ferrous systems

Safe, Reliable, and Future-Ready: The Business Case for Rust and Ferrocene in Safety-Critical Systems

Jonathan Pallant, Senior Embedded Systems Engineer, Ferrous Systems

Contents

Introduction 2

The Growing Need for Safer Programming 2

Rust's Advantages in Safety Critical Systems 5

The Road to Adoption: The Business Case for Rust and Ferrocene 8

Real-World Applications of Rust in Safety-Critical Systems 12

The Future of Safety-Critical Systems is Rust 14

Introduction

As embedded systems in safety-critical industries become increasingly complex, the demand for reliability, security, and maintainability has never been higher. In the past traditional programming languages like C and C++ have served these industries well, but they also bring significant risks that are only amplified as systems scale and the complexity of software increases. The need for safer, efficient and more reliable solutions is now an urgent necessity. At the same time, organizations face mounting pressure to ensure integrity and transparency across their software supply chain, making toolchain choices as critical as the code itself. Rust, coupled with the qualified Ferrocene toolchain, is poised to meet these needs. It blends the performance and low-level control of traditional languages with modern safety features and certification support, making it an ideal candidate for not only safety-critical embedded systems, but any application where safety and performance are paramount.

The Growing Need for Safer Programming

As software becomes increasingly embedded in critical infrastructure, transportation, healthcare, IoT, and industrial systems, the consequences of programming errors have never been more severe. To take automobiles as an example, even low-end vehicles today feature up to 100 separate ECUs, and 100 million lines of code (Motortrend).

At that kind of scale, and combined with the levels of connectivity demanded by the modern consumer, memory corruption, data races, and undefined behavior—once considered mere nuisances—are now potential vectors for catastrophic failures, security breaches, and even loss of life. In industries where reliability is paramount, traditional programming approaches reliant on manual memory management and runtime safeguards are proving insufficient and costly. The demand for safer and more efficient software development practices is no longer optional, it is an urgent necessity.

Historically, languages like C and C++ have dominated systems programming due to their low-level control and performance advantages. However, these languages offer little protection against entire classes of vulnerabilities that continue to plague critical software. Buffer overflows, use-after-free errors, and data races are persistent threats, leading to costly recalls, regulatory fines, and severe reputational damage. While best practices, rigorous testing, and static analysis tools can mitigate many of these risks, they come at a huge expense and are often reserved for only the most critical of systems. As we build ever larger, more complex, and more connected systems, this solution does not scale. The software industry needs a fundamental shift towards widespread, practical, software safety without sacrificing efficiency or control.

As an example, Google has been rolling out Rust into the Android OS (the codebase that powered around 1 billion new smartphones in 2024 (Canalys)). They report that their Safe-Coding approach, including the use of Rust when adding new features to Android, is responsible for a large and continued decline in memory safety vulnerabilities in the codebase – well below industry norms (Google).

> The demand for safer and more efficient software development practices is no longer optional, it is an urgent necessity.

Regulations Are Reshaping Software Development

These changes aren't just occurring in commercial software development - regulatory landscapes are also evolving to reflect these growing concerns. New safety and security standards are placing stricter requirements on software correctness, maintainability, and verifiability. Organizations developing software for aerospace, automotive, medical devices, and industrial automation are now required to provide stronger guarantees of reliability and compliance for ever larger codebases. This shift demands programming tools and languages that provide built-in safety mechanisms, enforce correctness at compile time, and minimize human error. One

example of the legislative changes that are coming is the European Cyber Resilience Act, or CRA (europa.eu). It is fundamentally changing software development by making security and compliance non-negotiable. With deadlines rapidly approaching, companies must adapt to new requirements: by 11 June 2026, rules for conformity assessment bodies take effect; by 11 September 2026, manufacturers must report exploitable vulnerabilities and significant incidents; and by 11 December 2027, all products with digital elements sold in the EU must meet cybersecurity standards and carry the CE marking to demonstrate compliance. These regulatory shifts underscore the growing need for software development practices that prioritize security, transparency, and resilience—pushing businesses to adopt safer programming languages and tools that inherently support these principles.

The European Cyber Resilience Act (CRA) is fundamentally changing software development by making security and compliance non-negotiable.

