT)

IoMT addresses significant challenges including:

- Lowering the cost of medical services.
- Optimizing patient outcomes.
- Improving operational efficiency.

The IoMT's primary function is remote patient monitoring, which comes with significant benefits.

- Reduces costly hospital stays.
- Increases the accuracy of diagnostics and interpretation of symptoms.
- Improves efficiencies in medical staff management.
- Notifies doctors immediately in the event of an emergency.

With all these advantages, medical companies are constantly looking for ways to expand the use and capabilities of IoMT technology.

ROBOTIC SURGICAL SYSTEMS

Today, robotic surgical systems, such as Da Vinci, can assist with an array of procedures.

- Precision and complex radical prostatectomy
- Inguinal hernia repair
- Ureter repair
- Cholecystectomy
- Other types of surgery performed remotely or nearby

Robotic systems reduce surgical errors and make surgery less invasive. Various kinds of medical robots and the levels of tasks they perform are expected to increase. In turn, the market is expected to rise at a compound annual growth rate (CAGR) of 9.31% from 2022 to 2030. Helping to fuel this innovation is the fascinating potential that artificial intelligence (AI) adds to the table.

AI/MACHINE LEARNING

AI and machine learning (ML) provide medical devices with the following capabilities.

1. Analyze the environment.
2. Collect data points.
3. Make decisions based on those analyses.

The ability to accurately diagnose patient problems can significantly impact patient outcomes. For example, training software algorithms to learn from real world use and having their behavior be adaptive or change over time can provide diagnostic information, such as identifying skin cancer, by having the ability to see and interpret images.

AI-enabled products can result in inaccurate, even potentially harmful, recommendations for treatment, caused by unanticipated sources of bias in the information used to build or train the AI. The FDA regulates some, but not all AI-enabled medical products. Since it plays an important role in ensuring the safety and effectiveness of medical products, it's currently considering how to adapt its review process of AI-enabled medical devices. Therefore, in 2019, the FDA published the discussion paper, "Proposed Regulatory Framework for Modifications to Artificial Intelligence/Machine Learning (AI/ML)-Based Software as a Medical Device (SaMD)."

SOFTWARE AS A MEDICAL DEVICE (SaMD)

Software is what we think of when we think of modern technology. It is all around us performing tasks faster, more efficiently and providing us with amazing capabilities and conveniences. In the case where software is intended to be used for medical purposes but it's capable of being run on various platforms, like an app on your mobile devices, laptop computers, the cloud and such, it's referred to as Software as a Medical Device (SaMD) by the International Medical Device Regulators Forum (IMDRF).

Software is also being used to put us in a quasi-realistic world where we can experience and learn more effectively by being immersed in a simulated medical scenario. Virtual reality is a fascinating technology that will only expand and change us in ways yet to be considered.

VIRTUAL REALITY (VR)

Virtual reality technology offers education through immersive learning. Put your headset on and instantly find yourself in one of many virtual worlds. One world could target the patient's anxiety or focus away from pain. Another world environment could be a collaborative scenario where you're training with other clinicians before performing an operation by visualizing and stepping through a surgical procedure.

The physics in the VR world can mimic that of our reality so users can learn how to interact with medical instruments and engage the patient. One amazing example is VR Technology Used To Help Separate Conjoined Twins. Virtual reality in the medical industry is still in its early stages, but a fascinating takeover for the greatest benefit of patients and doctors.

CYBERSECURITY CHALLENGES

The medical transformation taking place also comes with challenges. Cybersecurity is at the top of minds for medical device manufacturers. The FDA is now actively holding these manufacturers accountable for problems that arise related to security. Medical device manufactures have historically focused on functional safety, stakeholder requirements, patient convenience, and time to market rather than security.

The expanding set of connected medical devices along with healthcare systems have exacerbated security threats. Threats such as a hackers stealing patients' personal data, taking control of a medical device, and holding it for ransom. Worse yet, turn off or emit critical alerts, change drug administering conditions, and commit or cause loss of life.

Robust measures are needed to secure medical devices, including testing against the myriad of security threats. There is a new breed of cyberterrorism that's daunting and the reason why enhanced cybersecurity is a priority for the medical industry.

Organizations need to adopt security management principles. This means that they need to build teams to address security. For example, have a team of security engineers to stay on top of regulatory requirements and support systems engineering with security requirements. They need to work together in defining, controlling, and improving processes that lead to diminishing vulnerabilities in the IoMD.

THE ROLE OF STANDARDS & REGULATIONS

The FDA is responsible for protecting public health. This includes safeguarding against unsafe and insecure medical devices. Part of the solution integral to the execution of the FDA's mission is the use of standards, which include medical device standards for development and performance, the testing methodology, the compliance criteria, the manufacturing practices, and much more.

Standard ISO 13485 is the international standard for quality management systems in the medical device industry. It provides a practical foundation for manufacturers to organizationally address business and technical practices that help run your organization efficiently and effectively. Best practices and processes are offered that organize, reduce costs, mitigate risks, boost productivity, and drive continuous improvement. Organizations certified to this standard demonstrate a commitment to excellence and delivering quality.

Since software is the life blood of most medical devices, standard IEC 62304 covers the design, development, and maintenance of software to ensure safety. It's also important to understand that IEC 62304 expresses and expects that medical device manufacturers obtain and apply complementing and supporting information and practices from other standards.

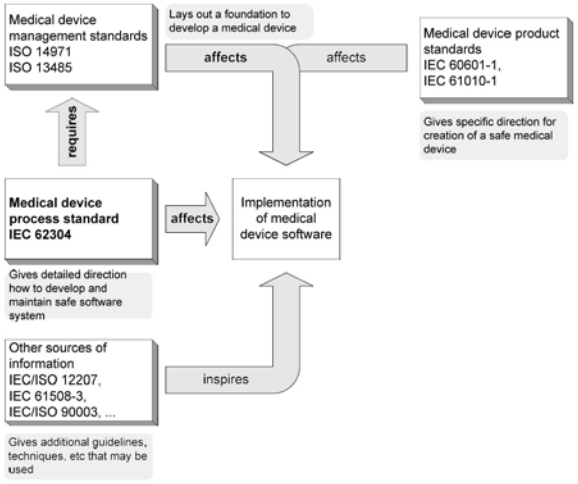


Figure 1-1: Relationship of key medical device standards to IEC 62304

Standard ISO 14971 on risk management ensures that the software risks are known and remediated.

IEC 60601 goes into additional detail on the software development life cycle, each phase, and practices of development regarding safety.

The FDA recognizes standard AAMI TIR 57, “Principles for medical device security – Risk management.”

IEC 62443 is the most recognized international cybersecurity standard and is heavily used in the industrial automation industry for IoT, which thoroughly applies to IoMD.

Standard IEC 62443 and UL 2900 have been officially recognized by the FDA. To put it in comparison, a product conforming to IEC 62443 would be 90% compliant with UL 2900-1, and in opposition, UL 2900-1 would be just 50% compliant with IEC 62443.

For healthcare IT cybersecurity, ISO 27799 is used.

In addition, to stay up to date on government regulations, the Food and Drug Administration provides a database with an arrangement of laws or rules into a systematic code. Regulations are updated each year. FDA Code of Federal Regulations (CFR) Title 21 provides FDA regulations on medical devices.

WHAT IS IEC 62304?

IEC 62304 is a functional safety standard that specifies the life cycle process and requirements for the development of medical software and software within medical devices. It covers development phases like the following:

- Requirements management
- Architectural design
- Implementation
- Integration
- Test verification and validation, including maintenance

For each phase of development, the standard lays out best practices with recommended activities and tasks to ensure delivery of safe, secure, and high-quality medical software systems.

The first edition of the standard was released in May 2006. It's an adaptation of the industrial IEC 61508 functional safety standard. IEC 62304 has had one update in 2015—Amendment 1 where a few requirements were added and others were amended, particularly those related to safety classification, the handling of legacy code and software item separation. These are discussed in "Reporting and & Analytics for Medical Devices Software."

The second edition of the standard 62304:2021 has been in a draft status for about six years and it was rejected. This draft version contains some great requirements towards cybersecurity, but one of the problematic aspects is that the scope of the standard has been extended into health software or health software systems, which is not qualified as a medical device. The latest version remains "IEC 62304:2006/AMD 1:2015."

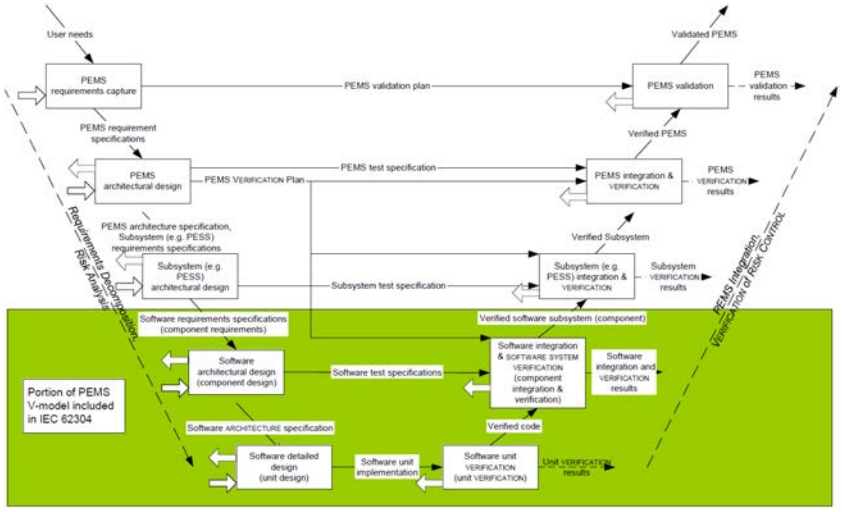


Figure 2-1: Programmable electrical mechanical systems (PEMS) V-model process life cycle

IEC 62304 is comprised of nine clauses. Like other formal documents, scope, normative references, terms, and definitions are provided to establish supporting context and terminology. This is covered by clauses 1 – 3.

The substance is found in the clauses that define software processes, which help identify risk and mitigation of those risks for the reason that testing alone is insufficient. Accumulated knowledge and best practices from the authors of the standard provide immense value.

1. Scope
2. Normative references
3. Terms and definitions
4. General requirements
5. Software development process
6. Software maintenance process
7. Software risk management process
8. Software configuration management process
9. Software problem resolution process

CLAUSE 4 – GENERAL REQUIREMENTS

Clause 4 has three requirements that relate to addressing risk from an organizational perspective and from a technical viewpoint as well.

4.1 – The first requirement is to demonstrate the use of a quality management system that complies with ISO 13485. A quality management system helps organizations manage all their quality, regulatory, clinical, and product development activities across the entire device life cycle in a single, purpose-built platform. Greenlight Guru is arguably the leading solution in the medical industry.

4.2 – The second requirement is to apply a risk management process that complies with ISO 14971. IEC 62304 Clause 7 goes into recommended process details regarding risk management and also references ISO 14971. ISO 14971 deals specifically with defining a framework for effective management of the risks, which include activities like identifying hazards and performing a risk analysis and risk evaluation. Various methods exist, such as failure mode event analysis (FMEA), fault tree analysis (FTA), hazard operability analysis (HAZOP) and others.

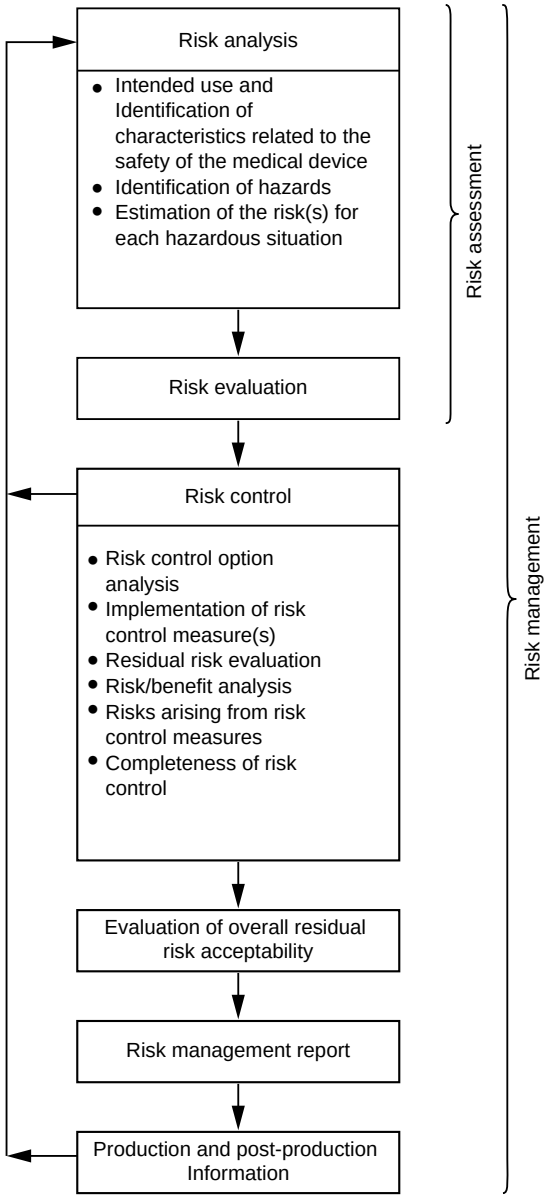


Figure 2-2: ISO 14971 risk management workflow

This analysis is a crucial task that leads to categorizing the severity level of potential hazard scenarios. Levels include:

- Catastrophic
- Critical serious
- Minor
- Negligible

Ultimately, the analysis derived from the risk management activities will impact the software system under development, such as having a software safety class assigned to the software items.

4.3 – The third requirement is software safety classification. Based on the risk analysis performed in 4.2 and the possible effects on the patient, operator, or other people resulting from a risk or hazard, the software will be assigned a class level based on severity. Classification affects the level of rigor in testing and other development factors.

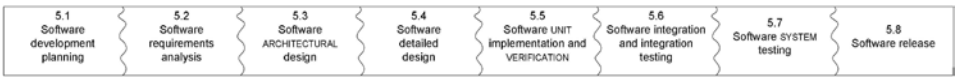
- Class A: No injury or damage to health is possible.
- Class B: Non serious injury is possible.
- Class C: Death or Serious injury is possible.

CLAUSE 5 – SOFTWARE DEVELOPMENT PROCESS

Clause 5 consists of eight subclauses and require the medical device manufacturer to consider the software development process, perform software planning and deliver a software development plan. It must contain things like the software development life cycle methodology selected, maybe Agile DevOps and scrum, an incremental/spiral methodology, or even waterfall. Modern Agile methodologies like DevOps CI/CD have become widely favored and adopted.

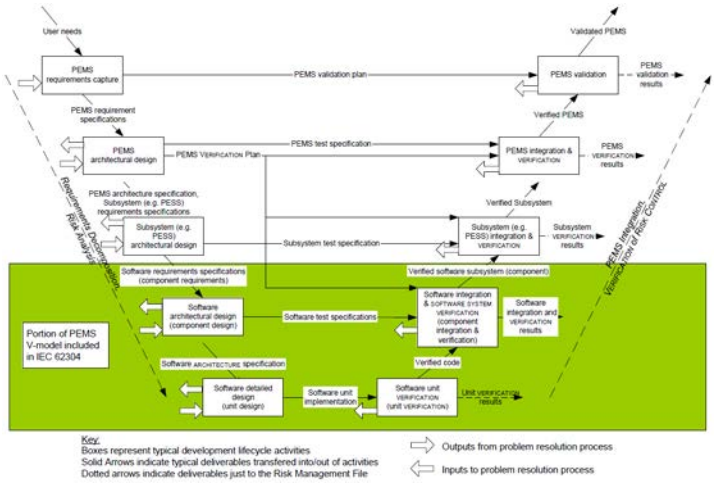
Documents and artifacts are to be produced for each of the subclauses in Clause 5 as the teams work through each phase of the V-model process using their chosen SDLC methodology. The V-model is not the waterfall methodology. It's a process and set of development phases that teams should include in their SDLC methodology to help ensure delivery of a safe, secure, and quality medical device.

Figure 2-3: Clause 5 SDLC phases



Documents and artifacts are to be produced for each of the subclauses in Clause 5 as the teams work through each phase of the V-model process using their chosen SDLC methodology. The V-model is not the waterfall methodology. It's a process and set of development phases that teams should include in their SDLC methodology to help ensure delivery of a safe, secure, and quality medical device.

Figure 2-4: IEC 62304 V-model software process life cycle (not a development methodology)



SUBCLAUSE 5.1 SOFTWARE DEVELOPMENT PLANNING

Subclause 5.1 is about planning and documentation. The table below captures each requirement. The far right columns show software risk classifications A, B, and C. If there's an "X" in a field, then the requirement or subclause task is recommended for that class. Note that subclause 5.1.4 list standards, methods, and tools are listed. MISRA C 2023 coding standard, DevOps, Git, and Parasoft are examples of standards, methods, and tools. Based on the class value, the tool may need to be TÜV certified for use on safety-critical systems.

Clause	Subclause	Class A	Class B	Class C
5.1 Software development planning	5.1.1 Software development plan	X	X	X
	5.1.2 Keep software development plan updated	X	X	X
	5.1.3 Software development plan reference to System design & development	X	X	X
	5.1.4 Software development standards, methods and tools planning			X
	5.1.5 Software Integration & Integration testing planning		X	X
	5.1.6 Software Verification Planning	X	X	X
	5.1.7 Software Risk Management Planning	X	X	X
	5.1.8 Dovumentation Planning	X	X	X
	5.1.9 Software configuration Management	X	X	X
	5.1.10 Supporting item to be controlled		X	X
	5.1.11 Software configuration item ontrol before verification		X	X
	5.1.12* Identification and avoidance of common software defects		X	X

Parasoft contributes to most of these subclauses by facilitating a reporting and analysis project dashboard with automated document generation on test verification results and change-based management through configuration management and more.

SUBCLAUSE 5.2 SOFTWARE REQUIREMENTS ANALYSIS

Subclause 5.2 activities shall have the manufacturer gather all requirements from:

- Regulations
- Customer
- Cybersecurity
- All others that apply to the device

Gathered requirements need to be partitioned into the following domains and decomposed into system requirements:

- Hardware
- Software
- Mechanical

Risk analysis is re-evaluated on all these system requirements, and then they must be verified through traceability back to the original requirements. System and acceptance test cases also need to be defined or created and traced back to requirements.

Clause	Subclause	Class A	Class B	Class C
5.2 Software requirement analysis	5.2.1 Define and document software requirements from system requirements	X	X	X
	5.2.2 Software requirements content	X	X	X
	5.2.3 Include Risk Control measure in software requirements		X	X
	5.2.4 Re-evaluate medical device risk analysis	X	X	X
	5.2.5 Update system requirements	X	X	X
	5.2.6 Verify software requirements	X	X	X

Parasoft has integrations with ALM tools like codebeamer, Jama, and Polarion, which provide robust solutions for requirements analysis, design, and partial verification. Parasoft completes the verification through traceability to test cases and down to the code, and validates requirements through test execution of the following testing methods:

- Unit
- Integration
- Systems
- Regression
- And more

Subclause 5.3 Software Architectural Design

Subclause 5.3 requires the manufacturer to define the software architecture of the device from system requirements. The system requirements are decomposed into high-level requirements and used to architect the software subsystems and their interfaces. Some software components may be legacy or software of unknow provenance (SOUP).

SOUP is off-the-shelf software that may have been developed with an unknown software development process that didn't address safety. SOUP software typically has been out in the field and working for quite some time, so there could be a sense of comfort in its reuse. However, the following are necessary for SOUP:

- Risk analysis.
- Assurance it will meet functional and performance requirements.
- Assurance it will operate properly on the hardware it's going to run on.

Clause	Subclause	Class A	Class B	Class C
5.3 Software architectural design	5.3.1 Transform software requirements into an architecture		X	X
	5.3.2 Develop an architecture for the interfaces of software items		X	X
	5.3.3 Specify functional and performance requirements of SOUP item		X	X
	5.3.4 Specify system hardware and software required by SOUP item		X	X
	5.3.5 Identify segregation necessary for risk control			X
	5.3.6 Verify software architecture		X	X

Parasoft C/C++test enables testing on target hardware. Many development ecosystems that include ARM, Intel, TASKING, microcontrollers from Microchip, TI, Renesas, compilers like IAR, Microsoft, Keil and so much more are supported. This also includes containerization with solutions from Docker and others.

Parasoft has a powerful integration with an architectural design tool called Lattix, which helps you understand, define, and control your software architecture. Parasoft then helps enforce the architecture with automatic detection of rule violations to control undesirable architectural deviations.

Watch the on-demand webinar and demo, Achieve Faster Compliance to ISO 26262 With Lattix and Parasoft in a GitLab CI Pipeline.

SUBCLAUSE 5.4 – SOFTWARE DETAIL DESIGN

Subclause 5.4 tasks require refining the architecture and its interfaces into software detail designs. Detail designs articulate the functional and nonfunctional capabilities that each unit must provide. This activity further refines requirements into a finer level set of low-level requirements and linked back to higher-level requirements.

Optionally, low-level requirements can be linked to code and detail design artifacts. Taking advantage of a requirements traceability matrix, engineers can quickly identify missing gaps between requirements, design, implementation, and the test cases that verify the requirements.

Clause	Subclause	Class A	Class B	Class C
5.4 Software detail design	5.4.1 Refine software architecture into software units		X	X
	5.4.2 Develop detail design for each software unit			X
	5.4.3 Develop detailed design for interfaces			X
	5.4.4 Verify detailed design			X

Parasoft provides capabilities to perform walkthroughs or code reviews on implemented detail designs. The refined requirements that link to the detail design artifacts can be verified and validated through test cases realized by Parasoft. These test cases test interfaces and validate design functionality and verify requirements.

