Document: P2026R0
Author: Ryan McDougall <rmcdougall@aurora.tech>
Co-authors: Bryce Adelstein Lelbach <blelbach@nvidia.com>
JF Bastien <jfbastien@apple.com>
Andreas Weis <Andreas.WA.Weis@bmw.de>
Ruslan Arutyunyan <ruslan.arutyunyan@intel.com>
Ilya Burylov <ilya.burylov@intel.com>
Glenn Elliott <glenn@aurora.tech>
Clint Liddick <clint@aurora.tech>
Philipp Robbel <philipp.robbel@aptiv.com>
Aditya Sreekumar <aditya.sreekumar@nutonomy.com>
Vasumathi Raman <vasumathi.raman@gmail.com>
Jake Askeland <jake.askeland@gmail.com>
Audience: WG21
Project: ISO/IEC JTC1/SC22/WG21 14882: Programming Language — C++

A Constituent Study Group for Safety-Critical Applications

Purpose

C++ runs banks, factories, airplanes, and cars. C++ is fast and malleable, and scales to both big and small problems. With a proliferation of devices embedded in our everyday lives, particularly in the area of robotics, C++-based systems will have an important effect on our world.

It is important that we engineer systems that are useful and safe. C++ is a fit-for-purpose solution for resource-constrained and real-time systems with safety implications. We have an ethical responsibility for such systems, and must make efforts to ensure software written in C++ is reliable and safe.

We propose a constituent study group that represents users of C++ for safety-critical applications.

Who We Are

C++ developers who care about systems for real-world applications with consequences, such as Robotics, Transportation, and Health. This includes industries such as:

- Autonomous Vehicles
- Consumer Robotics
- Internet of Things
- Industrial Automation and Control
- Heavy or High Energy Machinery

- Medical Equipment
- Aerospace
- Military or Personal Security

Safety

Safety is an emergent property of systems, measured statistically, and regulated by policy. The prerogative to decide if something is “safe” is usually reserved by governmental organizations.

Regulatory Agencies

In the United States, example regulatory agencies include the Federal Aviation Administration, and the National Highway Traffic Safety Administration. NHTSA declares its mission: “*As a Federal agency, NHTSA regulates the safety of motor vehicles and related equipment.*” and its oversight capacity: “*Laws Administered by NHTSA: Motor Vehicle Safety, Highway Safety, …*” (acts).

In the EU, there is a complex array of UN, EU, and national bodies. Generally each nation has an equivalent to the FAA or NHTSA.

Physical Safety

Physical safety is well defined by a number of regulatory agencies.

NHTSA

49 U.S. Code § 30102: *“motor vehicle safety” means the performance of a motor vehicle or motor vehicle equipment in a way that protects the public against unreasonable risk of accidents occurring because of the design, construction, or performance of a motor vehicle, and against unreasonable risk of death or injury in an accident, and includes nonoperational safety of a motor vehicle.*

EU

3.1 The Safe System
The Safe System approach aims for [...] a layered combination of measures to prevent people from dying [...] by taking the physics of human vulnerability into account. Better vehicle construction, improved road infrastructure, lower speeds for example all have the capacity to reduce the impact of crashes. Taken together, they should form layers of protection that ensure that, if one element fails, another one will compensate to prevent the worst outcome.This approach involves multi-sectoral and multi-disciplinary action and management by objectives, including timed targets and performance tracking.

International Electrotechnical Commission

A safety-related system comprises everything (hardware, software and human elements) necessary to carry out one or more safety functions, where failure of the safety function would give rise to a significant increase in the risk to the safety of persons and/or the environment.

ISO 45001

Injury and Ill Health

Adverse effect on the physical, mental or cognitive condition of a person.

Hazard

Source of potential injury and ill health (3.18).

Risk

Effect of uncertainty.

An effect is a deviation from the expected. Uncertainty is the state of deficiency of information related to, understanding or knowledge of an event, its consequence, or likelihood.

Risk is often characterized by reference to potential "events" and "consequences" or a combination of these, and is often expressed in terms of a combination of the consequences of an event and the associated "likelihood" of occurrence.

This constitutes one of the common terms and core definitions for ISO management system standards.

Health and Safety Risk

Combination of the likelihood of occurrence of a work-related hazardous event(s) or exposure(s) and the severity of injury and ill health (3.18) that can be caused by the event(s) or exposure(s).

