МЕЖГОСУДАРСТВЕННЫЙ СОВЕТ ПО СТАНДАРТИЗАЦИИ, МЕТРОЛОГИИ И СЕРТИФИКАЦИИ
(МГС)
INTERSTATE COUNCIL FOR STANDARDIZATION, METROLOGY AND CERTIFICATION
(ISC)

МЕЖГОСУДАРСТВЕННЫЙ СТАНДАРТ

**ГОСТ ISO 19014-4— 2024**

# МАШИНЫ ЗЕМЛЕРОЙНЫЕ

## Функциональная безопасность

### Часть 4

## Разработка и оценка программного обеспечения и передачи данных для элементов систем управления, связанных с обеспечением безопасности

(ISO 19014-4:2020, IDT)

Издание официальное

Москва
Российский институт стандартизации
2024

## Предисловие

Цели, основные принципы и общие правила проведения работ по межгосударственной стандартизации установлены ГОСТ 1.0 «Межгосударственная система стандартизации. Основные положения» и ГОСТ 1.2 «Межгосударственная система стандартизации. Стандарты межгосударственные, правила и рекомендации по межгосударственной стандартизации. Правила разработки, принятия, обновления и отмены»

**Сведения о стандарте**

1 ПОДГОТОВЛЕН Российской ассоциацией производителей специализированной техники и оборудования (Ассоциацией «Росспецмаш») на основе собственного перевода на русский язык англоязычной версии стандарта, указанного в пункте 5

2 ВНЕСЕН МТК 267 «Строительно-дорожные машины и оборудование»

3 ПРИНЯТ Межгосударственным советом по стандартизации, метрологии и сертификации (протокол от 30 августа 2024 г. № 176-П)

За принятие проголосовали:

| Краткое наименование страны по МК (ИСО 3166) 004—97 | Код страны по МК (ИСО 3166) 004—97 | Сокращенное наименование национального органа по стандартизации |
|---|---|---|
| Армения | AM | ЗАО «Национальный орган по стандартизации и метрологии» Республики Армения |
| Беларусь | BY | Госстандарт Республики Беларусь |
| Киргизия | KG | Кыргызстандарт |
| Россия | RU | Росстандарт |
| Таджикистан | TJ | Таджикстандарт |
| Узбекистан | UZ | Узбекское агентство по техническому регулированию |

4 Приказом Федерального агентства по техническому регулированию и метрологии от 3 октября 2024 г. № 1370-ст межгосударственный стандарт ГОСТ ISO 19014-4—2024 введен в действие в качестве национального стандарта Российской Федерации с 1 января 2025 г.

5 Настоящий стандарт идентичен международному стандарту ISO 19014-4:2020 «Машины землеройные. Функциональная безопасность Часть 4. Разработка и оценка программного обеспечения и передачи данных для элементов систем управления, связанных с обеспечением безопасности» («Earth-moving machinery — Functional safety — Part 4: Design and evaluation of software and data transmission for safety-related parts of the control system», IDT).

Международный стандарт разработан Техническим комитетом по стандартизации ISO/TC 127 «Машины землеройные» Международной организации по стандартизации (ISO).

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

6 ВВЕДЕН ВПЕРВЫЕ

*Информация о введении в действие (прекращении действия) настоящего стандарта и изменений к нему на территории указанных выше государств публикуется в указателях национальных стандартов, издаваемых в этих государствах, а также в сети Интернет на сайтах соответствующих национальных органов по стандартизации.*

*В случае пересмотра, изменения или отмены настоящего стандарта соответствующая информация будет опубликована на официальном интернет-сайте Межгосударственного совета по стандартизации, метрологии и сертификации в каталоге «Межгосударственные стандарты»*



## Содержание

М Е Ж Г О С У Д А Р С Т В Е Н Н Ы Й  С Т А Н Д А Р Т

# МАШИНЫ ЗЕМЛЕРОЙНЫЕ

## Функциональная безопасность

## Часть 4

## Разработка и оценка программного обеспечения и передачи данных для элементов систем управления, связанных с обеспечением безопасности

Earth-moving machinery. Functional safety. Part 4. Design and evaluation of software and data transmission for safety-related parts of the control system

**Дата введения — 2025—01—01**

## 1 Область применения

Настоящий стандарт определяет общие принципы разработки программного обеспечения (ПО) и требования к передаче сигналов элементов систем управления, связанных с обеспечением безопасности (MCS), в землеройных машинах (ЕММ) и их оборудовании, как определено в ISO 6165. Кроме того, в настоящем стандарте рассматриваются существенные опасности, определенные в ISO 12100, связанные с ПО, встроенным в систему управления машиной.

Существенными опасностями, на которые обращают внимание, являются неправильные реакции системы управления машиной на входные сигналы.

Кибербезопасность не включена в настоящий стандарт.

Примечание — Руководство по кибербезопасности см. в соответствующем стандарте по безопасности.

Настоящий стандарт не применим к ЕММ, изготовленным до даты его вступления в действие.

## 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)]:

ISO 6750-1, Earth-moving machinery — Operator's manual — Part 1: Contents and format (Машины землеройные. Руководство для оператора. Часть 1. Содержание и формат)

ISO 12100:2010, Safety of machinery — General principles for design — Risk assessment and risk reduction (Безопасность машин. Основные понятия, общие принципы конструирования. Оценка и снижение рисков)

ISO 13849-1:2015, Safety of machinery — Safety-related parts of control systems — Part 1: General principles for design (Безопасность машин. Детали систем управления, связанные с обеспечением безопасности. Часть 1. Общие принципы проектирования)

ISO 19014-1, Earth-moving machinery — Functional safety — Part 1: Methodology to determine safetyrelated parts of the control system and performance requirements (Машины землеройные. Функциональная безопасность. Часть 1. Методология определения частей систем контроля, связанных с обеспечением безопасности, и требования к рабочим характеристикам)

**Издание официальное**

ISO 19014-2, Earth-moving machinery — Functional safety — Part 2: Earth-moving machinery — Functional safety — Part 2: Design and evaluation of hardware and architecture requirements for safety-related parts of the control system (Машины землеройные. Функциональная безопасность. Часть 2. Проектирование и оценка оборудования и структуры систем управления, связанных с обеспечением безопасности)

## 3 Термины и определения

В настоящем стандарте применены термины по ISO 6165 и ISO 12100, а также следующие термины с соответствующими определениями.

ISO и IEC ведут терминологические базы данных для использования в стандартизации по следующим ссылкам:

- онлайн-платформа ISO: https://www.iso.org/obp;
- электропедия IEC: http://www.electropedia.org/.

3.1 **шинная система** (bus system): Подсистема, используемая в электронной системе управления для передачи сообщений (3.6).

Примечание 1 к статье — Шинная система состоит из системных блоков (источники и приемники информации), путей передачи/среды передачи (например, электрические линии, оптоволоконные линии, радиочастотная передача), интерфейса между источником/приемником сообщения и электронных элементов шины (например, интегральная схема, специфичная для протокола, приемопередатчики).

3.2 **закрытая шинная система** (encapsulated bus system): Шинная система (3.1), включающая фиксированное количество или предварительно определенное максимальное количество участников шины, соединенных друг с другом через передающую среду с четко определенными и фиксированными показателями и характеристиками.

3.3 **сбой связи между узлами связи** (failure of peer communication): Ситуация, в которой узел связи недоступен.

3.4 **непреднамеренный повтор сообщения** (unintended message repetition): Ситуация, в которой одно и то же сообщение (3.6) непреднамеренно отправляется снова.

3.5 **повтор сообщения** (message repetition): Ситуация, в которой одно и то же сообщение (3.6) намеренно отправляется снова.

Примечание 1 к статье — Этот метод повторной отправки одного и того же сообщения устраняет сбои, такие как потеря сообщения (3.10).

3.6 **сообщение** (message): Электронная передача данных.

Примечание 1 к статье — Передаваемые данные могут включать данные пользователя, адрес или данные идентификатора и данные для обеспечения целостности передачи.

3.7 **электронный блок управления;** ECU (electronic control unit): Электронное устройство (электронный программируемый контроллер), используемое в системе управления землеройной техникой.

3.8 **время реакции** (reaction time): Время от обнаружения события, связанного с безопасностью, до начала реакции системы безопасности.

3.9 **артефакт** (artifact): Рабочие продукты, которые производятся и используются во время проекта для сбора и передачи информации.

3.10 **потеря сообщения** (message loss): Непреднамеренное удаление сообщения (3.6) по вине участника шины.

3.11 **неправильная последовательность** (incorrect sequence): Непреднамеренное изменение последовательности сообщений (3.6) по вине участника шины.

Примечание 1 к статье — Шинные системы (3.1) могут содержать элементы с сохраненными сообщениями [по мере поступления (FIFO) и т. д.], которые могут изменить правильную последовательность.

3.12 **фальсификация сообщений** (message falsification): Непреднамеренная модификация сообщений (3.6) из-за ошибки участника шины или из-за ошибок в канале передачи.

3.13 **задержка сообщения** (message retardation): Непреднамеренная задержка или блокировка функции безопасности, вызванная перегрузкой пути передачи нормальным обменом данными или отправкой неправильных сообщений (3.6).

3.14 **счетчик активности** (alive counter): Компонент учета, инициализируемый «0» при создании или восстановлении контролируемого объекта.

Примечание 1 к статье — Счетчик увеличивается от времени $t - 1$ до времени $t$, пока объект активен. В конце работы счетчик активности показывает период времени, в течение которого объект был активен в сети.

3.15 **тестирование методом «черного ящика»** (black box testing): Тестирование объекта, не требующее знания его внутренней структуры или конкретной реализации.

3.16 **раздел** (partition): Ресурсный объект, выделяющий часть памяти, устройств ввода/вывода и использования центрального процессора одной или нескольким системным задачам (3.21).

Примечание 1 к статье — Разделы могут быть назначены одной или нескольким подсистемам в сети микроконтроллеров.

3.17 **программное разделение на разделы** (software partitioning): Метод сдерживания сбоев (3.26) ПО, состоящий в назначении ресурсов определенным компонентам ПО с целью предотвращения распространения сбоя программного обеспечения на несколько разделов (3.16).

3.18 **программный компонент** (software component): Один или несколько программных модулей (3.19).

3.19 **программный модуль** (software module): Независимая часть ПО, которую можно независимо протестировать и отследить до спецификации.

Примечание 1 к статье — Программный модуль является неделимым программным компонентом.

3.20 **программные разделы** (software partitions): Среда выполнения с отдельными назначенными системными ресурсами.

3.21 **системная задача** (system task): Задачи, которые выполняются в рамках ресурсов разделов (3.16) и с разными приоритетами.

3.22 **независимость программного обеспечения** (independence of software): Исключение непреднамеренных взаимодействий между программными компонентами, а также отсутствие влияния на корректную работу программного компонента ошибок другого программного компонента.

3.23 **история операций** (operational history): Эксплуатационные данные о программном компоненте или программном модуле (3.19) во время его эксплуатации.

3.24 **максимальное время цикла** (maximum cycle time): Статическое время для доступа к коммуникационной шине между узлами на уровне шины или узла.

Примечание 1 к статье — Применение протокола, запускаемого по времени, гарантирует, что это время цикла не будет превышено.

3.25 **максимальное время отклика** (maximum response time): Фиксированное время, назначенное системной активности для обмена глобально синхронизированными сообщениями (3.6) на шине в архитектуре, запускаемой по времени.

3.26 **программная ошибка** (software fault): Неправильный шаг, процесс или определение данных в ПО, из-за чего система выдает неожиданные результаты.

3.27 **анализ воздействия** (impact analysis): Документация, которая фиксирует понимание и последствия предлагаемого изменения.

3.28 **процесс управления конфигурацией** (configuration management process): Задача отслеживания и контроля изменений артефактов (3.9) в процессе разработки.

3.29 **постоянная передача сообщений** (constant transmission of messages): Ситуация, в которой неисправный узел постоянно передает сообщения (3.6), которые ставят под угрозу работу шины.

3.30 **блокировка доступа к шине данных** (blocking access to the data bus): Ситуация, в которой неисправный узел не придерживается ожидаемых моделей использования и предъявляет чрезмерные требования к использованию ресурсов, тем самым снижая их доступность для других узлов.

## 4 Разработка программного обеспечения

### 4.1 Общее

Основной целью следующих требований является достижение надежности ПО с помощью улучшения удобочитаемости, понятности, тестируемости и поддержки ПО. В настоящем разделе даны рекомендации по разработке ПО и последующему соответствующему тестированию. Предотвращение ошибок ПО должно рассматриваться в течение всего процесса разработки ПО.

Если существующий программный компонент был разработан в соответствии с ранее действовавшим стандартом и продемонстрирован посредством использования и проверки его применения для снижения риска до разумно возможного низкого уровня, не требуется обновлять документацию жизненного цикла ПО на уровне модуля ПО.

