ФЕДЕРАЛЬНОЕ АГЕНТСТВО

ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ И МЕТРОЛОГИИ

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ГОСТ Р МЭК 61508-7— 2012**

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

## Часть 7
## Методы и средства

**IEC 61508-7:2010**

**Functional safety of electrical/electronic/programmable electronic safety-related systems — Part 7: Overview of techniques and measures (IDT)**

**Издание официальное**

Москва
Стандартинформ
2014

gost.e-ecolog.ru

## Предисловие

1 ПОДГОТОВЛЕН Обществом с ограниченной ответственностью «Корпоративные электронные системы» и Федеральным бюджетным учреждением «Консультационно-внедренческая фирма в области международной стандартизации и сертификации — «Фирма «Интерстандарт» на основе собственного аутентичного перевода на русский язык международного стандарта, указанного в пункте 4

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

3 УТВЕРЖДЕН И ВВЕДЕН В ДЕЙСТВИЕ Приказом Федерального агентства по техническому регулированию и метрологии от 29 октября 2012 г. № 592-ст

4 Настоящий стандарт идентичен международному стандарту МЭК 61508-7:2010 «Функциональная безопасность систем электрических, электронных, программируемых электронных, связанных с безопасностью. Часть 7. Методы и средства» (IEC 61508-7:2010 «Functional safety of electrical/electronic/programmable electronic safety-related systems — Part 7: Overview of techniques and measures»).

Наименование настоящего стандарта изменено относительно наименования указанного международного стандарта для приведения в соответствие с ГОСТ Р 1.5 (подраздел 3.5).

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

5 ВЗАМЕН ГОСТ Р МЭК 61508-7—2007

*Правила применения настоящего стандарта установлены в ГОСТ Р 1.0—2012 (раздел 8). Информация об изменениях к настоящему стандарту публикуется в ежегодном (по состоянию на 1 января текущего года) информационном указателе «Национальные стандарты», а официальный текст изменений и поправок — в ежемесячном информационном указателе «Национальные стандарты». В случае пересмотра (замены) или отмены настоящего стандарта соответствующее уведомление будет опубликовано в ближайшем выпуске ежемесячного информационного указателя «Национальные стандарты». Соответствующая информация, уведомление и тексты размещаются также в информационной системе общего пользования — на официальном сайте Федерального агентства по техническому регулированию и метрологии в сети Интернет (gost.ru)*

© Стандартинформ, 2014

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

## Содержание

1 Область применения . . . . . 1
2 Нормативные ссылки . . . . . 2
3 Термины и определения . . . . . 3
Приложение А (справочное) Анализ методов и средств для Э/Э/ПЭ систем, связанных с безопасностью. Управление случайными отказами технических средств (см. МЭК 61508-2) . . . . . 4
Приложение В (справочное) Анализ методов и средств для Э/Э/ПЭ систем, связанных с безопасностью. Предотвращение систематических отказов (см. МЭК 61508-2 и МЭК 61508-3) . 16
Приложение С (справочное) Анализ методов и средств достижения полноты безопасности программного обеспечения (см. МЭК 61508-3) . . . . . 34
Приложение D (справочное) Вероятностный подход определения полноты безопасности предварительно разработанных программных средств . . . . . 70
Приложение Е (справочное) Краткий обзор методов и мер для проектирования СИС . . . . . 74
Приложение F (справочное) Определение свойств стадий жизненного цикла программного обеспечения . . . . . 84
Приложение G (справочное) Руководство по разработке связанного с безопасностью объектно-ориентированного программного обеспечения . . . . . 90
Приложение ДА (справочное) Сведения о соответствии ссылочных международных стандартов ссылочным национальным стандартам Российской Федерации . . . . . 92
Библиография . . . . . 93

## Введение

Системы, состоящие из электрических и/или электронных элементов, в течение многих лет используются для выполнения функций безопасности в большинстве областей применения. Компьютерные системы (обычно называемые «программируемые электронные системы»), применяемые во всех прикладных отраслях для выполнения функций, не связанных с безопасностью, во все более увеличивающихся количествах используются для выполнения функций обеспечения безопасности. Для эффективной и безопасной эксплуатации технологий, основанных на использовании компьютерных систем, чрезвычайно важно, чтобы лица, ответственные за принятие решений, имели в своем распоряжении руководства по вопросам безопасности, которые они могли бы использовать в своей работе.

Настоящий стандарт устанавливает общий подход к вопросам обеспечения безопасности для всего жизненного цикла систем, состоящих из электрических и/или электронных, и/или программируемых электронных (Э/Э/ПЭ) элементов, которые используются для выполнения функций обеспечения безопасности. Этот унифицированный подход был принят для разработки рациональной и последовательной технической политики для всех электрических систем обеспечения безопасности. При этом основной целью является содействие разработке стандартов для продукции и областей применения на основе стандартов серии МЭК 61508.

Примечание — Примерами стандартов для продукции и областей применения, разработанных на основе стандартов серии МЭК 61508, являются [1]—[3].

Обычно безопасность достигается за счет использования нескольких систем, в которых используются различные технологии (например механические, гидравлические, пневматические, электрические, электронные, программируемые электронные). Любая стратегия безопасности должна, следовательно, учитывать не только все элементы, входящие в состав отдельных систем (например датчики, управляющие устройства и исполнительные механизмы), но также и все подсистемы безопасности, входящие в состав общей системы обеспечения безопасности. Таким образом, хотя настоящий стандарт посвящен в основном Э/Э/ПЭ системам, связанным с безопасностью, он может также предоставлять общий подход, в рамках которого рассматриваются системы, связанные с безопасностью, базирующиеся на других технологиях.

Признанным фактом является существование огромного разнообразия использования Э/Э/ПЭ систем в различных областях применения, отличающихся различной степенью сложности, возможными рисками и опасностями. В каждом конкретном применении необходимые меры безопасности будут зависеть от многочисленных факторов, специфичных для конкретного применения. Настоящий стандарт, являясь базовым, позволит формулировать такие меры для областей применения будущих международных стандартов, а также для последующих редакций уже существующих стандартов.

Настоящий стандарт:

- рассматривает все соответствующие стадии жизненного цикла безопасности систем в целом, а также подсистем Э/Э/ПЭ системы и программного обеспечения (от первоначальной концепции, через проектирование, внедрение, эксплуатацию и техническое обеспечение до снятия с эксплуатации), в ходе которых Э/Э/ПЭ системы используются для выполнения функций безопасности;
- был задуман с учетом быстрого развития технологий; его основа является в значительной мере устойчивой и полной для будущих разработок;
- делает возможной разработку стандартов областей применения, в которых используются Э/Э/ПЭ системы, связанные с безопасностью; разработка стандартов для областей применения в рамках общей структуры, вводимой настоящим стандартом, должна привести к более высокому уровню согласованности (например основных принципов, терминологии и т. д.) как для отдельных областей применения, так и для их совокупностей, что даст преимущества в плане безопасности и экономики;
- предоставляет метод разработки спецификации требований к безопасности, необходимых для достижения заданной функциональной безопасности Э/Э/ПЭ систем, связанных с безопасностью;
- использует для определения требований к уровням полноты безопасности подход, основанный на оценке рисков;
- вводит уровни полноты безопасности для определения целевого уровня полноты безопасности для функций безопасности, которые должны быть реализованы Э/Э/ПЭ системами, связанными с безопасностью.

Примечание — Настоящий стандарт не устанавливает требований к уровню полноты безопасности для любой функции безопасности и не определяет то, как устанавливается уровень полноты безопасности. Однако настоящий стандарт формирует основанный на риске концептуальный подход и приводит примеры методов;

- – устанавливает целевые меры отказов для функций безопасности, реализуемых Э/Э/ПЭ системами, связанными с безопасностью, и связывает эти меры с уровнями полноты безопасности;

- устанавливает нижнюю границу для целевых мер отказов для функции безопасности, реализуемой одиночной Э/Э/ПЭ системой, связанной с безопасностью. Для Э/Э/ПЭ систем, связанных с безопасностью, работающих в режиме:

- низкой интенсивности запросов на обслуживание: нижняя граница для выполнения функции, для которой система предназначена, устанавливается в соответствии со средней вероятностью опасного отказа по запросу, равной $10^{-5}$,

- высокой интенсивности запросов на обслуживание или в непрерывном режиме: нижняя граница устанавливается в соответствии со средней частотой опасных отказов $10^{-9}$ в час.

Примечания

1 Одиночная Э/Э/ПЭ система, связанная с безопасностью, не обязательно предполагает одноканальную архитектуру.

2 В проектах систем, связанных с безопасностью и имеющих низкий уровень сложности, можно достигнуть более низких значений целевой полноты безопасности, но предполагается, что в настоящее время указанные предельные значения целевой полноты безопасности могут быть достигнуты для относительно сложных систем (например программируемые электронные системы, связанные с безопасностью);

- устанавливает требования по предотвращению и управлению систематическими отказами, основанные на опыте и заключениях из практического опыта. Учитывая, что вероятность возникновения систематических отказов в общем случае, не может быть определена количественно, настоящий стандарт позволяет утверждать для специфицируемой функции безопасности, что целевая мера отказов, связанных с этой функцией, может считаться достигнутой, если все требования стандарта были выполнены;

- вводит понятие «стойкость к систематическим отказам», применяемое к элементу, характеризующее уверенность в том, что полнота безопасности, касающаяся систематических отказов элемента, соответствует требованиям заданного уровня полноты безопасности;

- применяет широкий диапазон принципов, методов и средств для достижения функциональной безопасности Э/Э/ПЭ систем, связанных с безопасностью, но не использует явно понятие «безопасный отказ». В то же время понятия «безопасный отказ» и «безопасный в своей основе отказ» могут быть использованы, но для этого необходимо обеспечить соответствующие требования в конкретных разделах стандарта, которым эти понятия должны соответствовать.

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

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

## Часть 7
## Методы и средства

Functional safety of electrical electronic programmable electronic safety-related systems. Part 7. Techniques and measures

**Дата введения — 2013—08—01**

## 1 Область применения

1.1 Настоящий стандарт содержит общее описание различных методов и средств, обеспечивающих выполнение требований МЭК 61508-2 и МЭК 61508-3. Ссылки должны рассматриваться как базовые ссылки на методы и инструменты либо как примеры, и они могут не отражать современное состояние области.

1.2 МЭК 61508-1, МЭК 61508-2, МЭК 61508-3 и МЭК 61508-4 являются базовыми стандартами по безопасности, хотя этот статус не применим в контексте Э/Э/ПЭ систем, связанных с безопасностью, имеющих низкую сложность (см. пункт 3.4.3 МЭК 61508-4). В качестве базовых стандартов по безопасности, данные стандарты предназначены для использования техническими комитетами при подготовке стандартов в соответствии с принципами, изложенными в руководстве МЭК 104 и руководстве ИСО/МЭК 51. МЭК 61508-1, МЭК 61508-2, МЭК 61508-3 и МЭК 61508-4 предназначены для использования в качестве самостоятельных стандартов. Функция безопасности настоящего стандарта не применима к медицинскому оборудованию, соответствующему требованиям серии горизонтальных стандартов МЭК 60601 [4].

1.3 В круг обязанностей Технического комитета входит использование (там, где это возможно) основополагающих стандартов по безопасности при подготовке собственных стандартов. В этом случае требования, методы проверки или условия проверки настоящего основополагающего стандарта по безопасности не применяют, если на них нет конкретной ссылки или они не включены в стандарты, подготовленные этими техническими комитетами.

1.4 На рисунке 1 изображена общая структура стандартов серии МЭК 61508 и показана роль, которую играет настоящий стандарт в достижении функциональной безопасности Э/Э/ПЭ систем, связанных с безопасностью.

**Издание официальное**

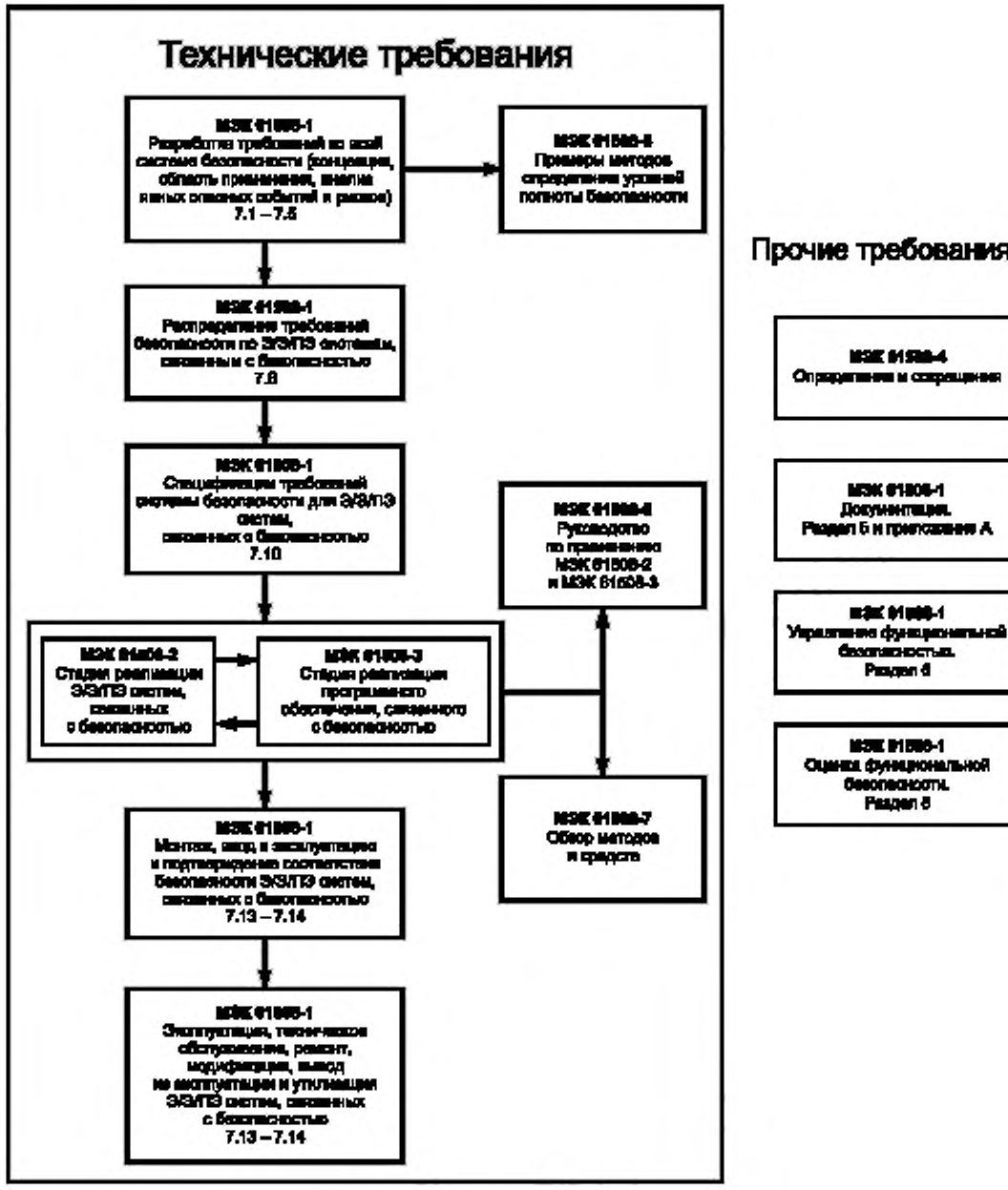



Рисунок 1 — Общая структура стандартов серии МЭК 61508

## 2 Нормативные ссылки

В настоящем стандарте использованы нормативные ссылки на следующие стандарты:

ИСО/МЭК Руководство 51:1990 Аспекты безопасности. Руководящие указания по включению в стандарты (ISO/IEC Guide 51:1999, Safety aspects — Guidelines for their inclusion in standards)

МЭК Руководство 104:2010 Подготовка публикаций по безопасности и использование базовых публикаций по безопасности и публикаций по безопасности групп (IEC Guide 104:2010, The preparation of safety publications and the use of basic safety publications and group safety publications)

МЭК 61508-1:2010 Функциональная безопасность систем электрических, электронных, программируемых электронных, связанных с безопасностью. Часть 1. Общие требования (IEC 61508-1:2010, Functional safety of electrical/electronic/programmable electronic safety-related systems — Part 1: General requirements)

МЭК 61508-2:2010 Функциональная безопасность систем электрических, электронных, программируемых электронных, связанных с безопасностью. Часть 2. Требования к системам электрическим, электронным, программируемым электронным, связанным с безопасностью (IEC 61508-4:2010, Functional safety of electrical/electronic/programmable electronic safety-related systems — Part 2: Requirements for electrical/electronic/programmable electronic safety-related systems)

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

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

## 3 Термины и определения

В настоящем стандарте применены термины, определения и сокращения по МЭК 61508-4.