Language Safety

Terms like "thread safety", "memory safety", "type safety", and "exception safety" are terms of art in computer science. Programs with these properties may be called "safe". However it is important to draw a distinction between the real world physical safety of a programmed system, and the low-level per-component properties that make up the system. Safety of a component is a *necessary*, but *not sufficient*, condition for the safety of the whole. It may even compromise systemic safety if this distinction is lost.

Safety Critical Systems

For our lay consideration, we define a safety critical system to be one whose failure or malfunction may result in:

- death or injury to people
- equipment or property damage potentially injurious to people
- environmental harm potentially injurious to people

Such systems are often designed to have less than one critical failure per billion (10^9) hours of operation.

Safety Critical Systems are defined by how they react to failure. We desire that systems be Fault-Tolerant, and Fail-safe. For software systems this requires three pillars:

1. Failure Mode Analysis: Look at all the ways the system might fail, and minimize the probability of failure with redundancy, or ameliorate the failure with a fallback.
2. Software Quality: The system is designed and implemented to meet those requirements
3. Software Assurance: The system operates as designed in expected conditions

All possible failure modes must be anticipated, fault-tolerance/fail-safe behavior must be formulated, and the system must be measured to behave in its operating domain as expected. This requires not only foresight to capture possible failure modes ahead of time, but also avoid creating any *new* or unforseen failure modes (i.e. bugs). Historically writing defect-free code has been difficult.

Quality

Quality is the surface area where software performance meets requirements. Reliability is quality over time. Defects are caused by failure of the software to perform. Defective software is unable to meet its fault-tolerant and fail-safe requirements. High quality systems are written to avoid, detect, report, isolate, and recover from defects.

Assurance

Assurance is a level of confidence established by repeated defect-free performance of software in its expected operating environment, including in the presence of failures.

Recapitulation

Safety is:

1. Property of a system
2. Measured statistically
3. Standardized and regulated

Safe Systems:

1. Minimize their known failure modes
2. Reliably handle their known failure modes
3. Fail gracefully in the presence of unforeseen failure modes
4. Limit defects in performance
5. Demonstrate expected performance, in a representative operating environment, up to a satisfactory confidence level

The astute reader will notice while our *constituency* is safety-conscious, most of what makes a system **safe** is *fundamentally* beyond the reach of the individual software developer.

Critical Reliability for C++

As demonstrated above, unlike another sensitive subject in computing applications -- security -- safety **cannot** be offered from behind a keyboard. What software developers *can* offer is **reliability**.

A study group or a working group *cannot* claim the prerogative of a government regulator. A study group or a working group *cannot* provide statistical assurance. A study group or a working group *cannot* standardize reliability by fiat. What a study group may offer is constituency consultation on the features that would improve the reliability of systems built with C++ that will end up in safety critical systems.

Critical Reliability Study Group

The name of the group sets the direction and scope. We believe the name that is most accurate, most inclusive, and most resilient to change, is “Critical Reliability”.

Scope

Incubate and review papers that improve suitability of C++ for critically reliable systems.

We wish to

- Provide feedback on how C++ evolution impacts critically reliable systems
- Identify unmet needs for the community and propose fixes for those gaps

We care about

- Predictable behavior
- Resource prioritization and constraints
- Completion guarantees
- Avoiding worst-case behavior
- Avoiding pathological behavior
- Minimizing programmer error
- Catching programming errors early as possible

- Isolating and containing failure
- Reproducing failure for diagnostic purposes
- Reasoning about failure modes
- Certifying or proving capability from source

Areas of potential collaboration

- Programming by Contract
- Deterministic Error Handling
- Deterministic Memory Allocation
- Type Safe Arithmetic
- Type Safe Units
- Tooling for Component Re-use
- Tooling for Source and Component Auditing
- Logging and Tracing
- Determinism and Reproducibility
- Redundancy and Isolation
- File and Process Privilege
- Task Priority and Executors
- Coroutine Performance

Non-Goals

- Performance or size optimization
- Coding standard development or enforcement
- Domain expertise outside reliable systems
- Dialects or niche solutions

Discarded Naming Choices

“Safety Critical” describes the constituency, but is overreaching in what we can promise, or what we may rightly claim authority over. Safety cannot be willed by a programmer, and safety regulation is a government prerogative in most jurisdictions.