ПО управления машиной должно соответствовать требованиям безопасности настоящего раздела. Кроме того, ПО управления машиной должно быть спроектировано и разработано в соответствии с принципами ISO 12100:2010 для учета соответствующих незначительных опасностей, которые не рассматриваются в настоящем стандарте.

### 4.2 Планирование

Должен быть разработан план для определения взаимосвязи между отдельными фазами разработки ПО и соответствующими артефактами.

Соответствующие методы и меры из таблиц 3—9 должны быть выбраны для разработки ПО в соответствии с требуемым уровнем эффективности защиты машины ($MPL_r$).

$MPL_r$ системы может быть достигнут путем параллельного добавления двух систем с более низким уровнем эффективности защиты. При параллельном добавлении (согласно ISO 19014-2) ПО в каждой системе может быть разработано с более низкими требованиями $MPL_r$. Это допустимо только в том случае, если у параллельных систем нет отказов по общей причине.

Пригодность выбранных методов или мер для применения должна быть обоснована в начале каждого запланированного этапа разработки. Для конкретного применения соответствующая комбинация методов или мер должна быть указана во время планирования разработки. Допускается использовать методы или меры, не указанные в таблицах 3—9.

На рисунке 1 представлена одна из возможных моделей разработки ПО (V-модель). Для разработки ПО может быть использован любой проверенный процесс разработки, отвечающий требованиям настоящего стандарта.

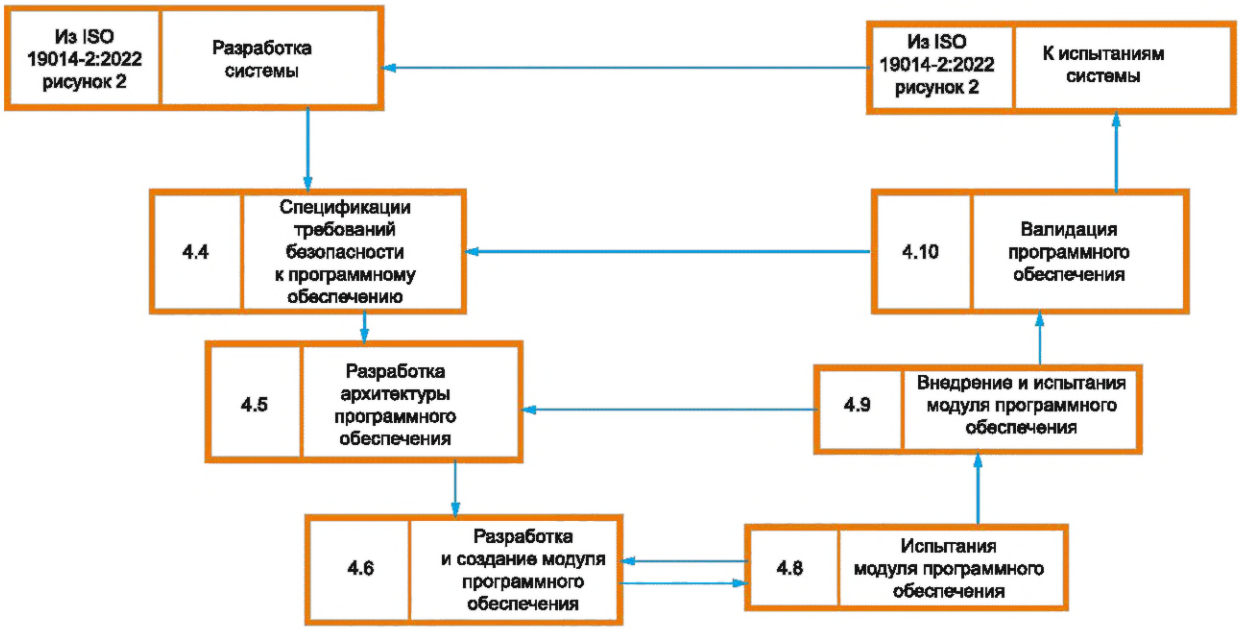



Рисунок 1 — V-модель разработки ПО

При выборе методов и мер помимо ручного кодирования может применяться разработка на основе моделей, когда исходный код автоматически генерируется из моделей.

Для каждого метода или меры в таблицах существует свой уровень обеспечения для каждого уровня эффективности защиты. В таблице 1 приведено описание спецификации требований безопасности ПО.

Таблица 1 — Спецификация требований безопасности ПО

| Символ | Спецификация требований безопасности ПО |
|---|---|
| + | Метод или меру следует использовать для данного MPLr<br>В случае, если этот метод или мера не используются, должно быть представлено соответствующее обоснование в фазе планирования безопасности |
| ○ | Метод или мера могут быть использованы для данного MPLr |
| – | Метод или мера не могут быть использованы для данного MPLr |

Должны быть выбраны методы и меры, соответствующие требуемому MPLr. Альтернативные или эквивалентные методы и меры обозначают буквами после номера. Должен быть выбран по крайней мере один из альтернативных или эквивалентных методов и мер, отмеченных знаком «+», и в этом случае обоснование не требуется. Пример спецификации требований безопасности ПО приведен в таблице 2.

Таблица 2 — Пример спецификации требований к безопасности ПО

| Метод/мера | | MPLr = a | MPLr = b, c | MPLr = d | MPLr = e |
|---|---|---|---|---|---|
| 1.a | Мера 1 | + | + | – | |
| 1.b | Мера 2 | + | + | + | + |
| 1.c | Мера 3 | + | + | + | + |

В этом случае:

- одна из мер, 1, 2 или 3, должна быть выполнена для MPLr = a, b, c;
- одна из мер, 2 или 3, должна быть выполнена для MPLr = d, e;
- в противном случае должно быть представлено обоснование неуказанного альтернативного метода/меры для выполнения требований стандарта для конкретного MPLr.

Обоснование должно быть предоставлено, если вместо перечисленных методов или мер используются другие эквивалентные методы или меры.

Если программный компонент оказывает какое-либо влияние на другие функции безопасности с другим MPLr, то должны применяться требования, относящиеся к самому высокому MPLr.

Если ПО содержит связанные и не связанные с безопасностью компоненты, то общий достигнутый уровень эффективности защиты машины со встроенным ПО (MPLa) должен быть ограничен программным компонентом с наименьшим MPLa; это требование не применяется, если достаточная независимость между программными компонентами может быть продемонстрирована в соответствии с разделом 7.

При повторном использовании программного компонента, предназначенного для модификации, должен быть выполнен анализ воздействия. План действий должен быть разработан и реализован для всего жизненного цикла ПО на основе результатов анализа воздействия, чтобы обеспечить достижение целей безопасности.

### 4.3 Артефакты

После определения отдельных фаз плана разработки ПО должны быть определены артефакты для каждой выполняемой фазы. Другие этапы и связанные артефакты могут быть добавлены путем распределения действий и задач. Принимая во внимание объем и сложность проекта, все артефакты могут быть изменены на отдельных этапах V-модели разработки, показанной на рисунке 1.

Примечание — Обычно отдельные фазы объединяют, если используемый метод/мера затрудняет четкое различие между фазами. Например, проектирование архитектуры ПО и реализация ПО могут быть созданы последовательно с помощью одного и того же автоматизированного средства разработки, как это делается в процессе разработки на основе модели.

В рамках процесса разработки ПО артефакты должны быть:

а) задокументированы в соответствии с результатами, ожидаемыми от запланированных фаз;

b) изменены в результате анализа воздействия, и только затронутое ПО должно подвергаться повторному тестированию;

c) объектами процесса управления конфигурацией.

Первым артефактом, применимым к процессу, является план разработки ПО. Последующие артефакты, определенные планом, должны включать:

- проектную спецификацию и соответствующий отчет о проверке для каждой фазы разработки ПО (нисходящая ветвь V-модели на рисунке 1);
- спецификацию тестирования и соответствующий отчет о тестировании для каждой фазы тестирования ПО (восходящая ветвь V-модели на рисунке 1);
- исполняемое программное обеспечение.

### 4.4 Спецификация требований безопасности программного обеспечения

Спецификация требований безопасности ПО должна содержать требования к следующему, если это применимо:

- к функциям, которые позволяют системе достигать или поддерживать безопасное состояние;
- функциям, связанным с обнаружением, индикацией и обработкой неисправностей элементов систем управления, связанных с безопасностью (SRP/CS);
- функциям, связанным с обнаружением, индикацией и обработкой ошибок в ПО;
- функциям, относящимся к оперативным и автономным проверкам функций безопасности.

Примечание 1 — Онлайн-тестирование выполняется во время использования тестируемой системы. Автономный тест выполняется, когда тестируемая система не используется.

Примечание 2 — Примером онлайн-теста может быть проверка неисправностей в системе рулевого управления во время движения машины. Примером проверки в автономном режиме может быть проверка на наличие неисправностей в системе рулевого управления до разрешения движения машины;

- функциям, позволяющим модифицировать параметры ПО, связанные с безопасностью;
- интерфейсам с функциями, не связанными с безопасностью;
- эффективности защиты и времени отклика;
- интерфейсам между ПО и аппаратной частью электронного блока управления.

Чтобы соответствовать установленному MPLr, соответствующий метод или меры должны быть выбраны с учетом спецификации требований безопасности ПО из таблицы 3.

Таблица 3 — Спецификация требований безопасности ПО

| Метод/мера | | Ссылка | MPLr = a | MPLr = b, c | MPLr = d | MPLr = e |
|---|---|---|---|---|---|---|
| 1 | Спецификация требований на естественном языке | A.1 | + | + | + | + |
| 2 | Автоматизированные инструменты составления спецификации | A.2 | ○ | ○ | ○ | + |
| 3.a | Неформальные методы | A.3 | + | + | + | – |
| 3.b | Полуформальные методы | A.4 | + | + | + | + |
| 3.c | Формальные методы | A.5 | + | + | + | + |
| 4 | Прямая прослеживаемость между требованиями безопасности системы и ПО | A.6 | ○ | ○ | ○ | + |
| 5 | Обратная прослеживаемость между требованиями безопасности системы и ПО | | ○ | ○ | ○ | + |
| 6.a | Пошаговое рассмотрение требований безопасности ПО | A.7 | + | + | + | – |
| 6.b | Проверка требований безопасности ПО | A.8 | + | + | + | + |
| Примечание — Описание методов и мер приведено в приложении A. | | | | | | |

### 4.5 Архитектура программного обеспечения

Архитектура ПО, описывающая иерархическую структуру всех связанных с безопасностью программных компонентов каждой системы управления, связанной с безопасностью (SCS), должна разрабатываться на основе требований безопасности ПО. Чтобы соответствовать установленному MPLr, соответствующие методы или меры должны быть выбраны с учетом разработки архитектуры ПО из таблицы 4.

Таблица 4 — Разработка архитектуры ПО

| Метод/мера | | Ссылка | MPLr = a | MPLr = b, c | MPLr = d | MPLr = e |
|---|---|---|---|---|---|---|
| 1.a | Неформальные методы | A.3 | + | + | + | – |
| 1.b | Полуформальные методы | A.4 | + | + | + | + |
| 1.c | Формальные методы | A.5 | + | + | + | + |
| 2 | Автоматизированные инструменты разработки | A.9 | ○ | ○ | ○ | + |
| 3.a | Циклический режим с гарантированным максимальным временем цикла | A.10 | ○ | ○ | + | + |
| 3.b | Синхронизация по времени | | ○ | ○ | + | + |
| 3.c | Управляемая событиями система с гарантированным максимальным временем отклика | | ○ | ○ | + | + |
| 4 | Прямая прослеживаемость между требованиями безопасности системы и архитектурой ПО | A.6 | ○ | ○ | ○ | + |
| 5 | Обратная прослеживаемость между требованиями безопасности системы и архитектурой ПО | | ○ | ○ | ○ | + |
| 6.a | Пошаговое рассмотрение архитектуры ПО | A.7 | + | + | + | – |
| 6.b | Проверка архитектуры ПО | A.8 | + | + | + | + |
| Примечание — Описание методов и мер приведено в приложении А. | | | | | | |

### 4.6 Разработка программного модуля и создание программного кода

Целями этого этапа разработки ПО являются:

- подробное описание поведения программных модулей, связанных с безопасностью, которые предписаны архитектурой ПО;
- создание читаемых, тестируемых и поддерживаемых программных модулей (например, созданный вручную код, модель и т. д.);
- проверка того, что архитектура ПО была полностью и правильно реализована.

Чтобы соответствовать установленному MPLr, при разработке программного модуля и создании программного кода соответствующие методы или меры должны быть выбраны из таблицы 5. Проверки автоматически сгенерированного кода необязательны.