**Приложение А**
**(справочное)**

## Анализ методов и средств для Э/Э/ПЭ систем, связанных с безопасностью. Управление случайными отказами технических средств (см. МЭК 61508-2)

### А.1 Электрические

Главная цель. Управление отказами в электромеханических компонентах.

#### А.1.1 Обнаружение отказов путем мониторинга в режиме онлайн

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблицы А.2, А.3, А.7 и А.13 — А.18).

Цель. Обнаружение отказов путем контроля поведения Э/Э/ПЭ системы, связанной с безопасностью, в процессе нормального (в режиме онлайн) функционирования управляемого оборудования (далее — УО).

Описание. При определенных условиях отказы могут быть обнаружены с помощью информации, например, о поведении во времени УО. Например, если коммутатор, который является частью Э/Э/ПЭ системы, связанной с безопасностью, нормально активизируется со стороны УО и если при этом коммутатор не изменяет состояния в предполагаемое время, то этот отказ может быть обнаружен. Обычными способами невозможно локализовать такой отказ.

#### А.1.2 Мониторинг контактов реле

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблицы А.2 и А.14).

Цель. Обнаружение отказов (например пайки) контактов реле.

Описание. Активизируемые контактные реле (или положительно управляемые контакты в реле) спроектированы так, что их контакты жестко связаны между собой. Рассмотрим два переключаемых контакта *a* и *b*. Если нормально разомкнутый контакт *a* оказался спаянным (залипшим), то нормально замкнутый контакт *b* не может замкнуться, если обмотка реле обесточивается. Следовательно, контроль замыкания нормально замкнутого контакта *b* при обесточенной обмотке реле может быть использован для подтверждения того, что нормально разомкнутый контакт *a* разомкнут. Отсутствие замыкания нормально замкнутого контакта *b* указывает на отказ контакта *a*, поэтому схема контроля должна обеспечить надежное отключение или обеспечить продолжение отключения при любом управлении оборудования контактом *a*.

Литература:

Zusammenstellung und Bewertung elektromechanischer Sicherheitsschaltungen für Ver-riegelungseinrichtungen. F. Kreutzkampf, W. Hertel, Sicherheitstechnisches Informations- und Arbeitsblatt 330212, BIA-Handbuch. 17. Lfg. X/91, Erich Schmidt Verlag, Bielefeld. www.BGIA-HANDBUCHdigital.de/330212.

#### А.1.3 Компаратор

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблицы А.2, А.3, А.4).

Цель. Оперативное обнаружение (не одновременное) отказов в независимом модуле обработки или в компараторе.

Описание. Сигналы независимых модулей обработки сравнивают циклически или непрерывно компаратором аппаратных средств. Сам компаратор может быть внешне тестируемым или же может использовать самоконтролируемую технологию. Обнаруживаемые компаратором различия в поведении процессоров независимых модулей формируют информацию для сообщений об отказах.

#### А.1.4 Схема голосования по мажоритарному принципу

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблицы А.2, А.3 и А.4).

Цель. Обнаружение и парирование отказов по меньшей мере в одном из трех аппаратных каналов.

Описание. Модуль голосования, использующий мажоритарный принцип (2 из 3, 3 из 4 или *m* из *n*), используется для обнаружения и парирования отказов. Сама схема голосования может внешне тестироваться или использовать самоконтролирующие технологии.

Литература:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6.

#### А.1.5 Отсутствие питания (отключение энергии)

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.16).

Цель. Выполнение функции безопасности при выключении или отсутствии питания.

Описание. Функция безопасности выполняется, если контакты реле разомкнуты и ток не проходит. Например, при использовании тормозов для остановки опасного вращения двигателя тормоза отпускаются замыкающими контактами в системах, связанных с безопасностью, и включаются размыкающими контактами в системах, связанных с безопасностью.

Литература:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6.

**А.2 Электроника**

Главная цель. Управление отказами в твердотельных компонентах.

**А.2.1 Тестирование избыточным оборудованием**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблицы А.3, А.15, А.16 и А.18).

Цель. Обнаружение отказов с использованием избыточных аппаратных средств, то есть с использованием дополнительных аппаратных средств, не требующихся для реализации функций обработки.

Описание. Избыточные аппаратные средства могут быть использованы для тестирования на соответствующей частоте заданных функций безопасности. Такой подход обычно требуется для реализации положений пунктов А.1.1 или А.2.2.

**А.2.2 Принципы динамического управления**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.3).

Цель. Обнаружение статических отказов путем динамической обработки сигналов.

Описание. Принудительное изменение других статических сигналов (генерируемых извне или внутри) помогает обнаруживать статические отказы в компонентах. Этот метод часто ассоциируется с электромеханическими компонентами.

Литература:

Elektronik in der Sicherheitstechnik. H. Jürs, D. Reinert, Sicherheitstechnisches Informations- und Arbeitsblatt 330220, BIA-Handbuch, Erich-Schmidt Verlag, Bielefeld, 1993, gost.e-ecolog.ru

**А.2.3 Стандартный тестовый порт доступа и архитектура граничного сканирования**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблицы А.3, А.15 и А.18).

Цель. Управление и наблюдение за происходящим на каждом контакте интегральной схемы (ИС).

Описание. Тестирование граничного сканирования представляет собой метод построения ИС, который повышает тестируемость ИС, разрешая проблему доступа к внутренним точкам тестируемой схемы. В типичной сканируемой по границам интегральной схеме, содержащей внутренние логические схемы, а также входные и выходные буферы, между внутренними логическими схемами и входными/выходными буферами, соединенными с внешними контактами ИС, размещается ступень сдвигового регистра. Содержимое каждого сдвигового регистра находится в ячейке граничного сканирования. Ячейка граничного сканирования может управлять и наблюдать за происходящим на каждом входном и выходном контакте ИС через стандартный тестовый порт доступа. Тестирование внутренних логических схем ИС проводится путем отключения размещенных на чипе внутренних логических схем от входных сигналов, получаемых от окружающих компонентов, и последующего выполнения внутреннего тестирования. Эти тесты могут быть использованы для обнаружения отказов в ИС.

Литература:

IEEE 1149—1:2001, IEEE standard test access port and boundary-scan architecture, IEEE Computer Society, 2001, ISBN: 0-7381-2944-5.

**А.2.4** (Не используется)

**А.2.5 Избыточный контроль**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.3).

Цель. Обнаружение отказов путем создания нескольких функциональных модулей, контроля поведения каждого из них для обнаружения отказов и последующего инициирования перехода в безопасное состояние при обнаружении какого-либо несоответствия в поведении.

Описание. Функция безопасности выполняется по меньшей мере двумя аппаратными каналами. Выходы этих каналов контролируются и безопасное состояние инициируется при обнаружении отказа (в случае, если выходные сигналы из всех каналов не идентичны).

Литература:

Elektronik in der Sicherheitstechnik. H. Jürs, D. Reinert, Sicherheitstechnisches Informations- und Arbeitsblatt 330220, BIA-Handbuch, Erich-Schmidt Verlag, Bielefeld, 1993, gost.e-ecolog.ru

Dependability of Critical Computer Systems 1. F. J. Redmill, Elsevier Applied Science, 1988, ISBN 1-85166-203-0.

**А.2.6 Электрические/электронные компоненты с автоматической проверкой**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.3).

Цель. Обнаружение отказов путем периодической проверки способности выполнения функции безопасности.

Описание. Аппаратные средства тестируются до запуска процесса и затем тестируются повторно через соответствующие интервалы. УО продолжает работу только при условии успешного прохождения каждого теста.

Литература:

Elektronik in der Sicherheitstechnik. H. Jürs, D. Reinert, Sicherheitstechnisches Informations- und Arbeitsblatt 330220, BIA-Handbuch, Erich-Schmidt Verlag, Bielefeld, 1993, http://www.bgia-handbuchdigital.de/330220.

Dependability of Critical Computer Systems 1. F.J. Redmill, Elsevier Applied Science, 1988, ISBN 1-85166-203-0.

**А.2.7 Текущий контроль аналоговых сигналов**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблицы А.3 и А.13).

Цель. Повышение достоверности результатов измерений аналоговых сигналов.

Описание. Везде, где возможно, используются аналоговые приборы, а не цифровые. В аналоговых приборах отключение или безопасные состояния представляются уровнями аналоговых сигналов обычно с контролем устойчивости уровня этого сигнала. Это обеспечивает непрерывный контроль и высокую степень достоверности передачи сигнала, снижает частоту необходимого гарантийного тестирования функции чувствительности передатчика. Внешние интерфейсы, например импульсные линии, также нуждаются в тестировании.

**А.2.8 Снижение максимальных значений**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2, подпункт 7.4.2.13.

Цель. Повышение надежности компонентов аппаратных средств.

Описание. Компоненты аппаратных средств успешно выполняют свои функции на уровнях напряжений, которые определены при проектировании системы и являются более низкими, чем их максимальные значения, определенные в спецификации. Снижение этих значений является обычной практикой, гарантирующей, что при всех нормальных условиях эксплуатации компоненты будут успешно функционировать при уровнях напряжений ниже их максимальных значений.

**А.3 Модули обработки**

Главная цель. Распознавать отказы, которые приводят к неправильным результатам в модулях обработки.

**А.3.1 Программное самотестирование: предельное количество комбинаций (одноканальное)**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.4).

Цель. Оперативное обнаружение отказов в модулях обработки.

Описание. Аппаратные средства создаются с использованием стандартных методов, не учитывающих специальных требований к безопасности. Обнаружение отказов реализуется целиком дополнительными программными функциями, которые выполняют самотестирование с использованием по меньшей мере двух дополнительных комбинаций данных (например 55hex и AAhex).

**А.3.2 Программное самотестирование: блуждающий бит (одноканальное)**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.4).

Цель. Оперативное обнаружение отказов в физической памяти (например в регистрах) и дешифраторе команд процессора.

Описание. Обнаружение отказов полностью реализуется дополнительными программными функциями, которые выполняют самотестирование с использованием комбинации данных (например комбинации блуждающих битов), которая тестирует физическую память (регистры данных и адресные регистры) и дешифратор команд. Однако диагностический охват составляет только 90 %.

**А.3.3 Самотестирование, обеспечиваемое оборудованием (одноканальное)**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.4).

Цель. Оперативное обнаружение отказов в процессоре с использованием специальных аппаратных средств, которые увеличивают скорость и расширяют область обнаружения отказов.

Описание. Дополнительные специальные аппаратные средства обеспечивают функции самотестирования для обнаружения отказов. Например, таким средством может быть аппаратный модуль, который циклически контролирует выход на наличие конкретной битовой комбинации, используя механизм сторожевой схемы.

**А.3.4 Запрограммированная обработка (одноканальная)**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.4).

Цель. Оперативное обнаружение отказов в процессоре.

Описание. Процессоры могут быть спроектированы со встроенными специальными функциями распознавания или исправления отказов. До сих пор эти функции применялись только в относительно простых схемах и не получили широкого распространения. Однако такие функции не должны исключаться в будущих разработках.

Литература:

Le Processeur Codé: un nouveau concept appliqué a la sécurité des systèmes de transports. Gabriel, Martin, Wartski, Revue Générale des chemins de fer, No. 6, June 1990.

Vital Coded Microprocessor Principles и Application for Various Transit Systems. P. Forin, IFAC Control Computers Communications in Transportation, 79—84, 1989.

**А.3.5 Программное обнаружение несовпадений**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.4).

Цель. Оперативное обнаружение отказов в процессоре путем динамического программного сравнения.

Описание. Два модуля взаимно обмениваются данными (включая результаты, промежуточные результаты и тестируемые данные). Если при сравнении данных, выдаваемых с использованием программных средств в каждом модуле, обнаруживаются различия, то это приводит к выдаче сообщений об отказе.

**А.4 Постоянная память**

Главная цель. Выявление модификаций информации в постоянной памяти.

**А.4.1 Сохранение слов с многобитовой избыточностью (например контроль ROM с модифицированным кодом Хэмминга)**

Примечание — См. также А.5.6 и С.3.2. Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.5).

Цель. Обнаружение всех однобитовых отказов, всех двухбитовых отказов и некоторых отказов во всех битах в 16-битовом слове.

Описание. Каждое слово в памяти расширяется несколькими избыточными битами для формирования модифицированного кода Хэмминга с расстоянием, равным 4 (по меньшей мере). При каждом считывании слова проверка избыточных битов может указывать, произошло ли искажение. При обнаружении различия вырабатывается сообщение об отказе. Эта процедура может также использоваться для обнаружения ошибок адресации путем вычисления избыточных битов для объединения слова данных с его адресом.

Литература:

Prüfbare und korrigierbare Codes. W.W. Peterson, München, Oldenburg, 1967.

Error detecting and error correcting codes. R.W. Hemming, The Bell System Technical Journal 29 (2), 147—160, 1950.

**А.4.2 Модифицируемая контрольная сумма**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.5).

Цель. Обнаружение всех отказов нечетных битов, то есть приблизительно 50 % всех возможных битовых отказов.

Описание. Контрольная сумма блока памяти образуется соответствующим алгоритмом, который обрабатывает все слова в блоке памяти. Эта контрольная сумма может храниться как дополнительное слово в ROM, либо может быть добавлена как дополнительное слово в блок памяти для того, чтобы алгоритм контрольной суммы выработал заранее заданное значение. В последнем тестировании памяти контрольная сумма создается снова с использованием того же алгоритма, и результат сравнивается с запомненным или заданным значением. При обнаружении различий вырабатывается сообщение об ошибке.

**А.4.3 Сигнатура из одного слова (8 бит)**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.5).

Цель. Обнаружение всех однобитовых отказов и всех многобитовых ошибок в слове при достижении приблизительно 99,6 % всех возможных битовых отказов.

Описание. Содержимое блока памяти сжимается (с использованием аппаратных или программных средств) в одно слово памяти с использованием алгоритма контроля с помощью избыточного циклического кода (CRC). Типичный алгоритм CRC рассматривает все содержимое блока памяти как побайтовый или побитовый последовательный поток данных, в котором выполняется непрерывное полиномиальное деление с использованием полиномиального генератора. Остаток от деления сохраняется и представляет собой сжатое содержимое памяти — «сигнатуру» памяти. Сигнатура вычисляется каждый раз при последующем тестировании и сравнивается с уже запомненным значением. При обнаружении различий выдается сообщение об ошибке.

**А.4.4 Сигнатура из двух слов (16 бит)**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.5).

Цель. Обнаружение всех однобитовых ошибок и всех многобитовых ошибок в слове составляет примерно 99,998 % всех возможных битовых ошибок.

Описание. Данная процедура вычисляет сигнатуру с использованием алгоритма контроля с помощью CRC, однако длина результирующего значения составляет по меньшей мере два слова. Расширенная сигнатура заносится в память, повторно вычисляется и сравнивается как одно слово. При обнаружении различий между сохраненной и повторно вычисленной сигнатурами выдается сообщение об ошибке.

**А.4.5 Дублирование блока (например дублирование ROM в аппаратном и программном исполнении)**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.5).

Цель. Обнаружение всех битовых ошибок.

Описание. Адресное пространство дублируется в двух областях памяти. Первая область памяти работает в нормальном режиме. Вторая содержит ту же информацию и доступна параллельно с первой. Их выходы сравниваются, и при обнаружении различий выдается сообщение об ошибке. Для обнаружения некоторых видов битовых ошибок данные должны запоминаться инверсно в одной из двух областей памяти и инвертироваться обратно при чтении.