CLAUSE 5.5 – SOFTWARE UNIT IMPLEMENTATION AND VERIFICATION

Subclause 5.5 activities require the manufacturer to write or implement the code for each software unit as expressed by the detail design spec and requirements. It provides acceptance criteria or verification that the code does what it's supposed to do, and that risk control measures have been implemented in the code.

Not captured in the below table are the additional acceptance criteria in Subclause 5.5.4 which includes test verification methods like data and control flow, initialization of variables, memory management, including memory overflows, boundary value conditions and more, which Parasoft supports through automated testing.

Clause	Subclause	Class A	Class B	Class C
5.5 Software unit implementation and verification	5.5.1 Implement each software unit	X	X	X
	5.5.2 Establish software unit verification process		X	X
	5.5.3 Software unit acceptance criteria		X	X
	5.5.4* Additional software unit acceptance criteria			X
	5.5.5 Software unit verification		X	X

Parasoft provides automated test verification methods, such as static analysis, that support data and control flow analysis, memory overflows and much more. It also supports coding standards like MISRA C/C++, AUTOSAR C++14, CERT C/C++, CWE, and more. In fact, organizations can create their own custom coding standard using C/C++test.

CLAUSE 5.6 – SOFTWARE INTEGRATION AND INTEGRATION TESTING

Subclause 5.6 activities require the manufacturer to perform software integration and integration testing. Here you've started to move up a level of abstraction from the unit to integrating software units together that form a higher level of functionality defined by high-level requirements.

These integrations need to be tested and ensure that the transfer of data and control across the integrated items function as intended, including the interfaces between them. Regression testing is also part of this activity. Since bugs may be found, the problem-resolution process needs to be applied here. Clause 9 addresses the problem-resolution process.

Clause	Subclause	Class A	Class B	Class C
5.6 Software integration and integration testing	5.6.1 Integrate software units		X	X
	5.6.2 Verify software integration		X	X
	5.6.3 Test integrated software		X	X
	5.6.4 Integration testing content		X	X
	5.6.5 Verify integration test procedures		X	X
	5.6.6 Conduct regression tests		X	X
	5.6.7 Integration test record contents		X	X
	5.6.8 Use software problem resolution process		X	X

Parasoft supports integration and regression testing. Our solutions integrate with tools like Jira so that bugs identified and captured during testing can be exported to Jira for defect management or problem resolution.

SUBCLAUSE 5.7 – SOFTWARE SYSTEM TESTING

Subclause 5.7 moves up the verification ladder another level of abstraction from integration testing to system level testing. Engineers know this as black box testing. This activity ensures that the system functionality and other quality of service requirements are verified. If software changes are made, including the implementation of new functional or nonfunction requirements and bug fixes, regression testing is required. Test results and progress should be recorded.

Clause	Subclause	Class A	Class B	Class C
5.7 Software system testing	5.7.1 Establish tests for software requirements	X	X	X
	5.7.2 Use software problem resolution process	X	X	X
	5.7.3 Retest after changes	X	X	X
	5.7.4 Verify software system testing	X	X	X
	5.7.5 Software system test record contents	X	X	X

Parasoft supports system testing and supports structural code coverage for statements, branch, and modified condition decision coverage to ensure that you've done enough testing. Change-based testing is another valuable feature and performed during regression testing. Instead of running all test cases, only the validating test cases run automatically to test only the code that has changed. All test results are tracked and documented, and compliance reports can be auto generated.

SUBCLAUSE 5.8 – SOFTWARE RELEASE

Subclause 5.8 covers activities associated with the release. The manufacturer is required to document test verification results, including the following:

- Compliance reports
- All anomalies and issues captured
- Answers and resolutions for all captured anomalies and issues

The software releases need to be archived and ensure repeatability of software releases.

Clause	Subclause	Class A	Class B	Class C
5.8 Software release	5.8.1 Ensure software verification is complete	X	X	X
	5.8.2 Document known residual anomalies	X	X	X
	5.8.3 Evaluate known residual anomalies		X	X
	5.8.4 Document released versions	X	X	X
	5.8.5 Document how released software was created		X	X
	5.8.6 Ensure activities and tasks are complete		X	X
	5.8.7 Archive software	X	X	X
	5.8.8 Assure repeatability of software release	X	X	X

Parasoft DTP is an intelligent dashboard that shows progress toward completion through charts and graphs. DTP consolidates testing results, generates compliance and detailed software verification reports, and performs actionable analytics.

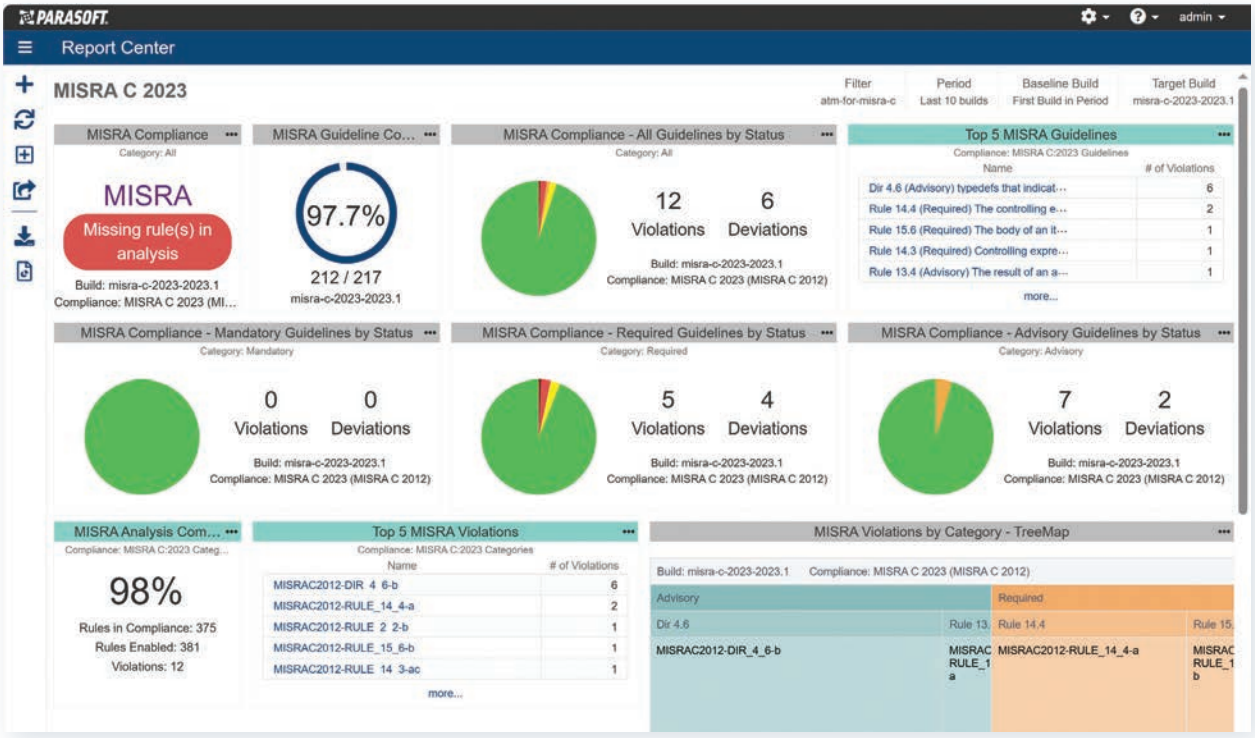


Figure 2-5: Example Parasoft DTP dashboard on progress towards MISRA compliance

CLAUSE 6 – SOFTWARE MAINTENANCE PROCESS

Clause 6 is all about software maintenance, which is a crucial part of the SDLC in medical devices. The manufacturer is expected to establish their software maintenance plan. Processes and assurances need to be in place to prevent inappropriate software updates and upgrades. In fact, many incidents in the field that happen after the product has been released in the market are related to product maintenance. Clause 6 is considered as important as the development phases in Clause 5. To demonstrate this, the diagram below shows that the Clause 6 process is also integrated into Clause 5.

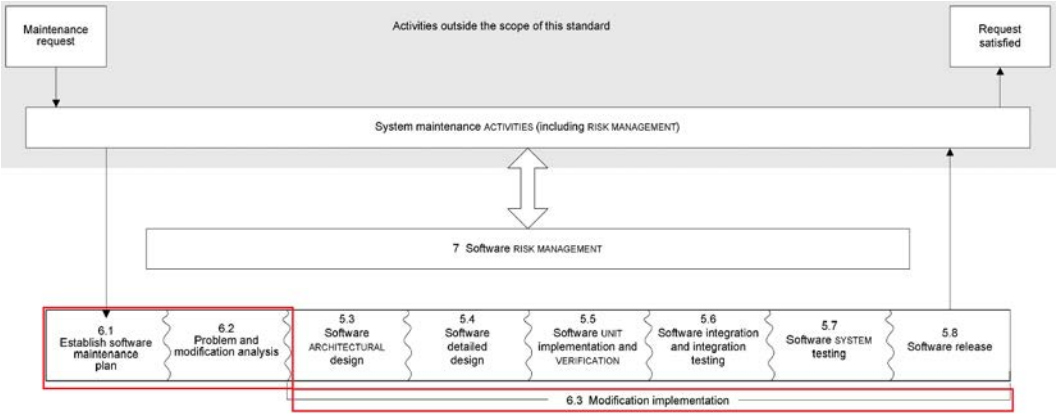


Figure 2-6: Overview IEC 62304 maintenance processes and activities

For example, let's say an issue is identified out in the field or an enhancement is to be added. The bug fix or software change needs to go through each of the development phases. This ensures a safe, secure, and reliable software change and a quality medical device.

CLAUSE 7 – SOFTWARE RISK MANAGEMENT PROCESS

Clause 7 lays out a process for risk management in collaboration with Subclause 4.2 and Clause 2 in the IEC 62304 standard. The process consists of:

- Performing analysis of software contributing to hazardous situations.
- Risk control measures.
- Verification of the risk control measures.
- Managing the risk of the software changes.

As captured in Subclause 4.2 of this document, the ISO 14971 standard is required because it defines the framework for managing risks and identifying hazards. ISO 14971 provides a framework for performing your hazard analysis and risk assessment. This is somewhat complex.

Examples of hazards

Examples of energy hazards	Examples of biological and chemical hazards	Examples of operational hazards	Examples of information hazards
Electromagnetic energy Line voltage Leakage current — enclosure leakage current — earth leakage current — patient leakage current Electric fields Magnetic fields **Radiation energy** Ionizing radiation Non-ionizing radiation	**Biological** Bacteria Viruses Other agents (e.g. prions) Re- or cross-infection **Chemical** Exposure of airway, tissues, environment or property, e.g. to foreign materials: — acids or alkalis — residues — contaminates — additives or processing aids	**Function** Incorrect or inappropriate output of functionality Incorrect measurement Erroneous data transfer Loss or deterioration of function **Use error** Attentional failure Memory failure Rule-base failure Knowledge-based failure Routine violation	**Labelling** Incomplete instruction for use Inadequate description of performance characteristics Inadequate specification of intended use Inadequate disclosure of limitations **Operating instructions** Inadequate specification of accessories to be used with the medical device Inadequate specification of pre-use checks Over-complicated operating

Figure 2-7: Examples of hazards

Example of semi-quantitative probability levels

Common terms	Examples of probability range
Frequent	$\geqslant 10^{-3}$
Probable	$< 10^{-3}$ and $\geqslant 10^{-4}$
Occasional	$< 10^{-4}$ and $\geqslant 10^{-5}$
Remote	$< 10^{-5}$ and $\geqslant 10^{-6}$
Improbable	$< 10^{-6}$

Figure 2-8: Example of semi-quantitative probability levels

Example of five qualitative severity levels

Common terms	Possible description
Catastrophic	Results in patient death
Critical	Results in permanent impairment or life-threatening injury
Serious	Results in injury or impairment requiring professional medical intervention
Minor	Results in temporary injury or impairment not requiring professional medical intervention
Negligible	Inconvenience or temporary discomfort

Figure 2-9: Example of qualitative severity levels

Analysis consists of identifying the risks, evaluating the risk, control options (reduce, eliminate), residual risk acceptability, and more. In plain English, the probability of a hazard and its severity to the patient and/or medical staff is examined. This is a crucial task that leads to categorizing the software severity level (Class A, B & C) of potential hazard scenarios.

Note: This is not equivalent to Class I, II, or III medical device classification, but analogous to safety integrity level (SIL) defined in IEC 61508.

- Class A: No injury or damage to health is possible
- Class B: Non serious injury is possible
- Class C: Death or serious injury is possible

IEC 63204:2015 Amendment 1 provides updates on software safety classification. There's a classification flow chart and there was additional clarification on segregating software item and assigning different safety classifications. The assigned software classification level to a software item ripples through the entire software development process regarding additional steps required to take to ensure quality.

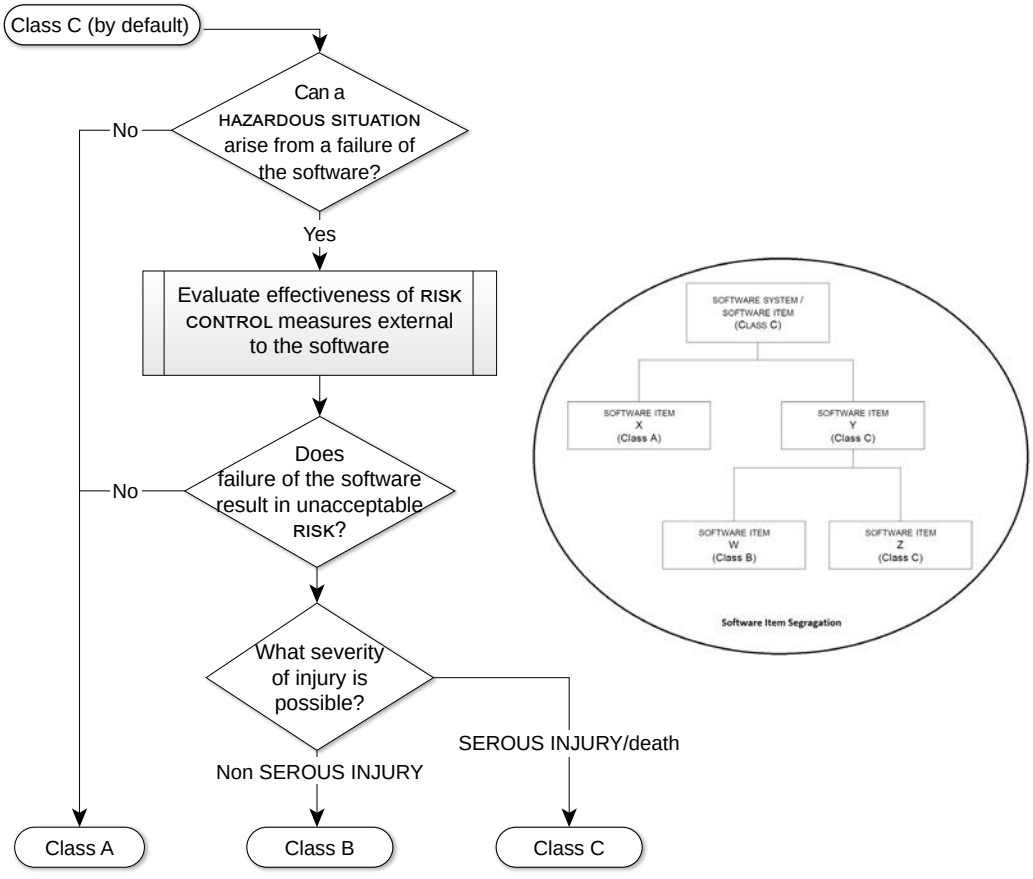


Figure 2-10: IEC 62304 classification flow chart and software item segregation sample

CLAUSE 8 – SOFTWARE CONFIGURATION MANAGEMENT PROCESS

Clause 8 is about integrating a configuration management process that establishes the means to identify configuration items. This means versioning of the software, how and when to baseline, and addressing other attributes that help identify configuration items, like title or name, archived history, and more.

Change management is also part of this.

- What, when, and who changed the item?
- Was it verified?
- Was it approved?

CLAUSE 9 – SOFTWARE PROBLEM RESOLUTION PROCESS

Clause 9 is about having a problem-resolution process in place. Have the relevant parties been advised?

Are all defects or problems captured, investigated, analysed for trends, remediated, verified, and ensure that documentation has been updated?

Requirements for Compliance in Testing

STATIC ANALYSIS

Static code analysis is the analysis of code without actual code execution. Static analysis exposes safety and security vulnerabilities in the code by applying a comprehensive set of code analysis techniques including:

- Pattern-based analysis
- Data flow analysis
- Control flow analysis
- Abstract interpretation
- Code metrics
- And more

These methods identify memory buffer overflows, divide by zero, use of insecure libraries, organization coding rules, and directive violations, and more.

IEC 62304, Clause 5.5 - Software Unit Implementation and Verification, includes additional measures to ensure quality code. For example, subclause 5.5.4 - Additional Software Unit Acceptance Criteria lists the following eight criteria.

1. Proper event sequence
2. Data and control flow
3. Planned resource allocation
4. Fault handling (error definition, isolation, and recovery)
5. Initialization of variables
6. Self-diagnostics
7. Memory management and memory overflows
8. Boundary conditions

Most of these verification activities are supported through the automation of static analysis using modern advanced tools like Parasoft C/C++test. In addition, Parasoft provides code metrics on maintainability, clarity, testability, portability, robustness, reusability, complexity, and support for team code peer reviews. Dynamic analysis, unit testing and other runtime error detection is also provided.

THE ROLE OF STATIC ANALYSIS IN IEC 62304 SOFTWARE VERIFICATION

The role of static analysis is to flush out defects early in the implementation phase where defects are the cheapest to fix, increasing code quality from the get-go. Static analysis solutions are available directly in a developer's IDE.

These solutions provide reports indicating that the analyzed software is in compliance with the selected standard, which can include MISRA C/C++, SEI CERT C/C++, CWE, and many others. Additionally, static analysis can be automated during the build process or within your CI/CD workflow to ensure a quality code baseline.

Parasoft can automatically generate compliance reports required as software verification evidence and all this data can be source controlled for historical and auditing purposes. Equally important is the use of a defect tracking and management system to provide meaningful analytical views and prioritization for the intent of solving the highest risk issues down to the lowest.

Clause	Subclause	Class A	Class B	Class C
5.5 Software unit implementation and verification	5.5.1 Implement each software unit	X	X	X
	5.5.2 Establish software unit verification process		X	X
	5.5.3 Software unit acceptance criteria		X	X
	5.5.4* Additional software unit acceptance criteria			X
	5.5.5 Software unit verification		X	X

Class C devices must conform to the specific guideline of IEC 62304, Section 5.5.4, additional software unit acceptance criteria. The appropriate recommendations that are addressed by static analysis tools are described below.

Data Flow Analysis

A technique for gathering information about the possible set of values calculated at various points in a computer program. Data flow analysis is a critical aspect of advanced static analysis tools that help detect complex errors such as tainted data vulnerabilities.

Control Flow Analysis

A static code analysis technique for determining the control flow of a program. Modern advanced static analysis tools, such Parasoft C/C++test, use sophisticated control and data flow analysis to detect complex defects and security vulnerabilities.

Initialization of Variables

Static analysis tools can detect and warn developers about uninitialized variables.

Memory Management & Memory Overflows

Static analysis can warn developers of various memory management bugs, such as buffer overflows, memory leaks, resource leaks, and variable overflows (integer number overflows).

Boundary Conditions

Static analysis can warn developers about poor type conversions where values might exceed type boundaries.

Walkthroughs & Inspections

Informal methods used to verify design and implementation. Static analysis tools automate much of the tedious aspects of code inspection such as coding standards compliance while flagging errors and possible software weaknesses.

CODING STANDARDS

IEC 62304, Annex B.5.5 requires "...coding standards should be used to specify a preferred coding style. Examples of coding standards include requirements for understandability, language usage rules or restrictions, and complexity management."

Although initially focused on automotive systems, MISRA is becoming a de facto standard in other industries, like medical device software. SEI CERT C/C++ is an important standard for creating more secure code and is gaining traction in medical device software as well. Static analysis tools like Parasoft C/C++test are essential in establishing and enforcing a compliant coding standard such as MISRA.

MISRA C 2023

MISRA C is a set of coding guidelines for the C programming language. The focus of the standard is increasing safety of software by pre-emptively preventing programmers from making coding mistakes that can lead to runtime failures (and possible safety concerns) by avoiding known problem constructs in the C language.

Over the years, many developers of embedded systems were (and still are) complaining that MISRA C was too stringent of a standard and that the cost of writing fully compliant code was difficult to justify. Realistically, given that MISRA C is applied in safety-critical software, the value of applying the standard to a project depends on factors such as:

- Risk of a system malfunction because of a software failure
- Cost of a system failure to the business
- Development tools and target platform
- Level of developer's expertise

Programmers must find a practical middle ground that satisfies the spirit of the standard and still claim MISRA compliance without wasting effort on non value added activities.

MISRA C Compliance

In the document, "MISRA Compliance: 2020," the MISRA Consortium provides the response needed by the community with a well-defined framework of what the statement, "MISRA Compliant," truly means.

The document helps organizations use a common language articulating the compliance requirements by defining the following artifacts.

- **The Guideline Enforcement Plan.** Demonstrates how each MISRA guideline is verified.
- **The Guideline Re-Categorization Plan.** Communicates the agreed upon severity of individual rules in the guidelines as part of the vendor/client relationship.
- **The Deviations Report.** Documents the violations of guidelines with appropriate rationale.
- **The Guidelines Compliance Summary.** This is the primary record of overall project compliance.

