ФЕДЕРАЛЬНОЕ АГЕНТСТВО

ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ И МЕТРОЛОГИИ

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСТ Р 72396— 2025/ IEC/TS 61508-3-2:2024

ФУНКЦИОНАЛЬНАЯ БЕЗОПАСНОСТЬ СИСТЕМ ЭЛЕКТРИЧЕСКИХ, ЭЛЕКТРОННЫХ, ПРОГРАММИРУЕМЫХ ЭЛЕКТРОННЫХ, СВЯЗАННЫХ С БЕЗОПАСНОСТЬЮ

Часть 3-2

Требования и руководство по применению математических и логических методов для установления точных свойств программного обеспечения и их документирование

(IEC/TS 61508-3-2:2024, IDT)

Издание официальное

Москва
Российский институт стандартизации
2025

gost.e-ecolog.ru

Предисловие

1 ПОДГОТОВЛЕН Обществом с ограниченной ответственностью «ЭОС Тех» (ООО «ЭОС Тех») и Федеральным государственным бюджетным учреждением «Российский институт стандартизации» (ФГБУ «Институт стандартизации») на основе собственного перевода на русский язык англоязычной версии стандарта, указанного в пункте 4

2 ВНЕСЕН Техническим комитетом по стандартизации ТК 58 «Функциональная безопасность»

3 УТВЕРЖДЕН И ВВЕДЕН В ДЕЙСТВИЕ Приказом Федерального агентства по техническому регулированию и метрологии от 20 ноября 2025 г. № 1442-ст

4 Настоящий стандарт идентичен международному документу IEC/TS 61508-3-2:2024 «Функциональная безопасность систем электрических, электронных, программируемых электронных, связанных с безопасностью. Часть 3-2. Требования и руководство по применению математических и логических методов для установления точных свойств программного обеспечения и их документирование» (IEC/TS 61508-3-2:2024 «Functional safety of electrical/electronic/programmable electronic safety-related systems — Part 3-2: Requirements and guidance in the use of mathematical and logical techniques for establishing exact properties of software and its documentation», IDT).

При применении настоящего стандарта рекомендуется использовать вместо ссылочных международных стандартов соответствующие им межгосударственные и национальные стандарты, сведения о которых приведены в дополнительном приложении ДА

5 ВВЕДЕН ВПЕРВЫЕ

*Правила применения настоящего стандарта установлены в статье 26 Федерального закона от 29 июня 2015 г. № 162-ФЗ «О стандартизации в Российской Федерации». Информация об изменениях к настоящему стандарту публикуется в ежегодном (по состоянию на 1 января текущего года) информационном указателе «Национальные стандарты», а официальный текст изменений и поправок — в ежемесячном информационном указателе «Национальные стандарты». В случае пересмотра (замены) или отмены настоящего стандарта соответствующее уведомление будет опубликовано в ближайшем выпуске ежемесячного информационного указателя «Национальные стандарты». Соответствующая информация, уведомление и тексты размещаются также в информационной системе общего пользования — на официальном сайте Федерального агентства по техническому регулированию и метрологии в сети Интернет (*gost.e-ecolog.ru

© IEC, 2024
© Оформление. ФГБУ «Институт стандартизации», 2025

Настоящий стандарт не может быть полностью или частично воспроизведен, тиражирован и распространен в качестве официального издания без разрешения Федерального агентства по техническому регулированию и метрологии

Содержание

1 Область применения 1
2 Нормативные ссылки 1
3 Термины, определения и сокращения 2
3.1 Термины и определения 2
3.2 Сокращения 8
4 Соответствие настоящему стандарту 8
5 Формальная спецификация требований безопасности 9
6 Формальная архитектура программного обеспечения/спецификация проекта 9
7 Языки программирования высокого уровня: выбор ESCL 10
8 Компиляция в объектный код 10
9 Ошибки и исключения времени выполнения 10
10 Применимые методы 10
Приложение А (обязательное) Применимые математические и логические методы 12
Приложение В (справочное) Конкретные математические и логические методы 14
Приложение С (справочное) Свойства, гарантируемые применением определенных методов M&LT . . . 17
Приложение D (справочное) Детализация программного обеспечения от спецификации безопасности к написанию кода 19
Приложение ДА (справочное) Сведения о соответствии ссылочных международных стандартов межгосударственным и национальным стандартам 20
Библиография 21

Введение

Стандарты МЭК 61508-1:2010 — МЭК 61508-7:2010 образуют серию базовых стандартов по функциональной безопасности электрических, электронных и программируемых электронных систем (систем Э/Э/ПЭ). Она охватывает жизненный цикл этих систем. Большая часть функционала таких систем, как правило, реализуется программно. МЭК 61508-3:2010 устанавливает требования к программному обеспечению.

Обязательное приложение А и справочные приложения В и С МЭК 61508-3:2010 содержат таблицы, в которых перечислены различные методы и меры, а также даны некоторые рекомендации по выбору этих методов для различных уровней полноты безопасности (УПБ). В нем перечислены общие категории и даны различные уровни рекомендаций для них, такие как «не рекомендуется», «рекомендуется» или «настоятельно рекомендуется», а также более конкретные методы для различных этапов разработки программного обеспечения.

Эти методы и меры представляют собой сочетание общих и конкретных. Термин «формальные методы», используемый в МЭК 61508-3, связан с использованием математических и логических подходов для спецификации, оценки, проектирования и верификации программного обеспечения. Такие методы используются для определения требований, оценки проекта, проверки исходного и объектного кода, а также для создания тестовых наборов и мониторинга корректной работы программного обеспечения времени выполнения. В настоящем стандарте их называют «математические и логические методы» (M&LT) или просто «методы M&LT». Некоторые из методов M&LT, представленных в настоящем стандарте, не ограничиваются разработкой программного обеспечения и в равной степени применимы к другим технологиям проектирования на основе цифровых систем. Ни один из методов M&LT не ограничивается областью программных систем, связанных с безопасностью, хотя в настоящем стандарте рассматриваются методы M&LT, только для приложений, связанных с безопасностью.

Использование методов, рекомендуемых в приложениях А, В и С МЭК 61508-3:2010, не исключает, например, подверженности программного обеспечения сбоям во время выполнения. Современные технологии разработки программного обеспечения позволяют исключить различные типы сбоев во время выполнения программного обеспечения за счет его тщательной разработки. Используя описанные в настоящем стандарте методы, можно гарантировать отсутствие многих типов сбоев во время выполнения программного обеспечения.

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФУНКЦИОНАЛЬНАЯ БЕЗОПАСНОСТЬ СИСТЕМ ЭЛЕКТРИЧЕСКИХ, ЭЛЕКТРОННЫХ, ПРОГРАММИРУЕМЫХ ЭЛЕКТРОННЫХ, СВЯЗАННЫХ С БЕЗОПАСНОСТЬЮ

Часть 3-2

Требования и руководство по применению математических и логических методов для установления точных свойств программного обеспечения и их документирование

Functional safety of electrical, electronic, programmable electronic safety-related systems. Part 3-2. Requirements and guidance in the use of mathematical and logical techniques for establishing exact properties of software and its documentation

Дата введения — 2026—07—01

1 Область применения

Настоящий стандарт является частью серии стандартов МЭК 61508 и охватывает общее обеспечение надежности программного обеспечения, используемого в критически важных операционных технологиях (ОТ) и выполняемого на аппаратных средствах, которые входят в состав приложения ОТ. В настоящем стандарте рассматривается только программное обеспечение, связанное с безопасностью, которое разрабатывается в соответствии со стандартом по функциональной безопасности программного обеспечения Э/Э/ПЭ МЭК 61508-3; в частности, рассматривается разработка программного обеспечения, которая осуществляется в соответствии с формальной спецификацией требований к безопасности. Успешное использование некоторых или всех свойств достоверности, определенных в настоящем стандарте, повышает уверенность в том, что конкретная часть программного обеспечения, связанного с безопасностью, соответствует требованиям УПБ функции безопасности, которую она частично или полностью реализует, и тем самым повышает стойкость к систематическим отказам программного обеспечения.