Таблица 5 — Разработка программного модуля и создание программного кода

| Метод/мера | | Ссылка | MPLr = a | MPLr = b, c | MPLr = d | MPLr = e |
|---|---|---|---|---|---|---|
| 1.a | Неформальные методы | A.3 | + | + | – | – |
| 1.b | Полуформальные методы | A.4 | + | + | + | + |
| 1.c | Формальные методы | A.5 | + | + | + | + |
| 2 | Автоматизированные инструменты разработки | A.9 | ○ | ○ | ○ | + |
| 3 | Использование стандартных разработок и кодов | A.11 | ○ | + | + | + |
| 4 | Отсутствие неструктурированных потоков в программах на высоких языках программирования[b] | | ○ | ○ | + | + |

*Окончание таблицы 5*

| Метод/мера | | Ссылка | MPLr = a | MPLr = b, c | MPLr = d | MPLr = e |
|---|---|---|---|---|---|---|
| 5 | Ограниченное автоматическое преобразование типов[b] | A.11 | ○ | ○ | + | + |
| 6 | Ограниченное использование прерываний[b] | | ○ | ○ | ○ | + |
| 7 | Ограниченное использование указателей[b] | | ○ | ○ | ○ | + |
| 8 | Ограниченная рекурсия | | ○ | ○ | ○ | + |
| 9.a | Динамические переменные или объекты с онлайн-проверкой[b] | A.12 | ○ | ○ | – | – |
| 9.b | Динамические переменные или объекты без онлайн-проверки[b] | A.13 | ○ | ○ | + | + |
| 10 | Ограничение размера модулей ПО | A.14 | + | + | + | + |
| 11 | Одна точка ввода и одна точка вывода данных для каждого маршрута и функции[b] | | ○ | + | + | + |
| 12 | Полностью определенный интерфейс | | ○ | + | + | + |
| 13 | Скрытие информации | | ○ | ○ | + | + |
| 14 | Контроль сложности ПО | | ○ | ○ | ○ | + |
| 15 | Структурированная разработка и программирование | A.15 | ○ | + | + | + |
| 16 | Защитные разработки или коды | A.16 | ○ | ○ | ○ | + |
| 17 | Использование проверенных или верифицированных компонентов ПО[a] | A.17 | ○ | ○ | ○ | ○ |
| 18 | Прямая прослеживаемость от спецификации требований безопасности к ПО до проектирования ПО | A.6 | ○ | ○ | ○ | + |
| 19.a | Пошаговое рассмотрение разработки, исходного кода или их обоих | A.7 | + | + | + | – |
| 19.b | Инспекция разработки, исходного кода или их обоих | A.8 | + | + | + | + |

Примечание — Описание методов и мер приведено в приложении А.

[a] Рекомендуется использование проверенных и верифицированных элементов ПО.

[b] Эти методы и меры не всегда применимы при графическом моделировании в разработках, основанных на моделировании.

### 4.7 Выбор языка программирования и инструментов

Полнота безопасности разрабатываемого ПО может напрямую зависеть от выбранного языка программирования, инструментов, используемых при разработке и тестировании, а также от использования существующих надежных и проверенных программных модулей. Чтобы соответствовать установленному MPLr, при выборе языка программирования и инструментов используют данные из таблицы 6.

Таблица 6 — Выбор языка программирования и инструментов

| Метод/мера | | Ссылка | MPLr = a | MPLr = b, c | MPLr = d | MPLr = e |
|---|---|---|---|---|---|---|
| 1 | Подходящий язык программирования | A.18 | + | + | + | + |
| 2 | Поддержка языковых подмножеств | A.19 | ○ | ○ | ○ | + |
| 3.a | Инструменты и интерпретаторы с повышенной надежностью в результате использования или валидации | A.20 | ○ | + | + | + |
| 3.b | Сертифицированные инструменты и интерпретаторы | A.21 | ○ | + | + | + |

Примечание — Описание методов и мер приведено в приложении А.

### 4.8 Тестирование программного модуля

Целью тестирования программных модулей является проверка того, что разработанные и реализованные программные модули соответствуют проекту безопасности ПО. На этом этапе должна быть разработана процедура тестирования программных модулей на соответствие их требованиям, и испытания должны проводиться в соответствии с этой процедурой.

Для систематического подхода к тестированию программных модулей, при использовании соответствующих инструментов для управления процессом и его автоматизации, поддерживают трудоемкие и подверженные ошибкам задачи в тестируемом модуле. Доступность инструментов поддержки способствует исчерпывающему подходу как к обычному, так и к регрессионному тестированию.

Для упрощения проверки, оценки и обслуживания все данные, решения и обоснования должны быть задокументированы на протяжении всего программного проекта. Документация на программные модули должна включать:

- выполненные тестирования;
- решения и их обоснование;
- проблемы и их решения.

Примечание — Запись данных важна для технического обслуживания компьютерных систем, поскольку обоснование определенных решений, принятых в ходе проекта разработки, не всегда известно инженерам по техническому обслуживанию.

Чтобы соответствовать установленному MPLr при тестировании программного модуля, используют методы или меры из таблицы 7.

Тестирование программных модулей может выполняться в разных средах, например:

- тесты модели в контуре,
- тесты ПО в контуре,
- тесты процессора в контуре,
- аппаратные тесты в контуре.

Таблица 7 — Тестирование программного модуля

| Метод/мера | | Ссылка | MPLr = a | MPLr = b, c | MPLr = d | MPLr = e |
|---|---|---|---|---|---|---|
| 1 | Анализ граничных значений | A.22 | ○ | ○ | + | + |
| 2 | Анализ потока управления | A.23 | ○ | ○ | + | + |
| 3 | Анализ потока данных | A.24 | ○ | ○ | + | + |
| 4 | Выполнение тестового примера из анализа граничных значений | A.25 | ○ | ○ | ○ | + |
| 5 | Функциональное тестирование/тестирование методом «черного ящика» [включая тестирование с внесением ошибок (FI)] | A.26 | + | + | + | + |
| 6.a | Тестирование структуры (точки входа) | A.27 | ○ | + | – | – |
| 6.b | Тестирование структуры (операторы) | | ○ | + | + | – |
| 6.c | Тестирование структуры (ветвление) | | ○ | + | + | + |
| 7 | Классы эквивалентности и тестирование входных разделов[a] | A.28 | ○ | ○ | + | + |
| 8 | Выполнение тестового примера из генерации тестового примера на основе модели | A.29 | ○ | ○ | ○ | + |
| 9 | Время отклика и тестирование ограничений памяти[a] | A.30 | ○ | + | + | + |
| 10 | Тестирование требований к эффективности защиты[a] | | ○ | + | + | + |
| 11 | Лавинное/нагрузочное тестирование[a] | | ○ | ○ | ○ | + |
| 12 | Тестирование интерфейса модуля ПО | A.31 | ○ | ○ | ○ | + |
| 13 | Последовательное сравнительное тестирование | A.32 | ○ | ○ | + | + |

*Окончание таблицы 7*

| Метод/мера | | Ссылка | MPLr = a | MPLr = b, c | MPLr = d | MPLr = e |
|---|---|---|---|---|---|---|
| 14 | Прямая прослеживаемость между разработкой модуля ПО и спецификацией модулей и тестов | A.6 | ○ | ○ | ○ | + |
| Примечание — Описание методов и мер приведено в приложении А.<br>[a] Разработчик может не использовать этот метод, но должен использовать его на уровне интеграции, при необходимости. | | | | | | |

### 4.9 Интеграция и тестирование программного модуля

Целями этого этапа разработки ПО являются:

- интеграция программных модулей в программные компоненты во встроенном ПО системы управления безопасностью;
- проверка того, что требования к ПО правильно реализованы во встроенном ПО.

На этом этапе конкретные шаги интеграции проверяют на соответствие требованиям безопасности ПО. Также тестируют интерфейсы между программными модулями и между программными модулями и компонентами. Этапы интеграции и тестирования программных компонентов должны напрямую соответствовать иерархической архитектуре ПО.

Чтобы соответствовать установленному MPLr при интеграции и тестировании программного модуля, выбирают подходящие методы и меры из таблицы 8.

Интеграционное тестирование программных модулей может выполняться в разных средах, например:

- тесты модели в контуре,
- тесты ПО в контуре,
- тесты процессора в контуре,
- аппаратные тесты в контуре.

Таблица 8 — Интеграция и тестирование программного модуля

| Метод/мера | | Ссылка | MPLr = a | MPLr = b, c | MPLr = d | MPLr = e |
|---|---|---|---|---|---|---|
| 1 | Функциональное тестирование/тестирование методом «черного ящика» [включая тестирование с внесением ошибок (FI)] | A.26 | + | + | + | + |
| 2 | Классы эквивалентности и тестирование входных разделов | A.28 | ○ | ○ | + | + |
| 3 | Время отклика и тестирование ограничений памяти | A.30 | ○ | + | + | + |
| 4 | Тестирование требований к эффективности защиты | | ○ | + | + | + |
| 5 | Лавинное/нагрузочное тестирование | | ○ | ○ | ○ | + |
| 6 | Последовательное сравнительное тестирование | A.32 | ○ | ○ | + | + |
| 7 | Прямая прослеживаемость между разработкой архитектуры ПО и спецификацией интеграционных тестов | A.6 | ○ | ○ | ○ | + |
| Примечание — Описание методов и мер приведено в приложении А. | | | | | | |

### 4.10 Валидация программного обеспечения

Целью этого этапа разработки ПО является демонстрация того, что встроенное ПО правильно реализует требования безопасности к программному обеспечению.

Тестирование должно быть основным методом валидации ПО. Анимация и моделирование могут использоваться в качестве дополнения к действиям по проверке.

ПО должно быть отработано путем имитации:

- входных сигналов, присутствующих во время нормальной работы;
- ожидаемых событий;
- нежелательных условий, требующих действия системы.

Эффективность процедур испытаний и любых других используемых мер должна оцениваться по сравнению со спецификациями требований безопасности по завершении процесса проверки.

Чтобы соответствовать установленному MPLr, методы и меры должны быть выбраны из таблицы 9.

Таблица 9 — Валидация ПО

| Метод/мера | | Ссылка | MPLr = a | MPLr = b, c | MPLr = d | MPLr = e |
|---|---|---|---|---|---|---|
| 1.a | Тестирование сети машины | B.1 | + | + | + | + |
| 1.b | Тестирование оборудования в контуре | B.2 | + | + | + | + |
| 1.c | Тестирование на уровне машины | B.3 | + | + | + | + |
| 2 | Тестирование требований к эффективности защиты [a] | A.6 | ○ | ○ | ○ | + |
| 3 | Лавинное/нагрузочное тестирование | | ○ | ○ | ○ | + |
| Примечание — Описание методов и мер приведено в приложении B. | | | | | | |

Методы испытаний приведены в приложении B.

## 5 Программная параметризация

### 5.1 Общее

Программная параметризация относится к возможности адаптации системы ПО к различным требованиям после завершения разработки путем изменения параметров для изменения функциональных возможностей ПО. Программная параметризация параметров, связанных с безопасностью, должна рассматриваться как часть SRP/CS и должна быть описана в спецификации требований к безопасности ПО.

Программные параметры включают в себя:

- варианты кодирования (например, код страны, левостороннее/правостороннее рулевое управление и т. д.);
- системные параметры (например, значение низких оборотов холостого хода, диаграммы характеристик двигателя и т. д.);
- данные калибровки (например, для конкретной машины, ограничитель для настройки дроссельной заслонки и т. д.).

### 5.2 Целостность данных

Должна поддерживаться целостность данных, используемых для параметризации, и должны предотвращаться несанкционированные модификации. Это достигается путем применения методов или мер для контроля:

- диапазона допустимых входных данных;
- повреждений данных до и после передачи;
- ошибок процесса передачи параметров;
- последствий неполной передачи параметров; и
- последствий сбоев и отказов аппаратных и программных средств инструмента, используемого для параметризации.

### 5.3 Программная проверка параметризации

Следующие действия по проверке должны быть выполнены для параметризации на основе ПО:

- проверка правильности настройки каждого параметра, связанного с безопасностью (минимальные, максимальные и репрезентативные значения);
- подтверждение того, что связанные с безопасностью параметры были проверены на правдоподобность путем использования недопустимых значений, записанных в ПО на этапе конфигурации, для проверки его поведения;
- проверка предотвращения несанкционированного изменения параметров, связанных с безопасностью;

- проверка того, что данные/сигналы для параметризации генерируются и обрабатываются таким образом, что отказы не могут привести к потере функции безопасности.

## 6 Защита передачи связанных с безопасностью сообщений по шинным системам

В настоящем разделе приведены рекомендации по защите передачи связанных с безопасностью сообщений, используемых в SCS, и в связи, которая может иметь место между различными компонентами (например, микроконтроллерами, интеллектуальными датчиками, интеллектуальными исполнительными механизмами) внутри SCS, как показано на рисунке 2.

Примечание 1 — На момент публикации рассматриваются только закрытые шинные системы, в которых изготовитель определил количество и тип участников шины (т. е. устройств, подключенных к шине). Шины данных и адреса, внутренние для внутренних устройств ЦП и ECU, не рассматриваются.