**А.5 Память с произвольным доступом**

Главная цель. Обнаружение отказов во время адресации, записи, запоминания и считывания.

Примечание — Случайные сбои, перечисленные в МЭК 61508-2, таблица А. 1, являются отказами, которые должны быть обнаружены в процессе эксплуатации или должны быть проанализированы при выводе доли безопасных отказов. Причинами случайных ошибок являются: (1) альфа-частицы, образовавшиеся в результате процесса распада, (2) нейтроны, (3) внешний источник электромагнитного излучения и (4) внутренние перекрестные помехи. Внешний источник электромагнитного излучения должен соответствовать другим требованиям настоящего стандарта.

Результаты воздействия альфа-частиц и нейтронов могут быть обработаны функционирующими средствами обеспечения полноты безопасности. Но такие средства обеспечения полноты безопасности эффективны для случайных отказов аппаратных средств и не эффективны для случайных сбоев, например тесты для ОЗУ, такие как «блуждающая траектория», GALPAT и т. д., не являются эффективными, тогда как методы, использующие контроль четности и коды с исправлением ошибок, возвращающие содержимое ячеек памяти, являются эффективными.

Случайный сбой происходит, когда излучение вызывает такой заряд, который может изменить состояние или переключить с низкого уровня напряжения на высокий ячейку полупроводниковой памяти, регистр, защелку или триггер. Такую случайную ошибку называют «исправимой», потому что сама схема излучением не повреждается. Такие ошибки разделяют на однобитовые нарушения (SBU) или однособытийные нарушения (SEU) и многобитовые нарушения (MBU).

Если схема, в которой произошел сбой, является запоминающим элементом, таким как ячейка памяти или триггер, то ее состояние сохранится до следующей (намеченной) операции записи. Новые данные будут храниться правильно. В комбинаторной схеме это приведет скорее к незначительному сбою, потому что существует постоянный поток энергии из компонента, управляющего этим узлом. Влияние на соединительные провода и линии связи также может быть незначительным. Однако из-за большей емкости воздействие на них альфа-частиц и нейтронов считают незначительным.

Такие случайные сбои могут происходить в переменной памяти любого вида, то есть в DRAM, SRAM, регистровой памяти в MP, кэш-памяти, конвейерах, регистрах конфигурации устройств, таких как ADC, DMA, MMU, контроллер прерываний, сложные таймеры. Чувствительность к альфа-частицам и нейтронам зависит от напряжения питания и геометрии. Небольшие конфигурации с напряжением питания 2,5 В и особенно ниже 1,8 В потребуют более серьезной оценки и более эффективных мер защиты.

Интенсивность случайных сбоев для (встроенной) памяти находится в диапазоне от 700 Fit/MBit до 1200 Fit/MBit (см. перечисления a) и i) ниже). Это эталонное значение для сравнения с данными, полученными для устройств, реализованных на основе кремниевой технологии. До недавнего времени полагали, что SBU были доминирующими, но последний прогноз (см. перечисление a) ниже) показывает растущий процент MBU в общей интенсивности случайных сбоев (SEP) для технологий менее 65 нм.

Более подробная информация о случайных сбоях дана в следующих источниках:

a) Altitude SEE Test European Platform (ASTEP) and First Results in CMOS 130 nm SRAM. J.-L. Autran, P. Roche, C. Sudre et al. Nuclear Science, IEEE Transactions on Volume 54, Issue 4, Aug. 2007 Page(s):1002—1009;

b) Radiation-Induced Soft Errors in Advanced Semiconductor Technologies, Robert C. Baumann, Fellow, IEEE, IEEE TRANSACTIONS ON DEVICE AND MATERIALS RELIABILITY, VOL. 5, NO. 3, SEPTEMBER 2005;

c) Soft errors' impact on system reliability, Ritesh Mastipuram and Edwin C Wee, Cypress Semiconductor, 2004;

d) Trends And Challenges In VLSI Circuit Reliability, C. Costantinescu, Intel, 2003, IEEE Computer Society;

e) Basic mechanisms and modeling of single-event upset in digital microelectronics, P.E. Dodd and L.W. Massengill, IEEE Trans. Nucl. Sci., vol. 50, no. 3, pp. 583—602, Jun. 2003;

f) Destructive single-event effects in semiconductor devices and ICs, F.W. Sexton, IEEE Trans. Nucl. Sci., vol. 50, no. 3, pp. 603—621, Jun. 2003;

g) Coming Challenges in Microarchitecture and Architecture, Ronen, Mendelson, Proceedings of the IEEE, Volume 89, Issue 3, Mar 2001 Page(s):325—340;

h) Scaling and Technology Issues for Soft Error Rates, A Johnston, 4th Annual Research Conference on Reliability Stanford University, October 2000;

i) International Technology Roadmap for Semiconductors (ITRS), several papers.

**А.5.1 Тесты «шахматная доска» или «марш» для памяти с произвольным доступом**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.6).

Цель. Обнаружение преимущественно статических битовых ошибок.

Описание. Расположенная в шахматном порядке битовая комбинация нулей и единиц записывается в ячейки памяти с битовой организацией. Затем эти ячейки анализируются попарно с тем, чтобы убедиться в их одинаковости и правильности. Адрес первой ячейки такой пары является переменным, а адрес второй ячейки этой пары образуется путем битового инвертирования первого адреса. При первом прохождении диапазон адресов памяти проходят в направлении более высоких адресов переменных адресов, а при втором прохождении — в направлении более низких адресов. После этого оба прохождения повторяются с заранее заданным инвертированием. При обнаружении какого-либо различия выдается сообщение об отказе.

При «маршевом» тестировании памяти с произвольным доступом ячейки памяти с битовой организацией инициализируются унифицированным потоком битов. При первом прохождении ячейки анализируются в нисходящей последовательности; проверяется правильность содержимого каждой ячейки и ее содержимое инвертируется. Основа, созданная в первом прохождении, рассматривается при втором прохождении в убывающем порядке и так же обрабатывается. Первые прохождения повторяются с инвертируемыми предварительными значениями в третьем и четвертом прохождениях. При обнаружении различий выдается сообщение об отказе.

**А.5.2 Тест «блуждающая траектория» для памяти с произвольным доступом**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.6).

Цель. Обнаружение статических и динамических ошибочных битов и перекрестных помех между ячейками памяти.

Описание. Тестируемая область памяти инициализируется унифицированным потоком битов. Затем первая ячейка инвертируется, и остальная часть памяти анализируется на правильность. После этого первая ячейка повторно инвертируется для возврата в исходное значение, и вся процедура повторяется для следующей ячейки. Второе прохождение «модели блуждающего бита» осуществляется при инверсии всех первоначально назначенных значений памяти. При обнаружении различий выдается сообщение об ошибке.

**А.5.3 Тест «GALPAT» или «Прозрачный GALPAT» для памяти с произвольным доступом**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.6).

Цель. Обнаружение статических битовых отказов и большой части динамических связей.

Описание. При тестировании памяти с произвольным доступом «попарной записью-считыванием» выбранная область памяти сначала инициализируется унифицированно (то есть все 0 или все 1). После этого первая ячейка памяти тестируется и затем инвертируется, и все остальные ячейки анализируются на правильность содержимого. После каждого доступа по чтению к одной из оставшихся ячеек инвертированная ячейка также проверяется. Эта процедура повторяется для каждой ячейки в выбранной области памяти. Второе прохождение выполняется противоположно первому. Любые различия приводят к выдаче сообщения об ошибке.

Тестирование «прозрачной попарной записью-считыванием» представляет собой вариацию описанной выше процедуры: вместо инициализации всех ячеек в выбранной области памяти существующее содержимое остается неизменным, а для сравнения содержимого набора ячеек используются контрольные суммы (сигнатуры). Выбирается первая тестируемая ячейка области памяти и вычисляется и сохраняется сигнатура $S1$ всех оставшихся ячеек области. Затем тестируемые ячейки инвертируются, и повторно вычисляется сигнатура $S2$. (После каждого доступа по чтению к одной из оставшихся ячеек инвертируемая ячейка также проверяется.) Сигнатура $S2$ сравнивается с сигнатурой $S1$ и при любом различии выдается сообщение об ошибке. Тестируемая ячейка повторно инвертируется для повторного установления исходного содержимого и сигнатура $S3$ всех оставшихся ячеек повторно вычисляется и сравнивается с сигнатурой $S1$. Любые различия приводят к выдаче сообщения об ошибке. Все ячейки памяти в выбранной области тестируются тем же способом.

**А.5.4 Тест «Абраам» для памяти с произвольным доступом**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.6).

Цель. Обнаружение всех постоянных отказов и отказов в соединениях между ячейками памяти.

Описание. Диагностический охват выше, чем при тесте «попарная запись-считывание». Число операций, необходимых для выполнения всего тестирования памяти, составляет примерно 30 $n$, где $n$ — число ячеек памяти. Тестирование может быть «прозрачным» при выполнении запоминания и тестирования в различных временных сегментах в периоде рабочего цикла.

Литература:

Efficient Algorithms for Testing Semiconductor Random-Access Memories. R. Nair, S.M. Thatte, J.A. Abraham, IEEE Trans. Comput. C-27 (6), 572—576, 1978.

**А.5.5 Однобитовая избыточность (например контроль памяти с произвольным доступом с помощью бита четности)**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.6).

Цель. Обнаружение 50 % всех возможных битовых отказов в тестируемой области памяти.

Описание. Каждое слово в памяти расширяется на один бит (бит четности), который дополняет каждое слово до четного или нечетного числа логических единиц. Четность слова данных проверяется при каждом чтении. При обнаружении ложного числа единиц выдается сообщение об ошибке. Выбор четности или нечетности должен осуществляться так, чтобы всякий раз в случае отказа не выдавалось ничего, кроме нулевого (0) и единичного (1) слова, вырабатывалось уведомление о том, что это слово неправильно закодировано. Контроль четности также может быть использован для обнаружения ошибок адресации, если четность определяется для объединения слова данных с его адресом.

**А.5.6 Контроль памяти с произвольным доступом с помощью модифицированного кода Хэмминга или обнаружение ошибок данных с помощью кодов обнаружения и исправления ошибок (EDC)**

Примечание — См. также А.4.1 и С.3.2. Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.6).

Цель. Обнаружение всех нечетных битовых отказов, всех двухбитовых отказов, некоторых трехбитовых отказов и некоторых многобитовых отказов.

Описание. Каждое слово в памяти расширяется несколькими избыточными битами для выработки модифицированного кода Хэмминга с расстоянием Хэмминга, равным по меньшей мере 4. При каждом считывании слова проверка избыточных битов может указывать, произошло ли искажение. При обнаружении различий выдается сообщение об отказе. Эта процедура может быть также использована для обнаружения ошибок адресации при вычислении избыточных битов для объединения данных с его адресом.

Литература:

Prüfbare und korrigierbare Codes. W.W. Peterson, München, Oldenburg, 1967.

Error detecting и error correcting codes. R.W. Hamming, The Bell System Technical Journal 29 (2), 147—160, 1950.

**А.5.7 Дублирование со сравнением памяти с произвольным доступом с аппаратными или программными средствами и тестирование чтением/записью**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.6).

Цель. Обнаружение всех битовых отказов.

Описание. Адресное пространство содержит две части. Первая часть памяти функционирует в нормальном режиме. Вторая часть памяти содержит ту же информацию и доступна параллельно с первой. Выходы этих частей памяти сравниваются. При обнаружении различий выдается сообщение об ошибке. Для обнаружения некоторых видов битовых ошибок данные должны сохраняться инверсно в одной из двух частей памяти и обратно инвертироваться при чтении.

**А.6 Устройства ввода-вывода и интерфейсы (внешний обмен)**

Главная цель. Обнаружение отказов на устройствах ввода и вывода (цифровые, аналоговые, последовательные или параллельные) и предотвращение дальнейшей передачи недопустимых выходных данных.

**А.6.1 Тестирующая комбинация**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблицы А.7, А.13 и А.14).

Цель. Обнаружение статических отказов (константные отказы) и перекрестных помех.

Описание. Этот метод реализует независимое от потока данных циклическое тестирование входных и выходных элементов. В нем используются определенные тестирующие комбинации для сравнения с соответствующими этим тестирующим комбинациям предполагаемыми значениями. Информация тестирующей комбинации, восприятие и оценка тестирующей комбинации должны быть независимы друг от друга. Тестирующие комбинации не должны неблагоприятно влиять на операции УО.

**А.6.2 Кодовая защита**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблицы А.7, А.15, А.16 и А.18).

Цель. Обнаружение случайных отказов аппаратных средств и систематических ошибок в потоке ввода/вывода данных.

Описание. Процедура, реализующая кодовую защиту, защищает вводимую и выводимую информацию от систематических и случайных отказов аппаратных средств. Кодовая защита обеспечивает зависимое от потока дан-

ных обнаружение отказов входных и выходных модулей, основываясь на избыточности информации и/или временной избыточности. Обычно избыточная информация налагается на входные и/или выходные данные; тем самым обеспечиваются средства для мониторинга правильности операций входных и выходных схем. Возможно применение многих методов — например, сигнал несущей частоты может налагаться на выходной сигнал датчика. После этого логический модуль может проверить наличие частоты несущей, либо на выходе канала могут быть добавлены избыточные кодовые биты для контроля действительности прохождения сигнала между логическим модулем и оконечным исполнительным механизмом.

**А.6.3 Многоканальное параллельное выходное устройство**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.7).

Цель. Обнаружение случайных отказов аппаратных средств (константные отказы), отказов, обусловленных внешними воздействиями, временных сбоев, отказов адресации, постепенных отказов и самоустраняющихся отказов.

Описание. Это зависимое от потока данных многоканальное параллельное выходное устройство с независимыми выходами для обнаружения случайных аппаратных отказов. Обнаружение отказов осуществляется с помощью внешних компараторов. При появлении отказа УО непосредственно отключается. Это устройство действует только в том случае, если поток данных изменяется в интервале диагностического тестирования.

**А.6.4 Средство контроля выходов**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.7).

Цель. Обнаружение случайных отказов, отказов, обусловленных внешними воздействиями, временных сбоев, отказов адресации, постепенных отказов (для аналоговых сигналов) и самоустраняющихся отказов.

Описание. Это устройство, зависимое от потока данных, сравнивает выходные данные с независимыми входными данными для обеспечения совместимости с областью допустимых значений (время, диапазон). Обнаруженный отказ не всегда относится к неправильному выходному сигналу. Это устройство действует только в том случае, если поток данных изменяется в интервале диагностического тестирования.

**А.6.5 Сравнение/голосование входных данных**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблицы А.7 и А.13).

Цель. Обнаружение случайных отказов, отказов, обусловленных внешними воздействиями, временных сбоев, отказов адресации, постепенных отказов (для аналоговых сигналов) и самоустраняющихся отказов.

Описание. Это устройство, зависимое от потока данных, сравнивает выходные данные с независимыми входными данными для обеспечения совместимости с областью допустимых значений (время, диапазон). Реализуемая избыточность может быть 1 из 2, 2 из 3 или более высокая. Это устройство действует только в том случае, если поток данных изменяется в интервале диагностического тестирования.

**А.7. Маршруты данных (внутренний обмен)**

Главная цель. Обнаружение отказов, обусловленных искажениями при передаче информации.

**А.7.1 Однобитовая аппаратная избыточность**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.8).

Цель. Обнаружение всех нечетно-битовых ошибок, то есть 50 % всех возможных битовых ошибок в потоке данных.

Описание. Шина расширяется на одну линию (бит) и эта дополнительная линия (бит) используется для обнаружения отказов путем проверки на четность.

**А.7.2 Многобитовая аппаратная избыточность**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.8).

Цель. Обнаружение отказов в процессе передачи по шине и в последовательных каналах связи.

Описание. Шина расширяется на две или более линий (битов) и эти дополнительные линии (биты) используются для обнаружения отказов методом кода Хэмминга.

**А.7.3 Полная аппаратная избыточность**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.8).

Цель. Обнаружение отказов в процессе передачи данных путем сравнения сигналов двух шин.