When first introducing MISRA C into a project, commonly where code already exists, the key document is The Guideline Re-Categorization Plan. This document captures all directives, rules, and identifies which categories have been recategorized. However, it's important to have the same rational categorization for newly developed code as well. For example, the following diagram shows part of a recategorization plan.

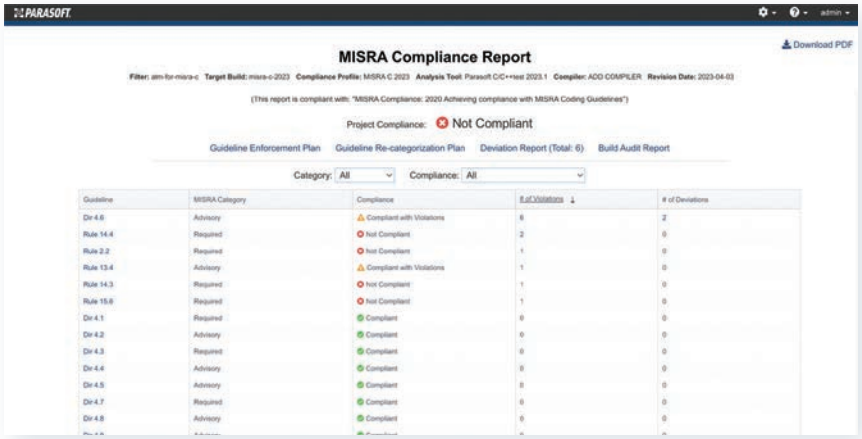
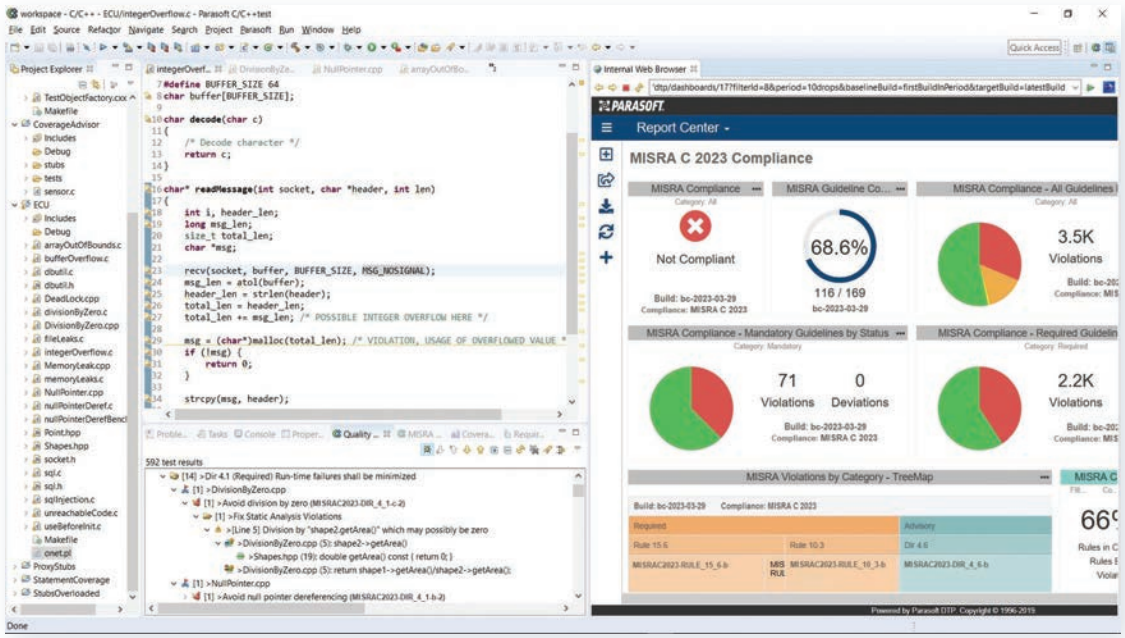


Figure 3-1: Parasoft DTP MISRA Compliance report

The "MISRA C 2023" compliance document recommends against recategorizations from a less stringent to a more stringent classification. In addition, it is possible to disapply advisory rules all together after reviewing the types of violations with the team.

The requirement to document deviations is only necessary for all required rules. Any violations in adopted code should be reviewed. Deviations need to clearly state that violations do not compromise safety and security. Regardless of recategorization, if there is a finding that compromises the safety or security of the system, the issue must be fixed. Also, modifications to the existing code may introduce other issues not clearly seen by the developer.

Figure 3-2: Parasoft C/C++test and DTP dashboard



SEI/CERT

The Software Engineering Institute (SEI) Computer Emergency Response Team (CERT) has a set of guidelines to help developers create safer, more secure, and more reliable software. Started in 2006 at a meeting of the C Standard Committee, the first CERT C standard was published in 2008 and is constantly developing and evolving.

There's a book form version published in 2016, but it doesn't include the latest updates. This standard doesn't have specific frozen releases like CWE Top 25 and OWASP Top 10. The standard arose from a large community of over 3,000 people with a focus on engineering and prevention. So the CERT secure coding standards focus on prevention of the root causes of security vulnerabilities rather than treating or managing the symptoms by searching for vulnerabilities.

The CERT coding guidelines are available for C, C++, Java, Perl, and Android. They fall into two main categories.

1. Rules
2. Recommendations

Rules are guidelines that are detectable by static analysis tools and require strict enforcement, while recommendations are guidelines that have a lower impact and are sometimes difficult to analyze automatically.

CERT includes a risk assessment system that combines likelihood of occurrence, severity, and relative difficulty of mitigation. This helps developers prioritize which guideline violations are the most important to investigate. The inclusion of mitigation effort to the guideline priority is an important addition to the CERT secure coding standards, which many other standards lack.

The cost factor allows for the creation of the CERT bullseye diagram in which the center bullseye is the highest severity guidelines that are more difficult to fix. The benefit of this prioritization is focusing on the most critical violations that provide the biggest bang for the buck in security improvement while helping the development team filter out less important warnings.

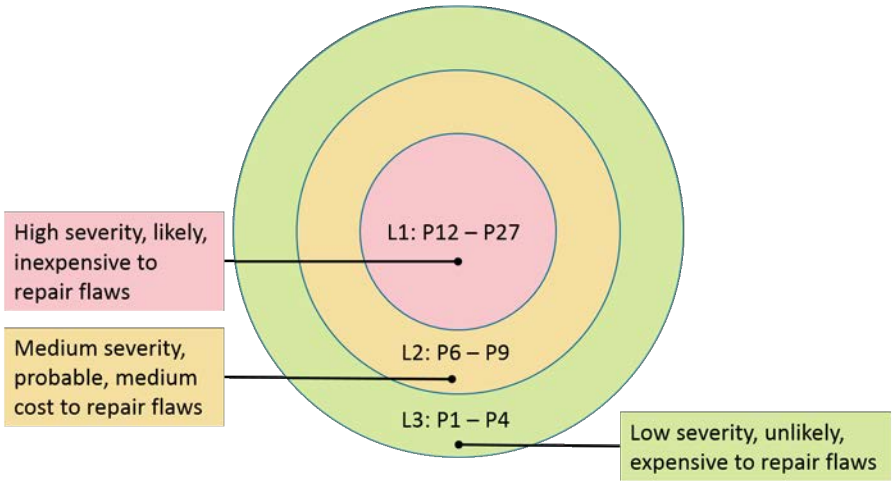


Figure 3-3: Cost factor CERT bullseye

SEI CERT C/C++ Conformance

According to the SEI CERT C documentation, conformance "requires that the code not contain any violations of the rules specified in this standard. If an exceptional condition is claimed, the exception must correspond to a predefined exceptional condition, and the application of this exception must be documented in the source code."

Although conformance is less specific than standards such as MISRA, the principles remain similar. Rules should be followed and deviations rare and well documentation. Recommendations should be used when possible and those that aren't needed to be documented.

Violations that persist in the source code need to be documented. However, no deviation is acceptable for performance or usability and the onus is on the developer to demonstrate that the deviation will not lead to a vulnerability.

Parasoft C/C++test provided comprehensive CERT compliance dashboard and reports. Individual compliance reports are available on demand based on the latest build of the software or any previous build.

These reports can be sorted and navigated to investigate violations in more detail. A conformance test plan is available to correlate the CERT guideline with the corresponding Parasoft static analysis checker, which is an important tool if conformance documentation is needed for audit purposes. In addition, all the interesting reports as specified by the team are in a single PDF available for download for auditors.

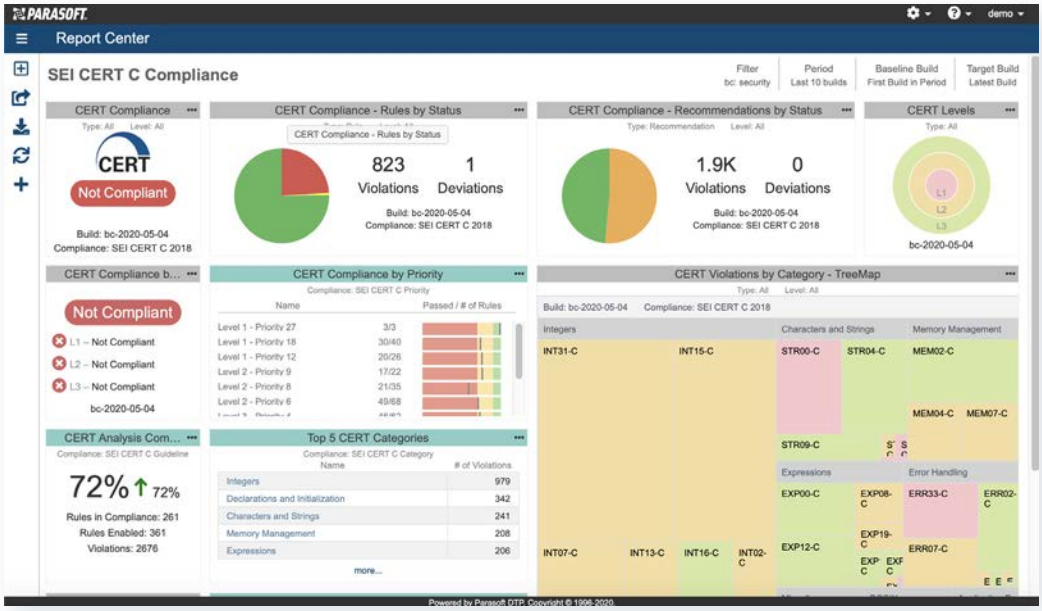


Figure 3-4: SEI CERT C DTP dashboard

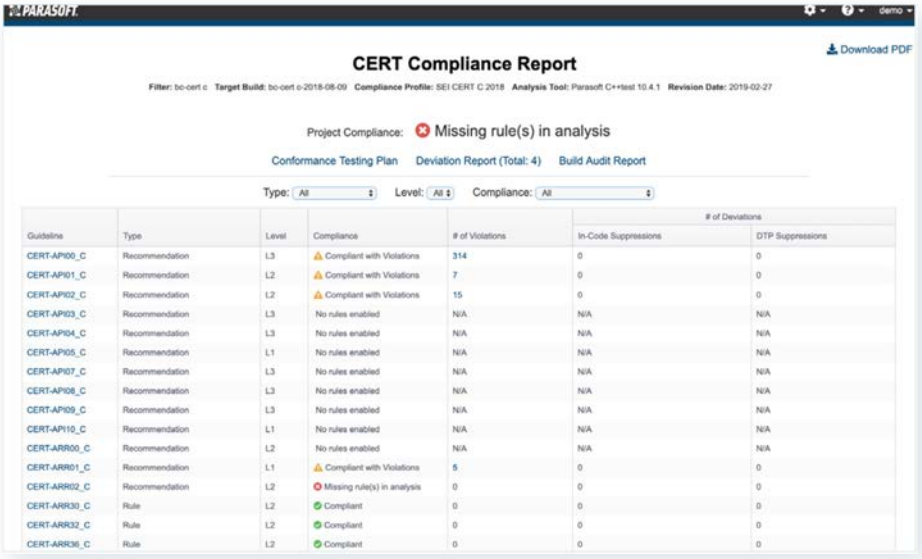


Figure 3-5: CERT C compliance report

Support for CERT C/C++ in Parasoft C/C++test

Parasoft provides comprehensive support for CERT C and CERT C++ secure coding standards with complete coverage of all the CERT C/C++ guidelines including both rules and recommendations that are detectable by static analysis. Checker names, dashboards, and reports use the CERT naming convention to make conformance and auditing easier. A CERT conformance dashboard, which includes the CERT risk score, helps developers focus on the most critical violations.

CWE

CWE (Common Weakness Enumeration) is a list of discovered software weaknesses based on the analysis of reported vulnerabilities (CVEs). The collection of CVEs and CWEs is a U.S. government-funded initiative developed by the software community and managed by the MITRE organization. In its entirety, the CWE list contains over 800 items.

These 800+ items are organized in more usable lists such as the well-known CWE Top 25. The Top 25 lists the most common and dangerous security weaknesses, which are all exploits that have a high chance of occurring and the impact of exploiting the weakness is large. The software weaknesses documented by a CWE are the software implicated in a set of discovered vulnerabilities (documented as CVEs) when analysis was performed to discover the root cause. CVEs are specific observed vulnerabilities in software products that have an exact definition of how to exploit them.

The current version of CWE Top 25 is from 2011. An updated Top 25 is currently in process with improved linking to CVEs and the NVD. Ranking considers realworld information so that it truly represents the Top 25 application security issues today. As soon as it is released, Parasoft will have updated support for the latest version.

Rank	Score	ID	Name
[1]	93.8	CWE-89	Improper Neutralization of Special Elements used in an SQL Command (‘SQL Injection’)
[2]	83.3	CWE-78	Improper Neutralization of Special Elements used in an OS Command (‘OS Command Injection’)
[3]	79.0	CWE-120	Buffer Copy without Checking Size of Input (‘Classic Buffer Overflow’)
[4]	77.7	CWE-79	Improper Neutralization of Input During Web Page Generation (‘Cross-site Scripting’)
[5]	76.9	CWE-306	Missing Authentication for Critical Function
[6]	76.8	CWE-862	Missing Authorization
[7]	75.0	CWE-798	Use of Hard-coded Credentials
[8]	75.0	CWE-311	Missing Encryption of Sensitive Data
[9]	74.0	CWE-414	Unrestricted Upload of File with Dangerous Type
[10]	73.8	CWE-807	Reliance on Untrusted Inputs in a Security Decision
[11]	73.1	CWE-250	Execution with Unnecessary Privileges
[12]	70.1	CWE-352	Cross-Site Request Forgery (CSRF)
[13]	69.3	CWE-22	Improper Limitation of a Pathname to a Restricted Directory (‘Path Traversal’)
[14]	68.5	CWE-494	Download of Code Without Integrity Check
[15]	67.8	CWE-863	Incorrect Authorization
[16]	66.0	CWE-829	Inclusion of Functionality from Untrusted Control Sphere
[17]	65.5	CWE-732	Incorrect Permission Assignment for Critical Resource
[18]	64.6	CWE-676	Use of Potentially Dangerous Function
[19]	64.1	CWE-327	Use of a Broken or Risky Cryptographic Algorithm
[20]	62.4	CWE-131	Incorrect calculation of Buffer Size
[21]	61.5	CWE-307	Improper Restriction of Excessive Authentication Attempts
[22]	61.1	CWE-601	URL Redirection to Untrusted Site (‘Open Redirect’)
[23]	61.0	CWE-134	Uncontrolled Format String
[24]	60.3	CWE-190	Integer Overflow or Wraparound
[25]	59.9	CWE-759	Use of a One-Way Hash without a Salt

Figure 3-6: The CWE Top 25

For software teams that have a good handle on the Top 25, there’s another grouping of the next most common and impactful vulnerabilities called the CWE CUSP. Another way to think of these are the top 25 honorable mentions.

The CWE uses a risk scoring method to rank the Top 25 and on the CUSP. This score takes into consideration the technical impact of a software weakness (how dangerous an exploit of the weakness is in the real world) as measured by the CWSS (common weakness scoring system). Examples of technical impacts from vulnerabilities may include:

- Denial of service (DoS)
- Distributed denial of service (DDoS)
- Read or write access to protected information
- Unauthorized access
- And more

The details of these methods aren't too important, but the sorted list is useful in understanding which vulnerabilities to be concerned about the most. As an example, it's possible that your application is purely internal and DoS issues aren't critical for you. Being able to prioritize on the most important weaknesses for your own application can help overcome overwhelm with static analysis violations.

CWE Top 25 and On the Cusp Compliance

Introducing the coding standard compliance process into the team development workflow isn't an easy task. As such, it's important to select a tool that will help in achieving compliance without imposing too much overhead and without the requirement for additional manual procedures. The following points are important decision-making factors when selecting the solution for static analysis.

The CWE Top 25 and its lesser known sibling, On the Cusp, are not a coding standards per se, but a list of weaknesses to avoid, improving security. To be CWE compliant, a project should be able to prove that it has made reasonable efforts to detect and avoid these common weaknesses.

Parasoft's advanced static analysis tools for C, C++, Java, and .NET are officially compatible with CWE, providing automated detection of both Top 25 and On the Cusp weaknesses and many more. CWE-centric dashboards give users quick access to standards violations and current project status. A built-in CWE Top 25 configuration is available for C, C++, .NET, and Java and has full coverage of all the 25 common weaknesses.

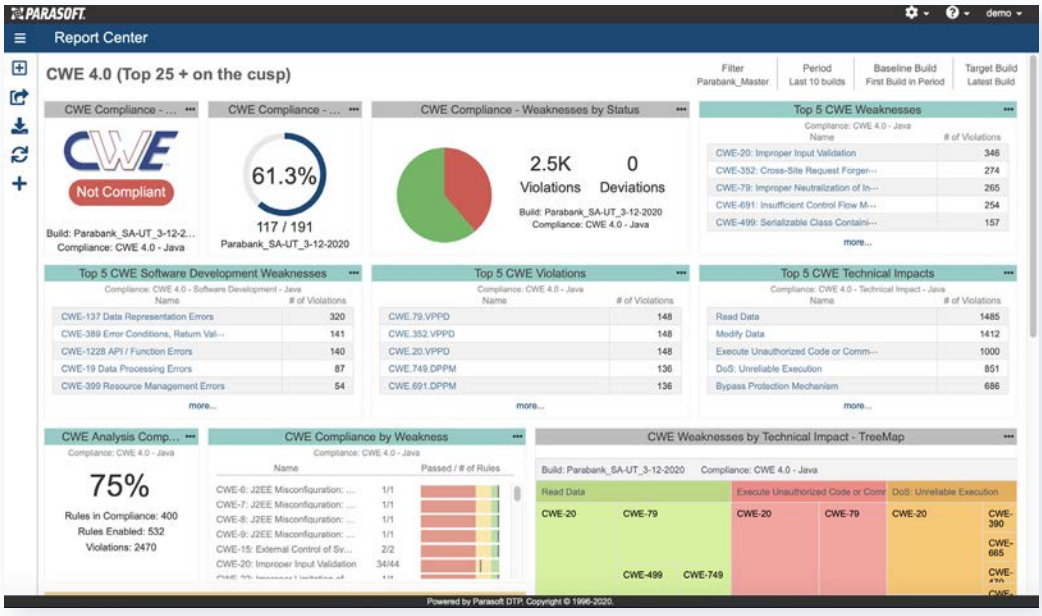


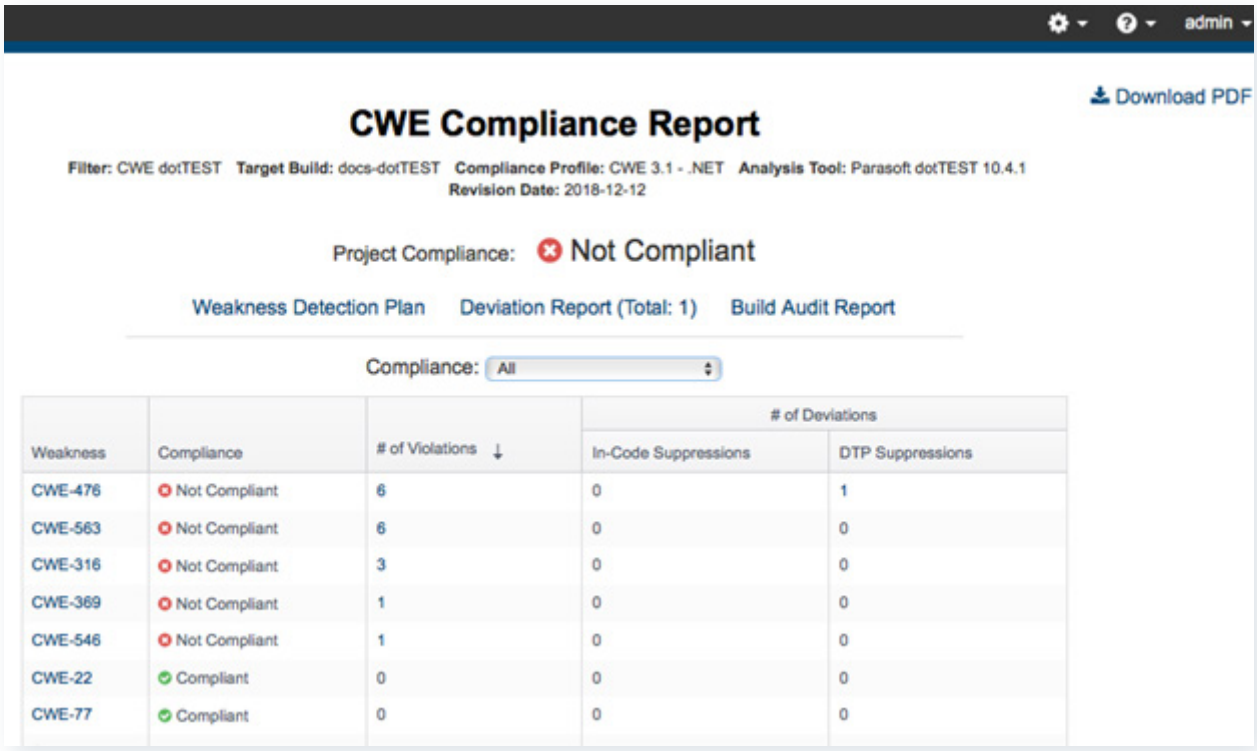
Figure 3-7: CWE Top 25 + on the cusp DTP dashboard

The Parasoft tools include information from the Common Weakness Risk Analysis Framework (CWRAF), such as technical impact, so you can benefit from the same type of prioritization based on risk and technical impact and weaknesses found in your own code.