Для контроля указанных в таблице 10 ошибок передачи и уровней эффективности защиты применяются соответствующие методы и меры из таблицы 11.

Таблица 10 — Контроль ошибок передачи и уровни эффективности защиты

| Ошибки передачи | MPLr = a, b, c | MPLr = d, e |
|---|---|---|
| Сбой связи между узлами | Да[a] | Да |
| Фальсификация сообщений | Да[a] | Да |
| Повтор сообщений | Нет | Да |
| Потеря сообщений | Да[a] | Да |
| Вставка сообщений | Нет | Да |
| Неправильная последовательность | Нет | Да |
| Задержка сообщений | Да[a] | Да |
| Блокирование доступа к шине | Да[a] | Да |
| Постоянная передача сообщений | Да[a] | Да |
| [a] Не применимо к системам категории B или 1, если диагностическое покрытие не требуется. | | |

Примечание 2 — Методы могут быть реализованы в таких протоколах, как SAE J1939, для устранения ошибок передачи.


Рисунок 2 — Сеть микроконтроллеров, состоящая из блоков ECU на шине данных

В таблице 11 определены некоторые методы и меры защиты от ошибок передачи путем исключения конкретных последствий ошибок, связанных с передачей или защитой данных, связанных с безопасностью, в SCS. Для защиты от ошибок могут использоваться другие методы и меры.

Таблица 11 — Методы и меры в рамках сети микроконтроллеров

| Методы/меры | | Ошибки передачи (см. раздел 6) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Сбой связи между узлами | Повтор сообщений | Потеря сообщений | Вставка сообщений | Неправильная последовательность | Фальсификация сообщений | Задержка сообщений | Блокирование доступа к шине | Постоянная передача сообщений |
| D.1 | Сообщения о сохранении активности | Да | Нет | Нет | Нет | Нет | Нет | Нет | Да | Да |
| D.2 | Счетчик активности | Нет | Да | Да | Да | Нет | Нет | Нет | Нет | Да |
| D.3 | Проверка циклическим избыточным кодом (CRC) | Нет | Нет | Нет | Нет | Нет | Да | Нет | Нет | Нет |
| D.4 | Порядковый номер | Нет | Да | Да | Да | Да | Нет | Нет | Нет | Да |
| D.5 | Повторение сообщения | Нет | Нет | Да | Нет | Да | Да | Нет | Нет | Нет |
| D.6 | Защитный таймер | Да | Нет | Да | Нет | Нет | Нет | Да | Да | Да |
| D.7 | Шина данных, управляемая по времени | Да | Нет | Нет | Да | Нет | Нет | Да | Нет | Нет |
| D.8 | Защита шины | Нет | Нет | Нет | Нет | Нет | Нет | Нет | Нет | Да |
| D.9 | Мини-интервалы | Нет | Нет | Нет | Нет | Нет | Нет | Да | Нет | Да |

Соответствие требуемому MPLr с учетом методов и мер, реализованных в сетевой связи, может быть проверено посредством расчета гарантии целостности данных, как указано в приложении C. В приложении D содержится дополнительная информация о таблице 11.

## 7 Независимость за счет программного разделения

### 7.1 Общее

Разделение ПО предназначено для того, чтобы помочь разработчику доказать независимость программных компонентов.

Адекватная независимость программных компонентов гарантирована за счет исключения отдельных последствий программных сбоев, нарушающих эту независимость.

С этой целью методы и меры должны применяться с достаточной эффективностью, чтобы:

- управлять опасностями, которые могут возникнуть в подсистемах, чтобы они не могли повлиять на другие подсистемы;
- иметь достаточную независимость от программных компонентов путем разделения ПО.

Для достижения адекватной независимости программных компонентов системные ресурсы должны быть назначены независимым разделам, каждый из которых представляет определенную среду выполнения. Использование разделения ПО не ограничивается сосуществованием ПО с разными MPL в одной и той же среде выполнения. Также могут поддерживаться:

a) внесение изменений в раздел без перепроверки немодифицированных программных разделов;

b) совместное существование ПО различного происхождения (собственной разработки или стороннего изготовителя).

Для программных разделов с MPLr, равным c, d, e, требуется независимость.

Примечание — Разделы могут быть распределены в пределах одного микроконтроллера или распределены между несколькими микроконтроллерами в сети микроконтроллеров.

В зависимости от выбранной архитектуры могут использоваться два подхода:

a) несколько разделов в пределах одного микроконтроллера;

b) несколько разделов в пределах сети ECU.

Концепция разделения ПО и связанные с ней методы и меры для обеспечения независимости компонентов ПО должны учитываться при определении архитектуры ПО.

Та часть ПО, которая реализует поддержку разделения, должна иметь такой же или более высокий MPLa, чем самый высокий MPLr, связанный с программными разделами.

### 7.2 Несколько разделов в одном микроконтроллере

Чтобы гарантировать достаточную независимость программных компонентов в одном микроконтроллере, как показано на рисунке 3, правильное выполнение функции, связанной с безопасностью, должно быть защищено от любого из следующих последствий неисправности:

- повреждения памяти (непреднамеренная запись в память другого раздела).

Примечание 1 — Повреждение памяти относится к отображаемому в память вводу-выводу (I/O), регистрам памяти и т. д.;

- блокировки разделов (из-за взаимоблокировок связи);
- неправильного распределения процессорного времени;
- ошибочного узла связи (отправитель отправляет сообщения не тому получателю или маскируется под отправителя, отличного от него самого);
- повреждения интерфейса ввода-вывода из-за непреднамеренной записи в интерфейс ввода-вывода другого раздела.

Примечание 2 — Повреждение интерфейса ввода-вывода относится только к внешним устройствам.

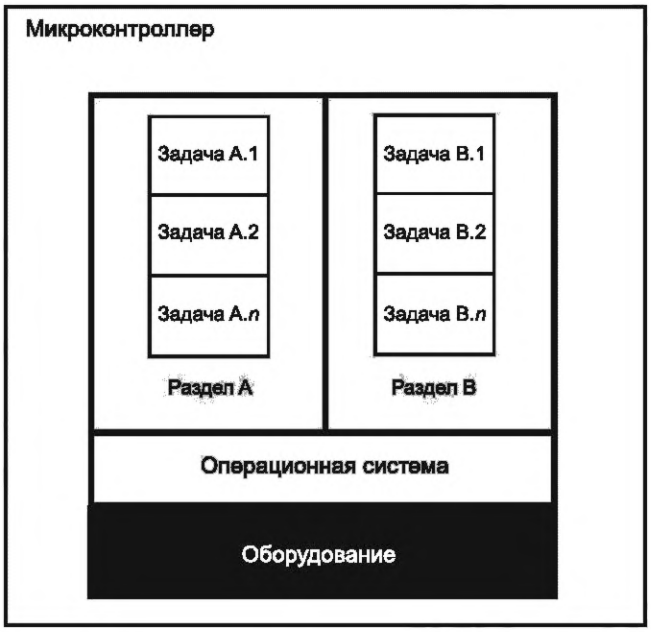



Рисунок 3 — Несколько разделов в одном микроконтроллере

В таблице 12 определены методы и меры внутри микроконтроллера, необходимые для того, чтобы можно было справляться со всеми соответствующими последствиями ошибок, перечисленными выше, чтобы обеспечить достаточную эффективность и достаточную независимость программных компонентов. Дополнительная информация приведена в приложении Е.

Таблица 12 — Методы и меры внутри микроконтроллера

| Методы/меры | | Эффект от ошибки | | | | |
|---|---|---|---|---|---|---|
| | | Повреждение памяти | Блокировка разделов | Неправильное выделение процессорного времени | Связь с неправильным узлом | Повреждение интерфейсов ввода-вывода |
| E.1 | Однозначный двунаправленный коммуникационный объект | Нет | Нет | Нет | Да | Нет |
| E.2 | Строго два объекта однонаправленной связи | Нет | Нет | Нет | Да | Нет |
| E.3 | Идентификаторы для идентификации, подтверждения или того и другого. | Нет | Нет | Нет | Да | Нет |
| E.4 | Асинхронная передача данных | Нет | Да | Нет | Нет | Нет |
| E.5 | Строгое планирование на основе приоритетов | Нет | Нет | Да | Нет | Нет |
| E.6 | Метод разделения времени | Нет | Нет | Да | Нет | Нет |
| E.7 | Механизмы защиты памяти | Да | Нет | Нет | Нет | Да |
| E.8 | Проверка важных для безопасности данных | Да | Нет | Нет | Нет | Нет |
| E.9 | Статический анализ | Да | Нет | Нет | Нет | Нет |
| E.10 | Статическое распределение | Да | Да | Да | Да | Да |

### 7.3 Несколько разделов в пределах сети ECU

Требования, гарантирующие адекватную связь по сети, когда независимость программных компонентов реализована посредством сети микроконтроллеров, приведены в разделе 6.

## 8 Информация по эксплуатации

### 8.1 Общее

Информация по эксплуатации должна быть приведена в соответствие с ISO 12100:2010, 6.4.

### 8.2 Руководство по эксплуатации

Информация по эксплуатации должна быть включена в руководство по эксплуатации машины (руководство оператора).

Информация должна быть приведена в соответствие с ISO 6750-1 и может включать следующее:

- описания символов, отображаемых оператору, и требуемых действий;
- описания сообщений об ошибках, которые отображаются оператору, и требуемых действий;
- описание предупредительных сообщений, отображаемых оператору, и требуемых действий;
- описание процедур калибровки, которые должен выполнять оператор.

## Приложение А
## (справочное)

## Описание программных методов/мер

### А.1 Спецификация требований на естественном языке

Спецификация требований безопасности ПО должна включать описание проблемы на естественном языке и, при необходимости, дополнительные неформальные методы, такие как рисунки и диаграммы.

### А.2 Автоматизированные инструменты составления спецификации

Следует использовать автоматизированные инструменты для облегчения автоматического обнаружения неоднозначности и обеспечения полноты в полуформальных и формальных методах спецификации. Инструмент должен создавать спецификации в виде базы данных, которую можно автоматически проверять для оценки согласованности и полноты. В целом этот метод поддерживает не только составление спецификации, но также проектирование и другие этапы жизненного цикла проекта.

### А.3 Неформальные методы

Неформальные методы должны предоставлять средства описания разработки системы на каком-либо этапе ее развития, т. е. составления спецификации, проектирования или кодирования, как правило, с помощью естественного языка, диаграмм, рисунков и т. д.

### А.4 Полуформальные методы

Полуформальные методы проектирования должны выражать концепцию, спецификацию или разработку недвусмысленно и последовательно, чтобы можно было обнаружить некоторые ошибки и упущения.

В некоторых случаях описание должно анализироваться машиной или анимироваться для отображения различных аспектов поведения системы. Анимация может дать дополнительную уверенность в том, что система соответствует требованиям.

Примеры полуформальных методов включают, но не ограничиваются: диаграммы потоков данных, псевдокод, инструменты моделирования, конечное состояние машины и диаграммы переходов состояний.

### А.5 Формальные методы

Формальные методы дают средства разработки, описывающие систему в строгих обозначениях, которые должны быть подвергнуты математическому анализу для выявления различных классов несоответствий или ошибок.

Более того, в некоторых случаях описание должно анализироваться машиной со строгостью, аналогичной проверке синтаксиса компилятором исходной программы, или анимироваться для отображения различных аспектов поведения описываемой системы. Анимация может дать дополнительную уверенность в том, что система соответствует реальным требованиям, а также формально заданным требованиям, поскольку она улучшает распознавание заданного поведения человеком.

Формальный метод обычно предлагает систему записи (обычно используется какая-то форма дискретной математики), способ получения описания в этой системе записи и различные формы анализа для проверки различных свойств описания.

### А.6 Прослеживаемость программного обеспечения безопасности

Чтобы гарантировать, что ПО, полученное в результате разработки, отвечает требованиям правильной работы системы, связанной с обеспечением безопасности, необходима согласованность между этапами разработки ПО. Прослеживаемость между действиями является ключевой концепцией, которая подтверждает, что:

- решения, принятые на более ранней стадии, адекватно реализуются на более поздних стадиях (прямая прослеживаемость),
- решения, принятые на более позднем этапе, необходимы для реализации более ранних решений (обратная прослеживаемость).

Прямая прослеживаемость в целом связана с проверкой того, что требование адекватно учтено на более поздних стадиях разработки ПО. Прямая прослеживаемость важна на нескольких этапах разработки программного обеспечения безопасности:

- от требований безопасности системы к требованиям безопасности ПО;
- от спецификации требований безопасности ПО к архитектуре ПО;
- от спецификации требований безопасности к ПО до проектирования ПО;
- от спецификации разработки ПО к спецификациям модулей и интеграционных тестов;
- от требований к разработке системы и ПО для интеграции аппаратного/программного обеспечения до спецификаций тестирования интеграции аппаратного/программного обеспечения;
- от спецификации требований к безопасности ПО до плана проверки безопасности ПО;
- от спецификации требований безопасности ПО до плана модификации ПО (включая повторную проверку и повторную валидацию);
- от спецификации проекта ПО до плана проверки ПО (включая проверку данных).