2 Нормативные ссылки

В настоящем стандарте использованы нормативные ссылки на следующие стандарты [для датированных ссылок применяют только указанное издание ссылочного стандарта, для недатированных — последнее издание (включая все изменения)]:

IEC 61508-3:2010, Functional safety of electrical/electronic/programmable electronic safety-related systems — Part 3: Software requirements (Функциональная безопасность систем электрических, электронных, программируемых электронных, связанных с безопасностью. Часть 3. Требования к программному обеспечению)

IEC 61508-4:2010, Functional safety of electrical/electronic/programmable electronic safety-related systems — Part 4: Definitions and abbreviations (Функциональная безопасность систем электрических, электронных, программируемых электронных, связанных с безопасностью. Часть 4. Определения и сокращения)

Издание официальное

3 Термины, определения и сокращения

3.1 Термины и определения

В настоящем стандарте применены следующие термины с соответствующими определениями.

ИСО и МЭК поддерживают терминологические базы данных, используемые в целях стандартизации, по следующим адресам:

- платформа онлайн-просмотра ИСО доступная по адресу: gost.e-ecolog.ru ;
- Электропедия МЭК доступная по адресу: gost.e-ecolog.ru .

3.1.1 **абстрактная интерпретация** (abstract interpretation) <компьютерной программы>: Статический анализ программы, использующий абстрактные состояния программы или машины, который обеспечивает правдоподобные результаты для заданного свойства, т. е. который никогда не сообщает о том, что свойство выполняется, если оно не выполняется.

3.1.2 **свойство достоверности** (assurance point) <в разработке программного обеспечения с использованием M&LT>: Триада, которая состоит из ПО и/или документации (S1), другого ПО и/или документации (S2) и такого их общего свойства P, что P(S1.S2) может быть формально математически доказано.

Примечание 1 — Несмотря на то, что математическое доказательство для свойства достоверности является формально возможным, оно может быть слишком сложным или требовать слишком много ресурсов для исчерпывающего выполнения и надежной проверки (например, оценщиком).

3.1.3 **инструмент автоматического доказательства теорем** (automated prover, automated theorem prover): Автоматизированная компьютерная программа доказательства теорем, которая выполняет формальный логический вывод на множестве предложений, представленных на формальном языке.

3.1.4 **автоматическое доказательство теорем** (automated proving): Использование инструмента автоматического доказательства теорем.

3.1.5 **характеристическая функция** (characteristic function) <множества или отношения>: Функция, заданная на области значений аргумента и принимающая значения {0,1}, такая, что ее значение равно единице, когда ее аргумент принадлежит некоторому множеству или отношению, и нулю, когда ее аргумент не принадлежит этому множеству или отношению.

3.1.6 **автоматический генератор кода** (code generator, automatic code generator): Программное обеспечение, которое преобразует программу на языке высокого уровня или спецификацию в обычную программу на языке третьего или четвертого поколения.

3.1.7 **стандарт кодирования, подмножество языка программирования** (coding standard, programming-language subsetting) <в методах M&LT>: Ограничения на конструкции, с помощью которых может быть написана программа на языке программирования высокого уровня.

Примечание 1 — Целью стандарта кодирования, ограничивающего разрешенные для использования конструкции языка программирования, является обеспечение однозначной семантики программы, написанной в соответствии с этим стандартом.

Примечание 2 — Типичный стандарт кодирования (в единственном числе эта фраза встречается редко) обеспечивает устранение известных причин ненадежного поведения в программном коде (например, запрещение переменных-указателей и отказ от использования неопределенных или зависящих от компилятора языковых возможностей). Стандарты кодирования для программ, написанных на языке без строгой типизации данных, вполне могут обеспечивать явную проверку ожидаемого или указанного диапазона входных или выходных данных на входе или выходе.

Примечание 3 — Термин «стандарты кодирования» в общем случае относится к дополнительным свойствам кода, а не к подмножеству языка программирования.

3.1.8 **компиляция** (compilation): Операция трансляции исходного кода уровня исполнения (ESCL) в объектный код (OC).

Примечание 1 — В определении явно упоминается ESCL — концепция, которая используется в настоящем стандарте, но не в общем случае, где используются компиляторы и практикуется компиляция.

3.1.9 **полнота** (completeness) <формального логического языка относительно заданного семантического свойства>: Качество формального логического языка с формальной семантикой, которое

выполняется для заданного семантического свойства, если можно показать наличие этого свойства у каждого предложения языка посредством логического вывода.

Примечание 1 — Алгоритм является полным относительно свойства, если он всегда доказывает свойство, которое выполняется.

3.1.10 **состав** (compositional) <формальной семантики>: Использование в качестве входных данных только предложения формального языка (а не какого-либо указания на контекст — например, ссылок на индексалы) и создание представления исключительно с помощью дерева синтаксического анализа предложения.

3.1.11 **вычислимый, рекурсивный** (computable, recursive) <согласно теории вычислимых функций>: Вычислимый по Тьюрингу.

Примечание 1 — Термин «рекурсивный» используется здесь в том же смысле, что в теории вычислимости по Тьюрингу и теории рекурсивных функций [1], [2].

Примечание 2 — Помимо понятия «вычислимости по Тьюрингу» в математической литературе существуют и другие понятия «вычислимости». Известно, что одни из них эквивалентны вычислимости по Тьюрингу, но для других этот вопрос остается открытым. Таким образом, указанное определение устраняет неоднозначность использования термина «вычислимый». Аналогично, в информатике и системной инженерии термин «рекурсивный» имеет различные значения; указанное определение устраняет неоднозначность его использования.

Примечание 3 — Термин «разрешимый» часто применяется к свойствам, а термин «вычислимый» — к функциям. Свойство разрешимо тогда и только тогда, когда его характеристическая функция вычислима.

Примечание 4 — Понятие «вычислимость по Тьюрингу» обычно определяется в учебниках по теории рекурсии на нескольких десятках страниц, часто с использованием большого количества вспомогательных понятий [1], [2]. Не существует краткого определения, которое можно было бы использовать здесь.

Примечание 5 —В логике и информатике существуют и другие понятия вычислимости, которые не сводятся к вычислимости по Тьюрингу, или способ такого сведения неизвестен.

3.1.12 **согласованность** (consistency): Свойство набора предложений формального языка, которое заключается в том, что они не противоречат друг другу.

3.1.13 **противоречивость** (contradictory): Свойство набора предложений формального языка, которое заключается в том, что их представления являются взаимоисключающими, т. е. в одной и той же структуре они не могут одновременно выполняться все или быть реализованы.

3.1.14 **разрешимый** (decidable): Имеющий характеристическую функцию, вычислимую по Тьюрингу.

3.1.15

элемент (element): Часть подсистемы, которая включает в себя отдельный компонент или любую группу компонентов, выполняющих одну или несколько функций безопасности элемента.

Примечание 1 — Элемент может включать в себя аппаратные средства и/или программное обеспечение.

Примечание 2 — Типичными элементами являются датчик, программируемый контроллер и исполнительный элемент.

[МЭК 61508-4:2010, статья 3.4.5]

3.1.16

функция безопасности элемента (element safety function): Часть функции безопасности, которая реализована элементом.