The On the Cusp guidelines are also available. When enabled, they're treated the same way as the Top 25 and reports provide the same level of detail. This is useful since the UL 2900 (formerly Underwriters Laboratory) and FDA recommend the full list of guidelines (Top 25 + On the Cusp + OWASP Top 10). It's possible to integrate other guidelines from CWE lists or other standards and guidelines using Parasoft's custom checker configurations as needed.

Parasoft also supports detailed compliance reporting to streamline audit processes. The web dashboards provide the link to compliance reports for a complete picture of where a project stands. In addition, the CWE Weakness Detection Plan maps the CWE entry against the checkers that are used to detect the weakness. This helps illustrate how compliance was achieved to an auditor, and the audit reports are available to download as PDFs for easy reporting.

Figure 3-8: CWE 3.2 - .NET compliace report



UNIT TESTING

Software verification is an inherent part of medical device software development. Testing, by way of execution, is a keyway to demonstrating implementation of requirements and delivery of quality software. Unit testing is the verification of low-level requirements. It ensures that each software unit does what it's required to do within its expected quality of service—safety, security, and reliability—requirements.

Safety and security requirements instruct that software units don't behave in unforeseen ways where the medical device is not susceptible to hijack, data manipulation, theft, or corruption.

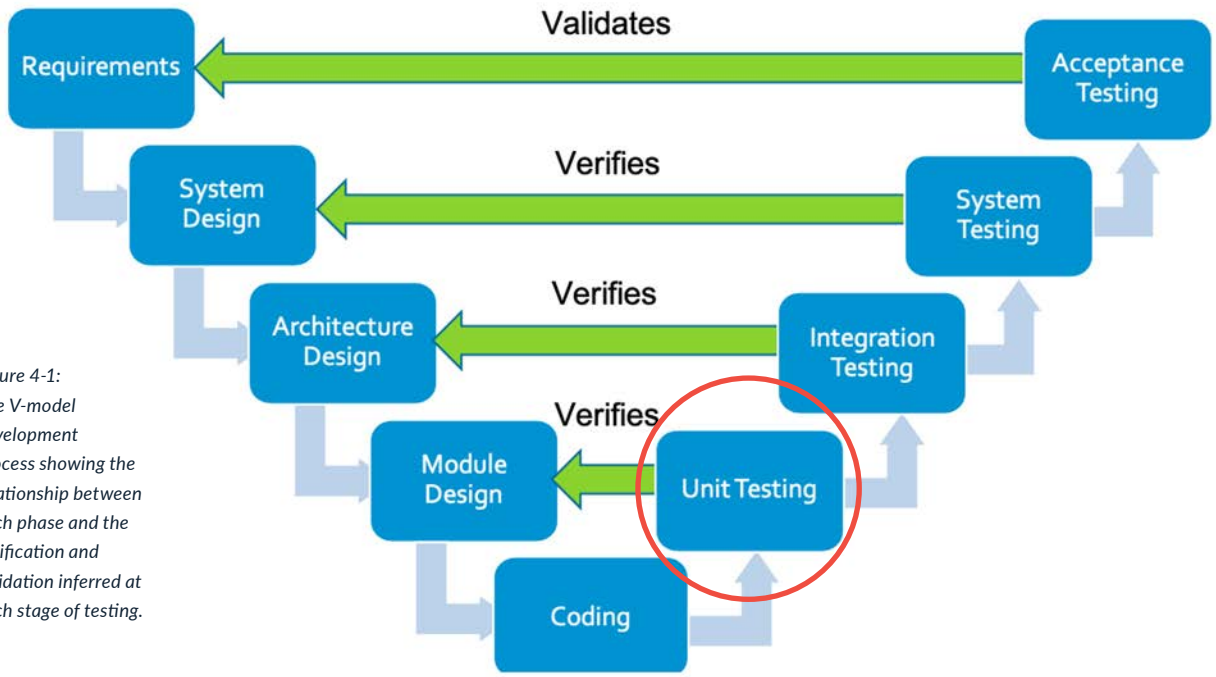


Figure 4-1: The V-model development process showing the relationship between each phase and the verification and validation inferred at each stage of testing.

In terms of the classic V-model process of development, unit test execution is a verification practice to ensure the module is designed correctly. IEC 62304 has fairly broad guidelines for what needs to be tested for unit testing:

"The manufacturer shall establish strategies, methods and procedures for verifying each software unit. Where verification is done by testing, the test procedures shall be evaluated for correctness."

These guidelines apply to Class B and C devices specifically.

Clause	Subclause	Class A	Class B	Class C
5.5 Software unit implementation and verification	5.5.1 Implement each software unit	X	X	X
	5.5.2 Establish software unit verification process		X	X
	5.5.3 Software unit acceptance criteria		X	X
	5.5.4* Additional software unit acceptance criteria			X
	5.5.5 Software unit verification		X	X

In keeping with established industry quality standards, the following unit testing techniques can be satisfied and accelerated with test automation tools like Parasoft C/C++test.

UNIT TEST METHODS

Requirement-Based Test

These tests directly test functionality as specified in each requirement. Test automation tools need to support bidirectional traceability of requirements to their tests and the requirements testing coverage reports to show compliance.

Interface Test

These tests ensure programming interfaces behave and perform as specified. Test tools need to create function stubs and data sources to emulate behavior of external components for automatic unit test execution.

Fault Injection Test

These tests use unexpected inputs and introduce failures in the execution of code to examine failure handling or lack thereof. Test automation tools must support injection of fault conditions using function stubs and automatic unit test generation using a diverse set of preconditions, such as min, max, and heuristic values.

Resource Usage Evaluation

These tests evaluate the amount of memory, file space, CPU execution, or other target hardware resources used by the application.

TEST CASE DRIVERS

Analysis of Requirements

Clearly, every requirement drives, at minimum, a single unit test case. Although test automation tools don't generate tests directly from requirements, they must support two-way traceability from requirements to code and requirements to tests, and maintain requirements, tests, and code coverage information.

Generation & Analysis of Equivalence Classes

Test cases must ensure that units behave in the same manner for a range of inputs, not just cherry-picked inputs for each unit. Test automation tools must support test case generation using data sources to efficiently use a wide range of input values. Parasoft C/C++test uses factory functions to prepare sets of input parameter values for automated unit test generation.

Analysis of Boundary Values

Automatically generated test cases, like heuristic values and boundary values, employ data sources to use a wide range of input values in tests.

Error Guessing

This method uses the function stubs mechanism to inject fault conditions into tested code flow analysis results and can be used to write additional tests.

AUTOMATED TEST EXECUTION AND TEST CASE GENERATION

Test automation provides large benefits to embedded medical device software. Moving away from test suites that require a lot of manual intervention means that testing can be done quicker, easier, and more often.

Offloading this manual testing effort frees up time for better test coverage and other safety and quality objectives. An important requirement for automated test suite execution is being able to run these tests on both host and target environments.

Target-Based Testing for Medical Devices

Automating testing for medical device software is more challenging due to the complexity of initiating and observing tests on embedded targets, not to mention the limited access to target hardware that software teams have.

Software test automation is essential to make embedded testing workable on a continuous basis from host development system to target system. Testing embedded software is particularly time consuming. Automating the regression test suite provides

considerable time and cost savings. In addition, test results and code coverage data collection from the target system are essential for validation and standards compliance.

Traceability between test cases, test results, source code, and requirements must be recorded and maintained. For those reasons, data collection is critical in test execution.

Parasoft C/C++test is offered with its test harness optimized to take minimal additional overhead for the binary footprint and provides it in the form of source code, where it can be customized if platform-specific modifications are required.

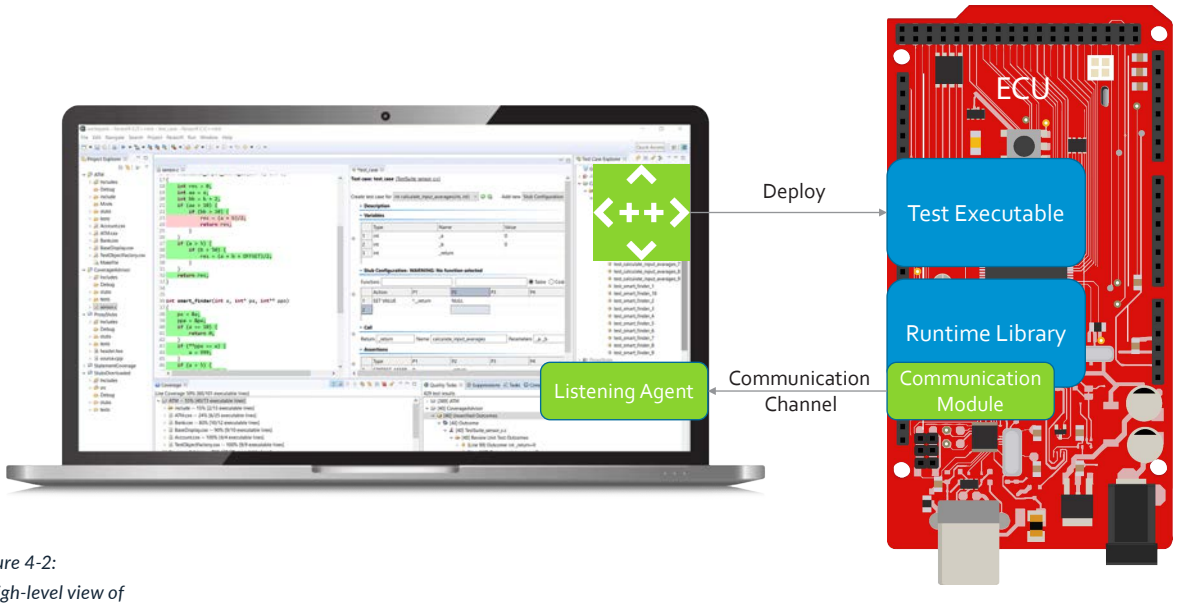


Figure 4-2: A high-level view of deploying, executing, and observing tests from host to target in Parasoft C/C++test.

One huge benefit that the Parasoft C/C++test solution offers is its dedicated integrations with embedded IDEs and debuggers that make the process of executing test cases smooth and automated. Supported IDE environments include:

- Eclipse
- VS Code
- Green Hills Multi
- Wind River Workbench
- IAR EW
- ARM MDK
- ARM DS-5
- TI CCS
- Visual Studio
- Many more

Automated Test Case Generation

Unit test automation tools universally support some sort of test framework, which provides the harness infrastructure to execute units in isolation while satisfying dependencies via stubs. Parasoft C/C++test is no exception. Part of its unit test capability is the automated generation of test harnesses and the executable components needed for host and target-based testing.

Test data generation and management is by far the biggest challenge in unit testing. Test cases are particularly important in safety-critical software development because they must ensure functional requirements and test for unpredictable behavior, security, and safety requirements. All while satisfying test coverage criteria.

Parasoft C/C++test automatically generates test cases like the popular CppUnit format. By default, C/C++test generates one test suite per source/header file. It can also be configured to generate one test suite per function or one test suite per source file.

Safe stub definitions are automatically generated to replace "dangerous" functions, which include system I/O routines such as rmdir(), remove(), rename(), and so on. In addition, stubs can be automatically generated for missing function and variable definitions. User defined stubs can be added as needed.

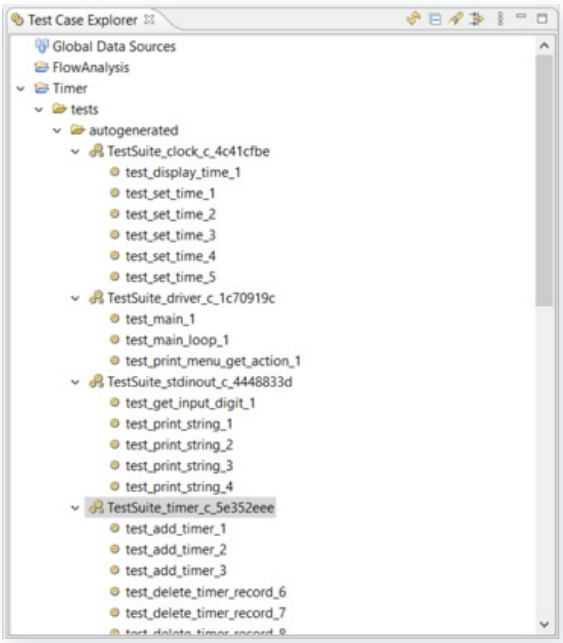


Figure 4-3: Parasoft C/C++ automated test case generation, in this case, one test suite per function.

REGRESSION TESTING

As part of most software development processes, regression testing is done after changes are made to software. These tests determine if the new changes had an impact on the existing operation of the software.

IEC 62304 requires that regression testing be used “to demonstrate that defects have not been introduced into previously integrated software” and that “when a change is made to a software system (even a small change), the degree of regression testing (not just the testing of the individual change) should be determined to ensure that no unintended side effects have been introduced.”

Regression tests are necessary, but they only indicate that recent code changes have not caused tests to fail. There's no assurance that these changes will work. In addition, the nature of the changes that motivate the need to do regression testing can go beyond the current application and include changes in hardware, operating system, and operating environment.

REGRESSION TESTING IN MEDICAL DEVICE SOFTWARE

In safety-critical software development, validation is critical in proving correct functionality, safety, and security. Tests are needed to confirm any changes to the application to ensure functionality and to verify there are no unforeseen impacts on the rest of the system.

If a test case passed in the past and now fails, a potential regression has been identified. The failure could be caused by new functionality, in which the test case may need to be updated so that it takes into consideration changes in input and output values.

Regression testing of embedded systems also includes the execution of the following types of test cases:

- Integration
- System
- Performance
- Stress and more

In fact, all previously created test cases may need to be executed to ensure that no regressions exist and that a new dependable software version release is constructed. This is critical because each new software system or subsystem release is built or developed upon. If you don’t have a solid foundation the whole thing can collapse.

Parasoft C/C++test supports the creation of regression testing baselines as an organized collection of tests and will automatically verify all outcomes. These tests are run automatically on a regular basis to verify whether code modifications change or break the functionality captured in the regression tests. If any changes are introduced, these test cases will fail to alert the team to the problem. During subsequent tests, C/C++test will report tasks if it detects changes to the behavior captured in the initial test.

HOW TO DECIDE WHAT TO REGRESSION TEST

The key challenge with regression testing is determining what parts of an application to test. It is common to default to executing all regression tests when there's doubt on what impacts recent code changes have had—the all or nothing approach.

For large software projects, this becomes a huge undertaking and drags down the productivity of the team. This inability to focus testing hinders much of the benefits of iterative and continuous processes, potentially exacerbated in embedded software where test targets are a limited resource.

A couple of tasks are required here.

1. Identify which tests need to be re-executed.
2. Focus the testing efforts (unit testing, automated functional testing, and manual testing) on validating the features and related code that are impacted by the most recent changes.

Developers and testers can get a clear understanding of the changes in the codebase between builds using the Process Intelligence Engine (PIE) within Parasoft DTP (Development Testing Platform) combined with Parasoft's proprietary coverage analysis engines:

- C/C++test for C and C++
- dotTEST for C#
- Jtest for Java

With this combination, teams can improve efficiency and achieve the promise of Agile.

This form of smart test execution is called test impact analysis. It's sometimes referred to as change-based testing.

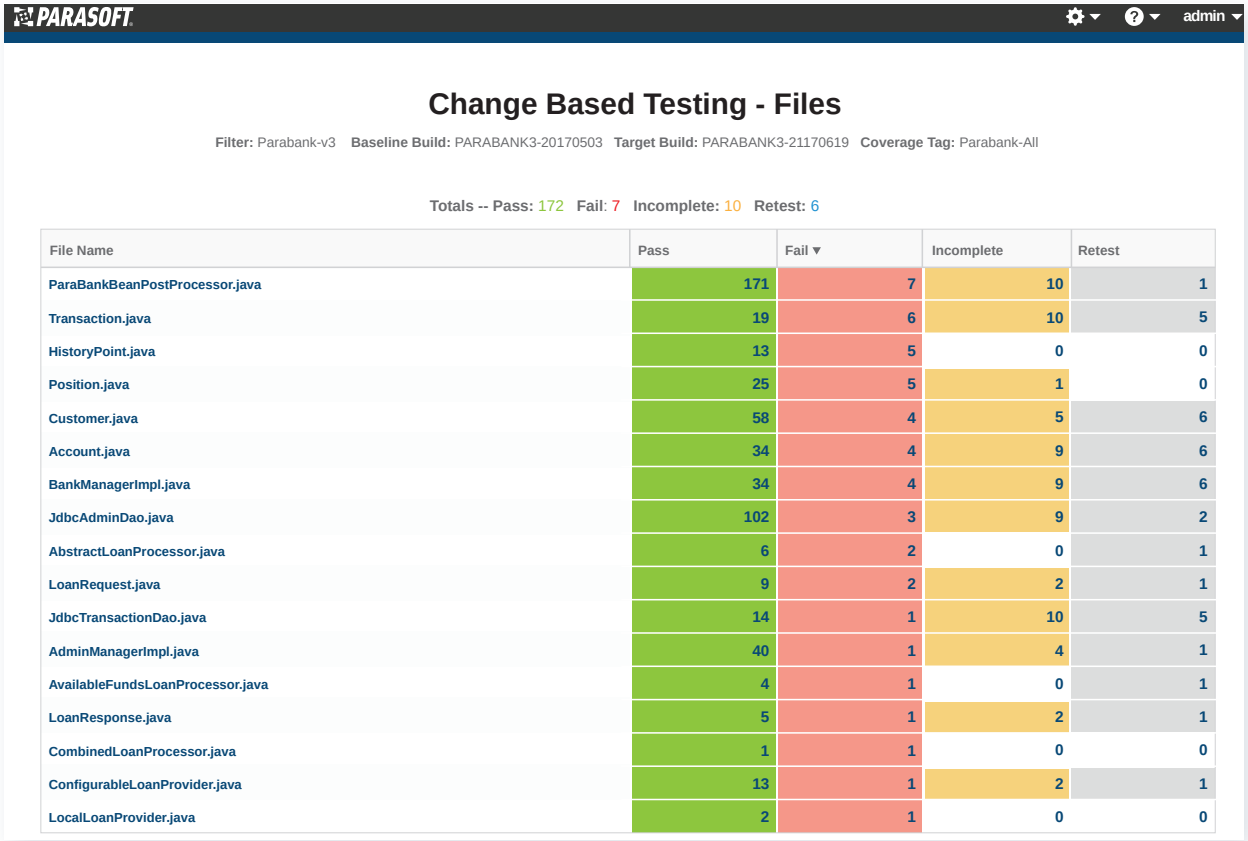
UNDERSTAND THE IMPACT OF CODE CHANGES ON TESTING WITH TEST IMPACT ANALYSIS

Test impact analysis uses data collected during test runs and changes in code between builds to determine which files have changed and which specific tests touched those files. Parasoft's analysis engine can:

- Analyze the delta between two builds.
- Identify the subset of regression tests that need to be executed.
- Understand the dependencies on the units modified to determine what ripple effect the changes have made on other units.

Parasoft Jtest and dotTEST provide insight into the impact of software changes. Each solution recommends where to add tests and where further regression testing is needed.

Figure 5-1: An example change-based testing report from Parasoft DTP showing areas of the code that are and are not tested.



File Name	Pass	Fail ▾	Incomplete	Retest
ParaBankBeanPostProcessor.java	171	7	10	1
Transaction.java	19	6	10	5
HistoryPoint.java	13	5	0	0
Position.java	25	5	1	0
Customer.java	58	4	5	6
Account.java	34	4	9	6
BankManagerImpl.java	34	4	9	6
JdbcAdminDao.java	102	3	9	2
AbstractLoanProcessor.java	6	2	0	1
LoanRequest.java	9	2	2	1
JdbcTransactionDao.java	14	1	10	5
AdminManagerImpl.java	40	1	4	1
AvailableFundsLoanProcessor.java	4	1	0	1
LoanResponse.java	5	1	2	1
CombinedLoanProcessor.java	1	1	0	0
ConfigurableLoanProvider.java	13	1	2	1
LocalLoanProvider.java	2	1	0	0

SOFTWARE INTEGRATION TESTING

Integration testing follows unit testing with the goal of validating the architectural design. Testing software integrations can be done bottom up and top down with a combination of approaches likely in many software organizations.

IEC 62304, Section 5.6 requires, "The manufacturer shall test the integrated software ITEMS in accordance with the integration plan and document the results. [Class B, C]." In addition, there are supporting requirements for test verification and documenting results in regression testing.

Clause	Subclause	Class A	Class B	Class C
5.6 Software integration and integration testing	5.6.1 Integrate software units		X	X
	5.6.2 Verify software integration		X	X
	5.6.3 Test integrated software		X	X
	5.6.4 Integration testing content		X	X
	5.6.5 Verify integration test procedures		X	X
	5.6.6 Conduct regression tests		X	X
	5.6.7 Integration test record contents		X	X
	5.6.8 Use software problem resolution process		X	X

BOTTOM-UP INTEGRATION

This testing begins with unit testing, followed by tests of progressively higher-level combinations of units called modules or builds. The approach follows a version of the testing pyramid where unit testing forms the foundation of a thorough testing regime. Integration tests follow the integration of units into larger architectural blocks.