The question is no longer whether safer programming is needed everywhere, it is how best to achieve it. The ideal solution must ensure safety and correctness without introducing performance bottlenecks or unnecessary complexity. It must also be maintainable, scalable, and capable of supporting long-term software lifecycles. This is where Rust offers a compelling alternative: by providing memory safety, thread safety, and modern tooling while maintaining the performance characteristics of low-level languages. By addressing the root causes of common software failures, Rust represents a transformative approach to systems programming, aligning with both industry needs and regulatory expectations.

ferrous systems

Rust's Advantages in Safety Critical Systems

Rust addresses critical challenges in safety-critical systems with its unique **model of memory ownership**, which helps to eliminate memory-related defects like null-pointer dereferencing, buffer overflows, and data races, at compile time. The early detection and mitigation of these issues reduce the amount of engineering effort wasted on debugging and ensures that systems are safer and more reliable overall. The major leap forward in language design is that Rust is able to provide this **memory safety** by using strict compile-time checks rather than garbage collection, making it ideal for real-time applications that require deterministic performance.

Rust is able to provide memory safety by using strict compile-time checks rather than garbage collection, making it ideal for real-time applications that require deterministic performance.

Rust also leverages its **ownership and borrowing system** to ensure **thread safety** at compile time, preventing data races and other issues common in concurrent systems. This safety feature is particularly valuable in embedded systems, where concurrency bugs can have severe, often hidden, consequences, and where the use of a high-level language like Python, C# or Java would add unacceptable overheads.

Rust further excels in constrained environments thanks to its **minimal runtime overhead** and **excellent compatibility with C based APIs**. This allows it to inter-operate with almost any existing Operating System or Real Time Operating System, or run straight on bare-metal. That makes Rust a natural choice on the microcontrollers and other resource-constrained systems common in safety-critical and real-time applications.

In addition to safety, Rust improves **long-term maintainability**. Its strong type system, modularity and explicit memory management make codebases more predictable, reducing technical debt and ensuring scalability. Unlike traditional systems programming languages that can lead to fragile, hard-to-debug code, Rust encourages a structured approach that results in more maintainable and sustainable software for industries with long product lifecycles. Indeed, Google reports that their Rust developers are twice as productive as teams working in C++ (The Register) – mainly due to them spending far less time debugging code they've already written.

Furthermore, Rust's ecosystem continues to evolve to support a broad range of safety-critical applications. The availability of formal verification tools, deterministic execution models, and qualified toolchains like Ferrocene, ensures that Rust can meet the highest standards for **correctness and compliance**. As organizations face increasing regulatory scrutiny and the rising costs of software failures, Rust provides a forward-looking solution that not only meets today's needs but is built to support the demands of the future.

Finally, Rust, Ferrocene, and the surrounding open-source ecosystem, offer a compelling response to the growing challenges of **managing modern software supply chains**, particularly in safety-critical industries such as automotive, medical, and industrial control. Rust's strong memory safety guarantees, absence of undefined behavior, and modern concurrency model help reduce the risk of defects that could propagate through a company's internal or external supply chain. Ferrocene builds on this foundation by providing a qualified Rust toolchain tailored for use in safety-critical contexts, enabling compliance with standards like ISO 26262 or DO-178C. By leveraging open-source technologies with transparent development practices and broad community support, companies gain increased visibility into their toolchain, reduce vendor lock-in, and ensure long-term maintainability. This approach supports both innovation and rigorous certification demands, empowering organizations to modernize their systems while maintaining confidence in the safety, reliability, and integrity of their software supply chain.

ferrous systems

Key Benefits of Rust:

Advantage	Description
Compliance with Industry Standards	With quality-managed and qualified toolchains like Ferrocene, Rust can be used in compliance with a number of key safety-critical standards, making it suitable for industries with stringent regulatory requirements. Ferrocene's use of Rust is covered by the rigorous Ferrocene Language Specification, and work is underway to transition towards an official Rust Specification.
Safer Concurrent Systems	Rust enforces thread safety at compile time, preventing data races that are difficult to detect in traditional languages. This makes it particularly valuable in concurrent systems and embedded applications.
Long-term Maintainability	Rust's strong type system, ownership model, and explicit memory management reduce technical debt, improve code predictability, and ensure maintainability over long product lifecycles, particularly in safety-critical industries. Rust toolchains like Ferrocene offer stable releases with long-term support – ideally suited to these kinds of long-lived developments.
Memory Safety Without Garbage Collection	Rust eliminates buffer overflows, use-after-free, and null pointer dereferences through its type system. Unlike dynamically managed languages, Rust achieves safety without runtime garbage collection, making it suitable for hard-real-time applications with tight timing constraints.
Minimal Runtime	Rust's 'no-std' mode allows embedded applications to run on bare-metal, making it ideal for microcontrollers and other resource-constrained environments. Using the same libraries across the cloud, the web, on mobile, and on deeply embedded targets, allows developers to truly write once and deploy anywhere.
Modern Tooling and Ecosystem	Cargo, Rust's package manager, streamlines dependency management and the firmware build workflow. Ecosystem supplied libraries like embedded-hal allow for effective hardware abstraction across embedded platforms, whilst services like kellnr and JFrog Artifactory allow product managers to know precisely what goes into each Rust firmware build. And even when you're not writing Rust, the best modern developer tools are written in Rust – like uv for Python.