[МЭК 61508-4:2010, статья 3.5.3]

3.1.17 **уровень исполняемого исходного кода,** ESCL (executable source code level, ESCL): Исходный код, в котором полностью описано точное поведение программного обеспечения.

3.1.18 **формальный вывод** (formal inference) <в логике>: Вывод предложения из посылок с использованием явных формальных правил логического вывода.

3.1.19 **формальный язык** (formal language) <в M&LT>: Язык с определенным синтаксисом, который без исключений поддается синтаксическому анализу с помощью цифровых вычислений, а также позволяет вычислять, образует ли данная последовательность символов предложение языка.

Примечание 1 — В настоящем стандарте для обозначения формального языка используется термин «предложение», но во многих формальных языках уместнее использовать другие термины. Если формальный

язык состоит только из строк символов (как в теории формальных языков, теории автоматов и формальной логике), правильно построенный элемент называется предложением, за исключением формальной логики, в которой он называется правильно построенной формулой. В языках программирования правильно построенный элемент называется корректной программой, в языках спецификаций — корректной спецификацией, а в диаграммных языках — корректной диаграммой.

Примечание 2 — Это понятие формального языка используется в логике, лингвистике, математике, теории формального языка, теории автоматов и теории компиляции. Формальные языки, такие как языки спецификаций (инженерии и программного обеспечения), также часто имеют предопределенную формальную семантику (например, Z, TLA), и использование термина в таких контекстах программной инженерии часто неявно включает формальную семантику.

Примечание 3 — В математике и теории автоматов термин «формальный язык» обозначает только набор строк символов из определенного набора символов без каких-либо ограничений на способ формирования этих строк. Тем не менее, все формальные языки, которые используются в M&LT, удовлетворяют условиям, указанным в определении.

Примечание 4 — В формальной логике термин «язык» (без приставки «формальный») обычно относится к набору нелогических символов. Набор логических символов формального логического языка обычно считается определенным несмотря на то, что он различается в логике предикатов первого порядка и логике более высокого порядка.

3.1.20 **формальная логика** (formal logic): Пропозициональная логика, логика предикатов, логика высшего порядка, комбинаторная логика, модальная логика, неклассическая логика или другая математическая языковая структура, основанная на формальном языке, в котором используются понятия формального вывода, согласованности и противоречия.

3.1.21 **формальное доказательство** (formal proof): <предложения из заданного множества предложений в формальной логике>: Описанный в формальной логике формальный вывод предложения, в котором в качестве посылок используются предложения из заданного множества предложений.

3.1.22 **формальная семантика** (formal semantics) <языка>: Точное математическое представление (предполагаемого) значения предложений формального языка, такое, что любые два одинаковые по значению предложения имеют одинаковое представление, а любые два различные по значению предложения — разные представления.

Примечание 1 — Формальная семантика частично или полностью включает в себя структуру семантики формальной логики.

Примечание 2 — В большинстве формальных семантик два предложения с одинаковым значением могут иметь разные символьные представления, и может возникнуть необходимость дополнительно обрабатывать эти символьные представления (например, с помощью формального «агента упрощения», такого как инструмент автоматического доказательства теорем или программы проверки), чтобы вынести суждение об идентичности представлений, а именно символьных представлений, сокращенных редуцирующим агентом. В этом случае представление состоит из символьного представления, за которым следует его упрощение, если требуется определить, имеют ли два предложения одинаковое значение.

Примечание 3 — Идентичность значения может не являться вычислимой, но должна быть как минимум полувычислимой.

3.1.23 **формальная семантика** (formal semantics) <предложения>: Представление предложения в формальной семантике (языка).

3.1.24 **формальная спецификация** (formal specification): Функциональная спецификация, которая написана на формальном языке с формальной семантикой.

3.1.25 **формальная верификация** (formal verification) <в M&LT>: Математически строгое доказательство или метод, который гарантирует выполнение требуемых свойств.

Примечание 1 — Некоторые методы формальной верификации работают с определенной доверительной вероятностью, меньшей, чем достоверность.

Примечание 2 — Примерами методов формальной верификации являются доказательство теорем, верификация моделей и абстрактная интерпретация.

3.1.26 **удовлетворять** (fulfil): <о программном объекте применительно к спецификации (объекта) или спецификации (субъекта) к спецификации (объекта)>: Выполнять так, чтобы было возможно формально доказать представление спецификации (объекта) из представления программного объекта/спецификации (субъекта) в формальной логике.

Примечание 1 — Здесь термины «спецификация (субъекта)» и «спецификация (объекта)» указывают на то, какая из двух спецификаций является субъектом термина «удовлетворять».

3.1.27 **промежуточная исполняемая спецификация,** IES (intermediate executable specification, IES): Исходный код уровня языка программирования, размещаемый над уровнем исполняемого исходного кода, когда существуют несколько уровней исходного кода согласно 6.1.

3.1.28 **уровень** (level) <языка программирования>: Подъязык языка программирования высокого уровня, связанный синтаксическим преобразованием с другими уровнями — другими подъязыками — языка программирования.

Пример 1 — Язык программирования Java имеет так называемые «исходный код» и байт-код. Байт-код Java генерируется компилятором Java из исходного кода Java. Исходный код Java и байт-код Java — это уровни; уровень исходного кода Java находится «выше» уровня байт-кода Java.

Пример 2 — Широко распространенные языки программирования (например, C и C++) имеют «стандартные библиотеки», которые состоят из более или менее сложных функций, которые вызываются в исходном коде с использованием одного идентификатора. Исходный код C, в котором используются идентификаторы библиотечных функций, является уровнем (например C.1). Предположим, что C.1 является подмножеством C, в котором используются только термины и операторы с явной однозначной семантикой. Код C.1, в котором библиотечные функции заменены встроенным кодом, не содержащим идентификаторы библиотечных функций или содержащим только некоторые из них, также образует уровень (например C.2). Уровень C1 находится «выше» C.2, поскольку исходный код C.1 преобразуется в исходный код C.2 с тем же значением путем замены библиотечных функций, не входящих в C.2, встроенным кодом, который выполняет эти функции.

Примечание 1 — Считается, что уровень языка программирования L.1 находится «выше» уровня L.2, если L.1 и L.2 строго синтаксически определены, и можно преобразовать программу уровня L.1 в программу уровня L.2 с той же семантикой при помощи общей технологии языка программирования.

3.1.29 **средство верификации моделей** (model checker): Программное обеспечение, которое выполняет верификацию моделей в программном коде.

3.1.30 **средство верификации моделей** (model checker) <набора состояний программы, машины или формальной модели>: Перечисление набора состояний и проверка выполнения заданного свойства состояния для каждого состояния в наборе.

Примечание 1 — Верификация моделей часто выполняется с использованием абстрактных состояний из практических соображений, связанных с комбинаторной сложностью. Возможности верификации моделей в значительной степени зависят от метода абстракции, который стремится охватить многочисленные фактические состояния программы с помощью минимально возможного количества абстрактных состояний для эффективной проверки свойства.

3.1.31 **объектный код** (object code): Исполнимый код, установленный и исполняемый непосредственно на цифровом вычислительном оборудовании.

3.1.32 **средство проверки доказательств, автоматизированное средство проверки доказательств** (proof checker, automated proof checker): Программное обеспечение, которое принимает машиночитаемое формальное доказательство на формальном логическом языке и возвращает значение, которое указывает, является ли доказательство корректным.