TOP-DOWN INTEGRATION

In this testing, the highest level modules are tested first. Progressively, testing of lower-level modules follows. This approach assumes significant subsystems are complete enough to be tested as a whole.

The V-model is good for illustrating the relationship between the stages of development and stages of validation. At each testing stage, more complete portions of the software are validated against the phase that defines it.

The V-model might imply a waterfall development method. However, there are ways to incorporate Agile, DevOps, and CI/CD into this type of product development while still being standards compliant.

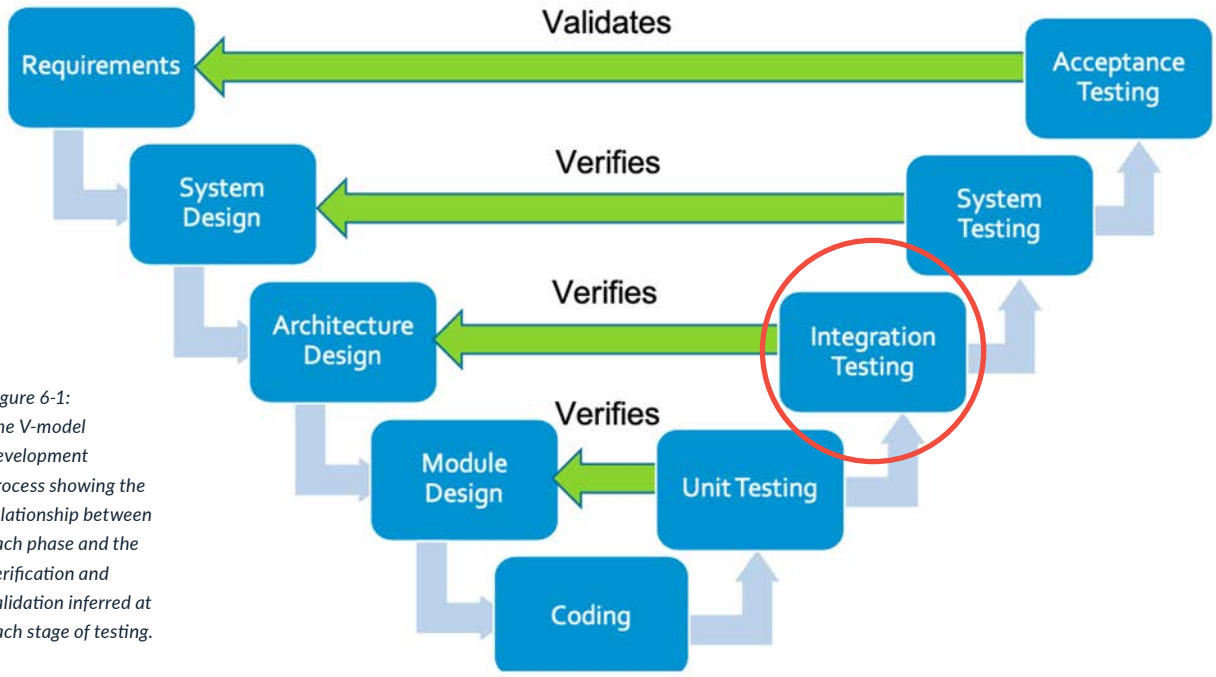


Figure 6-1: The V-model development process showing the relationship between each phase and the verification and validation inferred at each stage of testing.

While the act of performing tests is considered software validation, it's supported by a parallel verification process that involves the following activities to make sure teams are building the process and the product correctly.

- Reviews
- Walkthroughs
- Analysis
- Traceability
- Test
- Code coverage and more

The key role of verification is to ensure building delivered artifacts from the previous stage to specification in compliance with company and industry guidelines.

INTEGRATION & SYSTEM TESTING AS PART OF A CONTINUOUS TESTING PROCESS

Performing some level of test automation is foundational for continuous testing. Many organizations start by simply automating manual integration and system testing (top down) or unit testing (bottom up).

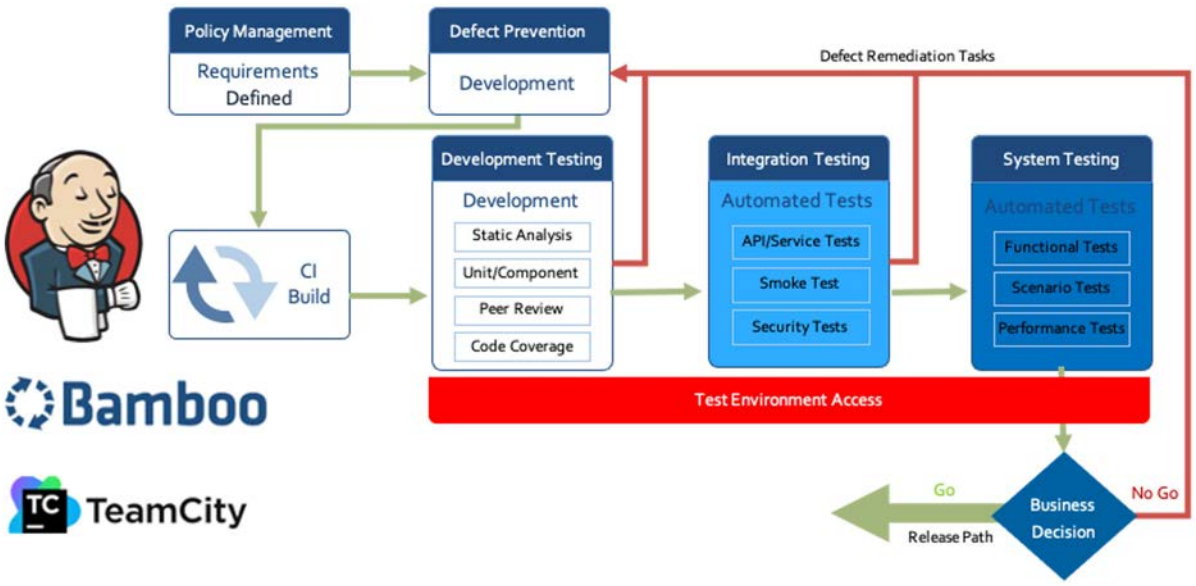
To enable continuous testing, organizations need to focus on creating a scalable test automation practice that builds on a foundation of unit tests that are isolated and faster to execute. Once unit testing is fully automated, the next step is integration testing and eventually system testing.

Continuous testing leverages automation and data derived from testing to provide a realtime, objective assessment of the risks associated with a system under development. Applied uniformly, it allows both business and technical managers to make better tradeoff decisions between release scope, time, and quality.

Continuous testing isn't just more automation. It's a larger reassessment of software quality practices that's driven by an organization's cost of quality and balanced for speed and agility. Even within the V-model used in safety-critical software development, continuous testing is still a viable approach, particularly during phases of testing, for example, during unit testing and integration testing.

Figure 6-2: A continuous testing cycle

The diagram below illustrates how different phases of testing are part of a continuous process that relies on a feedback loop of test results and analysis.



ANALYSIS & REPORTING IN SUPPORT OF INTEGRATION & SYSTEM TESTING

Parasoft test automation tools support the validation (actual testing activities) in terms of test automation and continuous testing. These tools also support the verification of these activities, which means supporting the process and standards requirements. Key aspects of safety-critical medical device software development are requirements traceability and code coverage.

IEC 62304 requires that the software development plan include "traceability between system requirements, software requirements, software system test, and risk control measures implemented in software."

Requirements analysis requires "All software requirements should be identified in such a way as to make it possible to demonstrate traceability between the requirement and software system testing."

Two-Way Traceability

Requirements in safety-critical software are the key driver for product design and development. These requirements include functional safety, application requirements, and nonfunctional requirements that fully define the product. This reliance on documented requirements is a mixed blessing because poor requirements are one of the critical causes of safety incidents in software. In other words, the implementation wasn't at fault, but poor or missing requirements were.

Automating Bidirectional Traceability

Maintaining traceability records on any sort of scale requires automation. Application life cycle management tools include requirements management capabilities that are mature and tend to be the hub for traceability.

Integrated software testing tools like Parasoft complete the verification and validation of requirements by providing an automated bidirectional traceability to the executable test case. This includes the pass or fail result and traces down to the source code that implements the requirement.

Parasoft integrates with market leading requirements management tools or ALM systems including:

- PTC/Intland codebeamer
- Polarion from Siemens
- Atlassian Jira
- Jama Connect
- And more

As shown in the image below, each of Parasoft's test automation solutions (C/C++test, Jtest, dotTEST, SOAtest, and Selenic) used within the development life cycle support the association of tests with work items defined in these systems, such as requirements, defects, and test case/test runs. Traceability is managed through Parasoft DTP, the central reporting and analytics dashboard.

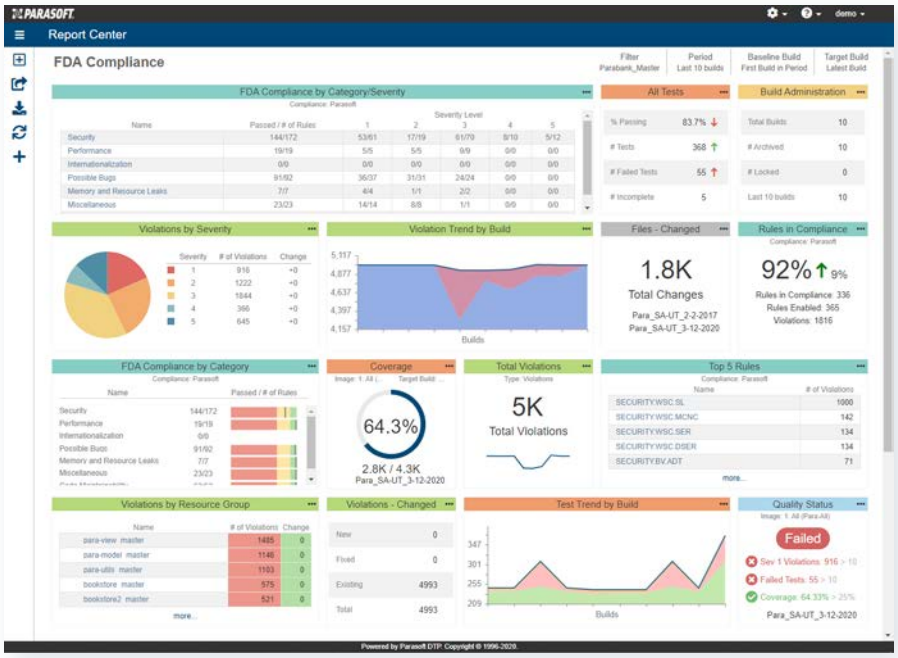


Figure 6-3:
An example of a reporting dashboard that captures the project's testing status and progress towards completion.

Parasoft DTP correlates the unique identifiers from the management system with:

- Static analysis findings
- Code coverage
- Results from unit, integration, and functional tests

Results are displayed within Parasoft DTP's traceability reports and sent back to the requirements management system. They provide full bidirectional traceability and reporting as part of the system's traceability matrix.

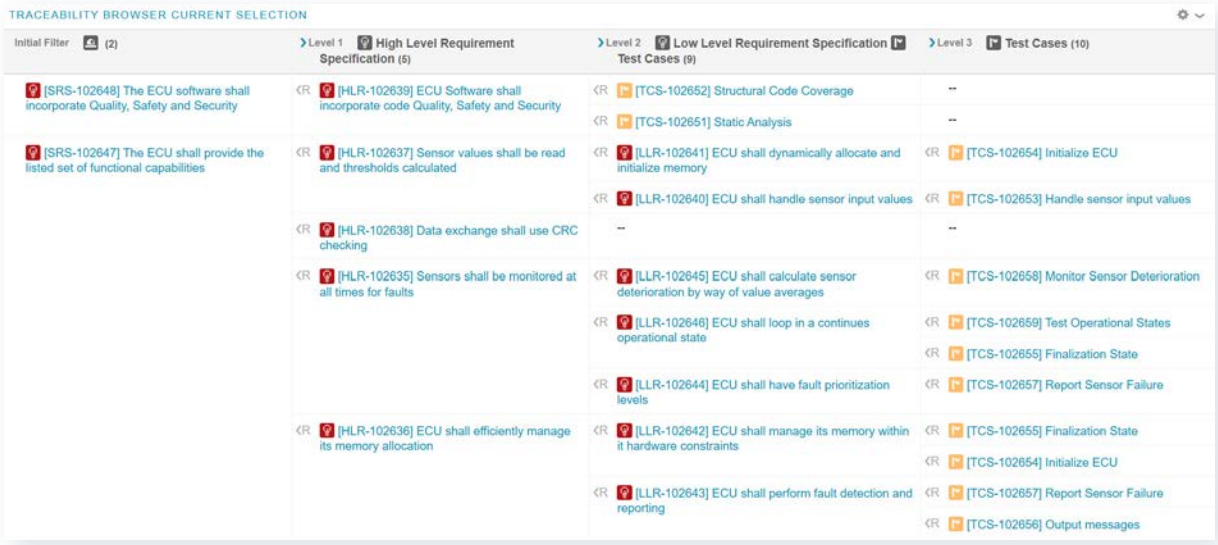


Figure 6-4: codebeamer traceability matrix, which lists system requirements from high level to low level and test cases and test results.

The traceability reporting in Parasoft DTP is highly customizable. The following image shows a requirements traceability matrix template for requirements authored in Polarion and traces to the test cases, static analysis findings, the source code files, and the manual code reviews.

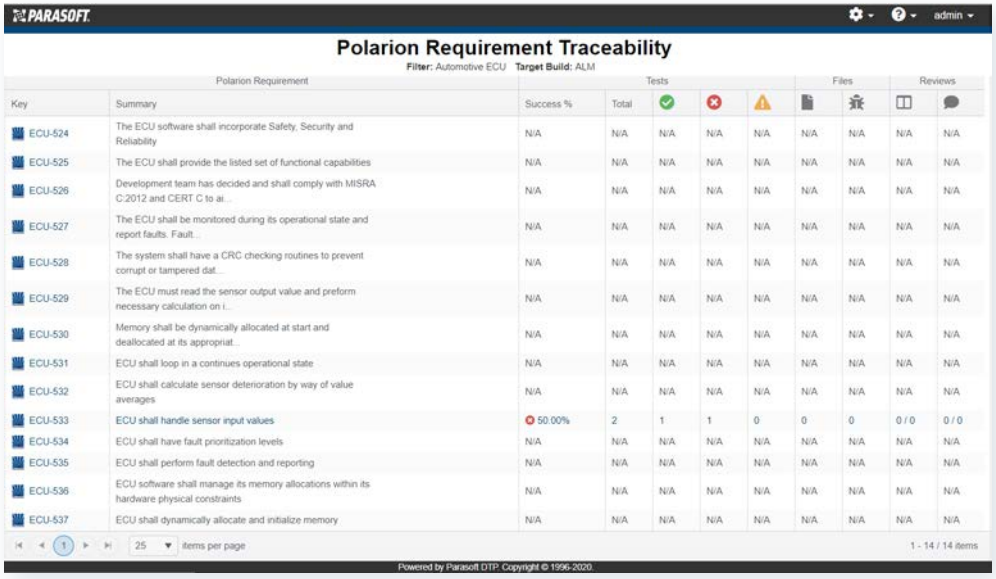


Figure 6-5: Requirements traceability matrix template from Parasoft DTP integrated with Siemens Polarion.

The bidirectional correlation between test results and work items provides the basis of requirements traceability. Parasoft DTP adds test and code coverage analysis to evaluate test completeness. Maintaining this bidirectional correlation between requirements, tests, and the artifacts that implement them is an essential component of traceability.

Code Coverage

Code coverage expresses the degree to which the application's source code is exercised by all testing practices, including unit, integration, and system testing—both automated and manual.

Collecting coverage data throughout the life cycle enables more accurate quality and coverage metrics, while exposing untested or under tested parts of the application. Depending on the medical device class, the depth and completeness of the code coverage will vary.

IEC 62304 specifies, "Software system testing focuses on functional (black box) testing, although it might be desirable to use white box (see previous section) methods to accomplish certain tests more efficiently, initiate stress conditions or faults, or increase code coverage of the qualification tests. The organization of testing by types and test stage is flexible, but coverage of requirements, risk control, usability, and test types (e.g., fault, installation, stress) should be demonstrated and documented."

Application coverage can also help organizations focus testing efforts when time constraints limit their ability to run the full suite of manual regression tests. Capturing coverage data on the running system on its target hardware during integration and system testing completes code coverage from unit testing.

Benefits of Aggregate Code Coverage

Captured coverage data is leveraged as part of the continuous integration (CI) process, as well as part of the tester's workflow. Parasoft DTP performs advanced analytics on code coverage from all tests, source code changes, static analysis results, and test results. The results help identify untested and undertested code and other high risk areas in the software.

Analyzing code, executing tests, tracking coverage, and reporting the data in a dashboard or chart is a useful first step toward assessing risk, but teams must still dedicate significant time and resources to reading the tea leaves and hope that they've interpreted the data correctly.

Understanding the potential risks in the application requires advanced analytics processes that merge and correlate the data. This provides greater visibility into the

true code coverage and helps identify testing gaps and overlapping tests. For example, what is the true coverage for the application under test when your tools report different coverage values for unit tests, automated functional tests, and manual tests?

The percentages cannot simply be added together because the tests overlap. This is a critical step for understanding the level of risk associated with the application under development.

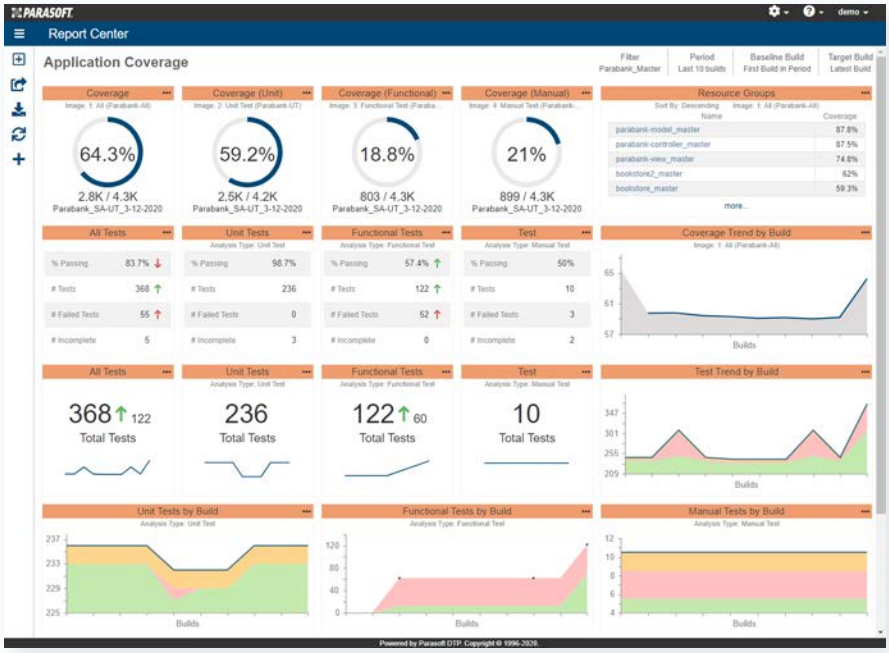


Figure 6-6: Aggregated code coverage from various testing methods

Understanding the Impact of Code Changes on Testing With Test Impact Analysis

Test Impact Analysis uses data collected during test runs and changes in code between builds to determine which files have changed and which specific tests touched those files. Parasoft's analysis engine can analyze the delta between two builds and identify the subset of regression tests that need to be executed. It also understands the dependencies on the units modified to determine the ripple effect the changes have made on other units.

Parasoft C/C++test, Jtest, and dotTEST provide insight into the impact of software changes and recommend where to add tests and where further regression testing is needed.

ACCELERATING INTEGRATION & SYSTEM TESTING WITH TEST AUTOMATION TOOLS

Parasoft's software test automation tools accelerate verification by automating the many tedious aspects of record keeping, documentation, reporting, analysis, and reporting.

- **Two-way traceability for all artifacts** ensures requirements have code and tests to prove they are being fulfilled. Metrics, test results, and static analysis results are traced to components and vice versa.
- **Code and test coverage** verifies all requirements are implemented and makes sure the implementation is tested as required.
- **Target and host-based test execution** supports different validation techniques as required.
- **Smart test execution** manages change with a focus on tests for only code that changed and any impacted dependents.
- **Reporting and analytics** provides insight to make important decisions and keeps track of progress. Decision making needs to be based on data collected from the automated processes.
- **Automated documentation generation** from analytics and test results support process and standards compliance.
- **Standards compliance automation** reduces the overhead and complexity by automating the most repetitive and tedious processes. The tools can keep track of the project history and relating results against requirements, software components, tests, and recorded deviations.

SOFTWARE SYSTEM TESTING

System testing tests the system as a whole. Once all the components are integrated, the entire system is tested rigorously to verify that it meets the specified functional, safety, security, and other nonfunctional requirements.

Clause	Subclause	Class A	Class B	Class C
5.7 Software system testing	5.7.1 Establish tests for software requirements	X	X	X
	5.7.2 Use software problem resolution process	X	X	X
	5.7.3 Retest after changes	X	X	X
	5.7.4 Verify software system testing	X	X	X
	5.7.5 Software system test record contents	X	X	X

IEC 62304 requires, "the manufacturer shall verify that:

A. all software requirements have been tested or otherwise verified;

B. the traceability between software requirements and tests or other verification is recorded; and

C. test results meet the required pass/fail criteria."

This type of testing in safety-critical software is performed by a specialized testing team. System testing falls within the scope of black box testing. As such, it shouldn't require any knowledge of the inner design of the code or logic.