Обратная прослеживаемость в широком смысле связана с проверкой того, что каждое решение по реализации (интерпретируемое в широком контексте и не ограничивающееся реализацией кода) четко обосновано некоторым требованием. Если это обоснование отсутствует, то реализация содержит что-то ненужное, что увеличивает сложность, но не обязательно отвечает каким-либо реальным требованиям системы, связанной с безопасностью. Обратная прослеживаемость важна на нескольких этапах разработки программного обеспечения безопасности:

- от требований безопасности к предполагаемым потребностям безопасности;
- от архитектуры ПО до спецификации требований безопасности ПО;
- от детального проекта ПО до архитектуры ПО;
- от программного кода до рабочего проекта ПО;
- от плана проверки безопасности ПО до спецификации требований безопасности ПО;
- от плана модификации ПО к спецификации требований безопасности ПО;
- от плана проверки ПО (включая проверку данных) до спецификации проекта ПО.

### А.7 Пошаговое рассмотрение

Пошаговое рассмотрение — это систематическая неформальная проверка, используемая для рассмотрения какого-либо аспекта проекта. Во время пошагового рассмотрения автор артефакта предоставляет пошаговый отчет одному или нескольким проверяющим.

Цель пошагового рассмотрения состоит в том, чтобы создать общее понимание артефакта и выявить любые ошибки, дефекты, несоответствия или проблемы в артефакте. Пошаговое рассмотрение — менее строгий метод, чем инспекция.

Пошаговое рассмотрение с экономической точки зрения следует использовать для обнаружения ошибок ПО на как можно более ранних этапах разработки. Оно состоит из команды проверяющих, выбирающей небольшой набор тестовых случаев, репрезентативных наборов входных данных и соответствующих ожидаемых результатов для программы. Затем тестовые данные вручную отслеживаются по логике программы.

### А.8 Инспекция

Инспекция — это систематический, формальный метод проверки, используемый для рассмотрения какого-либо аспекта проекта. Во время инспекции артефакт оценивается одним или несколькими проверяющими, чтобы убедиться, что он соответствует требованиям. Инспекцию организует и проводит руководитель разработки. Автор артефакта участвует в инспекции, но не может руководить процессом.

Инспекция проекта может выполняться на уровне проекта, чтобы выявить дефекты в разработке ПО. Инспекция проекта — это формальное, документированное, всестороннее и систематическое исследование разработки ПО для оценки требований к разработке, способности проекта соответствовать этим требованиям, выявления проблем и предложения решений. Такой метод в первую очередь предназначен для проверки работы разработчиков и должен рассматриваться как деятельность по подтверждению и уточнению.

Инспекция программного кода может выполняться на уровне кодирования для выявления дефектов в программном элементе. Формальная инспекция — это структурированный процесс проверки программного материала, который выполняется коллегами автора кода, для обнаружения дефектов и предоставления возможности улучшить ПО. Автор кода не должен принимать участия в процессе инспекции, кроме как информировать проверяющих на стадии ознакомления. Формальные инспекции могут проводиться на конкретных элементах ПО, произведенных на любом этапе жизненного цикла разработки ПО.

Перед проведением инспекции проверяющие должны ознакомиться с материалами, подлежащими проверке. Роли проверяющих в процессе инспекции должны быть четко определены. Должна быть подготовлена программа проверки. Входные и выходные критерии должны быть определены на основе свойств, необходимых для программного элемента. Входные критерии — это критерии или требования, которые должны быть выполнены до проведения инспекции. Выходные критерии — это критерии или требования, которые должны быть выполнены для завершения конкретного процесса.

Во время проверки результаты инспекции должны быть официально зарегистрированы модератором, роль которого заключается в содействии инспекции. Консенсус по результатам должен быть достигнут всеми инспекторами.

Дефекты следует классифицировать как:

a) требующие исправления до принятия, или

b) требующие исправления к определенному времени/событию.

Выявленные дефекты должны быть переданы автору кода для последующего устранения после завершения инспекции. В зависимости от количества и объема выявленных дефектов модератор может определить необходимость дополнительной инспекции программного материала.

### А.9 Средства автоматизированного проектирования

Этот тип инструмента может способствовать более систематическому выполнению процедуры проектирования за счет использования автоматических элементов, доступных для структурирования ПО.

Средства автоматизированного проектирования следует использовать при проектировании как аппаратного, так и программного обеспечения, если они доступны и оправданы сложностью системы. Корректность таких инструментов должна быть продемонстрирована специальными испытаниями, историей успешного использования

или независимой проверкой их выходных данных для конкретной разрабатываемой системы, связанной с безопасностью.

Инструменты поддержки следует выбирать по степени их интеграции. В этом контексте инструменты являются интегрированными, если они работают совместно, так что выходные данные одного инструмента имеют подходящее содержание и формат для автоматического ввода в последующий инструмент, что сводит к минимуму возможность человеческой ошибки при обработке промежуточных результатов.

Примечание — Интегрированная среда разработки (IDE) может быть использована для предоставления всесторонних средств для разработки ПО. Обычно они состоят из редактора исходного кода, средств автоматизации сборки и отладчика. Большинство из них имеют функции автоматического создания кода; некоторые содержат компилятор, интерпретатор или и то, и другое.

### А.10 Показатели безопасности в режиме реального времени

Реализация отказоустойчивости в критически важных для безопасности системах реального времени с предсказуемым поведением достигается за счет следующих решений:

- В циклическом режиме с гарантированным максимальным временем цикла связь между узлами осуществляется с использованием запускаемого по времени протокола класса С (TTP/C) в соответствии со статическим расписанием, при котором решается, когда передать сообщение и является ли полученное сообщение релевантным для конкретного электронного модуля или нет. Доступ к шине контролируется схемой циклического множественного доступа с временным разделением (TDMA), основанной на глобальном понятии времени.

- В системе с синхронизируемой по времени архитектурой (TTA) все системные действия инициируются и основаны на развитии глобально синхронизированной временной базы. Каждому приложению назначается фиксированный временной интервал на управляемой по времени шине, при этом сообщения, которыми обмениваются задания каждого приложения таким образом, могут пересылаться только в соответствии с определенным расписанием.

В системе, управляемой событиями, следует учитывать действия, вызванные произвольными событиями в непредсказуемое время.

Примечание — Другими протоколами с запуском по времени являются FlexRay и TT-Ethernet (Ethernet с запуском по времени).

### А.11 Правила проектирования

Следует использовать стандарты программирования для облегчения проверки полученного кода. Детальные правила должны быть полностью согласованы перед кодированием. Эти правила обычно требуют:

- стандарты детализации модульности, например интерфейсы между программными модулями, размеры программных модулей;

- ограниченное использование инкапсуляции, наследования (ограниченное по глубине) и полиморфизма в случае объектно-ориентированных языков;

- ограниченное использование или избегание определенных языковых конструкций, таких как «переход», «эквивалентность», динамические объекты, динамические данные, структуры динамических данных, рекурсия, указатели и выходы;

- ограничения на прерывания, разрешенные во время выполнения критического с точки зрения безопасности кода;

- макет кода (листинг);

- отсутствие безусловных переходов (например, «go-to») в программах на языках высокого уровня.

Эти правила упрощают тестирование, проверку, оценку и обслуживание программных модулей, поэтому они должны учитывать доступные инструменты в конкретных анализаторах.

Использование прерываний должно быть ограничено. Прерывания могут использоваться, если они упрощают систему. Программная обработка прерываний должна быть запрещена во время выполнения системных задач, связанных с безопасностью. Если используются прерывания, то элементы ПО, которые не могут быть прерваны, должны иметь указанное максимальное время вычисления, чтобы можно было рассчитать максимальное время, в течение которого прерывание запрещено. Использование и блокировка прерываний должны быть строго задокументированы.

В прикладном ПО арифметика указателя должна использоваться на уровне исходного кода только в том случае, если тип данных указателя и диапазон значений проверяют (чтобы убедиться, что ссылка указателя находится в правильном адресном пространстве).

Если используется рекурсия, должны быть установлены четкие критерии допустимой глубины рекурсии.

### А.12 Динамические переменные или объекты без онлайн-проверки

Поскольку количество динамических переменных и объектов, а также существующее свободное пространство памяти для размещения новых динамических переменных или объектов зависит от состояния системы в момент выделения ресурсов, при размещении или использовании переменных или объектов возможны программные

сбои. Например, когда объем свободной памяти в месте, выделенном системой, недостаточен, содержимое памяти другой переменной может быть непреднамеренно перезаписано. Если динамические переменные или объекты не используются, этих ошибок ПО можно избежать.

Ограничения на использование динамических объектов необходимы там, где динамическое поведение не может быть точно предсказано статическим анализом (т.е. до выполнения программы), и поэтому нельзя гарантировать предсказуемое выполнение программы. После использования динамической переменной или объекта (например, после выхода из подпрограммы) вся память, которая была выделена для него, должна быть освобождена.

### А.13 Динамические переменные или объекты с онлайн-проверкой

Онлайн-проверка во время выполнения определяет, не затронуты ли существующие переменные, данные или код распределения ресурсов. Если выделение ресурсов не разрешено (например, если памяти по определенному адресу недостаточно), должны быть предприняты соответствующие действия. Это альтернативный метод статического размещения всех необходимых переменных и объектов.

### А.14 Модульный подход

Модульный подход (модуляризация) предполагает ряд правил для этапов проектирования, кодирования и сопровождения программного проекта. Эти правила различаются в зависимости от используемого метода проектирования.

Для методов, описанных в этом документе, применяют следующие положения:

- Программный модуль должен выполнять одну четко определенную системную задачу или функцию.
- Связи между программными модулями должны быть ограничены и строго определены.
- Должны быть созданы наборы подпрограмм, обеспечивающие несколько уровней программных модулей.
- Размер программного модуля должен быть ограничен заданным значением (обычно от двух до четырех размеров экрана, но в соответствии с предварительно определенным стандартом программирования).
- Программные модули должны иметь один вход и один выход. Если требуется более одной точки входа или выхода, причина для этой стратегии должна быть задокументирована в ПО; это обеспечивает правильное выполнение и ремонтопригодность ПО.
- Программные модули должны взаимодействовать с другими программными модулями через их интерфейсы. Там, где используются глобальные или общие переменные, они должны быть хорошо структурированы, доступ должен контролироваться, и их использование должно быть оправдано в каждом конкретном случае.
- Все интерфейсы программных модулей должны быть полностью задокументированы.
- Интерфейс любого программного модуля должен содержать только те параметры, которые необходимы для его функционирования.
- Показатели сложности следует использовать для прогнозирования атрибутов программ на основе свойств самого ПО или истории его разработки или тестирования. Программные инструменты необходимы для оценки большинства мер. Некоторые из показателей, которые могут быть применены, включают, например теоретическую сложность графа, доступность, количество операторов и операндов, а также количество входов и выходов на программный модуль.
- Следует использовать сокрытие или инкапсуляцию информации, чтобы предотвратить непреднамеренный доступ к данным или процедурам и таким образом поддерживать структуру программы. Данные, глобально доступные всем программным элементам, могут быть случайно или неправильно изменены любым из этих элементов. Любые изменения в этих структурах данных могут потребовать детального изучения кода и обширных модификаций.

Сокрытие информации является общим подходом к минимизации этих трудностей. Ключевые структуры данных «скрыты», и ими можно манипулировать только с помощью определенного набора процедур доступа. Это позволяет изменять внутренние структуры или добавлять дополнительные процедуры, не влияя на функциональное поведение остального ПО.

### А.15 Структурированное программирование

Структурированное программирование следует использовать для разработки и реализации программы таким образом, чтобы ее можно было анализировать без выполнения.

Чтобы свести к минимуму структурную сложность, необходимо выполнить следующее:

a) разделить программу на соответствующие небольшие программные модули, гарантируя, что все взаимодействия будут явными;

b) составить поток управления программным модулем, используя структурированные конструкции (т. е. последовательности, итерации и операторы выбора);

c) количество возможных путей прохождения через программный модуль должно быть небольшим, а отношение между входными и выходными параметрами — максимально простым;

d) избегать сложных ветвлений, в частности избегать безусловных переходов (go-to) в языках более высокого уровня;

e) по возможности использовать входные параметры в качестве ограничений цикла для выполнения ветвления и избегать использования вычислений в качестве основы для решений ветвления и цикла.

Возможности языка программирования, которые поощряют описанный выше подход, следует использовать вместо других функций, которые (предположительно) более эффективны, за исключением тех случаев, когда эффективность имеет абсолютный приоритет.