3.1.33 **уточнение** (refinement): Преобразование промежуточной исполняемой спецификации в промежуточную исполняемую спецификацию или исполнимый исходный код другого уровня, при котором сохраняется формальная семантика исходной спецификации, но с более подробной информацией о выполнении.

Пример 1 — Преобразование исходного кода Java в байт-код Java является уточнением.

Пример 2 — Преобразование C.1 в C.2 в примере 2 из 3.1.28 является уточнением.

Пример 3 — Преобразование описания конечного автомата, например, в код C, реализующий этот конечный автомат, является уточнением.

3.1.34 **относительная полнота** (relative completeness) <формальной спецификации>: Полнота, при которой семантика предложений частично задана внешними ограничениями.

Примечание 1 — Формальная семантика является составной, если она выводит представления (семантики) исключительно из предложений (см. 3.1.17) без использования контекста только на основе дерева синтаксического разбора предложения. Некоторые полезные формальные семантики не являются составными; для определения значения утверждения требуется контекст. Такое использование контекста является примером семантики, частично заданной внешними ограничениями.

3.1.35 **представление** (rendering): <предложения в формальной семантике>: Форма, посредством которой предложение определяется как идентичное или отличное по смыслу от другого предложения.

3.1.36 **строгая** (rigorous) <спецификация>: Недвусмысленная, явно охватывающая все возможные варианты поведения и поддающаяся тщательной проверке на корректность и некорректность.

3.1.37 **строгая** (rigorous) <формальная верификация>: Выполняемая с использованием формального вывода и поддающаяся тщательной проверке на корректность и некорректность.

3.1.38 **строгий** (rigorous) <процесс>: Выполняемый с вниманием к корректности каждого этапа процесса.

3.1.39 **верификация времени исполнения** (runtime verification): Метод, при котором в программный код добавляются переменные мониторинга, частично описывающие состояние программы и с некоторой степенью детализации указывающие на корректность хода вычисления, а также генерирующий уведомление или исключение, если вычисление выполняется некорректно.

3.1.40

функция безопасности (safety function): функция, которая реализуется Э/Э/ПЭ системой, связанной с безопасностью, или другими мерами по снижению риска, и предназначена для достижения или поддержания безопасного состояния УО по отношению к конкретному опасному событию.

Примеры функций безопасности:

- функции, которые должны четко выполняться для снижения влияния опасной ситуации (например, выключение двигателя);

- функции, которые осуществляют превентивные действия, не допускающие возникновения опасных ситуаций (например, предотвращение запуска двигателя).

[МЭК 61508-4:2010, статья 3.5.1]

3.1.41 **анализ планируемости** (schedulability analysis): Анализ, который определяет, можно ли запланировать и выполнить в срок все задачи при использовании заданного алгоритма планирования.

3.1.42 **полувычислимый, полуразрешимый** (semi-computable, semi-decidable): Рекурсивно перечислимый в смысле, который используется в теории вычислимости Тьюринга и теории рекурсивных функций.

Примечание 1 — См. [1], [2].

3.1.43 **предложение** (sentence) <в теории формальных языков>: Строка символов, которая является правильно построенным элементом языка.

Примечание 1 — Если язык рассматривается только как множество строк символов, предложение языка является лишь элементом этого множества.

Примечание 2 — Если формальный язык является формальным языком спецификации, предложение в этом смысле часто называется утверждением, условием или модулем (языков спецификации, в которых используются модули). Если формальный язык является логическим, предложение в этом смысле называется корректно построенной формулой. Если формальный язык является языком программирования, предложение в этом смысле представляет собой (корректную) программу, что не является общепринятым в теории языков программирования.

Примечание 3 — В информатике существуют диаграммные языки (например, для некоторых форм автоматов, определения иерархий классов данных или иллюстрации взаимодействия между процессами), которые также можно считать формой формального языка, несмотря на то, что они состоят не из строк символов, а из определенных пространственных компоновок символов (часто двумерных). Называть их предложениями представляется неуместным; определение «правильно построенный» в равной степени применимо к таким языкам компоновки.

3.1.44 **правдоподобный** (sound): Демонстрирует свойство обоснованности.

3.1.45 **обоснованность** (soundness) <логикой формальной семантики>: Свойство логики и формальной семантики: если каждый раз, когда логически доказывается, что предложение языка истинно, это предложение логически корректно в формальной семантике.

3.1.46 **обоснованность** (soundness) <статическим анализатором заданной формальной семантики>: Свойство статического анализатора и формальной семантики: если каждый раз, когда анализатор доказывает, что утверждение S истинно, то S логически корректно в формальной семантике.

Примечание 1 — Если предположения A1, A2, …, An используются анализатором для обоснования S, S должно быть корректным при этих же предположениях, т. е. предложение (A1 ∧ A2 ∧…∧ An → S) логически выполнимо в формальной семантике.

Примечание 2 — Это определение применимо к любому инструменту статической формальной верификации, в том числе к средствам доказательства теорем и верификации моделей.

Примечание 3 —Поскольку используемая формальная семантика часто бывает неразрешимой, не существует алгоритма, который гарантированно проверяет логическую обоснованность S в рамках этой семантики. Неудачный результат практической проверки чаще всего обуславливается сложностью, а не неразрешимостью базовой семантики; как правило, неразрешимость не является препятствием для таких проверок.

3.1.47 **исходный код** (source code): Программа, написанная на формальном языке программирования более высокого уровня.

3.1.48 **статический анализ** (static analysis): Анализ свойств программы или спецификации на формальном языке посредством анализа ее текста, а не путем полного или частичного выполнения.

3.1.49

стойкость к систематическим отказам (systematic capability): Мера уверенности (выраженная в диапазоне ССО 1 — ССО 4) в том, что систематическая полнота безопасности элемента соответствует требованиям заданного значения УПБ для определенной функции безопасности элемента, если этот элемент применен в соответствии с указаниями, определенными для этого элемента в соответствующем руководстве по безопасности.

[МЭК 61508-4:2010, статья 3.5.9]

3.1.50 **средство доказательства теорем** (theorem prover): Программное обеспечение, первичные входные данные которого состоят из предполагаемых теорем и набора предпосылок в формальной логике, и которое пытается определять наличие формального доказательства предполагаемых теорем в логике.

Примечание 1 — Средство доказательства теорем предназначено для определенной (и заданной) формальной логики.

Примечание 2 — Средство доказательства теорем может обрабатывать заданные входные данные как успешно (находить доказательство), так и неуспешно (не находить доказательство). Последнее не означает, что доказательства в логике не существует. Средство доказательства теорем, которое не обнаруживает доказательство только при его отсутствии в логике, называется «процедурой принятия решения».

Примечание 3 — Некоторые логики исключают возможность существования вычислительной машины, способной всегда давать правильный ответ «да/нет», поскольку средства доказательства теорем часто основаны на заведомо неполных алгоритмах.

Примечание 4 — Существует множество форм средств доказательства теорем — от неинтерактивных «средств проверки доказательств», которые могут требовать от пользователя-человека трудоемкой предварительной работы с логикой, до «систем автоматического доказательства теорем» или «интерактивных средств доказательства теорем», которые демонстрируют различные уровни взаимодействия с пользователем, «направляющим» доказательство к желаемой цели.

3.1.51 **транскрипция** (transcription) <предложения формального языка в формальной семантике>: Письменная форма, в которую преобразуется предложение в формальной семантике до определения представления с использованием какого-либо сокращения.

3.1.52 **недвусмысленность** (unambiguous) <предложения, спецификации>: Наличие уникального представления в формальной семантике вплоть до логической или поведенческой эквивалентности.

Примечание 1 — Это определение формально не удовлетворяет критерию подстановочности для терминов: «X является недвусмысленным» представляется как «X является имеет уникальное …..». Однако читатели способны легко разрешить фразу «является имеет».