Описание. Шина дублируется и дополнительные линии (биты) используются для обнаружения отказов.

**А.7.4 Анализ с использованием тестирующих комбинаций**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.8).

Цель. Обнаружение статических отказов (постоянных отказов) и перекрестных помех.

Описание. Осуществляется независимое от потока данных циклическое тестирование маршрутов данных. Используется определенная тестирующая комбинация для сравнения наблюдаемых значений с соответствующими предполагаемыми значениями.

Восприятие информации о тестирующей комбинации и ее оценка должны быть независимы друг от друга. Тестирующие комбинации не должны неблагоприятно влиять на операции УО.

**А.7.5 Избыточность при передаче**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.8).

Цель. Обнаружение самоустраняющихся отказов в обмене по шине.

Описание. Информация передается последовательно несколько раз. Повторение осуществляется только для обнаружения самоустраняющихся отказов.

**А.7.6 Информационная избыточность**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.8).

Цель. Обнаружение отказов в обмене по шине.

Описание. Данные передаются блоками наряду с вычислениями контрольной суммы для каждого блока. После этого приемник повторно вычисляет контрольную сумму полученных данных. Результат сравнивается с полученной контрольной суммой.

**А.8 Источник питания**

Главная цель. Обнаружение или устойчивость к отказам, обусловленным источником питания.

**А.8.1 Защита от броска напряжения с помощью безопасного выключения**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.9).

Цель. Защита систем, связанных с безопасностью, от броска напряжения.

Описание. Бросок напряжения обнаруживается достаточно рано с тем, чтобы все выходы могли быть переключены в безопасное состояние процедурой отключения питания или переключились на второй блок питания.

Литература:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6.

**А.8.2 Управление напряжением (вторичным)**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.9).

Цель. Контроль вторичных напряжений и инициализация безопасного состояния, если значение напряжения не находится в заданном диапазоне.

Описание. Вторичное напряжение контролируется и питание отключается, либо происходит переключение на второй блок питания, если напряжение не находится в заданном диапазоне.

**А.8.3 Отключение системы безопасности при снижении питания**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.9).

Цель. Отключение питания с сохранением системой безопасности имеющейся информации.

Описание. Заранее выявляется бросок напряжения или слишком низкое напряжение питания для того, чтобы процедура отключения питания успела сохранить внутреннее состояние в незнергозависимой памяти (при необходимости) и тем самым установить все выходы в безопасное состояние, или же все выходы переключить в безопасное состояние, либо происходит переключение на второй блок питания.

**А.9 Временной и логический контроль последовательности выполнения программ**

Примечание — На эту группу методов или средств даны ссылки в МЭК 61508-2 (см. приложение А, таблицы А.15, А.16 и А.18).

Главная цель. Обнаружение искаженных программных последовательностей. Искаженная программная последовательность появляется, если отдельные элементы программы (например программные модули, подпрограммы или команды) обрабатываются в неправильной последовательности, или в несоответствующий период времени, или если сбилась тактовая частота процессора.

**А.9.1 Контрольный датчик времени с отдельной временной базой без временного окна**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблицы А.10 и А.11).

Цель. Контроль поведения и последовательности выполнения программ.

Описание. Внешние средства определения времени с отдельной базой времени (например контрольный датчик времени) периодически переключаются для контроля поведения компьютера и последовательности выполнения программ. Важно, чтобы моменты переключения были правильно расположены в программе. Контрольный датчик времени не переключается с некоторым фиксированным периодом, однако задается максимальный интервал.

**А.9.2 Контрольный датчик времени с отдельной временной базой и временным окном**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблицы А.10 и А.11).

Цель. Контроль поведения и последовательности выполнения программ.

Описание. Внешние средства определения времени с отдельной базой времени (например контрольный датчик времени) периодически переключаются для контроля поведения компьютера и последовательности выполнения программ. Важно, чтобы моменты переключения были правильно расположены в программе. Контрольный датчик времени не переключается с некоторым фиксированным периодом, однако задается максимальный интервал. Если последовательности программ выполняются больше или меньше ожидаемого времени, то выполняется действие чрезвычайного случая.

**А.9.3 Логический контроль последовательности выполнения программ**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблицы А.10 и А.11).

Цель. Контроль правильной последовательности выполнения отдельных частей программы.

Описание. Правильная последовательность выполнения отдельных частей программы контролируется с помощью программных средств (процедур учета, ключевых процедур) или с использованием внешних средств контроля. Важно, чтобы точки проверки располагались в программе правильно.

**А.9.4 Комбинация временного и логического контроля последовательности выполнения программ**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблицы А.10 и А.11).

Цель. Контроль поведения и правильной последовательности выполнения отдельных частей программы.

Описание. Средство определения времени (например контрольный датчик времени), контролирующее программную последовательность, вновь запускается только в случае, если последовательность модулей программы выполняется правильно.

**А.9.5 Первоначальный тест при включении**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблицы А.10 и А.11).

Цель. Обнаружение отказов при первоначальном тесте.

Описание. При запуске проводится первоначальный тест. Запуск возможен только в случае, если первоначальный тест прошел успешно. Например, датчик температуры может быть проверен нагретым резистором при запуске.

**А.10 Вентиляция и температура**

Примечание — На эту группу методов и средств дана ссылка в МЭК 61508-2 (см. приложение А, таблицы А.16 и А.17).

Главная цель. Управление отказами в системах вентиляции и температурных приборах и/или их контроль, если они связаны с безопасностью.

**А.10.1 Датчик температуры**

Цель. Обнаружение температурного перегрева или недогрева до того, как система начнет действовать вне заданных требований.

Описание. Датчик температуры контролирует температуру в наиболее критических точках Э/Э/ПЭ системы, связанной с безопасностью. Прежде чем температура выйдет из заданного диапазона, происходит аварийное действие.

**А.10.2 Управление вентиляцией**

Цель. Обнаружение отказов в работе вентилятора.

Описание. Работа вентиляторов контролируется. Если вентилятор находится в нерабочем состоянии, то предпринимаются действия по восстановлению его рабочего состояния (или его аварийному отключению).

**А.10.3 Безопасное выключение с использованием плавкого предохранителя**

Цель. Выключение системы, связанной с безопасностью, до того как параметры системы выйдут из заданных температурных режимов.

Описание. Плавкий предохранитель используется для выключения систем, связанных с безопасностью. В ПЭ системе выключение осуществляется процедурой отключения питания, которая хранит информацию, необходимую при аварийных действиях.

**А.10.4 Пороговые сообщения от термодатчиков и условная тревога**

Цель. Показать, что система, связанная с безопасностью, работает также за пределами допусков по температуре.

Описание. Измеряется температура, а при ее выходе из заданного диапазона выдается аварийный сигнал.

**А.10.5 Соединение устройства принудительного охлаждения воздуха и индикатора состояний**

Цель. Не допустить перегрева путем искусственного воздушного охлаждения.

Описание. Измеряется температура. Если температура превышает заданный предел, то включается искусственное воздушное охлаждение. Пользователь информируется об измеренном значении температуры.

**А.11 Обмен и запоминающее устройство большой емкости**

Главная цель. Контроль отказов в процессе обмена между внешними источниками и запоминающим устройством большой емкости.

**А.11.1 Разделение линий электрического питания и линий передачи информации**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.16).

Цель. Минимизировать перекрестные помехи информационных линий, индуцируемые сильным током системы питания.

Описание. Линии, обеспечивающие электрическое питание, отделяются от линий, переносящих информацию. Электрическое поле, которое может индуцировать на информационных линиях всплески напряжения, уменьшается с увеличением расстояния.

**А.11.2 Пространственное разделение групповых линий**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.16).

Цель. Минимизировать перекрестные помехи, индуцируемые током системы питания в групповых линиях.

Описание. Линии с дублирующими сигналами отделяются друг от друга. Электрическое поле, которое могут индуцировать броски напряжений в групповых линиях, уменьшается с увеличением расстояния. Это отделение линий снижает также отказы по общей причине.

**А.11.3 Повышение устойчивости к электромагнитным воздействиям**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблицы А.16 и А.18).

Цель. Минимизировать электромагнитные влияния на систему, связанную с безопасностью.

Описание. Создание таких методов, как экранирование и фильтрация, для уменьшения чувствительности систем, связанных с безопасностью, к электромагнитным полям, которые могут наводиться на линии питания или сигнальные линии, либо возникать в результате электростатических разрядов.

Примечание — Требования к устойчивости систем, связанных с безопасностью, и оборудования при выполнении функций, связанных с безопасностью (функциональная безопасность), для промышленных применений см. в [5] и [6].

Литература:

IEC/TR 61000-5-2:1997, Electromagnetic compatibility (EMC) — Part 5: Installation and mitigation guidelines — Section 2: Earthing and cabling.

Principles and Techniques of Electromagnetic Compatibility, Second Edition, C. Christopoulos, CRC Press, 2007, ISBN-10: 0849370353, ISBN-13: 978-0849370359.

Noise Reduction Techniques in Electronic Systems. H.W. Ott, John Wiley Interscience, 2nd Edition, 1988.

EMC for Product Designers. Tim Williams, Newnes, 2007, ISBN 0750681705.

Grounding and Shielding Techniques in Instrumentation, 3 edition, R. Morrison Wiley-Interscience, New York, 1986, ISBN-10: 0471838055, ISBN-13: 978-0471838050.

**А.11.4 Передача неэквивалентных сигналов**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблицы А.7 и А.16).

Цель. Обнаружение одинаковых индуцированных напряжений в групповых линиях передачи сигналов.

Описание. Вся дублируемая информация передается с неэквивалентными сигналами (например логические 1 и 0). Ошибки по общей причине (например вызванные электромагнитными излучениями) могут быть обнаружены неэквивалентным компаратором.

Литература:

Elektronik in der Sicherheitstechnik. H. Jürs, D. Reinert, Sicherheitstechnisches Informations- und Arbeitsblatt 330220, BIA-Handbuch. 20. Lfg. V/93, Erich Schmidt Verlag, Bielefeld. gost.e-ecolog.ru

**А.12 Датчики**

Главная цель. Управление отказами в датчиках систем, связанных с безопасностью.

**А.12.1 Эталонный датчик**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.13).

Цель. Обнаружение отказа датчика.

Описание. Для контроля работоспособности датчика используется независимый эталонный датчик. Все входные сигналы в подходящие временные интервалы проверяются эталонным датчиком для обнаружения отказов в работе проверяемого датчика.

Литература:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6.

**А.12.2 Положительно управляемый переключатель**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.14).

Цель. Разомкнуть контакт с помощью непосредственного механического соединения между кулачком переключателя и контактом.

Описание. Положительно управляемый переключатель размыкает свои обычно замкнутые контакты непосредственным механическим соединением между кулачком переключателя и контактом. Разомкнутость контактов переключателя обеспечивается всякий раз, когда кулачок переключателя находится в рабочем положении.

Литература:

Verriegelung beweglicher Schutzeinrichtungen. F. Kreutzkampf, K. Becker, Sicherheitstechnisches Informations- und Arbeitsblatt 330210, BIA-Handbuch. 1. Lfg. IX/85, Erich Schmidt Verlag, Bielefeld.

**А.13 Исполнительные элементы (приводы)**

Главная цель. Управление отказами в исполнительных элементах систем, связанных с безопасностью.

**А.13.1 Мониторинг**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.14).

Цель. Обнаружение отказа привода.

Описание. Операции привода контролируются (например положительно управляемыми контактами реле; см. контроль контактов реле в А.1.2). Избыточность, вносимая этим контролем, может быть использована для переключения на аварийный режим.

Литература:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6.

Zusammenstellung und Bewertung elektromechanischer Sicherheitschaltungen für Ver-riegelungseinrichtungen. F. Kreutzkampf, W. Hertel, Sicherheitstechnisches Informations- und Arbeitsblatt 330212, BIA-Handbuch. 17. Lfg. X/91, Erich Schmidt Verlag, Bielefeld.

**А.13.2 Перекрестный контроль групповых приводов**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблица А.14).

Цель. Обнаружение отказов в приводах путем сравнения результатов контроля.

Описание. Каждый групповой привод контролируется своим аппаратным каналом. При обнаружении различий вырабатывается аварийное действие.

**А.14 Средства против физического воздействия окружающей среды**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (см. приложение А, таблицы А.16 и А.18).

Цель. Предотвратить влияние физической окружающей среды (влажности, пыли, коррозийных субстанций), вызывающей отказы.

Описание. Покрытие оборудования должно противостоять чрезмерным внешним воздействиям.

Литература:

IEC 60529:1989, Degrees of protection provided by enclosures (IP Code).

**Приложение В**
**(справочное)**

## Анализ методов и средств для Э/Э/ПЭ систем, связанных с безопасностью. Предотвращение систематических отказов (см. МЭК 61508-2 и МЭК 61508-3)

Примечание — Многие методы, представленные в данном приложении, относятся и к программным средствам, но в приложении С они не описаны.

### В.1 Общие методы и средства

#### В.1.1 Управление проектами

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы В.1 — В.6).

Цель. Устранение отказов с использованием организационной модели, правил и средств по разработке и тестированию систем, связанных с безопасностью.

Описание. Наиболее значимыми и лучшими средствами являются:

- создание организационной модели в основном для обеспечения качества, описанное во многих работах по обеспечению качества, и
- установление правил и определение средств для создания и подтверждения соответствия систем, связанных с безопасностью, в руководствах по взаимосвязанным и отдельным проектам.

Для управления проектами установлены следующие важные базовые принципы:

- при выборе проектной организации определяются:
  - задачи и ответственности подразделений конкретной организации,
  - полномочия департаментов по обеспечению качества,
  - независимость гарантии качества (при выполнении внутренней проверки) от разработки;
- план последовательных действий (модель действий) формируется как:
  - определение действий по выполнению проекта, включая внутренние проверки и график их проведения,
  - обновление проекта;
- стандартная последовательность для внутренней проверки определяется как:
  - планирование, проведение и контроль проверки (теория проверки),
  - использование различных механизмов проверок для составных частей,
  - сохранение результатов повторных проверок;
- управление конфигурацией реализуется как:
  - администрирование и проверка версий,
  - выявление результатов модификаций,
  - проверки согласованности после модификаций;
- вводятся количественные оценки для средств обеспечения качества в виде:
  - установления требований,
  - статистики отказов;
- вводятся автоматизированные универсальные методы, инструменты и средства обучения персонала.

Литература:

ISO 9001:2008, Quality management systems — Requirements.

ISO/IEC 15504 (all parts), Information technology — Process assessment.

CMMI: Guidelines for Process Integration and Product Improvement, 2nd Edition. M.B. Chrissis, M. Konrad, S. Shrum, Addison-Wesley Professional, 2006, ISBN-10: 0-3212-7967-0, ISBN-13: 978-0-3212-7967-5.

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6.

Dependability of Critical Computer Systems 1. F.J. Redmill, Elsevier Applied Science, 1988, ISBN 1-85166-203-0.

#### В.1.2 Документация

Примечания

1 Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы В.1 — В.6).

2 См. также МЭК 61508-1 (раздел 5 и приложение А).

Цель. Предотвращение отказов и упрощение процедуры оценки безопасности с помощью систем документирования каждого шага процесса разработки.

Описание. Во время процедуры оценки, наряду со всеми составляющими, включенными в разработку, необходимо также уделять внимание эксплуатационным характеристикам и безопасности. В процессе разработки и обеспечения в любой момент времени проверки доказательств безопасности особое внимание уделяется документации на систему.

Основными общими подходами к документированию являются введение руководящих принципов создания документов и использование автоматизации, в том числе:

- руководящие принципы:
  - определяют структуру документа,
  - используют таблицы контрольных проверок для формирования содержания документа и
  - определяют формат документа;
- автоматизация управляет документированием и создается структурированная библиотека проекта.