The Road to Adoption: The Business Case for Rust and Ferrocene

Adopting a new programming language, especially in safety-critical industries, is a strategic decision that requires buy-in from multiple stakeholders. While challenges such as managing the transition from C or C++, or concerns about ecosystem maturity, may seem discouraging, these barriers can be overcome with a few key strategies and the right support. Advocating for Rust and Ferrocene within organizations by focusing on several key business, compliance, integration, and engineering advantages, can make a compelling case for adoption.

Business and Competitive Advantages

Rust's safety guarantees reduce debugging, testing, and validation efforts, leading to faster development cycles and lower costs. By eliminating entire classes of vulnerabilities, Rust minimizes the risk of costly recalls and compliance violations, offering long-term cost savings. Additionally, Rust's modern tooling and ecosystem ensure scalable, maintainable codebases, reducing technical debt and improving futureproofing. These advantages position Rust as a valuable asset in the competitive landscape of safety-critical software development.

Getting Cyber Resilient

Ferrocene addresses many of the concerns posed by the European Cyber Resilience Act. Even for products that aren't safety-critical, Ferrocene's ISO 9001 quality management means that it's a product you can trust. Ferrocene also has tools to help you understand precisely what goes into the product you are shipping, and how to stay on top of critical security issues.

Engineering and Talent Advantages

Rust's modern approach to safety and performance appeals to developers looking for cutting-edge tools that enhance productivity and code quality, so adopting Rust can position your organization to attract and retain top talent in an increasingly competitive tech landscape.

Incremental adoption strategies, coupled with quality, targeted training from experienced professionals, can streamline the transition ensuring teams quickly adapt to Rust while maintaining existing workflows.

Seamless Integration with Legacy Codebases

Rust is designed to integrate seamlessly with existing codebases, such as C and C++, making adoption incremental rather than all-or-nothing. Its robust FFI (Foreign Function Interface) allows developers to call C functions directly from Rust and vice versa, enabling teams to modernize critical components without a full rewrite. Tools like cbindgen and bindgen simplify interoperability by automating header generation and binding creation. With tools like cxx, bindings can also be generated for libraries written in C++. Rust's ownership system can be leveraged in these adaptations by providing software layers that encapsulate the raw C or C++ types and provide a memory-safe interface over the top. This incremental approach ensures that teams can continue leveraging their existing software stack, whilst enjoying the long-term benefits of Rust's robust safety and concurrency model for newer developments. This approach is perhaps exemplified by the addition of Rust into the Linux Kernel. There are of course challenges to be managed with any such transition, but developers who have been writing new Linux kernel modules in Rust have repeatedly stated that developing the equivalent modules in C would have taken much longer to write and debug (Asahi Linux).

The Benefits of Open Source with the Guarantees of a Quality-Managed Solution

Open source Rust, while widely supported by an active and growing community, may not yet provide the level of assurance required by many mission- and safety- product development processes. Traditional toolchains from dedicated providers offer more guarantees, but solutions are often proprietary and closed source, involving expensive licensing fees, ongoing maintenance costs, limited flexibility, and long-term contracts. Ferrocene, as the first qualified, open source, Rust compiler toolchain, offers a real alternative here.

Developed as a downstream Rust distribution, Ferrocene is 100% compatible with the upstream Rust compiler distributed by the Rust project, allowing developers to work seamlessly with both compilers on the same code base – with no changes, and none of the 'if this compiler then do that' style workarounds that are commonly seen in C and C++ codebases.

Ferrocene effectively bridges the gap between Rust's modern, safety-focused features and industry demand for quality-managed and certified software supply chains. By leveraging, and contributing to, the collective innovation of Rust's vibrant open-source community, Ferrocene is able to provide an open, transparent, and cost-effective alternative to proprietary solutions, while providing the long-term stability, rigorous compliance, and extended toolchain support that industrial applications require.

Ferrocene effectively bridges the gap between Rust's modern, safety-focused features and industry demand for quality-managed and certified software supply chains.