3.1.53 **неразрешимость** (undecidability): Категория сложности проблемы принятия решений, которая подразумевает отсутствие автоматического алгоритма, всегда доказывающего истинность решения, если оно выполняется (полнота), и никогда не доказывающего истинность решения, если оно не выполняется (обоснованность).

3.1.54 **неразрешимый** (undecidable): Обладающий неразрешимостью.

Примечание 1 — Из неразрешимости формы формального рассуждения следует, что существует формальная логика, которая выполняет это рассуждение (или, как принято считать, способна на это), и что не существует алгоритма определения доказуемости заданной правильно построенной формулы логики. Несмотря на то, что иногда важно знать о неразрешимости рассматриваемой проблемы, практические трудности, связанные с использованием средств доказательства теорем на конкретных проблемах, обычно значительно важнее любых

проблем, которые могут возникать из-за неразрешимости базовой логики. Для многих неразрешимых проблем практические решения обеспечиваются обоснованными алгоритмами.

3.1.55 **правильно построенный** (well-formed) <в формальном языке>: Корректный элемент формального языка, построенный в соответствии с правилами определения этого языка.

Примечание 1 — Если формальный язык состоит из простых строк символов, которые используются в регулярном языке или языке без контекста (например, языке формальной логики), правильно построенный элемент языка часто называется предложением. Если формальный язык является языком программирования, правильно построенный элемент называется корректной программой. Если формальный язык является языком формальной спецификации, правильно построенный элемент называется спецификацией.

Примечание 2 — В информатике существуют диаграммные языки (предназначенные, например, для некоторых форм автоматов, определения иерархий классов данных или иллюстрации взаимодействия между процессами), которые также можно считать формами формального языка, даже если они не состоят из строк символов. Структуры, построенные в соответствии с правилами формирования такого диаграммного языка, являются правильно построенными согласно данному определению.

3.1.56 **анализ времени выполнения в наихудшем случае;** анализ WCET (worst-case execution time analysis; WCET analysis): Анализ, который определяет надежную верхнюю границу длительности выполнения машинных инструкций конкретной задачи на конкретном оборудовании.

Примечание 1 — В анализе WCET предполагается, что задача выполняется непрерывно. Влияние вытеснения и блокировки задачи учитывается при анализе планируемости.

Примечание 2 — В WCET учитывается влияние помех на длительность непрерывного выполнения задачи (например, из-за конфликтов при доступе к общим ресурсам различных ядер многоядерных систем). Худшее время непрерывного выполнения задачи без учета таких помех обычно называется внутренним WCET.

3.1.57 **анализ времени ответа в наихудшем случае;** анализ WCRT (worst-case response time analysis; WCRT analysis): Анализ, который определяет надежную верхнюю границу максимального времени завершения задачи с момента ее запуска.

Примечание 1 — Момент запуска — время, когда вызов задачи становится готовым к выполнению.

Примечание 2 — Время ответа задачи в наихудшем случае включает в себя время ее выполнения в худшем случае, время, когда она вытесняется более приоритетными задачами, и время, когда она блокирована примитивами синхронизации.

3.2 Сокращения

ESCL — уровень исполняемого исходного кода (Executable Source Code Level);

FSRS — формальная спецификация требований безопасности (Formal Safety Requirements Specification);

IES — промежуточная исполняемая спецификация (Intermediate Executable Specification);

M&LT — математические и логические методы в системной инженерии (Mathematical and logical techniques in systems engineering);

OC — объектный код (Object Code);

SWA/DS — архитектура/спецификация проекта программного обеспечения (Software Architecture/Design Specification);

WCET — время выполнения в наихудшем случае (Worst-Case Execution Time).

4 Соответствие настоящему стандарту

Соответствие настоящему стандарту подтверждается демонстрацией того, что все соответствующие требования удовлетворяют всем заданным обязательным критериям и, следовательно, для каждого пункта достигнуты все цели. Такая демонстрация должна включать в себя обоснование выбора соответствующих требований согласно заявленному применению методов M&LT на протяжении всего жизненного цикла программного обеспечения.

Примечание — Для соответствия настоящему стандарту требуется удовлетворять не каждому пункту, а только пунктам, относящимся к аспектам жизненного цикла, к которым применяются методы M&LT.

5 Формальная спецификация требований безопасности

5.1 Для компонентов программного обеспечения должна быть написана однозначная строгая формальная спецификация требований безопасности (FSRS).

Примечание 1 — FSRS, которая определяется в 5.1, подходит для выполнения требований МЭК 61508-3:2010 (подраздел 7.2), т. е. является спецификацией требований безопасности программного обеспечения в соответствии с МЭК 61508-3:2010 (подраздел 7.2).

Примечание 2 — Согласно МЭК 61508-3:2010 (подпункт 7.4.2.8), если программное обеспечение реализует некоторую функцию безопасности, все программное обеспечение связано с безопасностью, а соответствующая спецификация требований к программному обеспечению включает в себя спецификацию требований безопасности.

Примечание 3 — Существуют три основных причины, по которым спецификация требований безопасности может в конечном счете оказаться неадекватной. Во-первых, спецификация требований безопасности может не включать в себя некоторое связанное с безопасностью поведение, которое должно быть предусмотрено в ней (т. е. она не исключает явно опасное поведение системы). Во-вторых, из-за неоднозначности спецификация требований безопасности может допускать опасное поведение, которое можно было бы исключить при однозначной спецификации. В-третьих, в спецификации требований безопасности может использоваться формальный язык, который не фиксирует некоторые различия между неопасным и опасным поведением и исключает некоторое неопасное поведение, которое на самом деле является приемлемым, что излишне ограничивает разработанную систему. Адекватность спецификации требований безопасности программного обеспечения регулируется требованиями МЭК 61508-1:2010 (подраздел 7.10) и МЭК 61508-3:2010 (подраздел 7.2) и не определяется только требованиями раздела 4 настоящего стандарта.

Примечание 4 — Язык для FSRS не указан в настоящем стандарте; разработчик сам выбирает и утверждает его. Тем не менее требование к однозначности FSRS ограничивает возможные способы ее написания в соответствии с 5.2. Для обеспечения этих свойств и выполнения этих задач подходят формальные языки спецификаций и их формальная семантика (как и контролируемые естественные языки, которые являются формой формального языка спецификаций).

Примечание 5 — Применение формальной спецификации требований безопасности означает, что используется язык спецификации, подходящий для формального анализа с использованием автоматизированных или полуавтоматических методов. Согласно 5.1 сам автоматизированный анализ не требуется; допускается выполнение ручного анализа.

5.2 FSRS должна проверяться с использованием математических и/или логических методов:

a) на согласованность;

b) относительную полноту.

Примечание 1 — Проверка на относительную полноту должна гарантировать, что в FSRS учтены все сценарии, которые могут приводить к опасностям.

Примечание 2 — Проверка на согласованность и относительную полноту может выполняться независимо от того, выполняется ли требуемое формальное рассуждение в разрешимой логике или она неразрешима. Например, предикатная логика неразрешима; булева (пропозициональная) логика разрешима, однако в предикатной логике часто требуется проверять согласованность утверждений. Попытки таких проверок часто осуществляются при помощи инструмента доказательства теорем с использованием метода, известного как сколемизация. При сколемизации генерируется формула, с которой могут работать средства доказательства теорем в пропозициональной логике, часто называемые решателями SAT (решателями задач выполнимости булевых формул).

5.3 Методы и результаты, используемые для проверки согласованности и относительной полноты, должны быть оформлены документально.