К конкретным методам создания документов относятся:

- разделение в документации описаний:
  - требований,
  - системы (документация пользователя) и
  - разработки (включая внутреннюю проверку);
- группирование разработанной документации в соответствии с жизненным циклом безопасности;
- определение стандартных модулей документации, из которых могут быть скомпилированы документы;
- ясная идентификация составных частей документа;
- формализованное обновление версий;
- выбор ясных и понятных средств описания:
  - формализованная нотация для определений,
  - естественный язык для введений, обоснований и представления намерений,
  - графическое представление для описания примеров,
  - семантическое определение для графических элементов и
  - терминологические справочники.

Литература:

IEC 61506:1997, Industrial-process measurement and control — Documentation of application software.

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6.

**B.1.3 Разделение систем, связанных с безопасностью, и систем, не связанных с безопасностью**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы В.1 и В.6).

Цель. Предотвращение влияния систем, связанных с безопасностью, на системы, не связанные с безопасностью, в непредвиденных ситуациях.

Описание. В спецификации должно быть определено, возможно ли разделение систем, связанных и не связанных с безопасностью. Должны быть установлены четкие спецификации взаимодействия между системами, связанными и не связанными с безопасностью. Четкое их разделение снижает затраты на тестирование систем, связанных с безопасностью.

Литература:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6.

**B.1.4 Разнообразие аппаратных средств**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы А.15, А.16 и А.18).

Цель. Обнаружение систематических отказов при выполнении операций УО с использованием разнообразных компонентов с различными частотами и типами отказов.

Описание. Для разнообразных каналов системы, связанной с безопасностью, используются различные типы компонентов. Это снижает вероятность отказов по общей причине (например увеличение напряжения по сравнению с номиналом, электромагнитные влияния) и повышает вероятность обнаружения таких отказов.

Существование различных средств выполнения требуемой функции, например, применение других физических принципов, предполагает возможность использования других способов решения проблемы обнаружения систематических отказов. Возникает разнообразие типов используемых способов. Это функциональное разнообразие дает возможность использовать различные подходы для достижения одного и того же результата.

Литература:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6.

**B.2 Спецификация требований к проектированию Э/Э/ПЭ системы**

Главная цель. Создание спецификации, которая по возможности была бы полна, свободна от ошибок, противоречий и проста для проверки.

**B.2.1 Структурирование спецификации**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы В.1 и В.6).

Цель. Уменьшение сложности путем создания иерархической структуры частичных требований. Предотвращение ошибок взаимосвязи между требованиями.

Описание. Данный метод разделяет функциональную спецификацию на частичные требования так, чтобы между ними существовали по возможности простейшие отношения. Этот метод применяется последовательно до тех пор, пока не будут получены небольшие четкие частичные требования. В результате получается иерархи-

ческая структура частичных требований, которая создает основу для спецификации полных требований. Данный метод выделяет взаимосвязи между частичными требованиями и особенно эффективен при его использовании для исключения ошибок в этих взаимосвязях.

Литература:

ESA PSS 05-02, Guide to the user requirements definition phase, Issue 1, Revision 1, ESA Board for Software Standardisation and Control (BSSC), ESA, Paris, March 1995, ftp://ftp.estec.esa.nl/pub/wm/wme/bssc/PSS0502.pdf.

Structured Analysis and System Specification. T. De Marco, Yourdon Press, Englewood Cliffs, 1979, ISBN-10: 0138543801, ISBN-13: 978-0138543808.

**B.2.2 Формальные методы**

Примечания

1 Подробные сведения о конкретных формальных методах приведены в С.2.4.

2 Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы В.1, В.2 и В.6).

Цель. Формальные методы позволяют строить спецификации и реализации технических систем на принципах математического рассуждения, что повышает завершенность, непротиворечивость или корректность спецификации или реализации.

Описание. Формальные методы обеспечивают средства разработки описания системы на конкретном этапе ее спецификации или проектирования. Такие формальные описания являются математическими моделями функции и/или структуры системы.

Поэтому может быть обеспечено однозначное описание системы (например любое состояние автомата описано его начальным состоянием, входами и уравнениями перехода автомата из одного состояния в другое), которое увеличивает понимание основной системы.

Выбор подходящего формального метода является трудной задачей, требующей полного понимания системы, ее процесса разработки и уровня используемых математических моделей (см. примечания).

Примечания

1 Теоремы модели (описывающие свойства) гарантированно представляют описание системы, которое обеспечивает гораздо большее доверие, чем моделирование, заключающееся в наблюдении отдельных действий системы.

2 Недостатками формальных методов могут быть:

- фиксированный уровень абстракции;
- ограничения в получении всей функциональности, которая относится к данному этапу;
- трудность понимания модели инженерами, которые ее реализуют;
- значительные усилия, необходимые для разработки, анализа и поддержки модели на всем жизненном цикле системы;
- недостаток эффективных инструментов, которые поддерживают создание и анализ модели;
- недостаток персонала, способного разрабатывать и анализировать модель.

3 Интерес сообщества, занимающегося формальными методами, был явно направлен на моделирование целевой функции системы, часто преуменьшая роль проблемы отказоустойчивости системы. Поэтому должны выбираться соответствующие формальные методы, включающие возможность решения проблемы устойчивости системы.

Литература:

Formal Specification: Techniques and Applications. N. Nissanke, Springer-Verlag Telos, 1999, ISBN-10: 1852330023.

**B.2.3 Полуформальные методы**

Примечание — В отличие от приведенного ниже в МЭК 61508-3, таблица В.7, представлен список полуформальных методов, расширенный методами, связанными с программным обеспечением:

- логические диаграммы/диаграммы функциональных блоков описаны в [7];
- циклограммы описаны в [7];
- диаграммы потоков данных: см. С.2.2;
- конечные автоматы/диаграммы переходов см. В.2.3.2;
- временные сети Петри см. В.2.3.3;
- модели данных сущность-связь-атрибут см. В.2.4.4;
- диаграммы последовательности сообщений см. С.2.14;
- таблицы решений/таблицы истинности см. С.6.1.

Цель. Создание недвусмысленных и согласованных частей спецификации в целях обнаружения ошибок, пропусков и неправильного поведения.

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы В.1, В.2 и В.6) и в МЭК 61508-3 (таблицы А.1, А.2, А.4, В.7, С.1, С.2, С.4 и С.17).

**B.2.3.1 Общие положения**

Цель. Убедиться в том, что проект соответствует своей спецификации.

Описание. Полуформальные методы обеспечивают средства создания описания системы на стадиях ее разработки (например спецификации, проектирования или кодирования). Описание может быть в некоторых случаях проанализировано на ЭВМ или для отображения различных аспектов поведения системы использована

анимация. Анимация может придать дополнительную уверенность в том, что система соответствует как реальным, так и специфицированным требованиям.

В следующих пунктах настоящего приложения описаны два полуформальных метода:

**В.2.3.2 Конечные автоматы/диаграммы переходов**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблицы В.5, В.7, С.15 и С.17).

Цель. Проведение моделирования, специфицирование или реализация структуры управления системы.

Описание. Многие системы могут быть описаны в терминах их состояний, входов и действий. Таким образом, находясь в состоянии *S*1 и при получении входа *I*, система может выполнить действие *A* и перейти в состояние *S*2. Путем описания действий системы для каждого входа в каждом состоянии можно полностью описать систему. Такая модель системы называется машиной с конечными состояниями (или конечными автоматами). Ее часто изображают в виде так называемой диаграммы переходов, которая показывает, как система переходит из одного состояния в другое, или в виде матрицы, в которой для каждого состояния и входа задаются действия по переходу в новое состояние.

В случае если система усложняется или имеет естественную структуру, то это может быть отражено в уровневой структуре конечного автомата. Диаграмма состояний — тип диаграммы переходов, в которой разрешены вложенные состояния (состояние объекта разделяется на два или больше подсостояний, которые могут развиваться параллельно, и, возможно, в некоторый момент могут опять объединиться в одно состояние); это добавляет выразительные возможности нотации переходов, но добавляет и дополнительную сложность, которая может быть нежелательной для системы, связанной с безопасностью. Диаграммы состояний имеют формальную (математическую) спецификацию. Диаграммы переходов могут применяться ко всей системе или к ее некоторому объекту или элементу.

Спецификация или проект, выраженные в виде конечного автомата, могут быть проверены на:

- полноту (система или объект должны иметь действие и новое состояние для каждого входа в каждом состоянии);
- согласованность (возможно только одно состояние для каждой пары состояние/вход);
- достижимость (возможно или нет перейти из одного состояния в другое с помощью некоторой последовательности входов) и
- отсутствие бесконечных циклов и тупиковых состояний и т. д.

Эти свойства являются важными для критических систем, так как легко разработать инструменты для обеспечения таких проверок, используя различные модели, основанные на теории конечных автоматов (формальные языки, сети Петри, графы Маркова и т. д.). Существуют также алгоритмы, позволяющие автоматически генерировать тестовые примеры для верификации реализаций конечных автоматов или анимации модели конечного автомата. Диаграммы переходов и диаграммы состояний широко поддерживаются инструментальными средствами, которые позволяют сформировать и проверить диаграммы, а также сгенерировать программный код для реализации описанного конечного автомата.

Они могут также использоваться для вычислений вероятности отказа, см. В.6 и С.6.

Литература:

Introduction to Automata Theory, Languages, and Computation (3rd Edition). J. Hopcroft, R. Motwani, J. Ullman, Addison-Wesley Longman Publishing Co, 2006, ISBN: 0321462254.

Sécurisation des architectures informatiques. Jean-Louis Boulanger, Hermès — Lavoisier, 2009, ISBN: 978-2-7462-1991-5.

**В.2.3.3 Моделирование во времени сетями Петри**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблицы В.5, В.7, С.15 и С.17).

Цель. Моделирование соответствующих аспектов поведения системы, оценка и, возможно, повышение безопасности и эксплуатационных требований путем анализа и повторного проектирования.

Описание. Метод сетей Петри является частным случаем метода конечных автоматов. Сети Петри относятся к классу моделей, описываемых теорией графов, и используются для представления информации и управления потоками в системах, в которых процессы конкурентны и асинхронны.

Сеть Петри — это сеть позиций и переходов. Позиции могут быть «маркированными» или «немаркированными». Переход считают «активизированным», если все его входы маркированы. В активизированном состоянии позиции разрешается (но не требуется) быть «возбужденной». Если позиция «возбуждена», то вход, поступающий на переход, становится немаркированным, а вместо него каждый выход из перехода оказывается маркированным.

Потенциальные опасности могут быть представлены в виде конкретных состояний (маркировок) в модели сети Петри. Модель может быть расширена с тем, чтобы обеспечить возможности моделирования систем во времени. И хотя «классические» сети Петри концентрируются на моделировании потоков управления, существуют некоторые расширения модели сети Петри, в которых моделируются потоки данных.

Литература:

Timed Petri Nets: Theory and Application. Jiacun Wang, Springer, 1998, ISBN 0792382706.

Sécurisation des architectures informatiques. Jean-Louis Boulanger, Hermès — Lavoisier, 2009, ISBN: 978-2-7462-1991-5.

**В.2.4 Автоматизированные средства разработки спецификации**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы В.1 и В.6) и в МЭК 61508-3 (таблицы А.1, А.2, С.1 и С.2).

**В.2.4.1 Общие положения**

Цель. Использование формальных методов спецификации для упрощения автоматического обнаружения неоднозначностей и полноты.

Описание. Данный метод создает спецификацию в виде базы данных, которая может автоматически анализироваться для оценки согласованности и полноты. Инструмент спецификации может в интересах пользователя использовать анимацию различных аспектов специфицированной системы. В общем случае данный метод поддерживает создание не только спецификаций, но и этап проектирования, а также другие этапы жизненного цикла. Инструменты спецификаций могут быть классифицированы в соответствии со следующими пунктами настоящего приложения.

**В.2.4.2 Инструменты, неориентированные на конкретный метод**

Цель. Помощь пользователю в составлении правильной спецификации, применяя подсказки и формируя связи между соответствующими частями.

Описание. Инструмент спецификаций освобождает пользователя от некоторой рутинной процедуры, поддерживает управление проектом и не представляет собой какую-либо конкретную методологию разработки спецификаций. Относительная независимость пользователей от метода позволяет быть более свободными при выборе конкретного метода и дает немного специальной поддержки, необходимой при создании спецификаций, что усложняет освоение системы.

**В.2.4.3 Процедура, ориентированная на модель с иерархическим анализом**

Цель. Предотвратить неполноту, неоднозначность и противоречивость в спецификации, например, помогая пользователю в создании правильной спецификации, обеспечении согласованности между описаниями процессов и данных на различных уровнях абстрагирования.

Описание. Данный метод дает функциональное представление о необходимой системе (структурный анализ) на различных уровнях абстрагирования (степень точности). Существует огромный арсенал таких моделей: конечные автоматы — класс таких моделей, широко используемых для описания построения дискретных/цифровых систем. Дифференциальные уравнения подобны по духу и целям для описания непрерывных/аналоговых систем. Структурный анализ проводится на различных уровнях абстрагирования как с процессами, так и с данными. Оценка неоднозначности и полноты возможна между иерархическими уровнями, а также между двумя функциональными единицами (модулями) на одном и том же уровне (например любое состояние модели системы описывается ее начальным состоянием, входами и уравнениями перехода автомата из одного состояния в другое).

Примечание — Проблемами при описании, основанном на моделях, могут быть: уровень абстракции; ограничения для получения всей функциональности, относящейся к данному этапу; трудности понимания модели практиками (от чтения синтаксиса до ее правильной интерпретации); значительные усилия, затрачиваемые на разработку, анализ и поддержку модели на всем жизненном цикле системы; доступность эффективных инструментов, поддерживающих создание и анализ модели (разработка таких инструментов, конечно, требует больших усилий); и наличие специалистов, способных разрабатывать и анализировать модели.

Литература:

System requirements analysis. Jeffrey O. Grady, Academic Press, 2006, ISBN 012088514X, 9780120885145.

**В.2.4.4 Модели данных сущность—связь—атрибут**

Цель. Помощь пользователю в создании правильной спецификации на основе использования при описании системы сущностей и отношений между ними.

Описание. Описание проектируемой системы в виде совокупности объектов и отношений между ними позволяет определять, какие отношения могут интерпретироваться системой. В общем случае эти отношения позволяют описывать иерархическую структуру объектов, поток данных, отношения между данными и данные, зависимые от конкретных производственных процессов. Этот классический подход расширен применением управления процессами. Возможности обследования и поддержка пользователя зависят от разнообразия проиллюстрированных отношений. Но множество возможностей представления усложняет применение этого метода.

Литература:

Software Requirements: Practical Techniques for Gathering and Managing Requirements Throughout the Product Development Cycle. Karl Eugene Wiegers, Microsoft Press, 2003, ISBN 0735618798, 9780735618794.

**В.2.4.5 Стимул и ответ**

Цель. Помощь пользователю в создании правильной спецификации путем идентификации взаимоотношений «стимул — ответ».

Описание. Взаимоотношения между объектами системы определены в нотации «стимулы» и «ответы». Используется простой и легко расширяемый язык, который содержит элементы языка, представляющие объекты, взаимоотношения, характеристики и структуры.

**В.2.5 Таблица контрольных проверок**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы В.1, В.2 и В.6) и в МЭК 61508-3 (таблицы А.10, В.8, С.10 и С.18).

Цель. Рассмотрение и управление критическими оценками всех важных аспектов системы на этапе жизненного цикла безопасности, обеспечивая исчерпывающий охват без установления точных требований.

Описание. Специалист, заполняющий таблицу контрольных проверок, должен дать ответ на ряд вопросов. Многие вопросы носят общий характер, и он должен интерпретировать их как наиболее подходящие к конкретной оцениваемой системе. Таблицы контрольных проверок допускается использовать на всех этапах полного жизненного цикла безопасности, жизненного цикла безопасности Э/Э/ПЭ системы и жизненного цикла безопасности программного обеспечения. Такие таблицы, в частности, полезны в качестве инструмента для оценки функциональной безопасности.

Для сокращения широкого разнообразия проходящих подтверждение соответствия систем большинство таблиц контрольных проверок содержат вопросы, которые применимы ко многим типам систем. Поэтому в используемой таблице контрольных проверок может оказаться множество вопросов, которые не уместны для конкретной системы и должны игнорироваться. Кроме того, может также возникнуть необходимость дополнить стандартную таблицу контрольных проверок вопросами, специально ориентированными на конкретную систему.