Ferrous Systems: **Product Portfolio**

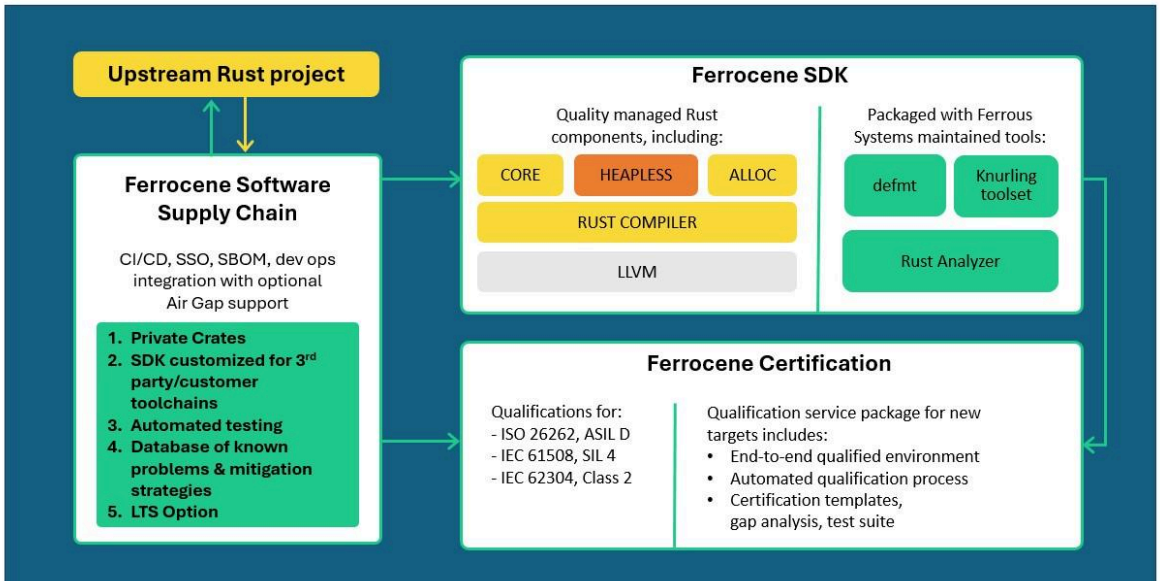

***Figure 1**: Ferrocene's primary product is the Ferrocene Software Supply Chain, which is used to manage downstream distributions of open-source software; this is used by Ferrous Systems to develop both the Ferrocene SDK and the Ferrocene Certification pack, but can also be used by customers directly to manage their own toolchains and locally managed open-source packages.*

Ensuring Compliance with Qualified Tooling

Rust's inherent advantages make it an appealing choice for safety-critical systems but adopting it in strictly regulated industries requires specialized tooling and certification pathways. Ferrocene has been proven to meet this challenge, and its open-source foundation and TÜV SÜD assessment provide a clear path to compliance with industry standards. Currently qualified for ISO 26262 (TCL 3/ASIL D), IEC 61508 (T3/SIL 4), and IEC 62304 (Class C), its compliance with these rigorous safety standards gives you a compelling combination of correctness, efficiency, and resilience.

By combining Rust's safety and performance with Ferrocene's industrial-grade reliability, organizations can confidently adopt Rust for mission-critical applications while securing a sustainable, verifiable development process.

Key Benefits of Ferrocene:

Feature	Description
Compatibility with Safety-Certified Workflows	Ferrocene provides a clear pathway for incorporating Rust into existing safety-certified workflows, addressing concerns around certification and toolchain validation.
Regulatory Alignment	A qualified compiler toolchain suitable for use in ISO 26262, IEC 61508, and other safety-critical standards. It is one compiler to suit all industries.
Open-Source Foundation	Ferrocene's open-source foundation brings continuous, community-driven improvements, without sacrificing quality or stability.
Long-Term Support & Maintenance	Ensures stability and compliance for industrial applications, providing ongoing updates and security patches.
Cost-Effective Licensing	Flexible and sustainable pricing for long-term projects.

Real-World Applications of Rust in Safety-Critical Systems

Rust's robust safety features and growing ecosystem are already making a significant impact in industries where reliability and security are non-negotiable. As software complexity in safety-critical systems continues to rise, companies across sectors are turning to Rust to address the challenges of memory safety, concurrency, and performance. Rust's ability to prevent common software errors like buffer overflows, data races, and null pointer dereferencing, combined with the added security of the Ferrocene toolchain, makes it an ideal choice for applications where failure is not an option. The following examples highlight how Rust is being applied to some of the most demanding industries in terms of safety and performance.