6 Формальная архитектура программного обеспечения/ спецификация проекта

6.1 Архитектура программного обеспечения/спецификация проекта (SWA/DS) должна быть строгой.

6.2 Соответствие SWA/DS формальной спецификации требований безопасности (FSRS) является свойством достоверности. Должна выполняться строгая и корректная формальная верификация того, что SWA/DS соответствует FSRS.

Примечание 1 — В соответствии с требованиями МЭК 61508-3:2010 (подраздел 7.4) SWA/DS является как проектом архитектуры программного обеспечения, так и спецификацией проекта программной системы.

Примечание 2 — В соответствии с 6.2 автоматизированная формальная верификация не требуется. Допускается использование ручной формальной верификации.

7 Языки программирования более высокого уровня: выбор ESCL

7.1 При использовании языка программирования более высокого уровня существуют один или несколько уровней, на которых полностью описано точное поведение программного обеспечения. Один из этих уровней должен выбираться в качестве уровня исполняемого исходного кода (ESCL).

Примечание 1 — Например, если выбран язык Java, в качестве ESCL можно выбирать исходный код Java либо байт-код. Если используется среда спецификации конечного автомата, а исходный код на языке программирования С автоматически генерируется из спецификаций конечного автомата, в качестве ESCL можно выбирать спецификацию конечного автомата либо полученный исходный код на языке С.

Примечание 2 — Например, императивный язык программирования со строгой семантикой — это язык, на котором полностью описывается точное поведение программного обеспечения в смысле 7.1.

Примечание 3 — При наличии нескольких вариантов выбора ESCL можно разделять процесс формальной верификации на цепочку отдельных формальных верификаций между уровнями. Уровни перед последним (нижним) уровнем называются промежуточными исполняемыми спецификациями (IES), а последний (нижний) уровень — ESCL.

7.2 Соответствие кода в ESCL SWA/DS является свойством достоверности. Должна выполняться строгая и корректная формальная верификация того, что код в ESCL соответствует SWA/DS.

Примечание — В соответствии с 7.2 автоматизированная формальная верификация не является обязательной. Допускается выполнение ручной формальной верификации.

8 Компиляция в объектный код

Соответствие объектного кода ESCL является свойством достоверности. Должна выполняться строгая и корректная формальная верификация того, что объектный код соответствует ESCL.

Примечание 1 — Ручная формальная верификация соответствия ОС ESCL обычно невыполнима.

Примечание 2 — Требуемая строгая и корректная формальная верификация может не являться полностью формальным математическим доказательством. Обычно верификация состоит из нескольких этапов, и некоторые из них имеют формальный статус. Одним из таких этапов с формальным статусом является компиляция ESCL с использованием сертифицированного компилятора: сертификация компилятора обычно придает формальный статус утверждению о том, что скомпилированный код соответствует ESCL. Тем не менее, скомпилированный код не обязательно является конечным объектным кодом. Он может содержать вызовы библиотечных функций, которые находятся вне компилятора, и код, сгенерированный компилятором, в любом случае компонуется с функциями библиотеки. Обычно в пользу этого свойства достоверности в документации приводятся аргументы о том, что вызовы библиотечных функций соответствуют (части) ESCL, а редактор связей сохраняет семантику ESCL.

9 Ошибки и исключения времени выполнения

9.1 Необходимо создавать список типов избегаемых ошибок времени выполнения.

9.2 Необходимо проводить строгую и корректную формальную проверку отсутствия ошибок времени выполнения, перечисленных в 9.1.

9.3 Необходимо проводить строгую и корректную формальную верификацию достижения указанных состояний системы обработчиками исключений времени выполнения при возникновении исключений.

10 Применимые методы

10.1 Методы M&LT, которые выбраны для обоснования требований к программному обеспечению, его документации и необходимы для выполнения свойств достоверности, должны быть правильно определены, обоснованы, правдоподобны и соответствовать требуемой задаче.

Примечание 1 — Применимые методы и методы достижения определенных свойств программного обеспечения во время разработки приведены в приложении А (см. таблицу А.1).

Примечание 2 — В приложении В (см. таблицу В.1) представлены конкретные методы, которые можно использовать для реализации каждого более общего метода из таблицы А.1.

Примечание 3 — В приложении С (см. таблицу С.1) представлены свойства программного обеспечения, которые могут быть гарантированы при использовании каждого конкретного метода из таблицы В.1.

10.2 Обязательные атрибуты методов M&LT, выбранных согласно 10.1, должны быть документированы.

10.3 Методы M&LT, указанные в приложении А (см. таблицу А.1), должны использоваться независимо от требуемого SC программного обеспечения.

Примечание — Применение надлежащих методов на каждом этапе разработки обеспечивает полную прослеживаемость спецификаций функциональных требований и требований безопасности вплоть до уровня исходного кода.

Приложение А
(обязательное)

Применимые математические и логические методы

Таблица А.1 — Методы M&LT

	Методы/средства
1	Формальная спецификация требований безопасности (FSRS)
2	Формальный анализ FSRS (относительная полнота, согласованность, целесообразность)
3	Автоматизированное доказательство/проверка обоснования свойств FSRS[a)]
4	Формальное моделирование, проверка модели и исследование модели FSRS
5	Формальная архитектура/спецификация проекта программного обеспечения (SWA/DS)
6	Формальный анализ SWA/DS
7	Автоматизированное доказательство/проверка обоснования выполнения FSRS для SWA/DS[a)]
8	Формальное моделирование, проверка модели и исследование модели SWA/DS
9	Абстрактная интерпретация (правдоподобный статический анализ)
10	Совместная разработка SWA/DS на ESCL[b)]
11	Автоматизированная генерация исходного кода из SWA/DS или промежуточной исполняемой спецификации (IES)[a)]
12	Автоматизированное доказательство/проверка обоснования выполнения SWA/DS для IES[a)]
13	Автоматизированная генерация условий верификации из/с использованием ESCL[a)]
14	Строгая формальная семантика ESCL
15	Автоматизированное доказательство/проверка обоснования свойств (таких как отсутствие восприимчивости к определенным видам ошибок времени выполнения) на уровне ESCL[a)]
16	Автоматизированное доказательство/проверка обоснования выполнения SWA/DS для ESCL[a)]
17	Формальная генерация тестовых примеров из FSRS[c)]
18	Формальная генерация тестовых примеров для SWA/DS[c)]
19	Формальная генерация тестовых примеров из IES[c)]
20	Формальная генерация тестовых примеров на ESCL[c)]
21	Формальный анализ стандартов кодирования
22	Анализ времени выполнения в наихудшем случае (WCET)/анализ времени отклика в наихудшем случае (WCRT)/анализ планируемости
23	Синтез мониторинга/верификация во время выполнения
24	Формально верифицированная компиляция
25	Автоматизированное доказательство/проверка обоснования выполнения на ESCL объектным кодом

[a)] «Автоматизированный» означает, что инструмент используется частично или полностью. Квалификация и использование инструментов рассматриваются в МЭК 61508-3:2010 (пункт 7.4.4).

[b)] «Совместная разработка» относится к разработке программного обеспечения с использованием аннотированных языков программирования, таких как ANNA (Stanford ANNotated Ada), SPARK и Eiffel™.

[c)] Тестирование не является методом, способным гарантировать обеспечение свойств достоверности. Согласно известной цитате 1969 года, тестирование может «демонстрировать наличие ошибок, но не их отсутствие». Тем не менее, M&LT указаны здесь, поскольку используются при генерации тестовых примеров. Генерация тестовых примеров выполняется с учетом требований к выходным данным теста, которые должны формулироваться и проверяться при выполнении теста.