### А.16 Защитное программирование

Во время программирования можно использовать различные методы для проверки аномалий управления или данных. Используемые методы следует систематически применять на протяжении всего программирования системы, чтобы уменьшить вероятность ошибочной обработки данных. Есть две пересекающиеся области защитных методов. Внутреннее безошибочное ПО разработано с учетом собственных конструктивных недостатков. Эти недостатки могут быть связаны с ошибками в разработке или кодировании, или с ошибочными требованиями. Методы включают следующее:

- проверку диапазона переменных;
- проверку значений на правдоподобие;
- тип, размерность и диапазон контрольных параметров процедур при входе в процедуру.

Данный набор защитных методов помогает гарантировать, что числа, которыми оперирует программа, являются допустимыми и для функции программы, и в смысле физического значения переменных.

Параметры «только для чтения» и «для чтения и записи» должны быть разделены, а доступ к ним должен проверяться. Функции должны обрабатывать все параметры как доступные только для чтения. Буквенные константы не должны быть доступны. Это помогает обнаружить случайную перезапись или ошибочное использование переменных. Отказоустойчивое ПО предназначено для учета сбоев в своей собственной среде или использования за пределами номинальных или ожидаемых условий и ведет себя заранее определенным образом.

Методы включают следующее:

- проверку входных переменных и промежуточных переменных, имеющих физическое значение, на правдоподобие;
- проверку влияния выходных переменных, предпочтительно путем непосредственного наблюдения за связанными изменениями состояния системы;
- проверку ПО, его конфигурации, в том числе как наличия и доступности ожидаемого оборудования, так и полноты самого ПО; это особенно важно для сохранения целостности после процедур технического обслуживания.

Некоторые защитные методы программирования, такие как проверка последовательности потока управления, также справляются с внешними отказами.

### А.17 Использование надежных/проверенных элементов программного обеспечения

Надежные/проверенные программные модули и программные компоненты могут быть повторно использованы в новых приложениях. Это позволяет разработчику воспользоваться преимуществами проектов, которые не были официально или тщательно проверены, но для которых имеется значительный опыт эксплуатации, тем самым уменьшая количество проверок, необходимых для проектов программных модулей и аппаратных компонентов в новых приложениях.

Программный компонент или модуль является в достаточной степени надежным, если его соответствие MPLr уже подтверждено или если он соответствует следующим критериям:

- функция, связанная с безопасностью, должна работать не менее одного года или 1000 часов без изменения спецификации;
- предыдущая история эксплуатации программного модуля связана с предполагаемой целью в новом приложении, таким образом устанавливая уверенность в пригодности программного модуля для нового приложения;
- отсутствие отказов функций, связанных с безопасностью, в повторно используемом ПО за всю историю эксплуатации.

Следует проводить тщательную оценку каждой функции, поскольку функция, не связанная с безопасностью в одном приложении, может быть функцией, связанной с безопасностью в другом приложении.

Для проверки того, что компонент или программный модуль соответствует вышеуказанным критериям, должны быть доступны следующие процедуры для предоставления доказательств, подлежащих документированию:

- идентификация каждой системы и ее компонентов, включая номера версий программного и аппаратного обеспечения, используемого в процессе проверки;
- выявление и отбор достаточной выборки пользователей и времени использования приложения (т. е. один год или 1000 часов работы);
- обнаружение и регистрация отказов с соответствующими корректирующими действиями.

Если используются альтернативные процедуры для сбора доказательств пригодности, должно быть предоставлено обоснование, чтобы продемонстрировать, что безопасность программного компонента или программного модуля все еще достигнута.

Описанный подход можно распространить на поставляемые сложные электронные компоненты (например, от поставщика к покупателю), когда немодифицированное ПО интегрируется с немодифицированным оборудованием и когда ранее описанные требования выполняются и документируются.

### А.18 Подходящий язык программирования

Цель состоит в том, чтобы выбрать язык программирования, который максимально поддерживает требования настоящего стандарта, в частности защитное программирование, структурированное программирование и, при наличии, операторы контроля отсутствия ошибок. Выбранный язык программирования должен вести к легко проверяемому коду и облегчать разработку, проверку и обслуживание программы. Широко используемые языки или их подмножества предпочтительнее языков специального назначения. Языки низкого уровня, в частности языки ассемблера, представляют проблемы из-за их ориентированности на конкретные процессоры/платформы.

Желательным свойством языка является то, что его разработка и использование должны приводить к программам, выполнение которых предсказуемо без запуска кода. В правильно определенном языке программирования существует подмножество, обеспечивающее предсказуемость выполнения программы. Это подмножество не может (в общем случае) быть определено статически, хотя многие статические ограничения могут помочь в обеспечении предсказуемого выполнения.

Примечание — См. IEC 61508-7: 2010, таблица С.1, где приведен список языков программирования.

### А.19 Поддержка языковых подмножеств

Использование языкового подмножества снижает вероятность внесения ошибок программирования и увеличивает вероятность обнаружения любых оставшихся ошибок ПО. Следует изучить язык программирования, чтобы определить программные конструкции, которые подвержены ошибкам либо трудны для анализа (например, с использованием методов статического анализа). Затем эти программные конструкции должны быть исключены и определено языковое подмножество. Кроме того, следует задокументировать, почему конструкции, используемые в языковом подмножестве, безопасны.

Примечание — Руководства MISRA ‘C’ (Ассоциации по надежности ПО автомобильной промышленности) являются примером руководств по разработке ПО, которые определяют подмножество языка и, как правило, направлены на обеспечение безопасности кода, защищенности, переносимости и надежности.

### А.20 Инструменты программирования с повышенной надежностью в результате использования или валидации

Инструменты программирования, которые проверены в использовании или прошли валидацию, должны применяться во избежание трудностей из-за сбоев инструментов, которые могут возникнуть во время разработки, проверки и обслуживания программных модулей. Следует избегать инструментов программирования без истории эксплуатации или с какими-либо серьезными известными ошибками, если нет документированных гарантий их правильной работы. Если в инструменте программирования обнаружены небольшие недостатки, соответствующие языковые конструкции записывают и их тщательно избегают во время реализации проекта, связанного с безопасностью.

Следует утверждать, что инструмент программирования имеет повышенную надежность в результате использования только в том случае, если представлены доказательства следующего:

a) инструмент программирования использовался ранее для той же цели с сопоставимыми вариантами использования, сопоставимой определенной операционной средой и с аналогичными функциональными ограничениями;

b) обоснование повышенной надежности в результате использования основано на достаточных и адекватных данных (по крайней мере, разработан один проект по созданию ПО со сроком эксплуатации не менее одного года или 1000 часов);

c) спецификация программного средства не изменилась;

d) случаи возникновения неисправностей и соответствующие ошибочные выходные данные инструмента программирования, полученные в ходе предыдущих разработок, накапливаются систематически.

Следует утверждать, что инструмент программирования имеет повышенную надежность в результате валидации только в том случае, если предоставлены свидетельства, показывающие, что валидация соответствует следующим критериям:

a) меры валидации демонстрируют, что инструмент программирования соответствует установленным требованиям (например, стандарт для языка программирования помогает определить требования для валидации соответствующего компилятора);

b) сбои и соответствующие им ошибочные выходные данные инструмента программирования, возникающие во время валидации, были проанализированы вместе с информацией об их возможных последствиях и с мерами по их предотвращению или обнаружению;

c) исследована реакция инструмента программирования на аномальные (не предусмотренные типом приложения) условия работы.

### А.21 Сертифицированные инструменты программирования и сертифицированные интерпретаторы

Инструменты программирования необходимы, чтобы помочь разработчикам на разных этапах разработки ПО. По возможности инструменты программирования должны быть сертифицированы, чтобы можно было предположить некоторый уровень уверенности в правильности результатов.

Сертификацию инструмента программирования обычно проводит независимая третья сторона в соответствии с независимо установленными критериями, как правило, национальными или международными стандартами. В идеальном случае инструменты программирования, используемые на всех этапах разработки (составление спецификации, проектирование, кодирование, тестирование и проверка), а также инструменты программирования, используемые в управлении конфигурацией, должны подлежать сертификации.

Инструменты программирования и интерпретаторы обычно сертифицируются только в соответствии со стандартами языка программирования или процесса; их обычно никак не сертифицируют в отношении безопасности.

### А.22 Анализ граничных значений

Анализ граничных значений следует использовать для обнаружения ошибок ПО, возникающих в пределах или границах параметров. Входная область программы делится на несколько входных классов (подмножеств входных значений и их комбинаций) в соответствии с отношением эквивалентности (каждое значение класса приводит к одному и тому же выходному значению). Тесты должны охватывать граничные и предельные значения классов. Тесты проверяют, совпадают ли границы входной области спецификации с границами программы.

Обычно границы для ввода имеют прямое соответствие с границами для диапазона вывода. Тестовые случаи должны быть написаны так, чтобы принудительно привести вывод к его граничным значениям. Также могут быть использованы тестовые примеры, которые приводят к превышению выходных значений граничных значений спецификации.

Если вывод представляет собой последовательность данных (например, распечатанную таблицу), особое внимание следует уделить первому и последнему элементам, а также спискам, не содержащим элементов, состоящим из одного элемента и двух элементов.

### А.23 Анализ потока управления

Анализ потока управления следует использовать для обнаружения ненадежных и потенциально неправильных структур программ.

Анализ потока управления — это метод статического тестирования для поиска областей кода, которые не соответствуют сложившимся практикам программирования. Анализируемая программа создает ориентированный граф, который может быть дополнительно проанализирован на предмет:

- недоступного кода, т.е. безусловных переходов, оставляющих блоки кода недоступными;
- запутанного кода, где, в отличие от хорошо структурированного кода с управляющим графом, сводимым последовательными уточнениями графа к одному узлу, слабо структурированный код может быть сведен только к блоку, состоящему из нескольких узлов.

### А.24 Анализ потока данных

Анализ потока данных следует использовать для обнаружения ненадежных и потенциально неправильных структур программ.

Анализ потока данных — это метод статического тестирования, который объединяет информацию, полученную в результате анализа потока управления, с информацией о том, какие переменные считываются или записываются в разных частях кода.

Анализ может проверять следующие типы переменных:

- те, которые можно прочитать до того, как им будет присвоено значение, чего можно избежать, всегда присваивая значение при объявлении новой переменной;
- написанные более одного раза без прочтения, что может указывать на пропущенный код;
- написанные, но никогда не прочитанные, что может указывать на избыточный код.

Аномалия потока данных не всегда напрямую связана с ошибкой программы; однако, если избежать аномалий, код с меньшей вероятностью будет содержать ошибки.

### А.25 Выполнение тестового примера из анализа граничных значений

Выполнение тестового примера из анализа граничных значений (см. А.22) следует использовать для обнаружения программных ошибок, возникающих при предельных или граничных значениях параметров.

### А.26 Функциональное тестирование и тестирование методом «черного ящика»

Функциональное тестирование следует использовать для выявления сбоев на этапах составления спецификации и разработки, а также для предотвращения сбоев во время реализации и интеграции программного и аппаратного обеспечения.

Во время функционального тестирования следует проверять, были ли достигнуты заданные параметры системы и были ли предоставлены входные данные, которые характерны для ожидаемого применения. Выходные данные сравнивают с указанными в спецификациях. Отклонения от спецификации и признаки неполного соответствия спецификации должны быть задокументированы. Функциональное тестирование электронных компонентов, разработанных для многокомпонентной архитектуры, обычно включает тестирование изготовленных компонентов совместно с предварительно проверенными взаимосвязанными компонентами.

Кроме того, рекомендуется, чтобы изготовленные компоненты были проверены в сочетании с другими взаимосвязанными компонентами, обрабатываемыми одновременно, чтобы выявить программные ошибки при совместном режиме использования, которые в противном случае остались бы скрытыми.

### А.27 Тестирование на основе структуры

Тестирование на основе структуры проверяет определенные подмножества структуры программы. На основе анализа программы выбирают такой набор входных данных, чтобы отрабатывался большой (и часто заранее заданный целевой) процент программного кода. Степени охвата кода различаются следующим образом в зависимости от требуемого уровня строгости тестирования.

Во всех случаях целью должно быть 100 % выбранного показателя охвата; если невозможно достичь 100 % охвата, причины, по которым 100 %-ный охват не может быть достигнут, должны быть задокументированы в отчете об испытаниях (например, защитный код, который можно ввести только в случае возникновения аппаратной проблемы). Методы в следующем списке широко поддерживаются инструментами тестирования:

- Охват точек входа (граф вызовов): убедитесь, что каждая подпрограмма (подмаршрут или функция) была вызвана хотя бы один раз (это минимальный охват структурного измерения).

В объектно-ориентированных языках может быть несколько подпрограмм с одним и тем же именем, которые применяются к разным вариантам полиморфного типа (переопределяющие подпрограммы), которые можно вызывать с помощью динамической диспетчеризации. В этих случаях каждая такая переопределяющая подпрограмма должна быть протестирована.