Использование таблицы контрольных проверок в большой степени зависит от экспертной оценки и суждения специалиста, который выбирает и применяет таблицу контрольных проверок. Принятые им решения относительно выбранных(ой) таблиц(ы) контрольных проверок и любые дополнительные или игнорируемые вопросы должны быть полностью документально оформлены и обоснованы. Необходимо стремиться к тому, что если применение таблиц контрольных проверок пересматривается, то гарантируется получение одних и тех же результатов, если только не используются различные критерии.

Описание системы в заполненной таблице контрольных проверок должно быть как можно более кратким. При необходимости исчерпывающего обоснования оно должно быть дано в виде ссылок на дополнительные документы. Для документирования результатов каждого вопроса должен использоваться ответ «успешно», «неуспешно» или «не завершено», либо аналогичный набор ответов. Эта лаконичность значительно упрощает процедуру оформления общего заключения результатов оценки в виде таблицы контрольных проверок.

Литература:

IEC 60880:2006, Nuclear power plants — Instrumentation and control systems important to safety — Software aspects for computer-based systems performing category A functions.

The Art of Software Testing, Second Edition. G. Myers et al., Wiley & Sons, New York, 2004, ISBN 0471469122, 9780471469124.

Software Quality Assurance: From Theory to Implementation. Daniel Galin, Pearson Education, 2004, ISBN 0201709457, 9780201709452.

IEC 61346 (all parts), Industrial systems, installations and equipment and industrial products — Structuring principles and reference designation.

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6.

Risk Assessment and Risk Management for the Chemical Process Industry. H.R. Greenberg, J.J. Cramer, John Wiley and Sons, 1991, ISBN 0471288829, 9780471288824.

**B.2.6 Экспертиза спецификации**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы B.1 и B.6).

Цель. Исключить некомплектность и противоречивость спецификации.

Описание. Общий метод анализа спецификации для ее оценки с различных сторон независимой командой экспертов. Такая команда задает вопросы разработчику, который должен дать ей удовлетворительные ответы. Анализ должен (по возможности) проводиться командой экспертов, которая не принимала участия в создании спецификации. Требуемая степень независимости оценки определяется уровнями полноты безопасности, задаваемыми для системы. Команда независимых экспертов должна быть способна реконструировать эксплуатационную функцию системы бесспорным способом без ссылок на любые последующие спецификации. Специалисты команды независимых экспертов должны также убедиться в том, что охвачены все уместные аспекты безопасности и технические аспекты в эксплуатационных и организационных средствах. Общий метод экспертизы спецификации доказал свою высокую эффективность на практике.

Литература:

IEC 61160:2005, Design review

The Art of Software Testing, Second Edition. G. Myers et al., Wiley & Sons, New York, 2004, ISBN 0471469122, 9780471469124.

Software Quality Assurance: From Theory to Implementation. D. Galin, Pearson Education, 2004, ISBN 0201709457, 9780201709452.

**B.3 Проектирование и разработка Э/Э/ПЭ системы**

Главная цель. Создание устойчивого проекта системы, связанной с безопасностью, в соответствии со спецификацией.

**B.3.1 Соблюдение руководящих материалов и стандартов**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблица B.2).

Цель. Рассмотрение стандартов сектора применения (не рассматриваемых в настоящем стандарте).

Описание. Во время проектирования системы, связанной с безопасностью, должны составляться руководящие материалы, которые должны, во-первых, приводить к созданию систем, связанных с безопасностью, и быть практически свободны от ошибок и, во-вторых, упрощать последующее подтверждение соответствия безопасности. Такие руководящие материалы могут быть универсальными, специальными для проекта или специальными только для отдельного этапа проекта.

Литература:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6.

**В.3.2 Структурное проектирование**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы В.2 и В.6).

Цель. Снижение сложности проектирования путем создания иерархической структуры частичных требований. Исключение ошибок взаимосвязей между требованиями. Упрощение верификации.

Описание. При проектировании аппаратных средств должны использоваться конкретные критерии или методы. Например, может потребоваться:

- проектирование иерархически структурированных схем;
- использование изготовленных и проверенных частей схем.

При проектировании программных средств использование структурных схем также позволяет создать однозначную структуру программных модулей. Данная структура показывает взаимосвязь модулей друг с другом, конкретные данные, которые передаются между модулями, и конкретное управление, существующее между модулями.

Литература:

IEC 61346 (all parts), Industrial systems, installations and equipment and industrial products — Structuring principles and reference designation.

Software Engineering for Real-time Systems. J.E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205.

Software Design. D. Budgen, Pearson Education, 2003, ISBN 0201722194, 9780201722192.

An Overview of JSD, J.R. Cameron, IEEE Trans SE-12 No. 2, February 1986.

Structured Development for Real-Time Systems (3 Volumes). P.T. Yourdon, P.T. Yourdon Press, 1985.

Structured Development for Real-Time Systems (3 Volumes). P.T. Ward, S.J. Mellor, Yourdon Press, 1985.

Applications and Extensions of SADT. D.T. Ross, Computer, 25—34, April 1985.

Essential Systems Analysis. St. M. McMenamin, F. Palmer, Yourdon Inc, 1984.

Structured Analysis (SA): A language for communicating ideas. D.T. Ross, IEEE Trans. Software Eng, Vol. SE-3 (1), 16—34.

**В.3.3 Использование достоверно испытанных компонентов**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы В.2 и В.6).

Цель. Снижение риска многих оригинальных и необнаруживаемых отказов путем использования компонентов с конкретными характеристиками.

Описание. Выбор достоверно испытанных компонентов для целей безопасности выполняется производителем в соответствии с надежностью компонентов (например использование эксплуатационно-тестируемых физических модулей для удовлетворения высоких требований безопасности или хранение относящихся к безопасности программ только в безопасной памяти). Обеспечение безопасности памяти может касаться устранения несанкционированного доступа, влияний несанкционированной среды (электромагнитная совместимость, радиация и т. д.), а также отклика компонентов в случае обнаружения отказов.

Литература:

IEC 61163-1:2006, Reliability stress screening — Part 1: Repairable assemblies manufactured in lots.

**В.3.4 Модульное проектирование**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы В.2 и В.6).

Цель. Снижение сложности и исключение ошибок, связанных с интерфейсами между подсистемами.

Описание. Каждая подсистема на всех уровнях проектирования четко определена и ограничена по размеру (только небольшим набором функций). Интерфейсы между подсистемами выполняются максимально простыми и пересечения (разделяемые данные, обмен информацией) минимизированы. Сложность отдельных подсистем также ограничивается.

Литература:

The Art of Software Testing, Second Edition. G. Myers et al., Wiley & Sons, New York, 2004, ISBN 0471469122, 9780471469124.

Software Engineering for Real-time Systems. J.E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205.

Software Reliability — Principles and Practices. G.J. Myers, Wiley-Interscience, New York, 1976, ISBN-10: 0471627658, ISBN-13: 978-0471627654.

**В.3.5 Средства автоматизированного проектирования**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы В.2 и В.6) и в МЭК 61508-3 (таблица А.4).

Цель. Более систематическое выполнение процедуры проектирования. Включение в проект подходящих автоматически сконструированных элементов, уже созданных и проверенных.

Описание. Инструменты автоматизированного проектирования (CAD) должны использоваться в процессе проектирования как аппаратных, так и программных средств, если они доступны и их использование обосновано сложностью системы. Корректность использования таких инструментов должна быть продемонстрирована конкретным тестированием, обширной предысторией удовлетворительного использования, либо независимой верификацией их результата для конкретной проектируемой системы, связанной с безопасностью.

Инструменты поддержки должны быть выбраны в соответствии с их уровнем интегрируемости. Инструменты считаются интегрируемыми, если они совместно работают так, что выходные данные одного инструмента по содержанию и формату подходят для автоматического ввода следующего инструментального средства, таким образом, минимизируя возможность внесения ошибки человеком в процессе его работы с промежуточными результатами.

Литература:

Overview of Technology Computer-Aided Design Tools and Applications in Technology Development, Manufacturing and Design. W. Fichtner, Journal of Computational and Theoretical Nanoscience, Volume 5, Number 6, June 2008, pp. 1089—1105(17).

The Electromagnetic Data Exchange: Much more than a Common Data Format. P.E. Frandsen et al. In Proceeding of the 2nd European Conference on Antennas and Propagation. The Institution of Engineering and Technology (IET), 2007, ISBN 978-0-86341-842-6.

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799.

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574.

**B.3.6 Моделирование**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы B.2, B.5 и B.6).

Цель. Проведение систематических и полных проверок как функционирования электрических/электронных схем, так и для корректного задания значений параметров их компонентов.

Описание. Функцию схемы, реализующую систему, связанную с безопасностью, имитируют на компьютере с помощью запрограммированной модели ее поведения. Поведение каждого отдельного компонента схемы моделируют отдельно, и отклик схемы, в которую он входит, анализируют при задании предельных значений параметров для каждого компонента.

**B.3.7 Проверка (обзор и анализ)**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы B.2 и B.6).

Цель. Выявление рассогласования между спецификацией и реализацией.

Описание. Проверяются заданные функции системы, связанной с безопасностью. Оценивается соответствие системы, связанной с безопасностью, требованиям, приведенным в спецификации. Все вызывающие сомнение ситуации при реализации и использовании изделий документируют в целях их последующего разрешения. В отличие от сквозного контроля во время процедуры проверки разработчик системы пассивен, а эксперт активен.

Литература:

IEC 61160:2005, Design Review.

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799.

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574.

The Art of Software Testing, Second Edition. G. Myers et al., Wiley & Sons, New York, 2004, ISBN 0471469122, 9780471469124.

ANSI/IEE 1028:1997, IEEE Standard for software reviews.

Dependability of Critical Computer Systems 3. P. G. Bishop et al., Elsevier Applied Science, 1990, ISBN 1-85166-544-7.

**B.3.8 Сквозной контроль**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблица B.6).

Цель. Выявление рассогласования между спецификацией и реализацией.

Описание. Проверяются заданные функции системы, связанной с безопасностью. Оценивается соответствие системы, связанной с безопасностью, требованиям, приведенным в спецификации. Все вызывающие сомнение ситуации при реализации и использовании изделий документируются в целях их последующего разрешения. В отличие от процедуры проверки во время сквозного контроля автор должен быть активен, а эксперт — пассивен.

Литература:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799.

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574.

ANSI/IEE 1028:1997, IEEE Standard for software reviews.

Dependability of Critical Computer Systems 3. P. G. Bishop et al., Elsevier Applied Science, 1990, ISBN 1-85166-544-7.

Methodisches Testen von Programmen. G. J. Myers, Oldenbourg Verlag, München, Wien, 1987.

### В.4 Процедуры эксплуатации и технического обслуживания Э/Э/ПЭ системы

Главная цель. Разработка процедур, которые исключают ошибки во время эксплуатации и обслуживания системы, связанной с безопасностью.

#### В.4.1 Инструкции по эксплуатации и техническому обслуживанию

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблица В.4).

Цель. Исключение ошибок во время эксплуатации и технического обслуживания систем, связанных с безопасностью.

Описание. Инструкции пользователя содержат важную информацию о способах использования и поддержки систем. В особых случаях эти инструкции могут содержать также примеры общих способов установки систем, относящихся к безопасности. Все инструкции должны легко восприниматься. Для описания сложных процедур и зависимостей должны использоваться рисунки и схемы.

Литература:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6.

#### В.4.2 Удобство общения с пользователем

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблица В.4).

Цель. Снижение сложности во время эксплуатации систем, связанных с безопасностью.

Описание. Правильность эксплуатации систем, связанных с безопасностью, в определенной степени зависит от оператора. Рассматривая конкретный проекты системы и рабочего места, разработчик систем, связанных с безопасностью, должен предусмотреть:

- необходимость минимального вмешательства человека;
- необходимое вмешательство наиболее простым способом;
- возможность минимального ущерба от ошибок оператора;
- эргономические требования при проектировании средств вмешательства и индикации;
- простые, имеющие четкую маркировку и удобные для использования средства оператора;
- неперенапряженность оператора даже в экстремальной ситуации;
- адаптированность обучения процедурам и средствам процесса вмешательства к уровню знаний и мотивации обучаемого пользователя.

#### В.4.3 Удобство общения с обслуживающим персоналом

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблица В.4).

Цель. Упрощение процедуры обслуживания систем, связанных с безопасностью, и проектирование необходимых средств для эффективной диагностики и ремонта.

Описание. Профилактическое обслуживание и ремонт часто проводятся в сложных условиях давления предельных сроков. Поэтому разработчик систем, связанных с безопасностью, должен предусмотреть, чтобы:

- средства, относящиеся к обслуживанию безопасности, требовались как можно реже или вообще не требовались;
- использовались достаточно чувствительные и легко управляемые диагностирующие инструменты для неизбежных ремонтов; эти инструменты должны включать в себя все необходимые интерфейсы;
- было достаточно времени (если отдельные инструменты диагностики необходимо разработать или приобрести).

#### В.4.4 Сокращение работ на стадии эксплуатации

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы В.4 и В.6).

Цель. Снизить эксплуатационные возможности для обычного пользователя.

Описание. Этот подход снижает эксплуатационные возможности путем:

- ограничения операций в рабочих режимах, например, коммутаторами ключей;
- ограничения числа используемых в работе элементов;
- ограничения числа возможных в общем случае рабочих режимов.

Литература:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6.

#### В.4.5 Эксплуатация только квалифицированным оператором

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы В.4 и В.6).

Цель. Исключение отказов, обусловленных ошибками оператора.

Описание. Оператор системы, связанной с безопасностью, обучен до степени, соответствующей уровню сложности и уровню полноты безопасности системы, связанной с безопасностью. В обучение входит изучение основ процесса производства и взаимосвязей между системами, связанными с безопасностью и УО.

Литература:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6.

**В.4.6 Защита от ошибок оператора**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы В.4 и В.6).

Цель. Защита системы от всех видов ошибок оператора.

Описание. Ложные входные сообщения (значение, время, и т. д.) обнаруживаются проверками достоверности или контролем УО. Для того, чтобы объединить эти средства в проекте, необходимо на самом раннем этапе определить, какие из входных сообщений возможны и какие допустимы.

**В.4.7** (Не используется)

**В.4.8 Защита от модификаций**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы А.17 и А.18).

Цель. Защита системы, связанной с безопасностью, от модификаций аппаратных средств техническими способами.

Описание. Модификации или манипуляции обнаруживаются автоматически, например, проверками достоверности сигналов датчиков, обнаружением техническим процессом и автоматическим запуском тестирования. При обнаружении модификации выдается аварийный сигнал.

**В.4.9 Подтверждение ввода**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы А.17 и А.18).

Цель. Обнаружение ошибки во время работы самим оператором до активизации УО.

Описание. Информация, вводимая в УО через систему, связанную с безопасностью, представляется оператору до передачи в УО с тем, чтобы оператор имел возможность обнаружить и исправить ошибки. Проектируемая система должна реагировать на неправильные, самопроизвольные действия оператора и учитывать нижние/верхние пределы скорости и направление реакции оператора. Это позволит исключить, например, более быстрое, чем предполагается, нажатие клавиш оператором, и настроить систему воспринимать двойное нажатие клавиши как одинарное или двойное за счет того, что система (изображение на экране) слишком медленно реагирует на разовое нажатие клавиши. Последовательное нажатие одной и той же клавиши при вводе критических данных должно восприниматься системой как одноразовое; нажатие клавиш «ввод» (enter) или «да» (yes) неограниченное число раз не должно приводить к нарушению безопасности системы.

Должны быть предусмотрены процедуры формирования временных пауз с возможностью выбора разных ответов (да/нет и т. п.) с тем, чтобы обеспечить возможность для размышления оператору, а системе — режим ожидания.

Любая перезагрузка ПЭ системы, связанной с безопасностью, делает эту систему уязвимой, если программные/аппаратные средства не спроектированы с учетом данной ситуации.