An important distinction with system level testing is that the system is tested in an environment that is close to the production environment where the application will be deployed. At this stage, specific safety functions are verified, and system wide security testing is run.

MEDICAL DEVICE TESTING AT THE SERVICE LEVEL

Individual medical devices often have connectivity into larger systems that, as an example, collect and analyze patient data. Health care providers can track this data and provide medical advice based on it. This ecosystem for medical devices is an important aspect of system testing since verification of data integrity, security, and confidentiality are required. System testing needs to include these environments to full verify the new device.

Instead of viewing system quality in terms of meeting individual device requirements, the scope is broadened to consider the quality of the services provided. Testing at the service level ensures nonfunctional requirements are met. For example, performance and reliability are difficult to assess at the device level or during software unit testing. Service based testing can simulate the operational environment of a device to provide realistic loads.

Security is a significant concern in medical devices. Cyberattacks most likely originate from the network itself by attacking the exposed APIs. Service based testing can create simulated environments for robust security testing, either through fuzzing (random and erroneous data inputs) or denial-of-service attacks.

VIRTUAL TEST ENVIRONMENT & SERVICE LEVEL TESTING

A real test lab requires the closest physical manifestation of the environment in which a medical device is planned to work. Even in the most sophisticated lab, it's difficult to scale to a realistic environment. A virtual lab fixes this problem.

Virtual labs evolve past the need for hard-to-find (or non-existent) hardware dependencies. They use sophisticated service virtualization with other key test automation tools.

Service Virtualization

Service virtualization simulates all of the dependencies needed by the device under test in order to perform full system testing. This includes all connections and protocols used by the device with realistic responses to communication. For example, service

virtualization can simulate an enterprise server backend with which a medical device communicates. Similarly, virtualization can control and simulate a dependent system, like patient information, in a realistic manner.

Service & API Testing

This testing drives the system under test in a manner that ensures the services and APIs it provides perform flawlessly. These tests can be manipulated via the automation platform to run performance and security tests as needed.

Runtime Monitoring

This detects errors in realtime on the system under test and captures important trace information.

Test Lab Management and Analytics

Once virtualized, an entire lab setup can be replicated as needed, providing overarching control of the virtual labs. Test runs can be automated and repeated. Analytics provide the necessary summary of activities and outcomes.

PARASOFT SOATEST & VIRTUALIZE FOR SERVICE LEVEL TESTING OF MEDICAL DEVICE SOFTWARE

Developers can build integrations earlier, stabilize dependencies, and gain full control of their test data with Parasoft Virtualize. Teams can move forward quickly without waiting for access to dependent services that are either incomplete or unavailable. Companies can enable partners to test against their applications with a dedicated sandbox environment.

Parasoft SOAtest delivers fully integrated API and web service testing tools that automate end-to-end functional API testing. Teams can streamline automated testing with advanced functional test creation capabilities for applications with multiple interfaces and protocols.

SOAtest and Virtualize are well suited for network-based, system-level testing of various types, including the following:

- Comprehensive protocol stack that supports, HTTP, MQTT, RabbitMQ, JMS, XML, JSON, REST, SOAP, and more.
- Security and performance testing during integration and system testing with integration into the existing CI/CD process.
- End-to-end testing that combines API, web, mobile, and database interactions into virtual test environments.

STRUCTURAL CODE COVERAGE

Collecting and analyzing code coverage metrics is an important aspect of safety-critical medical device software development. Code coverage measures the completion of test cases and executed tests. It provides evidence that verification is complete, at least as specified by the software design.

It also identifies dead code. This is code that can logically never be reached. It demonstrates the absence of unintended behavior. Code that isn't covered by any test is a liability because its behavior and functionality are unknown.

IEC 62304 specifies, "Software system testing focuses on functional (black box) testing, although it might be desirable to use white box (see previous section) methods to more efficiently accomplish certain tests, initiate stress conditions or faults, or increase code coverage of the qualification tests. The organization of testing by types and test stage is flexible, but coverage of requirements, risk control, usability, and test types (e.g., fault, installation, stress) should be demonstrated and documented."

How this translates to types and amounts of coverage is somewhat open to interpretation. However, in medical device software development, the onus is on the manufacturer to plan for code coverage, adhere to the plan, and complete it.

TYPES OF CODE COVERAGE

Following are the different types of code coverage.

- **Statement coverage** requires that each program statement be executed at least once. Branch and MC/DC coverage encompass statement coverage.
- **Branch coverage** ensures that each decision branch (if-then-else constructs) is executed.
- **Modified condition/decision coverage (MC/DC)** requires the most complete code coverage to ensure test cases execute each decision branch and all the possible combinations of inputs that affect the outcome of decision logic. For complex logic, the number of test cases can explode, so the modified condition restrictions are used to limit test cases to those that result in standalone logical expressions changing.

Advanced unit test automation tools such as Parasoft C/C++test provide all these code coverage metrics and more. C/C++test automates this data collection on host and target testing and accumulates test coverage history over time. This code coverage history can span unit, integration, and system testing to ensure coverage is complete and traceable at all levels of testing.

COVERAGE FROM SYSTEM TESTING

Obtaining code coverage through system testing is an excellent method to determine if enough testing has been performed. The approach is to run all your system tests, and then examine what parts of the code have not been exercised.

The unexecuted code implies that there may be need for new test cases to exercise the untouched code where a defect may be lurking and helps answer the question: Have I done enough testing?

When system testing is performed, the average resulting metric is 60% coverage. Much of the 40% unexecuted code is due to defensive code in your application. Defensive is code that will only execute upon the system triggering a fault or entering a problematic state that may be difficult to produce. Conditions like memory leakage or other types of faults caused by hardware failure, may take weeks, months, or years to encounter.

There's also defensive code mandated by your coding guidelines where system test cases can never get you to execute. For these reasons, system testing cannot take you to 100% structural code coverage. You'll need to employ other testing methods like manual and/or unit testing to get you to 100%.

COVERAGE FROM UNIT TESTING

As mentioned, unit testing can be used as a complementary approach to system testing to obtain 100% coverage. Obtaining code coverage through unit testing is one of the more popular methods used, but it doesn't expose whether you have done enough testing of the system because the focus is at the unit level (function/procedure).

The goal here is to create a set of unit test cases that exercise the entire unit at the required coverage need (statement, branch, and MC/DC) in order to reach 100% coverage for that single unit. This is repeated for every unit until the entire code base is covered. However, to get the most out of unit testing, do not solely focus on obtaining code coverage. That can generally be accomplished through sunny day scenario test cases.

Truly exercise the unit through sunny and rainy day scenarios, ensuring robustness, safety, security, and low-level requirements traceability. Let code coverage be a biproduct of your test cases and fill in coverage where needed.

To help expedite code coverage through unit testing, configurable and automated test case generation capabilities exist in Parasoft C/C++test. Test cases can be automatically generated to test for use of null pointers, min-mid-max ranges, boundary values, and much more. This automation can get you far. In minutes, you'll obtain a substantial amount of code coverage.

However, as in system testing, obtaining 100% code coverage is elusive due to the use of defensive code or formal language semantics. At the granular level of a unit, defensive code may come in the form of a *default* statement in a *switch*. If every possible case in a *switch* is captured, this leaves the *default* statement unreachable. In the example below, the *return 0;* will never get executed because the *while (1) is infinite*.

```
boolean mainLoop()
{
   int sensorValue;
   int status = 1;
   while (1)
   {
      status = readSensor(&sensorValue);
      if (status == STATUS_STOPPED) {
         handleSensorValue(sensorValue);
      }
      else if (status == STATUS_FAILED) {
         reportSensorFailure();
         handleSensorValue (sensorValue);
      }
   }
   return 0;
}
```

Figure 8-1: Unreachable return 0; Statement

Question: How does one obtain 100% coverage for these special cases?

Answer: Deploying manual methods.

You can follow these steps.

1. Label or notate the statement as covered by using a debugger.
2. Modify the call stack and execute the return 0; statement.
3. Visually witness the execution and, at minimum, document the file name, line of code, and code statement that is now considered covered.

This coverage performed through manual/visual inspection and reports can be used to supplement the coverage captured through unit testing. The addition of both coverage reports can be used to prove 100% structural code coverage.

The goal of obtaining code coverage is an added means to help ensure code safety, security, and reliability.

CODE INSTRUMENTATION

Code coverage is more often than not, identified through having the code instrumented. Instrumented refers to having the user code adorned with additional code to ascertain during execution if that statement, branch, or MC/CD has been executed.

Based on the target or medical device, the coverage data can be stored in the file system, written to memory, or sent out through various communication channels, such as the serial port, TCP/IP port, USB, and even JTAG.

Partial Instrumentation

Be aware that code instrumentation causes code bloat and the increase in code size may impact the ability to load the code onto memory-constrained target hardware for testing.

The workaround is to instrument part of the code.

1. Run your tests and capture the coverage.
2. Instrument the other part of the code.
3. Run your tests again.
4. Capture the coverage.
5. Merge the coverage from the previous test execution.

COVERAGE ADVISOR

Parasoft C/C++test resolves coverage gaps in test suites. Parasoft discovered how to use advanced static code analysis (data and control flow analysis) to find values for the input parameters required to execute specific lines of uncovered code.

In complex code, there are always those elusive code statements for which it is exceedingly difficult to obtain coverage. It's likely there are multiple input values with various permutations and possible paths that make it mind twisting and time consuming to decipher. But only one combination can get you the coverage you need. Parasoft makes it easy to obtain coverage of those difficult to reach lines of code.

When you select the line of code you want to cover, the Coverage Advisor will tell you what input values, global variables, and external calls you need to stimulate the code and obtain coverage.

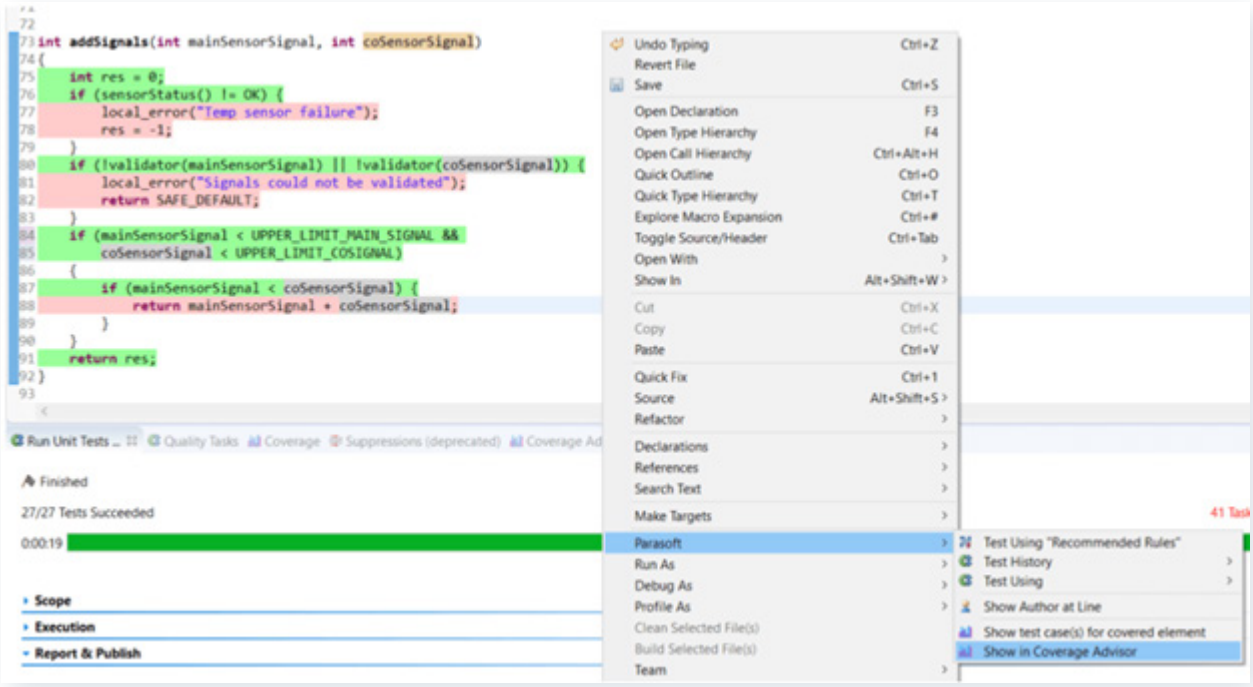


Figure 8-2: Invoking Coverage Advisor by right-clicking on the line of code.

The figure below shows an analysis report providing the user with a solution. The Pre-conditions field expresses:

- The range and input values for `mainSensorSignal` and `coSensorSignal`
- The expected outputs from the external calls

Upon creating the unit test case with these set parameter values and stubs for external calls, you get coverage of the line selected, plus the additional lines expressed in the Expected Coverage field.

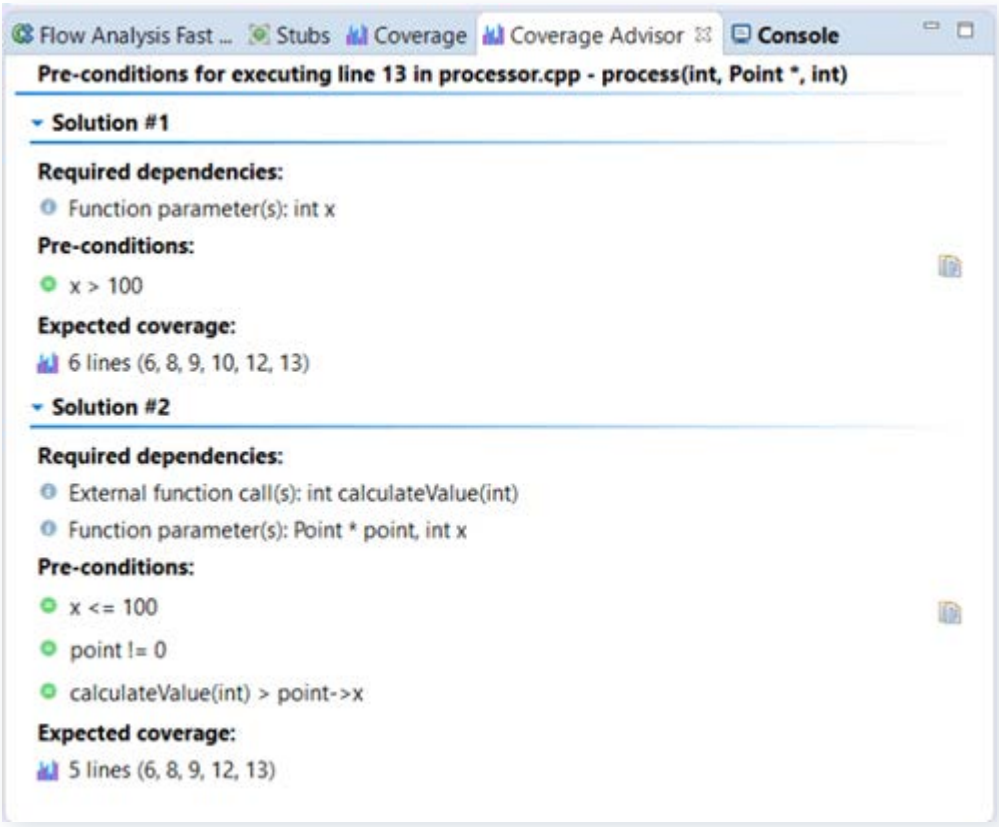


Figure 8-3: Two test case solutions provided by Coverage Advisor.

REQUIREMENTS & THE TRACEABILITY MATRIX

In medical device software, requirements management is a mandatory part of the software development process and the traceability of those requirements to implementation. Subsequently, proof of correct implementation needs to be ensured.

Requirements traceability is defined as "the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through periods of on-going refinement and iteration in any of these phases)."[1]

In the simplest sense, requirements traceability is needed to keep track of exactly what you're building when writing software. This means making sure the software does what it's supposed to and that you're only building what is needed.

Traceability works both to prove you satisfied the requirements and to identify what doesn't. If there are architectural elements or source code that can't be traced to a requirement, then it's a risk and shouldn't be there. The benefits go beyond providing proof of the implementation. Disciplined traceability is an important visibility into development progress.

IEC 62304 requires that the software development plan include "traceability between system requirements, software requirements, software system test, and risk control measures implemented in software."

Requirements analysis requires, "All software requirements should be identified in such a way as to make it possible to demonstrate traceability between the requirement and software system testing."

It's important to realize that many requirements in safety-critical software are derived from safety analysis and risk management. The system must perform it's intended functions, of course, but it must also mitigate risks to greatly reduce the possibility of injury. Moreover, in order to document and prove that these safety functions are implemented and tested fully and correctly, traceability is critical.

Tracing requirements isn't simply linking a paragraph from a document to a section of code or a test. Traceability must be maintained throughout the phases of development as requirements manifest into design, architecture, and implementation. Consider the typical V-model of software.

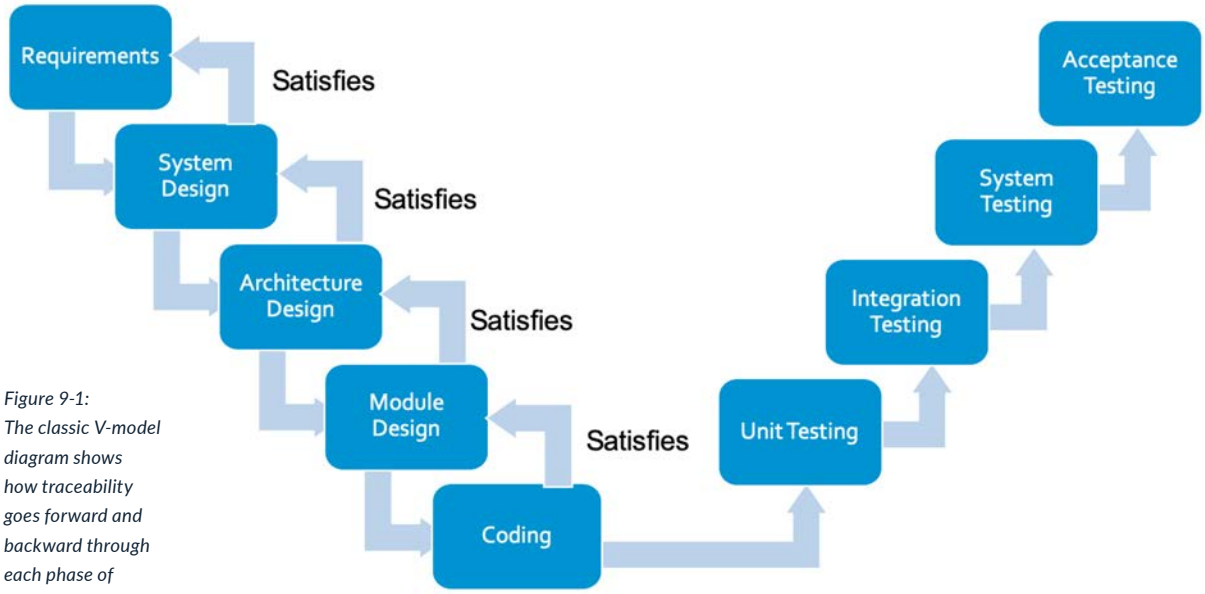


Figure 9-1: The classic V-model diagram shows how traceability goes forward and backward through each phase of development.

Each phase drives the subsequent phase. In turn, the work items in these phases must satisfy the requirements from the previous phase. System design is driven from requirements. System design satisfies the requirements, and so on.

Requirements traceability management (RTM) proves that each phase is satisfying the requirements of each subsequent phase. However, this is only half of the picture. None of this traceability demonstrates that requirements are being met. That requires testing.

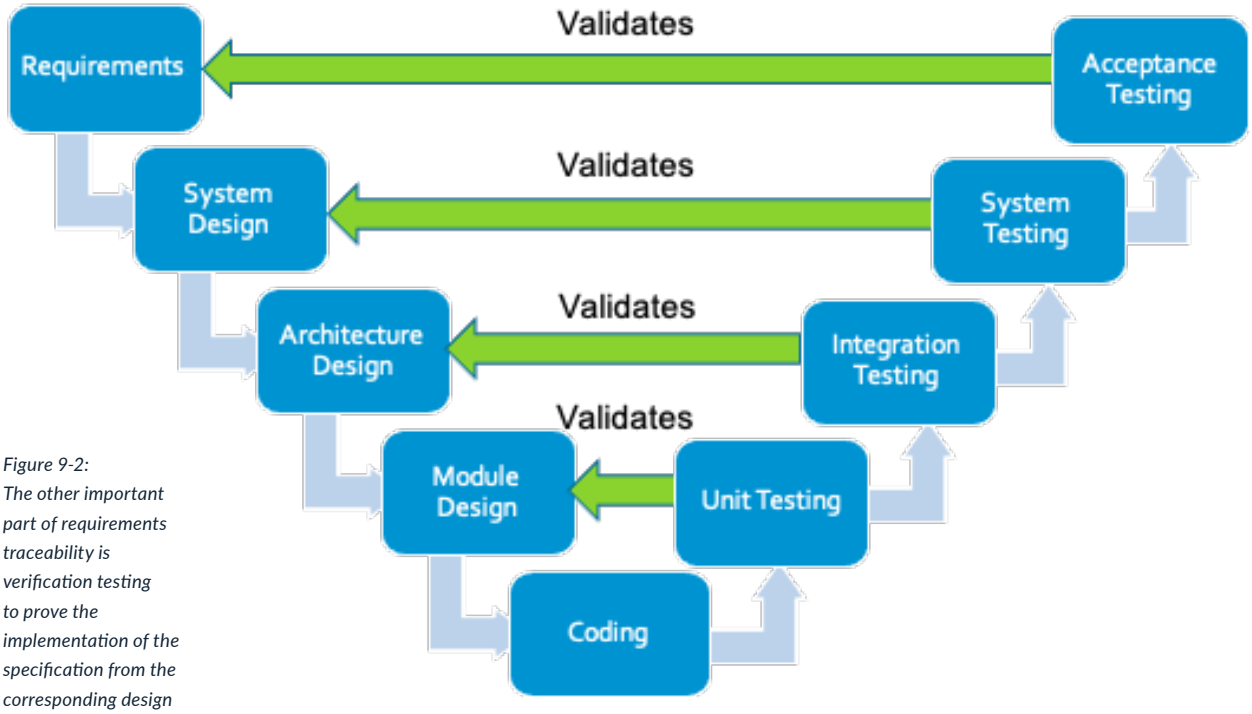


Figure 9-2: The other important part of requirements traceability is verification testing to prove the implementation of the specification from the corresponding design phase.