- Операторы: убедитесь, что все операторы в коде были выполнены хотя бы один раз.
- Ветвление: необходимо проверить все ветви каждого ветвления. Это может быть непрактично для некоторых типов защитного кода.

### А.28 Классы эквивалентности и тестирование входных разделов

Классы эквивалентности и тестирование входных разделов позволяют адекватно тестировать ПО при минимуме тестовых данных. Тестовые данные получаются путем выбора разделов входного домена, необходимых для тестирования ПО.

Эта стратегия тестирования основана на отношении эквивалентности входных данных, которое определяет раздел входной области. Тестовые случаи выбираются с целью охвата всех ранее указанных разделов. Из каждого класса эквивалентности используется не менее одного теста.

Есть две основные возможности для разделения входных данных:

- классы эквивалентности, производные от спецификации, — интерпретация спецификации может быть ориентирована на ввод (например, выбранные значения обрабатываются одинаково) либо на вывод (например, набор значений приводит к одному и тому же функциональному результату);
- классы эквивалентности, производные от внутренней структуры программы, — результаты классов эквивалентности определяются из статического анализа программы, например набор значений, ведущих к одному и тому же исполняемому пути.

### А.29 Выполнение тестового примера, сгенерированного на основе модели

Выполнение тестовых примеров на основе создания моделей тестовых сценариев призвано облегчить эффективную автоматическую генерацию тестовых сценариев на основе системных моделей и генерировать наборы тестов с высокой повторяемостью.

Тестирование на основе моделей (MBT) — это подход, при котором общие задачи тестирования, такие как создание тестовых примеров (TCG) и оценка результатов тестирования, основаны на модели тестируемой системы (приложения) (SUT).

Кроме того, тестирование на основе моделей можно сочетать с измерением тестового охвата на уровне исходного кода; функциональные модели могут быть основаны на существующем исходном коде.

Поскольку тестирование является дорогостоящим, существует огромный спрос на инструменты автоматического создания тестовых сценариев.

Таким образом, тестирование на основе моделей в настоящее время является очень активной областью исследований, в результате чего появилось большое количество доступных инструментов TCG. Эти инструменты обычно извлекают набор тестов из поведенческой части модели, гарантируя соответствие определенным требованиям охвата.

Модель представляет собой абстрактное, частичное представление желаемого поведения SUT. На основе модели создаются тестовые модели, образующие абстрактный набор тестов. Тестовые случаи извлекают из этого абстрактного набора тестов и выполняют в системе; тесты также можно запускать для модели системы.

Тестирование на основе моделей в последнее время специально нацелено на критически важную область безопасности; это позволяет заранее выявить неясности в спецификациях и дизайне. Тестирование на основе моделей также дает возможность автоматически генерировать множество неповторяющихся эффективных тестов, оценивать наборы регрессионных тестов, оценивать надежность и качество ПО, а также упрощает обновление наборов тестов.

### А.30 Тестирование производительности

Тестирование производительности следует использовать для того, чтобы убедиться, что работоспособность системы достаточна для выполнения заданных требований.

Спецификация должна включать требования к пропускной способности и времени отклика для функций, связанных с безопасностью. Спецификация требований также должна включать ограничения на использование общих ресурсов системы. Предлагаемый проект системы сравнивается с заявленными требованиями путем:

- создания модели системных процессов и их взаимодействия;
- определения использования ресурсов каждым процессом (стек, процессорное время, полоса пропускания связи, устройства хранения и т. д.);
- определения распределения требований, предъявляемых к системе при средних и наихудших условиях; и
- вычисления средней и наихудшей пропускной способности и времени отклика для отдельных функций системы.

Можно выполнить тестирование времени отклика и ограничений памяти, чтобы убедиться, что система соответствует требованиям к времени и памяти. Время отклика — это общее время, которое проходит с момента отправки пользователем/системой запроса до момента получения ответа. При тестировании времени отклика необходимо выяснить, как приложение обрабатывает все запросы и как время отклика увеличивается/уменьшается с увеличением нагрузки и выполнения по времени. Более того, тестирование ограничений памяти проверяет, что вся память инициализируется перед ее использованием и блокируется для использования другими процессами. Анализ проводится для определения потребностей распределения памяти в средних и наихудших условиях. Этот анализ требует оценок использования ресурсов и затраченного времени каждой функции системы. Эти оценки можно получить несколькими способами; например, сравнением с существующей системой или прототипированием и сравнительным анализом систем, критичных ко времени.

Тестирование требований к эффективности защиты выполняют для установления доказуемых требований к эффективности защиты программной системы. Выполняется анализ спецификаций системных и программных требований для определения всех общих и конкретных, явных и неявных требований к эффективности защиты. Каждое требование к эффективности защиты проверяют, в свою очередь, чтобы определить:

- критерии успеха, которые должны быть достигнуты;
- могут ли быть достигнуты значения по критериям успеха;
- потенциальную точность таких значений;
- этапы проекта, на которых можно оценить значения; и
- этапы проекта, на которых может быть выполнена оценка значений.

Лавинное/стресс-тестирование может быть выполнено, чтобы нагрузить тестовый объект исключительно высокой рабочей нагрузкой, чтобы показать, что тестовый объект выдерживает нормальные рабочие нагрузки. Это помогает разработчикам определить, достаточно ли работоспособна система, если нагрузка значительно превышает ожидаемый максимум.

Этот тип теста обычно используется для понимания верхних пределов пропускной способности системы. Существует множество условий испытаний, применимых к лавинным/нагрузочным испытаниям, включая следующие:

- При работе в режиме опроса тестовый объект получает гораздо больше входных значений в единицу времени, чем в обычных условиях.
- При работе по заявкам количество требований в единицу времени к объекту контроля превышает нормальные условия.
- Если большую роль играет размер базы данных, то он увеличен сверх обычных условий.
- Влияющие устройства настроены на максимальную или минимальную скорость соответственно.
- В экстремальных случаях все воздействующие факторы, насколько это возможно, ставятся в граничные условия одновременно.

В этих условиях тестирования можно оценить поведение тестируемого объекта во времени, наблюдать влияние изменений нагрузки и проверить правильность размеров внутренних буферов или динамических переменных, массивов данных и т. д.

### А.31 Тестирование интерфейса модуля программного обеспечения

Тестирование интерфейсов модулей используется для выявления ошибок в интерфейсах программных компонентов и модулей.

Возможны несколько уровней детализации или полноты тестирования. Наиболее важными уровнями являются:

- тестирование всех переменных интерфейса в их экстремальных значениях;
- тестирование всех переменных интерфейса по отдельности в их экстремальных значениях с другими переменными интерфейса в нормальных значениях;
- тестирование всех значений домена каждой переменной интерфейса при нормальных значениях других переменных;
- тестирование всех значений всех переменных в комбинации (это возможно только для небольших интерфейсов);
- заданные условия тестирования, относящиеся к каждому вызову каждой подпрограммы.

Эти тесты особенно важны, если интерфейсы не содержат операторов подтверждения отсутствия ошибок, обнаруживающих ошибочные значения параметров; они также важны после создания новых конфигураций уже существующих программных компонентов и модулей.

### А.32 Последовательное сравнительное тестирование

Последовательное сравнительное тестирование предназначено для определения того, дают ли физическая реализация и модель одинаковые выходные данные при одинаковых входных данных; это применимо только для разработки на основе моделей.

## Приложение В
## (обязательное)

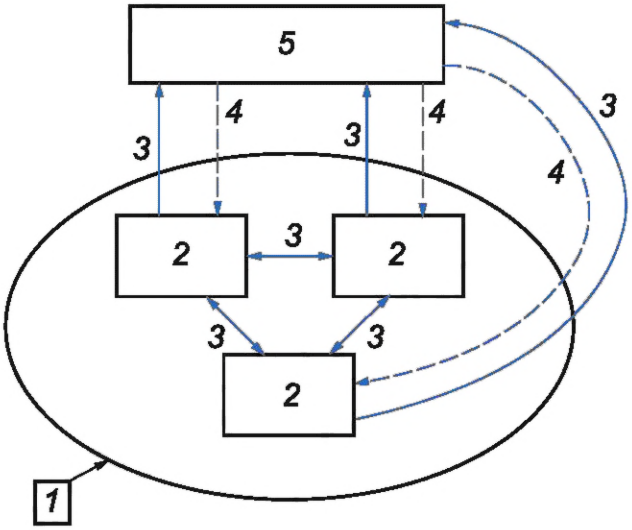
## Тестовые среды валидации программного обеспечения

### В.1 Тестирование в сети машины

Программное обеспечение должно быть интегрировано с использующим его микропроцессором в связанном с ним ECU; этот ECU должен быть интегрирован с остальными ECU, которые являются частью полной электрической системы машины. Затем ПО должно быть протестировано на интерфейсе с сетью ECU, чтобы продемонстрировать, что ПО работает в соответствии со спецификацией.

Программное обеспечение должно быть протестировано, как показано на рисунке В.1, путем моделирования:

- входных сигналов, присутствующих во время нормальной работы;
- ожидаемых событий;
- нежелательных условий, требующих действия системы.




*1* — полная система E/E/PES; *2* — реальный ECU с реальным тестируемым ПО; *3* — реальные отслеживаемые сигналы; *4* — симулируемые сигналы; *5* — тестовое оборудование

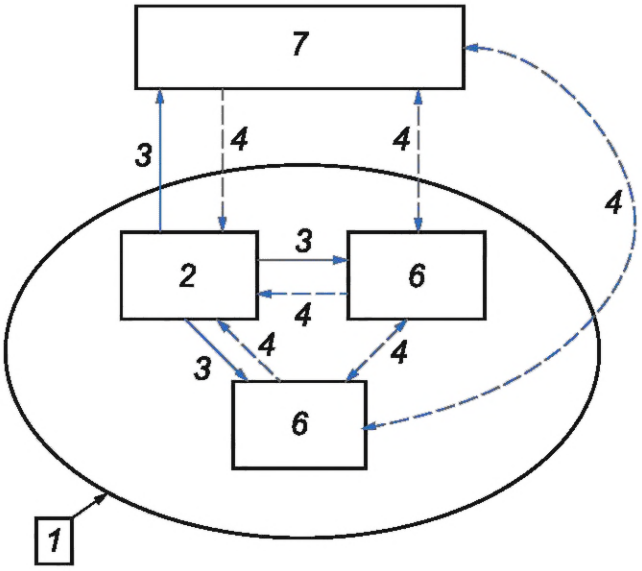
Рисунок В.1 — Тестирование сети машины

### В.2 Тестирование оборудования в контуре

Программное обеспечение должно быть интегрировано с использующим его микропроцессором в связанном с ним ECU, в то время как остальная часть связанной электрической системы машины и ее окружение должны быть смоделированы. Затем ПО должно быть протестировано в этой смоделированной среде, чтобы продемонстрировать, что ПО работает в соответствии со спецификацией.

Программное обеспечение должно быть протестировано, как показано на рисунке В.2, путем моделирования:

- входных сигналов, присутствующих во время нормальной работы;
- ожидаемых событий;
- нежелательных условий, требующих действия системы.




*1* — полная система E/E/PES; *2* — реальный ECU с реальным тестируемым ПО;
*3* — реальные отслеживаемые сигналы; *4* — симулируемые сигналы;
*6* — симулируемый ECU; *7* — смоделированная среда

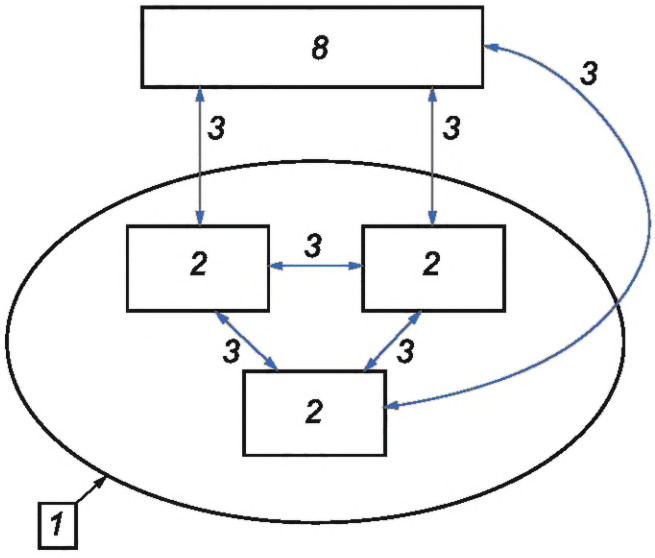
Рисунок В.2 — Тестирование оборудования в контуре

### В.3 Тестирование на уровне машины

Программное обеспечение и связанная с ним электрическая система машины должны быть интегрированы в соответствующую архитектуру машины. Затем система должна быть протестирована на машине, чтобы продемонстрировать, что ПО работает в соответствии со спецификацией.

Программное обеспечение должно быть протестировано, как показано на рисунке В.3, путем:

- входных сигналов, присутствующих во время нормальной работы;
- ожидаемых событий;
- нежелательных условий, требующих действия системы.