**В.5 Интеграция Э/Э/ПЭ системы**

Главная цель. Исключение отказов системы на стадии интеграции и обнаружение любых отказов во время этой и предыдущей стадий.

**В.5.1 Функциональное тестирование**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы В.3 и В.5) и в МЭК 61508-3 (таблицы А.5 — А.7 и С.5 — С.7).

Цель. Обнаружение отказов на стадиях создания спецификации и проектирования. Исключение отказов во время реализации и интеграции программных и аппаратных средств.

Описание. В процессе функционального тестирования определяется, достигнуты ли заданные характеристики системы. В систему поступают входные данные, которые адекватно характеризуют обычное выполнение операций. Наблюдаемые выходные результаты сравниваются с заданными в спецификации. Отклонения от спецификации и указания на неполноту спецификации документально оформляются.

Функциональное тестирование электронных компонентов, предназначенных для многоканальной архитектуры, обычно включает в себя промышленные компоненты, каждый из которых поставщик уже протестировал и предварительно подтвердил соответствие. Помимо этого рекомендуется, чтобы покупные промышленные компоненты были протестированы в сочетании с другими компонентами поставщика из той же партии, чтобы выявить неисправности группового типа, которые в противном случае остались бы невыявленными.

Также о достаточных рабочих возможностях системы см. руководящие материалы (приложение С, С.5.20).

Литература:

Software Testing and Quality Assurance. K. Naik, P. Tripathy, Wiley Interscience, 2008, Print ISBN: 9780471789116 Online ISBN: 9780470382844.

The Art of Software Testing, Second Edition. G. Myers et al., Wiley & Sons, New York, 2004, ISBN 0471469122, 9780471469124.

Practical Software Testing: A Process-oriented Approach. I. Burnstein, Springer, 2003, ISBN 0387951318, 9780387951317.

Dependability of Critical Computer Systems 3. P. G. Bishop et al., Elsevier Applied Science, 1990, ISBN 1-85166-544-7.

**В.5.2 Тестирование методом «черного ящика»**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы В.3, В.5 и В.6) и в МЭК 61508-3 (таблицы А.5 — А.7 и С.5 — С.7).

Цель. Проверка динамического поведения системы в реальных условиях функционирования. Выявление несоответствия функциональной спецификации и оценка ее полезности и устойчивости.

Описание. Функции системы или программы выполняются в заданном окружении с заданными данными тестирования, которые систематически формируются из спецификации в соответствии с установленными критериями. Это позволяет сравнить поведение системы с ее спецификацией. При проведении тестирования никакие сведения о внутренней структуре системы не используются. Основная цель состоит в том, чтобы определить, правильно ли выполняет функциональный модуль функции, требуемые спецификацией. Метод формирования эквивалентных классов служит примером критерия тестирования данных методом «черного ящика». Массив входных данных подразделяется на конкретные диапазоны входных значений (эквивалентные классы) на основе спецификации. После этого формируются тестовые примеры из:

- данных из допустимых диапазонов;
- данных из недопустимых диапазонов;
- данных предельных значений диапазонов;
- экстремальных значений и
- комбинаций из перечисленных выше классов.

Могут оказаться эффективными также другие критерии выбора тестовых примеров в различных режимах тестирования (модуля, интеграции и системы). Например, критерий «экстремальные эксплуатационные условия» используется при тестировании системы в процессе подтверждения соответствия.

Литература:

Software Testing and Quality Assurance. K. Naik, P. Tripathy, Wiley Interscience, 2008, Print ISBN: 9780471789116 Online ISBN: 9780470382844.

Essentials of Software Engineering. Frank F. Tsui, Orlando Karam. Jones & Bartlett, 2006. ISBN 076373537X, 9780763735371.

The Art of Software Testing, Second Edition. G. Myers et al., Wiley & Sons, New York, 2004, ISBN 0471469122, 9780471469124.

Systematic Software Testing. Rick D. Craig, Stefan P. Jaskiel. Artech House, 2002. ISBN 1580535089, 9781580535083.

**В.5.3 Статистическое тестирование**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы В.3, В.5 и В.6).

Цель. Проверка динамического поведения системы, связанной с безопасностью, и оценка ее полезности и устойчивости.

Описание. При этом подходе тестируется система или программа с входными данными, выбранными в соответствии с предполагаемым статистическим распределением реальных эксплуатационных входных данных — эксплуатационный профиль.

Литература:

A discussion of statistical testing on a safety-related application. S Kuball, J. H. R. May, Proc. IMechE Vol. 221 Part O: J. Risk and Reliability, Institution of Mechanical Engineers, 2007.

Practical Reliability Engineering. P. O'Connor, D. Newton, R. Bromley, John Wiley and Sons, 2002, ISBN 0470844639, 9780470844632.

Dependability of Critical Computer Systems 3. P. G. Bishop et al., Elsevier Applied Science, 1990, ISBN 1-85166-544-7.

Dependability of Critical Computer Systems 1. F. J. Redmill, Elsevier Applied Science, 1988, ISBN 1-85166-203-0.

**В.5.4 Полевые испытания**

Примечания

1 Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы В.3, В.5 и В.6).

2 См. также приложение С, С.2.10 — аналогичные средства, а в приложении D — статистический подход — то и другое в контексте программного обеспечения.

Цель. Использование результатов полевых испытаний из различных областей применения в качестве одного из средств исключения сбоев во время интеграции Э/Э/ПЭ системы и/или в процессе подтверждения соответствия Э/Э/ПЭ системы безопасности.

Описание. Использование компонентов или подсистем, которые при их использовании показали путем испытаний отсутствие или наличие только несущественных ошибок и существенно не изменялись в течение достаточно длительного периода времени во многих различных применениях. В частности, для сложных компонентов с множеством функций (например операционной системы, интегральных схем) разработчик должен обратить внимание на функции, которые были фактически протестированы методом полевых испытаний. Например, должны быть рассмотрены подпрограммы самотестирования для обнаружения сбоев: при отсутствии сбоев аппаратных средств в период эксплуатации о подпрограммах нельзя сказать, что они протестированы, поскольку они никогда не выполняли функций обнаружения своих сбоев.

При использовании полевых испытаний должны быть соблюдены следующие требования:
- неизменность спецификации;
- наличие 10 систем в различных применениях;
- длительность работы $10^5$ ч и по меньшей мере один год сервисной поддержки.

Примечание — В стандарте сектора применения могут быть определены другие значения этих параметров.

Полевые испытания документируются поставщиком и/или эксплуатирующей организацией; документация должна по меньшей мере содержать:
- точное обозначение системы и ее компонентов, включая управление версиями аппаратных средств;
- сведения о пользователях и времени применения;
- отработанное время в часах;
- процедуры выбора системы и прикладные программы, использованные при испытаниях;
- процедуры обнаружения и регистрации сбоев, а также процедуры устранения их последствий и причин возникновения.

Литература:

IEC 60300-3-2:2004, Dependability management — Part 3-2: Application guide — Collection of dependability data from the field.

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6.

**В.6 Подтверждение соответствия Э/Э/ПЭ системы безопасности**

Главная цель. Подтвердить, что Э/Э/ПЭ система, связанная с безопасностью, соответствует спецификации требований к Э/Э/ПЭ системе безопасности и спецификации требований проектирования Э/Э/ПЭ системы.

**В.6.1 Функциональные испытания в условиях окружающей среды**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблица В.5).

Цель. Оценка защищенности системы, связанная с безопасностью, от типовых воздействий окружающей среды.

Описание. Систему помещают в различные условия окружающей среды (например в соответствии со стандартами серии МЭК 60068 или серии МЭК 61000) и оценивают способности системы выполнять функции безопасности (на соответствие требованиям стандартов, указанных выше).

Литература:

IEC 60068-1:1988, Environmental testing — Part 1: General and guidance/Amendment 1(1992).

IEC 61000-4-1:2006, Electromagnetic compatibility (EMC) — Part 4-1: Testing and measurement techniques — Overview of IEC 61000-4 series.

Dependability of Critical Computer Systems 3. P. G. Bishop et al., Elsevier Applied Science, 1990, ISBN 1-85166-544-7.

**В.6.2 Испытания на устойчивость к пиковым выбросам внешних воздействий**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы В.5 и В.6).

Цель. Проверка способности систем, связанных с безопасностью, справляться с пиковыми выбросами внешних воздействий.

Описание. В систему загружается типичная прикладная программа и все периферийные линии (цифровые, аналоговые и последовательные интерфейсы, шины, источники питания и т. д.) подвергаются воздействию стандартных шумовых сигналов. Для того, чтобы получить их количественную оценку, целесообразно внимательно подходить к предельным значениям выбросов внешних влияний. Класс помех считается выбранным неверно, если функция системы не выполняется.

Литература:

IEC 61000-4-5:2005, Electromagnetic compatibility (EMC) — Part 4-5: Testing and measurement techniques — Surge immunity test.

C37.90.1—2002, IEEE Standard for Surge Withstand Capability (SWC) Tests for Relays and Relay Systems Associated with Electric Power Apparatus.

**В.6.3** (Не используется.)

**В.6.4 Статический анализ**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы В.5 и В.6) и в МЭК 61508-3 (таблицы А.9, В.8, С.9 и С.18).

Цель. Исключение систематических дефектов, которые могут приводить к отказам в тестируемой системе вначале либо после продолжительной эксплуатации.

Описание. Этот систематический и, возможно, автоматизированный метод позволяет исследовать конкретные статические характеристики опытных образцов системы для обеспечения полноты, согласованности, отсутствия неоднозначностей относительно сформулированных требований (например в руководящих материалах по конструированию, системных спецификациях и инструкциях о применении). Статический анализ должен быть

воспроизводим и применим к опытному образцу, который доведен до четко определенной завершающей стадии. Ниже приведены некоторые примеры статического анализа аппаратных и программных средств:

- анализ согласованности потока данных (например при тестировании, если данные об объекте интерпретируются как имеющие одно значение);
- анализ управления потоком (например определение маршрутов, кода недоступности);
- анализ интерфейсов (например исследование передачи переменных между различными программными модулями);
- анализ потока данных для обнаружения вызывающих сомнения последовательностей для переменных: создание — использование для обращения — удаление;
- тестирование строгого соблюдения конкретных руководящих материалов (например по вопросам: длина пути утечки тока и зазоры, расстояние между группами модулей, физическое расположение модулей, механически чувствительные физические модули, индивидуальное использование физических модулей при их внедрении).

Литература:

Static Analysis and Software Assurance. D. Wagner, Lecture Notes in Computer Science, Volume 2126/2001, Springer, 2001, ISBN 978-3-540-42314-0.

An Industrial Perspective on Static Analysis. B. A. Wichmann, A. A. Canning, D. L. Clutterbuck, L. A. Winsborrow, N. J. Ward and D. W. R. Marsh. Software Engineering Journal., 69—75, March 1995.

Dependability of Critical Computer Systems 3. P. G. Bishop et al.. Elsevier Applied Science, 1990, ISBN 1-85166-544-7.

**B.6.5 Динамический анализ и тестирование**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы B.5 и B.6) и в МЭК 61508-3 (таблицы A.5, A.9, B.2, C.5, C.9 и C.12).

Цель. Обнаружение ошибок в спецификации путем исследования динамического поведения опытных образцов на завершающих стадиях.

Описание. Динамический анализ систем, связанных с безопасностью, проводится при подаче на вход опытного образца системы, связанной с безопасностью, входных данных, которые типичны для заданного эксплуатационного окружения. Анализ будет удовлетворительным, если наблюдаемое поведение системы, связанной с безопасностью, соответствует требуемому поведению. Любой отказ системы, связанной с безопасностью, должен быть устранен, после чего новые варианты эксплуатации системы должны быть проанализированы.

Литература:

The Concept of Dynamic Analysis. T. Ball, ESEC/FSE '99, Lecture Notes in Computer Science, Springer, 1999, ISBN 978-3-540-66538-0.

Dependability of Critical Computer Systems 3. P. G. Bishop et al., Elsevier Applied Science, 1990, ISBN 1-85166-544-7.

**B.6.6 Анализ отказов**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы B.5 и B.6).

**B.6.6.1 Анализ видов и последствий отказов**

Цель. Проведение анализа проекта системы с систематическим исследованием всех возможных причин отказов компонентов системы и определением влияния этих отказов на поведение и безопасность системы.

Описание. Анализ обычно проводится экспертным методом. Каждый компонент системы анализируется по очереди с тем, чтобы выявить набор режимов отказов для компонента, их причины и результаты (на локальном уровне и на уровне всей системы), процедуры обнаружения и рекомендации. При выдаче рекомендаций они документально оформляются в виде корректирующих действий.

Литература:

IEC 60812:2006, Analysis techniques for system reliability — Procedure for failure mode and effects analysis (FMEA).

Risk Assessment and Risk Management for the Chemical Process Industry. H.R. Greenberg, J.J. Cramer, John Wiley and Sons, 1991, ISBN 0471288829, 9780471288824.

Reliability Technology. A. E. Green, A. J. Bourne, Wiley-Interscience, 1972, ISBN 0471324809.

**B.6.6.2 Причинно-следственные диаграммы**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблицы B.3, B.4, C.13 и C.14).

Цель. Моделирование с помощью причинно-следственных диаграмм, которые могут представить проект системы в виде последовательности комбинаций базовых событий.

Описание. Данное средство может рассматриваться как комбинация процедур анализа с помощью дерева отказов и дерева событий. Начиная с критического (начального) события, граф последствий просматривается в прямом направлении, используя логические элементы ДА/НЕТ, описывающие успех и неудачу некоторых операций. Это позволяет сформировать последовательность событий, ведущую или к аварии или к корректной ситуации. Затем для каждого отказа строятся графы причин (то есть деревья отказов). Прохождение в обратном направлении эквивалентно дереву отказов, где критическое событие представлено в виде события, описанного на верхнем уровне. Прохождение в прямом направлении позволяет определять возможные последствия, возникающие из события. В узле графа могут быть символы, описывающие условия распространения причин по различным ветвям от этого узла. Временные задержки также могут учитываться. Эти условия распространения причин также могут быть описаны с помощью деревьев отказов. Для того чтобы диаграмма выглядела более компактной, пути распро-

странения причин могут быть объединены с логическими символами. Должен быть определен набор стандартных символов для использования в причинно-следственных диаграммах. Такие диаграммы могут быть использованы для генерации деревьев отказов и для вычисления вероятности появления определенных критических последовательностей. Они также могут быть использованы для генерации деревьев событий.

Литература:

IEC 62502:2010, Analysis techniques for dependability — Event tree analysis (ETA).

The Cause Consequence Diagram Method as a Basis for Quantitative Accident Analysis. B.S. Nielsen, Danish Atomic Energy Commission, Riso-M-1374, 1971.

**B.6.6.3 Анализ дерева событий**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица В.4 и С.14).

Цель. Моделирование с помощью диаграмм последовательности событий, которая может возникнуть в системе после появления инициализирующего события и указать на возможные опасные последствия. Дерево события трудно создать с нуля, поэтому полезно использовать схему последовательности событий.

Описание. В верхней части диаграммы записывается последовательность условий, относящихся к формированию последовательности событий, следующих за инициализирующим событием. Начиная с инициализирующего события, являющегося целью анализа, проводится прямая линия к первому условию последовательности. Наличие ветвей «да» и «нет» диаграммы указывает на зависимость будущего события от условий. Каждая из двух ветвей продолжается до следующего условия. Однако не все условия выполняются на этих ветвях. Какая-то из них продолжится до окончания последовательности условий, но каждая ветвь дерева, построенная таким способом, представляет возможную последовательность. Если условия в последовательностях независимы, дерево событий может быть использовано для вычисления вероятностей различных последовательностей, основываясь на значениях вероятностей условий и их числе в последовательности. Поскольку условия редко бывают полностью независимыми, такие вычисления необходимо тщательно рассмотреть и это должны выполнять квалифицированные аналитики.

Литература:

IEC 62502:2010, Analysis techniques for dependability — Event tree analysis (ETA).