In the V-model, each testing phase verifies the satisfaction of the specifications associated with the corresponding design/implementation phase. In the example, you see:

- Acceptance testing validates requirements.
- Integration testing verifies architecture design.
- Unit testing verifies module design, and so on.

Validation typically occurs at the end of the development life cycle during acceptance testing with the customer.

Requirements traceability needs both the link to implementation and verification, plus all the associated artifacts from the development process. Software development on any realistic scale will have many requirements, complex design and architecture, and possibly thousands of units and unit tests. Automation of RTM in testing is necessary, especially for safety-critical software that requires documentation of traceability for certifications and audits.

REQUIREMENTS TRACEABILITY MATRIX

A requirement traceability matrix is a document that illustrates the satisfaction of requirements with a corresponding work item, like a unit test, module source code, architecture design element, and so on.

The matrix is often displayed as a table, which shows how each requirement is “checked off” by a corresponding part of the product. Creation and maintenance of these matrices are often automated with requirements management tools with the ability to display them visually in many forms and even hard copy, if required.

Below is a requirements traceability matrix example from PTC/Intland codebeamer. It shows system level requirements decomposed to high-level and low-level requirements, and the test cases that verify each.

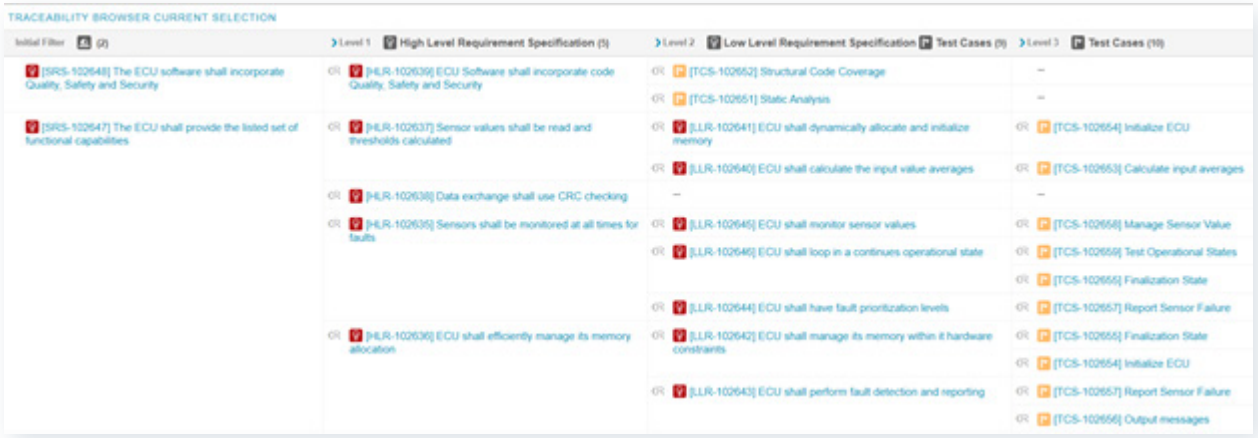


Figure 9-3: Requirements traceability matrix example in PTC/ Intland codebeamer

AUTOMATING BIDIRECTIONAL TRACEABILITY

Maintaining traceability records on any sort of scale requires automation. Application life cycle management tools include requirements management capabilities that are mature and tend to be the hub for traceability. Integrated software testing tools like Parasoft complete the verification and validation of requirements by providing an automated bidirectional traceability to the executable test case, which includes the pass or fail result and traces down to the source code that implements the requirement.

Parasoft integrates with market-leading requirements management and Agile planning systems including:

- PTC/Intland codebeamer
- Polarion from Siemens
- Jama Connect
- Atlassian Jira
- CollabNet VersionOne
- TeamForge

As shown in the image below, each of Parasoft's test automation tools, C/C++test, Jtest, dotTEST, SOAtest, and Selenic, support the association of tests with work items defined in these systems, such as:

- Requirements
- Stories
- Defects
- Test case definitions

Traceability is managed through the central reporting and analytics dashboard, Parasoft DTP.

Figure 9-4: Parasoft provides bidirectional traceability from work items to test cases and test results, displaying traceability reports with Parasoft DTP and reporting results back to the requirements management system.

Parasoft DTP correlates the unique identifiers from the management system with the following:

- Static analysis findings
- Code coverage
- Test results from unit, integration, and functional tests.

Results are displayed within Parasoft DTP's traceability reports and sent back to the requirements management system. They provide full bidirectional traceability and reporting as part of the system's traceability matrix.

The traceability reporting in Parasoft DTP is highly customizable. The following image shows a requirements traceability matrix template with requirements authored in Polarion that trace to the following:

- Test cases
- Static analysis findings
- Source code files
- Manual code reviews

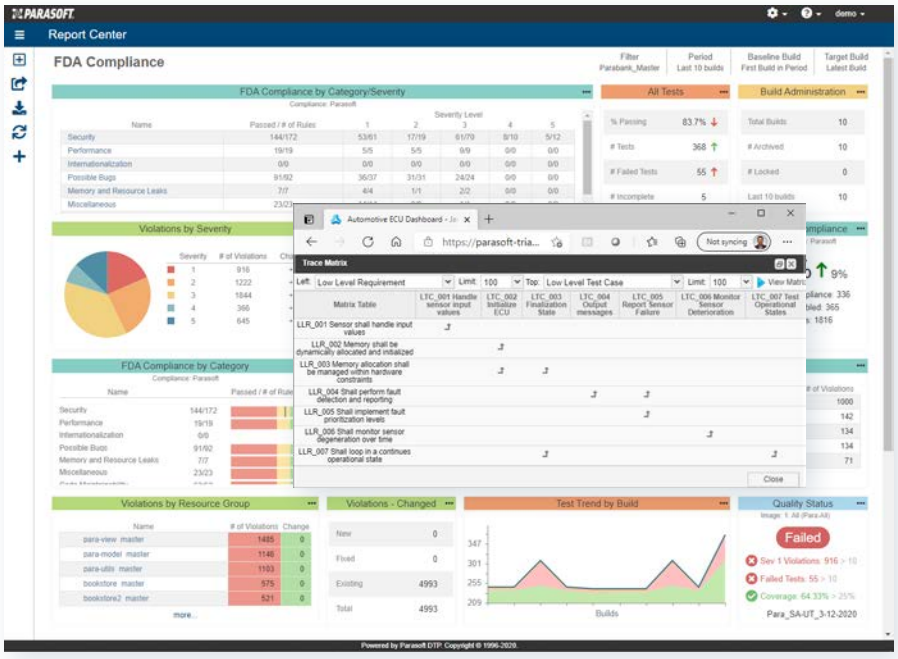


Figure 9-5: Jama requirements matrix and integration with Parasoft DTP

The bidirectional correlation between test results and work items provides the basis of requirements traceability. Parasoft DTP adds test and code coverage analysis to evaluate test completeness. Maintaining this bidirectional correlation between requirements, tests, and the artifacts that implement them is an essential component of traceability.

Bidirectional traceability is important so that requirement management tools and other life cycle tools can correlate results and align them with requirements and associated work items.

The complexity of modern software projects requires automation to scale requirements traceability. Parasoft tools are built to integrate with best-of-breed requirement management tools to aid traceability into test automation results and complete the software test verification and validation of requirements.

[1]*Gotel O.C.Z and Finklestein A.C.W., "An analysis of the requirements traceability problem", in Proceedings of ICRE94, 1st International Conference on Requirements Engineering, Colorado Springs, Co, IEEE CS Press, 1994*

A Unified, Fully Integrated Testing Solution for C/C++ Software Development

TOOL QUALIFICATION FOR SAFETY-CRITICAL MEDICAL DEVICES

Safety-critical software development standards, like IEC 62304, recommend that manufacturers prove that the tools they're using to develop software don't introduce issues and do provide correct, predictable results.

The process of providing such evidence is known as tool qualification. While it's a necessary process, tool qualification is often a tedious and time-consuming activity for which many organizations fail to plan. To make this painless, select tools are certified and have a history of being used in the development of safety-critical medical devices.

IEC 62304, Clause 5.1.4 Software Development Standards, Methods and Tools Planning, requires documentation related to the identification of specific tools and methods to be used for development in general.

To further elaborate and, in support of this risk control, is IEC 60601-1, Clause 14.6.2.: "Suitably validated tools and procedures shall be selected and identified to implement each risk control measure. These tools and procedures shall be appropriate to assure that each risk control measure satisfactorily reduces the identified risk(s)."

The end deliverable is proof in the form of documentation, but there's more to the qualification process than just delivering a big pile of static documentation. Parasoft's Qualification Kits for C/C++test includes a convenient tool wizard that brings automation into the picture and reduces the time and effort required for tool qualification.

PRE-CERTIFIED TOOLS

Tool qualification needs to start with tool selection, ensuring you are using a development tool that is certified by an organization, such as TÜV SÜD. This will significantly reduce the effort when it comes to tool qualification.

Parasoft C/C++test and Parasoft DTP are certified by TÜV SÜD for functional safety according to IEC 62304 and other industry standards for both host based and embedded target applications. The fully integrated testing solution for C/C++ software development paves the way for a streamlined qualification of static analysis, unit testing, and coverage requirements for the safety-critical standards.

ZERTIFIKAT ◆ CERTIFICATE ◆ СЕРТИФИКАТ ◆ CERTIFICADO ◆ CERTIFICAT

CERTIFICATE
No. Z10 075084 0005 Rev. 03

Holder of Certificate: Parasoft Polska Sp. z o. o.
Kelkowskiego 9
30-704 Krakow
POLAND

Certification Mark:

Product: Software Tool for Safety Related Development

Model(s): C/C++test

Parameters: Tool classification: T2 (acc. to IEC 61508)

The verification tool fulfills the requirements for support tools according to IEC 61508-3. The tool is qualified to be used in safety-related software development according to IEC 61508, EN 50128 and ISO 26262. It is suitably validated for use in safety-related development according to IEC 62304. The test report is a mandatory part of this certificate.

Tested according to: IEC 61508-3:2010
IEC 62304:2015
ISO 26262-8:2018
EN 50128:2011/A2:2020

The product was tested on a voluntary basis and complies with the essential requirements. The certification mark shown above can be affixed on the product. It is not permitted to alter the certification mark in any way. In addition the certification holder must not transfer the certificate to third parties. This certificate is valid until the listed date, unless it is cancelled earlier. All applicable requirements of the testing and certification regulations of TÜV SÜD Group have to be complied. For details see: www.tuvsud.com/ps-cert

Test report no.: PK83996C

Valid until: 2026-07-05

Date, 2021-07-06

(Christian Dirmeier)

Page 1 of 1
TÜV SÜD Product Service GmbH · Certification Body · Ridlerstraße 65 · 80339 Munich · Germany

TÜV®

Figure 10-1: Parasoft C/C++test TÜV SÜD certificate

TOOL QUALIFICATION REQUIRES MORE TESTING

Traditionally, tool qualification has meant significant amounts of manual labor, testing, and documenting to satisfy a certification audit. But this documentation-heavy process requires manual interpretation and completion. As a result, it's time consuming and adds to an organization's already heavy testing schedule and budget.

Parasoft leverages its own software test automation tool qualification with Qualification Kits, which include a documented workflow to dramatically reduce the amount of effort required.

Benefits of Using the Qualification Kits

- Automatically reduce the scope of qualification to only the parts of the tool in use.
- Automate tests required for qualification as much as possible.
- Manage any manual tests as eloquently as possible and integrate results alongside automated tests.
- Automatically generate audit-ready documentation that reports on exactly what's being qualified—not more, not less!

QUALIFY ONLY THE TOOLS USED

There should be no need to do any extra work for qualifying capabilities not used during development. Reducing the scope of testing, reporting, and documentation is a key way to reduce the qualification workload.

For example, as part of the IEC 62304 tool qualification kit and process, users can select Parasoft C/C++test for static analysis of C/C++ code to check its compliance to the MISRA C:2012 standard. The tool then selects only the parts of the qualification suite needed for this function.

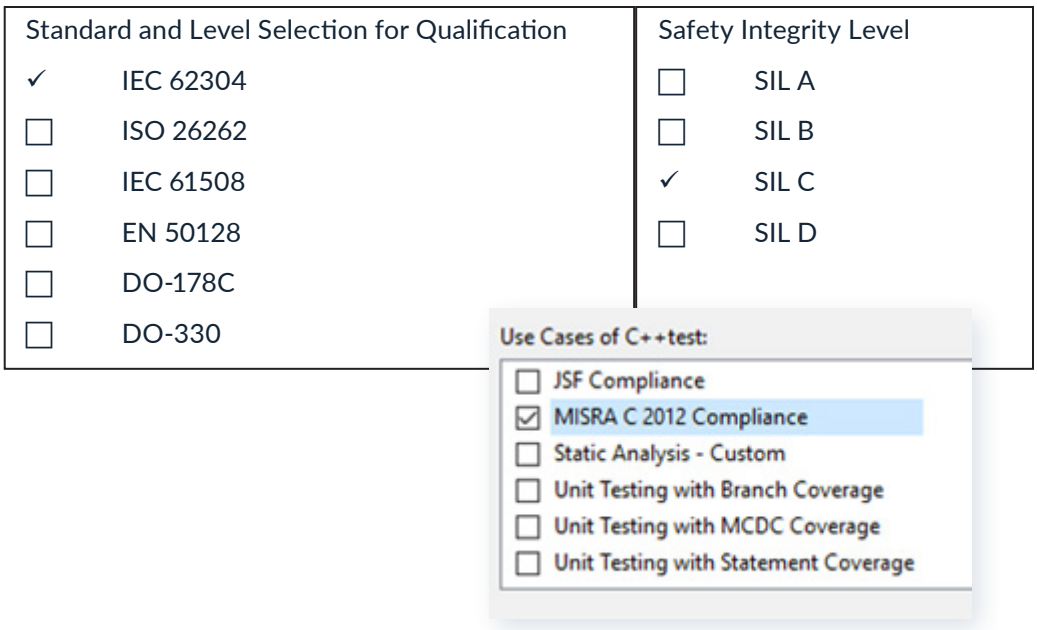


Figure 10-2: Functional compliance selection with additional use case settings

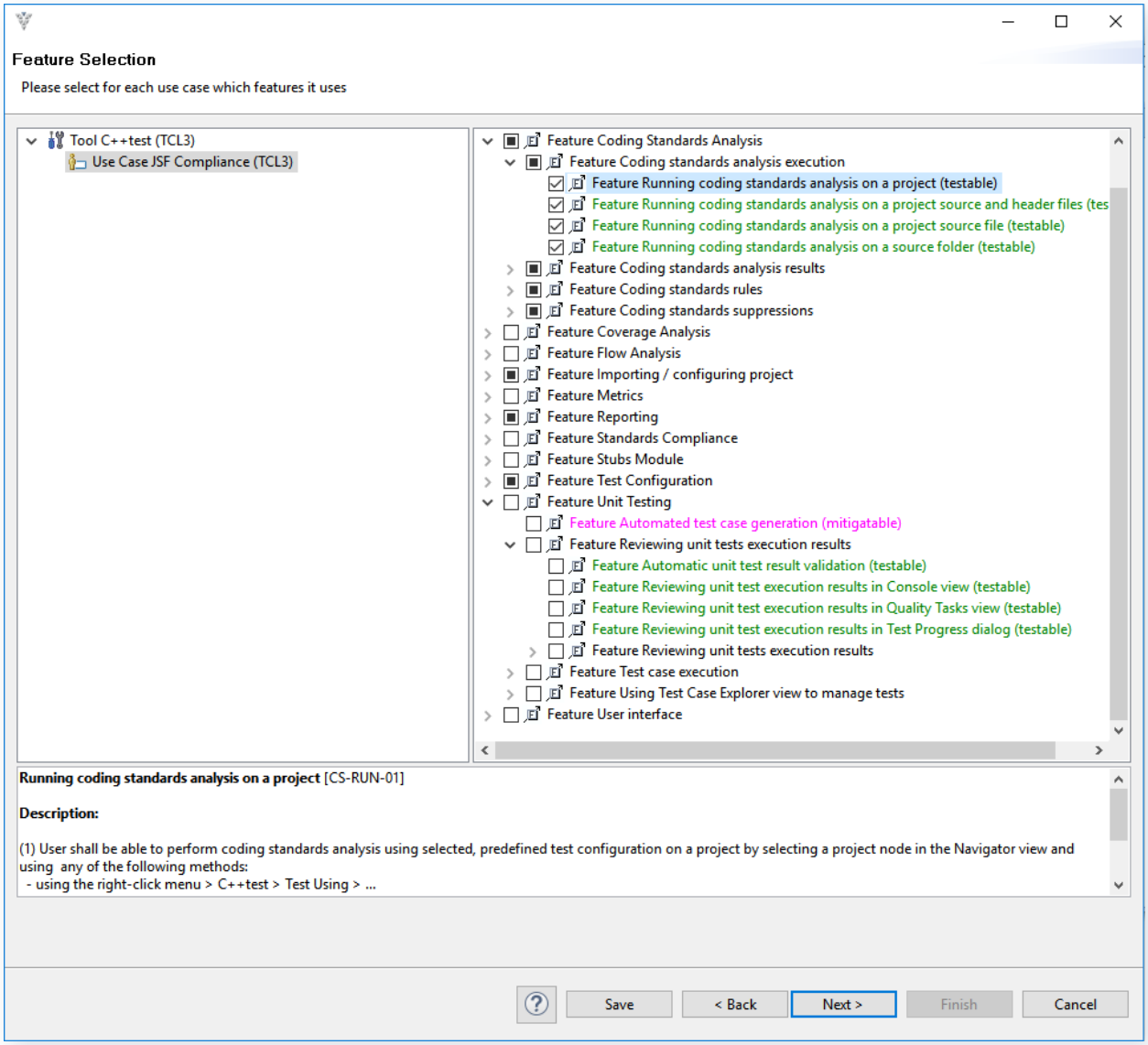


Figure 10-3: Parasoft Qualification Kits allow users to select the options required for their project. Upon selection, only tests and documentation are used and provided from this point forward.

LEVERAGE TEST AUTOMATION & ANALYTICS

A unique advantage to qualifying test automation tools is that the tools can be used to automate their own testing. Automating this as much as possible is key to making it as painless as possible. Even manual tests, which are inevitable for any development tool, are handled as efficiently as possible. Step by step instructions are provided and results are entered and stored as part of the qualification record.

Parasoft C/C++test collects and stores all test results from each build. Tests run as they do for any type of project. These results are brought into the test status wizard in the Parasoft Qualification Kits to provide a comprehensive overview of the results like those shown below.

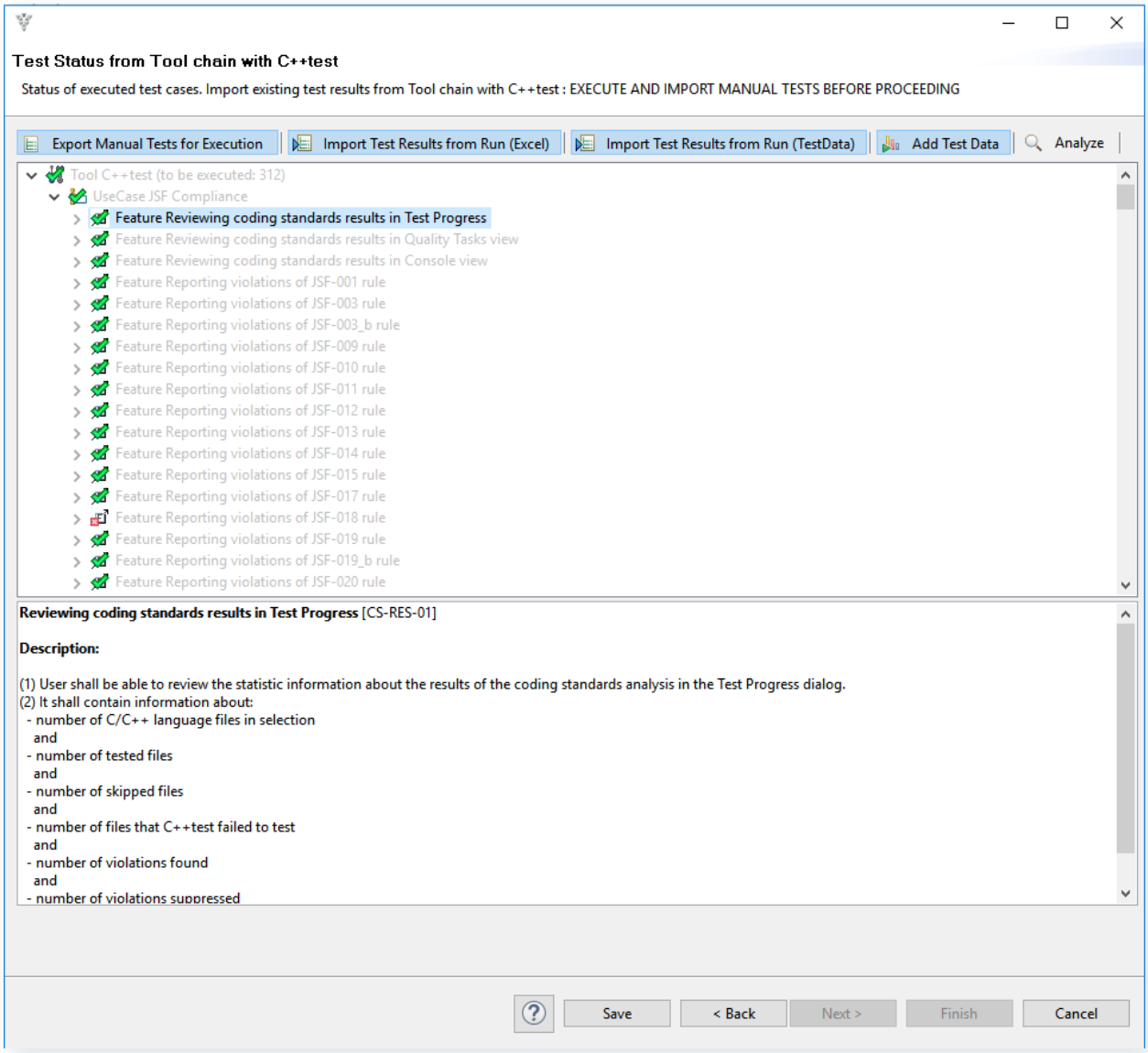


Figure 10-4: Leveraging centralized data collection and automating the qualification process greatly reduces manual tracking of compliance progress.