Примечание — Таблица А.1 предназначена для классификации промышленно зрелых M&LT. Цель этой таблицы —включить каждую промышленно зрелую технологию M&LT в одну из категорий, указанных во втором столбце. Тем не менее методы развиваются и эволюционируют: например, анализ WCET, который был включен в исходные версии этой таблицы в 2010 году, был с течением времени дополнен анализом WCRT и планируемости. Следует рассчитывать, что технологии, которые еще не включены в эту таблицу, развиваются и становятся промышленно зрелыми, и рассматривать таблицу как попытку максимально полной классификации на момент публикации.

Приложение В
(справочное)

Конкретные математические и логические методы

Таблица В.1 — Конкретные методы/инструменты M&LT

	Наименование метода или средства	Конкретные методы и инструменты M&LT	Существующий/новый
1	Формальная спецификация требований безопасности (FSRS)	Формальный язык описания. Формальная нотация. Контролируемый естественный язык. Формальная логика. Онтологический анализ опасностей	В МЭК 61508-3:2010 (таблица А.1)
2	Формальный анализ FSRS	Автоматизированная проверка согласованности. Средство доказательства теорем/ система автоматического доказательства теорем/ средство проверки доказательств	Новый
3	Автоматизированное доказательство/проверка обоснования свойств (согласованности, полноты определенных типов) FSRS	Автоматизированная проверка согласованности. Средство доказательства теорем/ система автоматического доказательства теорем/ средство проверки доказательств	Новый
4	Формальное моделирование, проверка модели и исследование модели FSRS	Язык моделирования. Формальное моделирование функций. Иерархическое моделирование. Проверка моделей	В МЭК 61508-3:2010 (таблица А.2)
5	Формальная спецификация проекта (SWA/DS)	Графический язык проектирования. Формальный язык описания. Формальная логика. Формальное уточнение	В МЭК 61508-3:2010 (таблица А.2)
6	Формальный анализ SWA/DS	Автоматизированная проверка согласованности. Средство доказательства теорем/ система автоматического доказательства теорем/ средство проверки доказательств	В МЭК 61508-7:2010 (пункт В.2.4)
7	Автоматизированное доказательство/проверка обоснования выполнения FSRS для SWA/DS	Формальное уточнение. Средство доказательства теорем. Система автоматического доказательства теорем. Средство проверки доказательств. Проверка модели	Новый
8	Формальное моделирование, проверка модели и исследование модели SWA/DS	Моделирование функций. Иерархические конечные автоматы. Моделирование с использованием сетей Петри. Проверка модели. Исследование состояния модели	В МЭК 61508-3:2010 (таблицы А.2, А.4, А.7, В.5, В.7)

Продолжение таблицы В.1

	Наименование метода или средства	Конкретные методы и инструменты M&LT	Существующий/новый
9	Абстрактная интерпретация (правдоподобный статический анализ)	Анализ потока данных. Анализ потока информации. Проверка руководства по написанию программного кода. Анализ ошибок времени выполнения. Анализ времени выполнения в наихудшем случае. Анализ использования стека в наихудшем случае	Новый
10	Совместная разработка SWA/DS на ESCL	Инструменты совместной разработки исходного кода программного обеспечения. Языки, позволяющие одновременно разрабатывать исходный код и формальные спецификации проекта	Новый
11	Автоматизированная генерация исходного кода из SWA/DS или промежуточной исполняемой спецификации (IES)	Генераторы кода, сохраняющие формальную семантику, из текстовых формальных спецификаций проекта. Генераторы кода, сохраняющие формальную семантику, из графических спецификаций проекта	В МЭК 61508-7:2010 (пункт С.4.6)
12	Автоматизированное или основанное на доказательстве/проверке обоснование выполнения SWA/DS для IES[a)]	Средство проверки доказательств. Средство доказательства теорем	Новый
13	Автоматизированная генерация формальных условий верификации из/с использованием ESCL	Средство проверки доказательств. Средство доказательства теорем	Новый
14	Строгая формальная семантика ESCL	Язык программирования со строгой формальной семантикой. Безопасное подмножество языка программирования со строгой формальной семантикой	Новый
15	Автоматизированное доказательство/проверка обоснования свойств (таких как отсутствие восприимчивости к определенным видам ошибок времени выполнения) на уровне ESCL	Инструменты статического анализа кода	Новый
16	Автоматизированное доказательство/проверка обоснования выполнения ESCL требований SWA/DS	Автоматизированное средство проверки доказательств	Новый
17	Формальная генерация тестовых примеров из FSRS	Автоматические генераторы тестовых примеров	Новый
18	Формальная генерация тестовых примеров из SWA/DS	Автоматические генераторы тестовых примеров	Новый
19	Формальная генерация тестовых примеров из IES	Автоматические генераторы тестовых примеров	Новый
20	Формальная генерация тестовых примеров из ESCL	Автоматические генераторы тестовых примеров	Новый

Окончание таблицы В.1

	Наименование метода или средства	Конкретные методы и инструменты M&LT	Существующий/новый
21	Формальный анализ стандартов кодирования (SPARK, MISRA C и т. д.)	Стандарты кодирования. Анализ стандартов кодирования	В МЭК 61508-3:2010 (таблицы А.3, А.4, В.1)
22	Анализ времени выполнения в наихудшем случае (WCET)/ анализ времени отклика в наихудшем случае (WCRT)/анализ планируемости	Инструменты анализа WCET. Инструменты анализа WCRT. Инструменты анализа планируемости	Новый
23	Синтез мониторинга/верификация времени выполнения	Синтез мониторинга/верификация времени выполнения	Новый
24	Формально верифицированная компиляция	Формально верифицированный компилятор	Новый
25	Автоматизированное доказательство/проверка обоснования выполнения ESCL объектным кодом	Автоматизированное средство проверки доказательств. Автоматизированное средство доказательства теорем	Новый

Приложение С
(справочное)

Свойства, гарантируемые применением определенных методов M&LT

Таблица С.1 — Свойства, гарантируемые методами M&LT

	Конкретные методы/ инструменты M&LT	Гарантируемые свойства получаемого программного обеспечения
1	Формальный язык описания	Обеспечение формальной верификации относительной полноты. Обеспечение формальной верификации согласованности
2	Контролируемый естественный язык	Обеспечение формальной верификации относительной полноты. Обеспечение формальной верификации согласованности
3	Формальные графические языки	При поддержке строгой формальной семантики: - обеспечение формальной верификации относительной полноты; - обеспечение формальной верификации согласованности
4	Формальная логика	Обеспечение формальной верификации относительной полноты. Обеспечение формальной верификации согласованности
5	Конечные автоматы	Обеспечение формальной верификации относительной полноты. Обеспечение формальной верификации согласованности. Обеспечение формальной верификации/проверки модели функциональных требований
6	Автоматизированное или основанное на проверке согласованности средство	Гарантированная согласованность спецификаций требований безопасности программного обеспечения
7	Автоматизированное или основанное на доказательстве теорем средство/ средство проверки доказательств	Гарантированное выполнение свойств FSRS. Гарантированное выполнение SWA/DS требований FSRS. Гарантированное выполнение свойств SWA/DS. Гарантированное выполнение ESCL требований SWA/DS. Гарантированное выполнение свойств ESCL. Гарантированное выполнение требований ESCL объектным кодом
8	Онтологический анализ опасностей	Гарантированное выполнение требований безопасности вплоть до уровня объектного кода
9	Формальный язык моделирования/проверка модели	Формальная верификация выполнения требований безопасности проектом системы
10	Формальная детализация	Формально заданная реализация более абстрактных спецификаций более конкретным проектом
11	Исследование состояния модели	См. проверка модели
12	Инструменты совместной разработки исходного кода программного обеспечения	Раннее обнаружение проблем со спецификацией реализации
13	Языки, позволяющие одновременно разрабатывать исходный код и формальные спецификации проекта	Раннее обнаружение проблем со спецификацией реализации
14	Подмножество языка программирования со строгой формальной семантикой	Гарантированная семантика объектного кода по отношению к исходному коду