Automotive

In the automotive industry, a number of vehicle manufacturers and Tier 1 suppliers are exploring Rust for developing software for advanced driver-assistance systems (ADAS) and electronic control units (ECUs), where safety and performance are crucial. Even today, the Low Power ECU fitted to every Volvo EX90 or Polestar 3 electric vehicle off the production line is running firmware written Rust, and running on top of a pure-Rust real-time framework – the result of a development program that has been running for a number of years (Tweede golf). Rust's memory safety features help mitigate risks associated with software failures, such as those that could lead to vehicle recalls, and Ferrocene's compliance with ISO 26262 (TCL 3/ASIL D) ensures that it meets the rigorous safety requirements for future automotive applications.

Industrial Automation

In industrial automation, Rust is being adopted in control systems for robotics, programmable logic controllers (PLCs), and real-time data acquisition systems. The language's concurrency safety and predictable performance characteristics are critical in these environments, where system failures could have wide-reaching consequences. Ferrocene's compliance with IEC 61508 (T3/SIL 4) ensures that it meets the rigorous safety requirements for industrial applications.

Medical Devices

In the medical device space, Rust's safety features are being used to develop firmware for systems like imaging devices, infusion pumps, and implantable medical devices. By reducing memory-related vulnerabilities, Rust enhances the safety of these devices. Rust's strong track record in cybersecurity also aligns with increasing regulatory demands, including Section 524B of the FD&C Act, which requires manufacturers to maintain a strong cybersecurity posture, and the FDA's final guidance on cybersecurity in medical devices emphasizes the importance of secure software development practices, risk management, and cybersecurity measures throughout the medical device lifecycle. Ferrocene's compliance with IEC 62304 (Class C) positions it as an ideal

solution for medical device developers wanting to streamline compliance efforts while ensuring the highest standards of safety and reliability.

Aerospace

Aerospace systems demand high reliability, deterministic performance, and strict safety compliance. Software used in avionics, flight control, and satellite firmware must adhere to rigorous safety and verification processes to ensure correctness, robustness, and traceability. DO-178C, the primary certification standard for airborne software, requires compliance with stringent objectives such as formal software verification, traceability from requirements to implementation, structural coverage analysis, and rigorous testing at various levels of integration. With Ferrocene's qualified compiler, Rust can now meet all the analysis requirements under DO-178C. This enables Rust to be adopted in certified aerospace applications, from commercial and military aircraft to unmanned aerial vehicles (UAVs) and space systems. Additionally, Rust's modern tooling, strong ecosystem, and built-in concurrency model make it well-suited for real-time embedded systems found in aerospace.

IoT and Connected Devices

Embassy, the popular Asynchronous Embedded Rust framework, was developed with Smart Locks (Akiles) in mind. Whilst not at the highest levels of criticality, Rust gives these low-power, long-lived devices the security they need to offer remote connectivity, even in places where firmware upgrades are typically considered highly challenging.

The Future of Safety-Critical Systems is Rust

As regulatory requirements tighten and software complexity grows, Rust is poised to become increasingly integral in safety-critical domains. Its memory safety guarantees, modern tooling, and growing ecosystem make it an ideal choice for industries that demand high-assurance software.

Ferrocene also plays a critical role in this adoption by providing a compliance-ready solution that effectively bridges the gap between the flexibility of open source development and the rigorous demands of regulated environments. Looking forward, the expansion of ecosystem support, formal verification tools, and safety-critical libraries will further solidify Rust's position as a leading language for secure, reliable, and efficient systems that can meet the growing demands for safety, performance, and compliance.

The complexity of embedded systems in safety-critical applications is increasing, alongside the demand for more reliable and secure software. Rust provides a solution that combines memory safety, concurrency guarantees, and real-time performance in a modern, efficient language. Investing in Rust and Ferrocene today isn't just a matter of keeping up with technology. It's a strategic decision to ensure the correctness, efficiency, and resilience of future systems. By adopting these tools now, companies can improve their programming model while investing in supply chain integrity, future-proof tooling, and long-term maintainability. In doing so, they can confidently meet today's regulatory requirements while preparing for the challenges and demands of the safety-critical industries of tomorrow.

Jonathan Pallant is a Senior Embedded Systems Engineer at Ferrous Systems and has been applying Rust to Bare Metal systems since 2016. Jonathan was a founding member of the Rust Embedded Working Group and spent a year on the Rust Leadership Council. He has also given numerous conference talks on Rust and Embedded.