MANAGING KNOWN DEFECTS

Every development tool has known bugs and any vendor selling products for safety-critical development must have these documented. There's more to dealing with known defects than just documenting them.

Tool qualification requires proof that these defects are not affecting the results used for verification and validation. For each known defect, the manufacturer must provide a mitigation for each one and document it to the satisfaction of the certifying auditor.

It's incumbent on the tool vendor to automate the handling of known defects as much as possible. After all, the vendor is expecting customers to deal with third-party software bugs as part of their workload!

The Parasoft C/C++test qualification kits include a wizard to automate the recording of mitigation for known defects as shown in the example below.

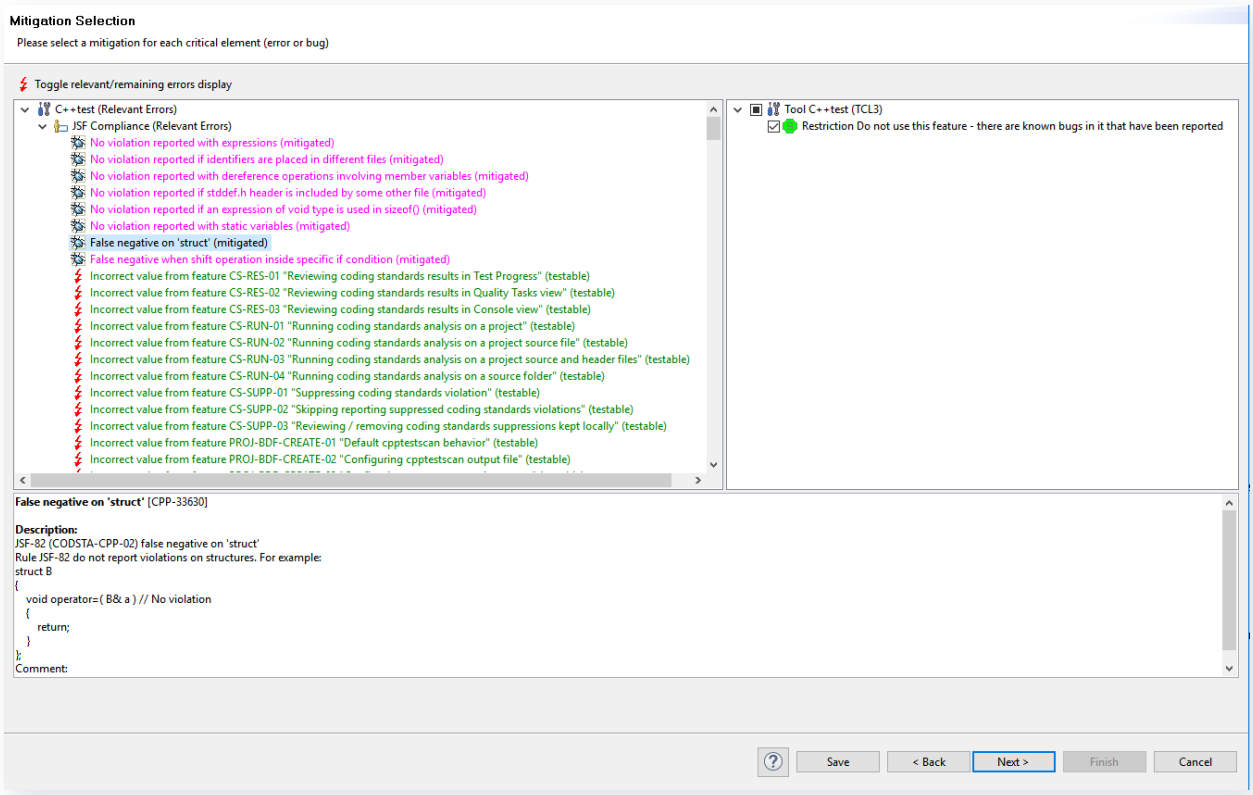


Figure 10-5: Known defects are managed directly in Parasoft C/C++test.

AUTOMATION OF TOOL QUALIFICATION DOCUMENTATION

The end result of tool qualification is documentation, and lots of it. Every test executed with results, every known defect with mitigation, manual test results, and exceptions are all recorded and reported. Qualification kits from other vendors can be just documentation alone and, without automation, documenting compliance is tedious.

Instead, using the Qualification Kits for C/C++test, the critical documents are generated automatically as part of the workflow.

- **Tool Classification Report** determines the qualification needed and presents the maximum safety level classification for C/C++test based on the use cases selected by the user.
- **Tool Qualification Plan** describes how C/C++test is going to be qualified for use in a safety relevant development project.
- **Tool Qualification Report** demonstrates that C/C++test has been qualified according to the tool qualification plan.
- **Tool Safety Manual** describes how C/C++test should be used safely, for example compliant to safety standards, like IEC 62304, in safety-critical projects.

In each of these documents, only the documentation required for the tool featured in use is generated because the scope of the qualification was narrowed down at the beginning of the project. Automation and narrowing the scope of qualification greatly reduces the documentation burden.

REPORTING & ANALYTICS FOR MEDICAL DEVICES SOFTWARE

Parasoft's extensive reporting capabilities bring the results of Parasoft C/C++test into context. Test results can quickly be accessed within the IDE or exported into the web-based reporting system, DTP.

In DTP, reports can be automatically generated as part of CI builds and printed for code audits in safety-critical organizations. Results from across builds can be aggregated to give the team a detailed view without requiring access to the code within their IDE.

In the reporting dashboard, Parasoft's Process Intelligence Engine (PIE) helps managers understand the quality of a project over time. It illustrates the impact of change after each new code change. Integrating with the overall toolchain, PIE provides advanced analytics that pinpoint areas of risk.

DEVELOPER'S VIEW IN THE IDE

Parasoft C/C++test helps teams efficiently understand results from software testing by reporting and analyzing results in multiple ways. Directly in the developer's IDE, users can view:

- Static analysis findings: warnings and coding standard violations
- Unit testing details: passed/failed assertions, exceptions with stack traces, info/debug messages
- Runtime analysis failures with allocation stack traces
- Code coverage details: percentage values, code highlights, including coverage test case correlation

The Quality Tasks view in the IDE makes it easy for developers to sort and filter the results, for example group per file, per rule, or per project. Developers can make annotations directly in the source code editors to correlate issues with the source code. This provides context and more details about reported issues and how to apply a fix.

Code coverage information is presented with visual green and red highlights displayed in the code editor, together with percentage values (for project, file, and function) in a dedicated Coverage view.

Analysis results for both IDE and command line workflows can also be exported to standard HTML and PDF reports for local reporting. For safety-critical software development, C/C++test provides an additional dedicated report format. It details unit test case configuration and includes the log of results from test execution. Users get a complete report of how the test case was constructed and what happened during runtime.

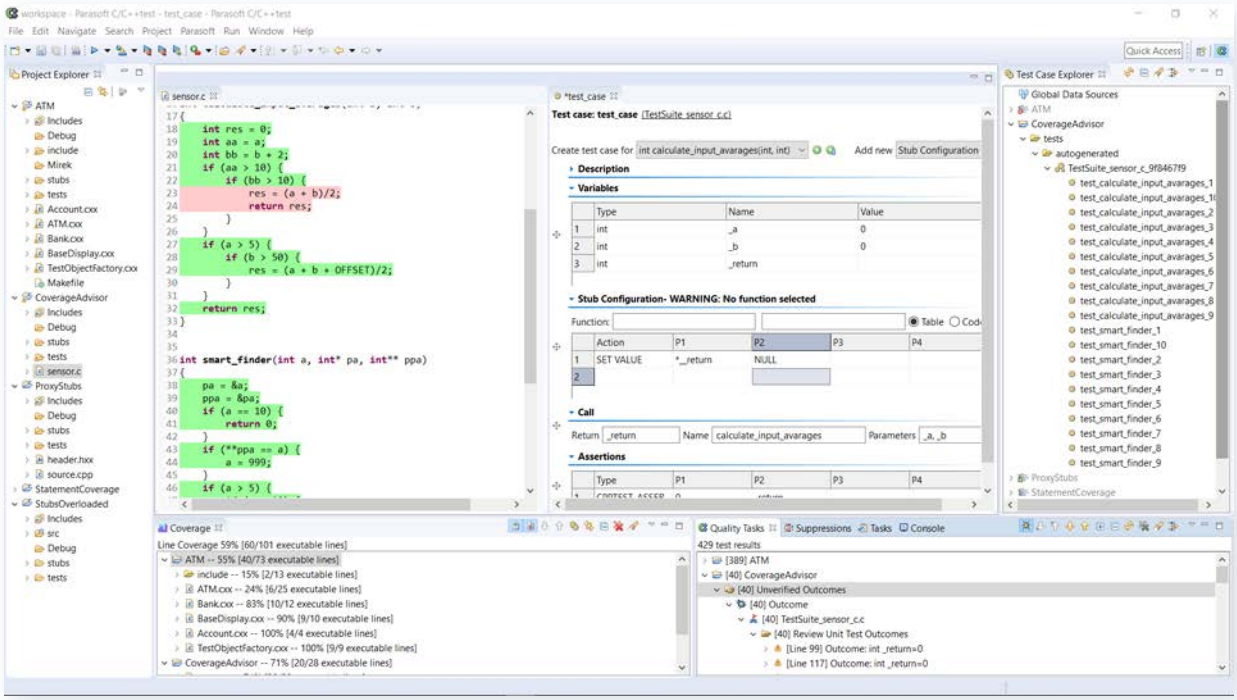


Figure 11-1: Parasoft C/C++test IDE unified code coverage and unit testing view.

TEAM WEB-BASED REPORTING

For team collaboration, Parasoft C/C++test publishes analysis results to DTP, a centralized server. Developers can access test results from automated runs and project managers can quickly assess the quality of the project. Reported results are stored with a build identifier for full traceability between the results and the build. Those results include details about the following:

- Static analysis findings
- Metric analysis
- Unit testing
- Code coverage
- Source code

When integrating into CI/CD workflows, Parasoft users benefit from a centralized and flexible web-based interface for browsing results. The dynamic web-based reporting dashboard includes:

- Customizable reporting widgets
- Source code navigation
- Advanced filtering
- Advanced analytics from the Process Intelligence Engine

Users can access historical data and trends, apply baselining and test impact analysis, and integrate with external systems like those for test requirements traceability.

Figure 11-2: Centralized web based dashboard for test impact analysis and more.

TEST IMPACT ANALYSIS

Each and every test performed, including manual, system level, and UI-based, is recorded as a pass/fail result, including the coverage impact on the code base. Each additional test is overlaid on this existing information, creating a complete picture of test success and coverage.

As code is changed, the impact is clearly visible on the underlying record, highlighting tests that now fail or code that is now untested. Raising this information in various degrees of detail allows developers and testers to quickly identify what needs to be altered or fixed for the next test run.

RISK-BASED ASSESSMENT

In addition to change impact analysis, static analysis can be used to highlight areas of the code that appear riskier than others. Risk can take a variety of forms including:

- Highly complex code
- Unusually high number of coding standard violations
- High number of reported static analysis warnings

These are areas of code that may require additional test coverage and even refactoring.

FUNCTIONAL SAFETY REPORTING

Parasoft C/C++test provides specific reporting capabilities suited to functional safety development. Here are two report examples:

- Unit Testing Execution Details Tests to Requirements Traceability
- Test to Code Coverage Traceability

CODE COVERAGE METRICS

There are various coverage metrics to consider. For medical devices, coverage may be one of the following:

- Statements
- Branch
- Modified condition decision coverage (MC/DC)
- Object/assembly code for the strictest requirements

Parasoft supports gathering all of these coverage metrics, including terms other industries use like block: call, function, path, decision, and more.

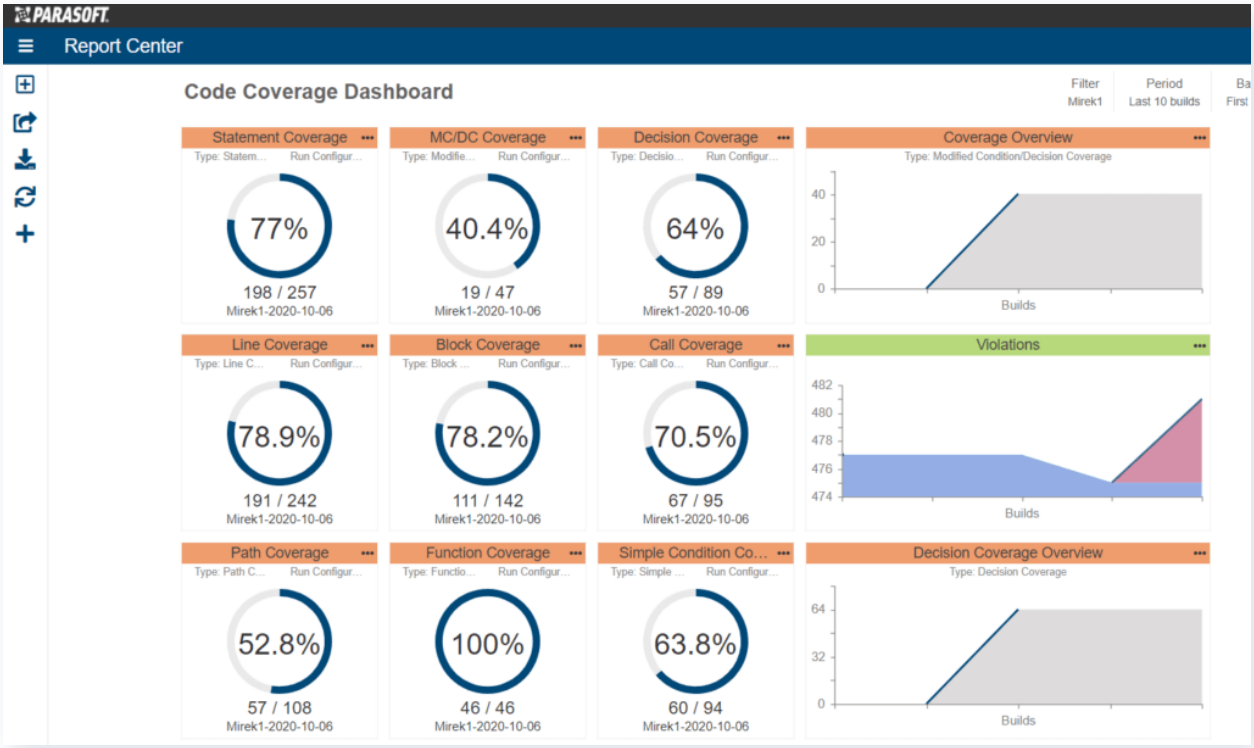


Figure 11-3: Individual code coverage metrics available within reporting dashboard.

CUSTOM ANALYTICS, REPORTS, & DASHBOARDS

Parasoft DTP is highly customizable and supports user configured custom processor for project-specific analysis, custom widgets, and dashboards.

Benefits of Centralized, Aggregated Data Analysis & Reporting

Development teams with one analysis and reporting system for compliance reap the following benefits.

- Efficiency, visibility, and ease of use
- Reduced overhead
- Clear insight into new and legacy code

Manage Compliance With Efficiency, Visibility, & Ease

Instead of just providing static analysis checkers with basic reporting and trends visualization, Parasoft's solution for coding standards compliance provides a complete framework for building a stable and sustainable compliance process.

In addition to standard reporting, Parasoft provides a dedicated compliance reporting module that gives users a dynamic view into the compliance process. Users can see results grouped according to categorizations from the original coding standard, manage the deviations process, and generate compliance documents required for code audits and certification as defined by the MISRA Compliance:2020 specification.

Reduce the Overhead of Testing

With a unified reporting framework, Parasoft C/C++test efficiently provides multiple testing methodologies required by the functional safety standards including static analysis, unit testing, and code coverage.

By presenting cumulative results from the multiple testing techniques, Parasoft provides consistent reporting that reduces the overhead of testing activities. The analytics, reports, and dashboards:

- Simplify code audits and the certification process.
- Eliminate the need for users to manually process reporting to build documentation for the certification process.
- Focus testing efforts where needed by eliminating extraneous testing and guesswork from test management.
- Reduce the costs of testing while improving test outcomes with better tests, more coverage, and streamlined test execution.
- Minimize the impact of changes by efficiently managing the change itself.

Pinpoint Priority & Risk Between New & Legacy Code

Parasoft's Process Intelligence Engine enables users to look at the changes between two builds to understand, for example, the level of code coverage or static analysis violations on the code that has been modified between development iterations, different releases, or an incremental development step from the baseline set on the legacy code.

Teams can converge on better quality over time by improving test coverage but by reducing the potential risky code. The technical debt due to untested code, missed coding guidelines, and potential bugs and security vulnerabilities can be reduced gradually build by build. Using the information provided by Parasoft tools, teams can focus in on the riskiest code for better testing and maintenance.

TAKE THE NEXT STEP

Try Parasoft C/C++test to see how your embedded development team can accelerate the delivery of high-quality, compliant software for medical devices.

ABOUT PARASOFT

Parasoft helps organizations continuously deliver quality software with its market-proven, integrated suite of automated software testing tools. Supporting the embedded, enterprise, and IoT markets, Parasoft's technologies reduce the time, effort, and cost of delivering secure, reliable, and compliant software by integrating everything from deep code analysis and unit testing to web UI and API testing, plus service virtualization and complete code coverage, into the delivery pipeline. Bringing all this together, Parasoft's award-winning reporting and analytics dashboard delivers a centralized view of quality enabling organizations to deliver with confidence and succeed in today's most strategic ecosystems and development initiatives—security, safety-critical, Agile, DevOps, and continuous testing.

More Resources

SAFETY-CRITICAL MEDICAL DEVICE SOFTWARE DEVELOPMENT

CASE STUDIES

- Smiths Medical Delivers Safe, High-Quality Medical Devices With Test-Driven Development
- Inovytec Achieves FDA Certification With Customized Static Code Analysis Solution

WEBSITE

- Medical Device Software Development Testing
- IEC 62304 Compliance With Parasoft
- Application Security Testing – Security Testing Made Simple
- MISRA Compliance With Parasoft
- Easily Automate the Tool Qualification Process

WHITEPAPERS

- Medical Device Software Development - Following FDA Guidelines for Software Validation
- Static Analysis for FDA Software Validation Compliance
- A Practical Guide to Accelerate MISRA C 2023 Compliance With Test Automation
- Accelerating MISRA C & SEI CERT C Compliance With Dedicated Reporting and Workflow Management
- Streamlining Unit Testing for Embedded and Safety-Critical Systems
- Embedded Cybersecurity Through Secure Coding Standards CWE and CERT

BLOG POSTS

- What Is IEC 62304 & How Is It Used in Medical Device Compliance?
- Prepare Your Medical Device Software for the New FDA Cybersecurity Guidance
- The Cure for Software Defects and Vulnerabilities in Medical Devices
- How a Medical Device Company Cut Testing Costs in Half
- 5 Tips for Static and Dynamic Analysis in Medical Device Software
- Expedite Your Code Coverage Task With a Coverage Advisor
- Regression Testing of Embedded Systems
- Verification vs Validation in Embedded Software
- Reducing the Risk and Cost of Achieving Compliant Software
- Qualifying a Software Testing Tool With the TÜV Certificate
- Breaking Down the AUTOSAR C++14 Coding Guidelines for Adaptive AUTOSAR
- A Smoother Road to MISRA Compliance
- The Two Big Traps of Code Coverage
- Shift-Left Your Safety-Critical Software Testing With Test Automation
- Requirements Management and the Traceability Matrix

WEBINARS

- How to Tackle Software Regulatory Compliance for Medical Devices
- Smiths Medical: How Does Your Unit Testing Stack Up
- How to Mitigate Cybersecurity Software Vulnerabilities in Medical Devices
- Learn How to Shorten Time-To-Market for New Medical Devices
- Cut Compliance Costs and Ensure Lifecycle Traceability With codebeamer ALM & Parasoft
- Make Your C/C++ Applications Safe and Secure With MISRA and CERT
- Automate Essential Testing to Verify & Validate Polarion Requirements
- Requirement Traceability for Safety-Critical Applications