*1* — полная система E/E/PES; *2* — реальный ECU с реальным тестируемым ПО;
*3* — реальные отслеживаемые сигналы; *8* — реальная машина

Рисунок В.3 — Тестирование на уровне машины

## Приложение С
## (справочное)

## Расчет обеспечения целостности данных

Гарантия целостности данных во время передачи может быть рассчитана для проверки того, обеспечивается ли MPLr мерами, реализованными в сетевой связи.

Коэффициент остаточной ошибки $\Delta$ следует рассчитывать на основе вероятности остаточной ошибки $R(p)$ (см. таблицу С.1) контролирующего механизма обеспечения целостности данных, связанных с безопасностью, и скорости передачи $v$ сообщений, связанных с безопасностью. Используют следующую формулу для расчета частоты остаточных ошибок из вероятности остаточной ошибки:

$$\Delta = 3\,600\ R(p)\ v\ m\ 100\ [\text{ошибки передачи в час}]$$

Таблица С.1 — Определения параметров целостности данных

| Параметр | Определение |
|---|---|
| 3600 | Коэффициент для расчета количества сообщений в час |
| $R(p)$ | Вероятность остаточной опасной ошибки.<br>Это вероятность получить сообщение об опасной ошибке, не обнаруженной средствами стандартной защиты шины.<br>Для оценки $R(p)$, основанной на информации изготовителя, следует принять вероятность битовой ошибки как $p$ = 0,01, если нет других доказательств.<br>$$R(p) \cong \frac{1}{2^r} \cdot \sum_{k=d}^{n} \binom{n}{k} \cdot p^k \cdot (1-p)^{n-k},$$<br>где $r$ — битовая длина CRC<br>$d$ — расстояние Хэмминга<br>$n$ — количество битов в блоке, включая его подпись CRC |
| $v$ | Необходимая частота (1/с) сообщений, относящихся к безопасности, для достижения требуемого времени реакции |
| $m$ | Количество сообщений, необходимое для реализации функции безопасности |
| 100 | Коэффициент 100 гарантирует, что в передачах обеспечивается 1 % (запас безопасности) рекомендуемой полноты безопасности. Это позволяет сделать вывод, что передача достаточно безопасна. Изготовитель может использовать менее консервативный запас безопасности (т. е. > 1 %), принимая во внимание остаточную частоту ошибок. |

В IEC 61784-3-2:2016, 9.5.3, указано следующее: «IEC 61784-3 требует, чтобы в расчетах PDF/PFH использовалась частота ошибок по битам (BER), равная 0,01, если только не будет дано доказательство того, что этот BER нецелесообразен. BER, равный 0,01, означает, что 1 из каждых 100 передаваемых битов передается неправильно. Это, конечно, нереалистичное условие для сети промышленной системы управления, где необходимо поддерживать доступность».

Примечание 1 — В настоящем стандарте BER эквивалентен $p$.

Примечание 2 — Сеть промышленной системы управления эквивалентна сети EMM.

Можно использовать другой BER/$p$, если имеется достаточно данных (например, фактические измеренные значения BER); использование измеренных значений $p$ является предпочтительным.

Требуемая частота остаточных ошибок $\Delta_{req}$ зависит от корреляции между уровнем эффективности защиты и частотой остаточных ошибок, как указано в таблице С.2:

Таблица С.2 — Целостность данных в сравнении с MPL

| MPL | Требуемая частота остаточных ошибок $\Delta_{req}$ [h-1] | MPL | Требуемая частота остаточных ошибок $\Delta_{req}$ [h-1] |
|---|---|---|---|
| a | $10^{-5} \le \Delta < 1^{-4}$ | d | $1^{-7} \le \Delta < 1^{-6}$ |
| b | $3 \cdot 1^{-6} \le \Delta < 1^{-5}$ | e | $1^{-8} \le \Delta < 1^{-7}$ |
| c | $1^{-6} \le \Delta < 3 \cdot 1^{-6}$ | | |

## Приложение D
## (справочное)

## Методы и меры защиты передачи данных

### D.1 Сообщения о сохранении активности

Периодически отправляется сообщение для информирования других устройств в сети о том, что устройство включено и обменивается данными по сети. Если при проверке получателем задержка между двумя такими сообщениями превышает заданное значение, сообщения считаются ошибочными.

### D.2 Счетчик активности

Компонент учета, инициализирующийся со значением «0» при создании контролируемого объекта. Сообщение отправляется с этим порядковым номером, определяемым как дополнительное поле данных, содержащее счетчик, который увеличивается от времени $t - 1$ до времени $t$, пока объект активен, в каждом следующем сообщении. Если при проверке получателем выявляется несогласованность счетчика, сообщения считаются ошибочными.

### D.3 Проверка циклическим избыточным кодом (CRC)

Проверка циклическим избыточным кодом позволяет определить разницу в содержании сообщения между отправителем и получателем. Содержимое сообщения сжимается с помощью алгоритма. Типичный алгоритм CRC рассматривает все содержимое блока как последовательный поток байтов или последовательностей битов, над которым выполняется непрерывное полиномиальное деление с использованием генератора полиномов. Остальная часть раздела представляет собой сжатое информационное содержимое и добавляется к соответствующему сообщению отправителем. Подпись вычисляется получателем и сравнивается с ранее сохраненной. При наличии расхождений сообщения считаются ошибочными.

При использовании CRC следует учитывать, что остаточная частота ошибок CRC, реализованной в шинной системе, может быть недостаточной, и в этом случае рекомендуется дополнительная CRC на прикладном уровне.

### D.4 Порядковый номер

Текущий номер встроен в связанное с безопасностью сообщение, которым обмениваются отправитель и получатель. Этот порядковый номер определяется как дополнительное поле данных, содержащее число, которое изменяется заданным образом от одного сообщения к другому. Если при проверке получателем выявляется несогласованность номера, сообщения считаются ошибочными.

### D.5 Повторение сообщения

Информация, связанная с безопасностью, отправляется более одного раза. Опасные действия выполняются только при подтверждении непротиворечивости сообщений.

### D.6 Защитный таймер

Этот таймер также называют «тайм-аутом». Во время передачи сообщения получатель проверяет, превышает ли задержка между двумя сообщениями заданное значение. В этом случае получатель считает сообщения ошибочными.

### D.7 Шина данных, управляемая по времени

Коммуникация, инициируемая по времени, означает, что действия инициируются по прошествии временных интервалов. В системе связи, управляемой по времени, все моменты времени передачи сообщений определяются при разработке системы. Система связи с синхронизацией по времени идеальна для приложений, в которых трафик данных носит периодический характер.

### D.8 Защита шины

Защита шины — независимый компонент между узлом и шиной, который позволяет данному узлу передавать информацию по шине только тогда, когда ему это разрешено; это предотвращает перегрузку шины неисправными узлами.

Примечание — Защита шины должна иметь информацию, когда узлу разрешен доступ к шине; этого трудно достичь в системе, запускаемой по событию, но концептуально просто в системе, запускаемой по времени.

### D.9 Мини-интервалы

Метод планирования шины, при котором каждый узел, подключенный к шине, ждет определенный период времени, прежде чем ему снова будет разрешен доступ к шине. Это стратегия доступа к среде с управлением по времени, в которой время разделено на последовательность мини-интервалов, каждый из которых длиннее, чем расчетная задержка в канале данных. Каждому узлу назначается уникальное количество мини-интервалов, которые проходят без передачи данных, прежде чем ему снова будет разрешено передавать данные.

## Приложение Е
## (справочное)

## Методы и меры защиты данных внутри микроконтроллера

### Е.1 Ограничение одного объекта двунаправленной связи

Один и только один объект двунаправленной связи (например, глобальная переменная ПО) используется между двумя соответствующими разделами для обмена данными.

### Е.2 Ограничение двух объектов однонаправленной связи

Два и только два объекта однонаправленной связи (например, локальные переменные ПО) используются между двумя соответствующими разделами для обмена данными.

### Е.3 ID для идентификации и подтверждения

Использование уникальных номеров для идентификации коммуникационных узлов или получения подтверждения сообщения коммуникационным узлом.

### Е.4 Асинхронная передача данных

При использовании асинхронной передачи данных состояние ожидания не завершается самой передачей.

### Е.5 Планирование на основе приоритета

При планировании на основе приоритета разделы, важные для безопасности, могут иметь приоритет при распределении процессорного времени. При распределении процессорного времени можно предусмотреть некоторое свободное время (буфер) в каждом процессорном цикле для управления входящими прерываниями.

### Е.6 Метод временных интервалов

Метод временных интервалов задает алгоритм планирования, основанный на предварительно определенном фиксированном расписании, повторяющемся с фиксированным периодом. При использовании метода временных интервалов распределение процессорного времени происходит через статическую таблицу распределения. Таким образом, для каждой системной задачи предопределяется фиксированный момент времени для ее активации.

### Е.7 Механизмы защиты памяти

Механизмы защиты памяти относятся к спецификациям процессоров, например с MMU (блок управления памятью) или MPU (блок защиты памяти).

MMU может обеспечивать виртуальную адресацию памяти. Это предотвращает повреждение системной задачей одного раздела области памяти другой системной задачи путем непреднамеренной записи в это пространство памяти, поскольку каждый раздел имеет свое собственное адресное пространство.

Использование MMU требует поддержки этой функции операционной системой. Должны быть приняты меры, чтобы MMU нельзя было игнорировать. Поэтому системные задачи выполняются в так называемом пользовательском режиме и реальный режим адресации не используется. Этот механизм защиты памяти может предотвратить непреднамеренную запись в интерфейсы ввода/вывода.

### Е.8 Проверка данных, критически важных для безопасности

Ячейки оперативной памяти (ОЗУ), содержащие важные для безопасности данные, проверяются с помощью дополнительных мер. Этого можно добиться, например, с помощью CRC или избыточного хранилища. Эффективность этой меры очень сильно зависит от качества проверки.

### Е.9 Статический анализ

Методы статического анализа используются для просмотра фрагментов кода, которые обращаются к ячейкам памяти, содержащим данные, связанные с безопасностью.

### Е.10 Статическое распределение

Ресурсы выделяются статически во время инициализации.

## Приложение ДА
## (справочное)

## Сведения о соответствии ссылочных международных стандартов межгосударственным стандартам

Таблица ДА.1

| Обозначение ссылочного международного стандарта | Степень соответствия | Обозначение и наименование соответствующего межгосударственного стандарта |
|---|---|---|
| ISO 6750-1 | — | * |
| ISO 12100:2010 | IDT | ГОСТ ISO 12100—2013 «Безопасность машин. Основные принципы конструирования. Оценки риска и снижения риска» |
| ISO 13849-1:2015 | — | * |
| ISO 19014-1 | IDT | ГОСТ ISO 19014-1—2024 «Машины землеройные. Функциональная безопасность. Часть 1. Методика определения элементов систем управления, связанных с обеспечением безопасности, и технические требования» |
| ISO 19014-2 | IDT | ГОСТ ISO 19014-2—2024 «Машины землеройные. Функциональная безопасность. Часть 2. Проектирование и оценка оборудования и структуры систем управления, связанных с обеспечением безопасности» |

* Соответствующий межгосударственный стандарт отсутствует. До его принятия рекомендуется использовать перевод на русский язык данного международного стандарта.

Примечание — В настоящей таблице использовано следующее условное обозначение степени соответствия стандартов:

- IDT — идентичные стандарты.

## Библиография


| | | |
|---|---|---|
| [1] | ISO 6165 | Earth-moving machinery — Basic types — Identification and terms and definitions |
| [2] | IEC 61508-7:2010 | Functional safety of electrical/electronic/programmable electronic safetyrelated systems — Part 7: Overview of Techniques and Measures |
| [3] | ISO 22448:2010 | Earth-moving machinery — Anti-theft systems — Classification and performance |
| [4] | ISO 26262-1:2018 | Functional safety — Part 1: Vocabulary |
| [5] | IEC 61784-3-2:2016 | Industrial communication networks — Profiles — Part 3-2: Functional safety fieldbuses — Additional specifications for CPF 2 |
| [6] | SAE J1939 | Recommended Practice for a Serial Control and Communications Vehicle Network |



УДК 631.3:006.354 МКС 53.100 IDT

Ключевые слова: машины землеройные, разработка и оценка программного обеспечения и передачи данных для элементов систем управления, связанных с обеспечением безопасности

Редактор *Н.А. Аргунова*
Технический редактор *И.Е. Черепкова*
Корректор *Л.С. Лысенко*
Компьютерная верстка *Л.А. Круговой*

Сдано в набор 07.10.2024. Подписано в печать 11.10.2024. Формат 60×84⅛. Гарнитура Ариал.
Усл. печ. л. 4,18. Уч.-изд. л. 3,55.

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.
www.gostinfo.ru info@gostinfo.ru