Окончание таблицы С.1

	Конкретные методы/ инструменты M&LT	Гарантируемые свойства получаемого программного обеспечения
15	Правдоподобный анализ статического кода	Гарантированное отсутствие определенных классов ошибок времени выполнения, таких как доступ за пределы массива, нарушения диапазона значений типов данных, деление на ноль. Выполнение IES/ESCL требований SWA/DS. Гарантированные верхние границы потребления ресурсов (например, использование стека во время выполнения и наихудшее время выполнения)
16	Автоматические генераторы тестовых примеров	Тестовые режимы и шаблоны, соответствующие спецификациям и шаблонам использования, обеспечивают достижение оптимального тестового охвата для конкретного использования
17	Стандарты кодирования	Возможность предотвращения типичных классов ошибок и ограничений ресурсов времени выполнения, таких как висячие указатели и доступ к неинициализированной памяти
18	Анализ стандартов кодирования	Предотвращение типичных классов ошибок и ограничений ресурсов на этапе выполнения. Этот анализ менее достоверен, чем формальная проверка, поскольку некоторые стандарты невозможно проверять автоматически
19	Инструменты анализа WCET/инструменты анализа WCRT/анализ планируемости	Гарантированное максимальное время выполнения для приложений жесткого реального времени
20	Синтез мониторинга/верификация времени выполнения	Верификация правильности работы функций безопасности во время выполнения
21	Формально верифицированный компилятор	Гарантированное выполнение требований ESCL объектным кодом

Приложение D
(справочное)

Детализация программного обеспечения от спецификации безопасности к написанию кода

D.1 Преобразование

В качестве примера рассмотрим конечный автомат (FSM), который хорошо известен из теории информатики и может быть определен на таком языке спецификаций, как Lustre [3]. Также можно программировать конечные автоматы на языках программирования, таких как SPARK Ada или C. FSM, который определен на языке спецификаций, и программа на языке C, реализующая этот FSM, называются эквивалентными, если они определяют и соответственно реализуют идентичные FSM (с одинаковыми множествами состояний, начальным и конечным состояниями, переходами между состояниями при одинаковых входных значениях для переходов и выходными значениями во время перехода, если FSM является преобразователем).

Существуют программные инструменты, которые принимают на вход спецификацию FSM, написанную на языке спецификации FSM, а на выходе генерируют программу (например, на языке C), которая реализует эквивалентный FSM. Говорят, что такие инструменты *преобразуют* спецификацию FSM в программу на языке C. Говорят, что формальный язык, используемый для спецификации источника преобразования, является языком более высокого уровня, чем формальный язык, который используется для преобразования. В общем случае уровень детализации конструкции на низкоуровневом языке неинтерпретируем на высокоуровневом языке, и, следовательно, не имеет к нему отношения.

Метод, который обеспечивает сохранение свойств спецификации, полученной в результирующем программном коде, посредством детализации и преобразования, является методом B [4], [5], [6]. Существуют инструментальные средства, которые реализуют метод B.

Ключевое свойство преобразования между двумя языками со строго определенной формальной семантикой заключается в том, что семантики конструкций высокоуровневого языка сохраняются конструкциями низкоуровневого языка. Когда формальная семантика является поведенческой (т. е. определяет строгое вычислительное поведение), например, в случае FSM, она сохраняется, если поведение конструкции низкоуровневого языка точно имитирует поведение, определенное семантикой конструкции языка более высокого уровня (включая поведение отказа на более низком уровне). В случае конечного автомата это означает, что FSM низкого уровня эквивалентен FSM более высокого уровня.

D.2 Детализация

Детализация — это преобразование или последовательность преобразований. Можно ввести более формальное определение следующим образом. Предположим, что T.1: D.1 → D.2, T.2: D.2 → D.3, …, T.k: D.k → D.(k+1) — это последовательность преобразований из D.1 в языке L.1 в D.(k+1) языка L.(k+1). Тогда детализацией является последовательное применение преобразований T.1, T.2, …, T.k к D.1, приводящее к D.(k+1) в языке L.(k+1).

В этой детализации уровень языка L.1 выше, чем уровень языка L.2, уровень языка L.2 выше, чем уровень языка L.3, и т. д. Для определения преобразований T.1, T.2, …, T.k каждый объект преобразования D.1, D.2, … D.k должен иметь строго определенную и недвусмысленную формальную семантику.

Целью свойств достоверности, представленных в настоящем стандарте, является максимально возможное приближение процесса вывода OC из FSRS к детализации от FSRS до OC.

Приложение ДА
(справочное)

Сведения о соответствии ссылочных международных стандартов межгосударственным и национальным стандартам

Таблица ДА.1

Обозначение ссылочного международного стандарта	Степень соответствия	Обозначение и наименование соответствующего межгосударственного и национального стандарта
IEC 61508-3:2010	IDT	ГОСТ IEC 61508-3—2018 «Функциональная безопасность систем электрических, электронных, программируемых электронных, связанных с безопасностью. Часть 3. Требования к программному обеспечению»
IEC 61508-4:2010	IDT	ГОСТ Р МЭК 61508-4—2012 «Функциональная безопасность систем электрических, электронных, программируемых электронных, связанных с безопасностью. Часть 4. Термины и определения»
Примечание — В настоящей таблице использовано следующее условное обозначение степени соответствия стандарта: - IDT — идентичный стандарт.		

Библиография

[1] Hartley Rogers Jr., Theory of Recursive Functions and Effective Computability, MIT Press, 1987 (Reprint of 1967 edition)

[2] George S. Boolos, John P. Burgess and Richard C. Jeffrey, Computability and Logic, 5th Edition, Cambridge University Press, 2007

[3] N. Halbwachs et al., The Synchronous Data Flow Programming Language LUSTRE, Proc. IEEE 79(9), 1991

[4] Jean-Raymond Abrial, The B-Book: Assigning Programs to Meanings, Jean-Raymond Abrial, Cambridge University Press, 1996

[5] Steve Schneider, The B-Method: An Introduction, Steve SchneiderPalgrave Macmillan, 2001

[6] Jean-Raymond Abrial, Modeling in Event-B: System and Software Engineering, Jean- Raymond Abrial, Cambridge University Press, 2010

УДК 62-783:614.8:331.454:006.354 ОКС 25.040.40

Ключевые слова: функциональная безопасность, программное обеспечение, математические и логические методы, свойства программного обеспечения

Редактор *Е.Ю. Митрофанова*
Технический редактор *В.Н. Прусакова*
Корректор *С.И. Фирсова*
Компьютерная верстка *М.В. Малеевой*

Сдано в набор 24.11.2025. Подписано в печать 11.12.2025. Формат 60×84⅛. Гарнитура Ариал.
Усл. печ. л. 3,26. Уч.-изд. л. 2,64.

Подготовлено на основе электронной версии, предоставленной разработчиком стандарта

Создано в единичном исполнении в ФГБУ «Институт стандартизации»
для комплектования Федерального информационного фонда стандартов,
117418 Москва, Нахимовский пр-т, д. 31, к. 2.

gost.e-ecolog.ru gost.e-ecolog.ru