“Robotics” is where the most new users of C++ for safety critical systems come from, and indeed it’s an attractive name currently. However we wish to avoid changes in fashion, or unnecessarily restrict our scope to neglect important constituents such as aerospace or medical.

Relationship to Embedded, Real-time, or Low-latency Development

Fast, predictable, response times are often a necessary requirement for real world systems that affect safety. Due to size, or other environmental constraints, those systems may be embedded into resource constrained devices. However small program size and good response time are **not *sufficient*** conditions for a system to be considered reliable and safe.

Game developers, scientific developers, high-frequency traders, or microcontroller programmers may also enjoy reliable software -- but they have more immediate priorities beyond physical safety, and those priorities are already represented by other study groups.

Relationship to Coding Standards, Guidelines, and Avoiding Undefined Behavior

Coding standards attempt to prophylactically avoid failure modes by mandating best practice. We have many examples of C++ coding standards, and know of many “bad habits” that lead to costly bugs. Having this information alone appears to be insufficient to ensure quality. Moreover rigid rules in place of careful design may in fact make things worse, as defects are typically proportional to new lines of code, and introducing an enforced standard could cause churn. The price to recertify the modified software would be worth it if we were certain the quality had improved, but it seems that’s not necessarily guaranteed.

Coding standards attempt to simplify creation of C++ by offering rules of thumb. Yet we know rules-of-thumb have exceptions. Undefined behavior can be eliminated from a program, but we know programs without undefined behavior may be incorrect, or programs with undefined behavior (technically) may pass assurance. Simple prophylactic measures are **not *sufficient*** conditions for a system to be considered reliable and safe.

MISRA C++, Autosar 14, High Integrity C++, and C++ Core Guidelines are all important pieces of a reliable assurance regime, and should be pursued and refined in parallel as worthy efforts. However more than being insufficient for system level reliability, they define de facto dialects. Our goal is to help C++ develop in a manner in which **modern** standard C++ doesn’t require special rules and exceptions to develop reliable systems.

Existing Organization

[Safety] Critical Reliability group spontaneously organized at CppCon 2019 by presenters and attendees, and has been running regular teleconferences since then. Papers being discussed are being collected and archived as well as keeping regular minutes. There have been 4 telecons as of this writing, with more scheduled. The average attendance has been consistently over 20 attendees. There are over 50 members subscribed to the mailing list, representing over a dozen current and active players in autonomous vehicles, automotive, modern AI, and computer hardware -- including some of the largest tech companies in the world.

Industry Scope

Major players or interested parties in current or future attendance (may) include:

Autonomous Vehicles

- Waymo
- Cruise
- Argo
- Uber
- Lyft
- Aurora Innovation
- Zoox
- Mobileye
- Nuro
- Aptiv
- Apex.ai
- Bosch
- Daimler
- Voyage
- Auto-X
- Torc
- Ghost

Automotive

- Tesla
- Toyota
- Honda
- GM
- Ford
- Daimler
- BMW
- Volkswagen
- Hyundai
- Fiat Chrysler
- Volvo

Aerospace

- NASA
- SpaceX
- Virgin Galactic
- Blue Origin
- Sierra Nevada Corporation
- FlightSafety International
- Boeing
- Airbus
- Lockheed Martin

- Northrop Grumman
- Bombardier
- Embraer
- Dassault
- BAE Systems
- Gulfstream

Medical Instruments

- Intuitive Surgical
- Auris Surgical
- Stryker
- Accuray
- Mazor Robotics
- Corindus Vascular Robotics
- Stereotaxis
- Zimmer Biomet
- TransEnterix
- Titan Medical
- Medrobotics
- Hansen Medical
- Medtronic
- Verb Surgical
- Canon Medical
- Toshiba Medical

Robotics

- Boston Dynamics
- Universal Robots
- Miso Robotics
- Stanley Robotics

Drones

- AeroVironment
- DJI
- Skydio
- Shield.ai
- Staaker
- Zero Zero Robotics
- Zipline

Military

- General Atomics

- General Dynamics
- Raytheon

General

- Intel
- Nvidia
- Google
- Apple
- Amazon
- Microsoft
- IBM
- Oracle