Risk Assessment and Risk Management for the Chemical Process Industry. H.R. Greenberg, J.J. Cramer, John Wiley and Sons, 1991, ISBN 0471288829, 9780471288824.

**B.6.6.4 Анализ видов, последствий и критичности отказов**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблицы А.10, В.4, С.10 и С.14).

Цель. Ранжирование критичности компонентов, которые могут вызвать нарушения, повреждения или ухудшение работы системы при одиночных ошибках в целях определить, каким компонентам может потребоваться особое внимание и какие средства управления необходимы в процессе проектирования или эксплуатации.

Описание. Этот метод сопоставим с FMEA, но отличается наличием одного или нескольких столбцов для описания критичности, которая может быть ранжирована многими методами. Наиболее сложный метод описан Обществом автомобильных инженеров (Society for Automotive Engineers — SAE) в ARP 926. В этом методе значение критичности для любого компонента определяется числом отказов конкретного вида, предполагаемым в процессе выполнения каждого миллиона операций, реализуемых в критическом режиме. Критичность является функцией девяти параметров, большинство из которых должны быть измерены. Очень простой метод определения критичности состоит в умножении вероятности отказа компонента на величину ущерба, который может быть при этом нанесен; этот метод аналогичен простой оценке показателя риска.

Литература:

IEC 60812:2006, Analysis techniques for system reliability — Procedure for failure mode and effects analysis (FMEA).

Software criticality analysis of COTS/SOUP. P. Bishop, T. Clement, S. Guerra. In Reliability Engineering & System Safety, Volume 81, Issue 3, September 2003, Elsevier Ltd., 2003.

Software FMEA techniques. P.L. Goddard. In Proc Annual 2000 Reliability and Maintainability Symposium, IEEE, 2000, ISBN: 0-7803-5848-1.

SAE-ARP-926—1997 Fault/failure analysis procedure.

**B.6.6.5 Анализ дерева отказов**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица В.4 и В.14).

Цель. Помощь в анализе событий или комбинации событий, которые вызывают угрозы или опасные последствия, и в выполнении вычисления вероятности главного события.

Описание. Начиная с главного события, которое может непосредственно вызвать угрозу или опасные последствия («событие вершины дерева»), выполняется анализ, чтобы идентифицировать причины этого события. Комбинации причины описываются логическими операторами («И», «ИЛИ» и т. д.). Затем анализируются промежуточные причины тем же способом и т. д., возвращаясь к базовым событиям, где анализ прекращается.

Данный метод является графическим, и для изображения дерева отказов используется набор стандартизованных символов. В результате анализа дерево отказа представляет собой логическую функцию, объединяющую базовые события (обычно отказы компонентов) с главным событием (полный отказ системы). Рассматриваемый

метод предназначен в основном для анализа аппаратных средств, но допускается также применять его к анализу ошибок программного обеспечения. Этот метод может использоваться для качественного анализа отказов (идентификация сценариев отказа: минимальные сечения или простые импликанты), полуколичественно (оценивая сценарии их вероятностями) и количественно для вычислений вероятности главного события (см. С.6).

Литература:

IEC 61025:2006, Fault tree analysis (FTA).

From safety analysis to software requirements. K. M. Hansen, A. P. Ravn, A. P. V Stavridou. IEEE Trans Software Engineering, Volume 24, Issue 7, Jul 1998.

**В.6.6.6 Модели Маркова**

Примечание — Краткое сравнение данного метода с методом, основанным на блок-схемах надежности, при анализе полноты безопасности аппаратных средств см. в МЭК 61508-6 (приложение В.1).

Цель. Моделирование поведения систем, используя граф состояний-переходов, и оценка общесистемных параметров (ненадежность, неготовность, безопасности MTTF, MUT, MDT и др.) системы.

Описание. Это — конечный автомат (см. В.2.3.2), представленный направленным графом. Узлы (круги) представляют состояния, а ребра (стрелки) между узлами представляют переходы (отказы, ремонты и т. д.), происходящие между состояниями. Ребра имеют весовые коэффициенты, соответствующие частотам отказов или частотам восстановлений. Фундаментальное свойство однородных процессов Маркова заключается в том, что будущее состояние зависит только от настоящего состояния, то есть переход из состояния $N$ к последующему состоянию $N+1$ не зависит от предыдущего состояния $N-1$. Это означает, что все вероятностные законы моделей экспоненциальны.

Следует заметить, что события, состояния и частоты отказов могут быть детализированы так, что может быть получено точное описание системы, например, обнаруженные или необнаруженные отказы, обнаружение наибольшего отказа и т. п. Интервалы контрольных проверок также могут быть смоделированы должным образом при помощи так называемых многофазных процессов Маркова, где вероятности состояний в конце одной фазы (например как раз перед контрольным испытанием) могут использоваться для вычисления начальных условий для следующей фазы (например вероятности различных состояний после того, как контрольная проверка была выполнена).

Метод Маркова подходит для моделирования многих систем, уровень избыточности которых изменяется со временем вследствие нахождения компонента в состоянии отказа или восстановления. Другие классические методы, например, FMEA и FTA, не могут быть адаптированы к моделированию влияний отказов в течение жизненного цикла системы, поскольку не существует простой комбинаторной формулы для вычисления соответствующих вероятностей.

В простейших случаях такую формулу, описывающую вероятности системы, можно найти в литературе или вывести самостоятельно. В более сложных случаях существуют методы упрощения (то есть сокращение числа состояний).

Тем не менее, однородный граф Маркова описывается системой линейных дифференциальных уравнений с постоянными коэффициентами. Был проведен серьезный анализ таких систем, в результате для их решения были разработаны мощные алгоритмы, которые доступны. Поэтому с увеличением размера модели очень эффективно использовать вышеупомянутые алгоритмы, которые реализованы в виде различных пакетов программного обеспечения.

Нужно отметить, что размер графа растет экспоненциально с числом компонентов, это — так называемый комбинаторный взрыв. Поэтому данный метод применим без апроксимаций только для небольших систем.

Если законы распределения неэкспоненциальные (полумарковские процессы), то необходимо использовать метод моделирования Монте-Карло (см. В.6.6.8).

Литература:

IEC 61165:1995 Application of Markov techniques.

The Theory of Stochastic Processes. R. E. Cox and H. D. Miller, Methuen and Co. Ltd., London, UK, 1963.

Finite MARKOV Chains. J. G. Kemeny and J. L. Snell. D. Van Nostrand Company Inc, Princeton, 1959.

The Theory and Practice of Reliable System Design. D. P. Siewiorek and R. S. Swarz, Digital Press, 1982.

Sécurisation des architectures informatiques. Jean-Louis Boulanger, Hermès — Lavoisier 2009, ISBN: 978-2-7462-1991-5.

**В.6.6.7 Структурные схемы надежности**

Примечания

1 Данный метод/средство используется в МЭК 61508-6 (приложение В).

2 См. также С.6.4 «Структурные схемы надежности».

Цель. Моделирование в форме диаграмм набора событий, которые должны происходить, и условий, которые должны быть удовлетворены, для успешного выполнения операций системы или задач. Данный метод в большей степени является методом представления, чем методом анализа.

Описание. Данный метод позволяет сформировать успешный маршрут, состоящий из блоков, линий и логических переходов. Такой успешный маршрут начинается от одной стороны диаграммы и проходит через блоки и логические переходы до другой стороны диаграммы. Блок представляет собой условие или событие, маршрут проходит через него, если условие истинно или событие произошло. Когда маршрут подходит к логическому переходу, то он продолжается, если критерий логического перехода выполняется. Если маршрут достигает какой-либо

вершины, то он может продолжаться по всем исходящим из нее путям. Если существует по меньшей мере один успешный маршрут через всю диаграмму, то цель анализа считается достигнутой.

Данный метод позволяет сформировать структурное представление моделируемой системы. Эта структура напоминает электрическую схему, в которой ток протекает от входа к выходу, что означает, что моделируемая система работает должным образом. Если в схеме есть разрыв, то это означает, что в моделируемой системе произошел отказ. В результате появляется концепция наборов минимальных сечений, которые представляют комбинации отказов (т.е. места, где структурная схема надежности имеет «разрыв»), приводящих к отказу моделируемой системы.

Математически данный метод подобен дереву отказов. Он представляет логическую функцию, связывающую состояния отдельных компонентов (отказавших или работающих) с состоянием всей системы (отказавшей или работающей). Поэтому вычисления подобны тем, которые описаны для дерева отказов.

Литература:

IEC 61078:2006 Analysis techniques for dependability — Reliability block diagram method.

Sécurisation des architectures informatiques. Jean-Louis Boulanger, Hermès — Lavoisier 2009, ISBN: 978-2-7462-1991-5.

**В.6.6.8 Моделирование методом Монте-Карло**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица В.4) и используется в МЭК 61508-6 (приложение В).

Цель. Моделирование ситуаций реального мира методом генерации случайных чисел, когда аналитические методы не применимы.

Описание. Моделирование методом Монте-Карло используется для решения двух классов проблем:

- вероятностного, в котором для генерации стохастических ситуаций используются случайные числа;
- детерминистического, который математически преобразуется в эквивалентную вероятностную форму.

Принцип моделирования Монте-Карло использует случайные числа для анимации модели поведения правильно и неправильно функционирующей исследуемой системы. Такие поведенческие модели реализуются моделями состояния-переходы (граф Маркова, сети Петри, формальные языки и т. д.). Моделирование Монте-Карло позволяет получить большую статистическую выборку, из которой формируются статистические результаты.

При использовании моделирования Монте-Карло необходимо заботиться о том, чтобы гарантировать, что смещения, допуски или шум были в приемлемых диапазонах. Этим необходимо управлять через доверительный интервал, который легко может быть получен из моделирований. Вопреки аналитическим методам моделирование Монте-Карло является самоаппраксимирующимся. Для упрощения модели незначительные события просто не появляются без необходимости их идентификации.

Общие принципы моделирования методом Монте-Карло заключаются в переформулировании проблемы так, чтобы полученные результаты были как можно более точными, что позволяет отказаться от решения проблемы в ее исходной постановке.

В контексте настоящего стандарта моделирование Монте-Карло может использоваться для вычислений УПБ и учитывать неопределенности данных о надежности. Используя современные компьютеры, можно легко выполнить моделирование системы с УПБ 4.

Литература:

Monte Carlo Methods. J. M. Hammersley, D. C Handscomb, Chapman & Hall, 1979.

Sécurisation des architectures informatiques. Jean-Louis Boulanger, Hermès — Lavoisier 2009, ISBN: 978-2-7462-1991-5.

**В.6.6.9 Модели дерева отказов**

Примечания

1 О применении данного метода для анализа полноты безопасности аппаратного обеспечения см. в МЭК 61508-6.

2 Применение дерева отказов в качестве средства подтверждения соответствия безопасности уже было описано в В.6.6.5. Данный метод также широко используется для анализа отказов и вероятностных расчетов.

Цель. При помощи систематического нисходящего графического (следствие — причина) подхода построить логическую функцию, связывающую базовые события (виды отказов) с главным событием (нежелательное событие).

Описание. Это одновременно и метод анализа, помогающий аналитику шаг за шагом разработать модель, и математическая модель для вероятностных расчетов. Данный метод позволяет выполнять:

- качественный анализ путем выявления и сортировки сценариев отказов (минимальные сечения или простейшие импликанты);
- полукачественный анализ путем ранжирования сценариев в соответствии с их вероятностями возникновения;
- количественный анализ путем расчета вероятности главного события.

Подобно блок-схемам надежности, дерево отказов представляет логическую (булеву) функцию, связывающую состояния индивидуальных компонентов (отказал или работает) с состоянием всей системы (отказала или работает). Если компоненты являются независимыми, то вероятностные расчеты для логической функции могут быть выполнены только с учетом вероятностных свойств базовых компонентов. Это не так просто, поскольку это статическая модель в основном работает только с постоянными вероятностями. Расчет вероятностей, зависимых от времени, должен быть проведен особенно внимательно. Например, PFDavg систем безопасности, включающих периодическое контрольное тестирование компонентов, не может быть рассчитан непосредственно, кроме того,

еще более сложно рассчитать PFH для систем безопасности, работающих в непрерывном режиме. Поэтому с помощью данного метода только инженеры по надежности с глубоким пониманием математики, лежащей в основе данного метода, должны проводить расчеты значений неготовность/PFD и ненадежность/PFH.

Для очень простых деревьев отказов расчеты могут быть проведены вручную, однако за последние 50 лет было разработано и реализовано довольно большое количество алгоритмов для решения сложных логических уравнений. Наиболее современным на текущий момент является метод двоичных диаграмм решений (Binary Decision Diagrams, BDD), который основан на технике компактного кодирования логических уравнений в памяти компьютера. В настоящее время это единственный метод, способный выполнять вероятностные расчеты без приближений для систем промышленных размеров. Он также достаточно эффективен для обработки неопределенностей при моделировании методом Монте-Карло.

Литература:

IEC 61025:2006, Fault tree analysis (FTA).

**B.6.6.10 Обобщенные стохастические модели сетей Петри**

Примечания

1 О применении данного метода для анализа полноты безопасности аппаратного обеспечения см. в МЭК 61508-6.

2 Метод сетей Петри уже был описан в B.2.3.3 как полуформальный метод. Данный метод также может быть эффективно использован для анализа полноты безопасности аппаратного обеспечения.

Цель. Графически построить модель поведения правильно и неправильно функционирующую системы настолько близко к реальной модели системы, насколько это возможно, в целях обеспечения эффективной поддержки моделирования методом Монте-Карло.

Описание. Применяется асинхронный конечный автомат, описанный в B.2.3.3, за исключением того, что хорошее свойство, отслеживаемое при полуформальном подтверждении соответствия, не существует, когда моделируется поведение неправильно функционирующей системы безопасности. Так называемые позиции (изображаются кружочками) представляют возможные состояния, а так называемые переходы (изображаются прямоугольниками) представляют события, которые могут произойти. Кроме маркирования позиций (см. B.2.3.3) могут быть использованы сообщения или предикаты для подтверждения соответствия (активизации) переходов, а продолжительность задержки между активизацией перехода и его «возбуждением» может быть детерминированной или стохастической величиной. Поэтому такие сети Петри называются «обобщенными стохастическими» сетями Петри.

Сети Петри являются гибкими поведенческими моделями, которые подтверждают свою высокую эффективность для поддержки моделирования методом Монте-Карло (см. B.6.6.8). Кроме точности самого метода Монте-Карло, которая, так или иначе, всегда известна, все ограничения других методов (зависимости, комбинаторный взрыв, незкспоненциальность законов распределения и т. д.) преодолеваются. Для современных компьютеров больше не является проблемой даже оценки для УПБ 4.

Литература:

IEC 62551:2012, Analysis techniques for dependability — Petri net modeling.

Sécurisation des architectures informatiques. Jean-Louis Boulanger, Hermès — Lavoisier, 2009, ISBN: 978-2-7462-1991-5.

**B.6.7 Анализ наихудшего случая**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы B.5 и B.6).

Цель. Исключение систематических ошибок, возникающих в результате неблагоприятных сочетаний условий окружающей среды и допусков на параметры компонентов системы.

Описание. Эксплуатационные возможности системы и размеры компонентов исследуются или вычисляются теоретически. При этом для условий окружающей среды задаются их допустимые предельные значения. Анализируются и сопоставляются со спецификацией наиболее существенные характеристики системы.

**B.6.8 Расширенное функциональное тестирование**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы B.5 и B.6).

Цель. Обнаружение отказов на стадиях спецификации, проектирования и разработки системы. Проверка поведения системы, связанной с безопасностью, в случае редких или неспецифицированных операций ввода информации.

Описание. Расширенное функциональное тестирование проверяет функциональное поведение системы, связанной с безопасностью, как реакцию на входные условия, которые ожидаются только в редких случаях (например глобального отказа) или не охватываются спецификацией системы, связанной с безопасностью, (например некорректные операции). Для редко встречающихся условий наблюдаемое поведение системы, связанной с безопасностью, сравнивается со спецификацией. В тех случаях, когда реакция системы, связанной с безопасностью, не специфицирована, следует убедиться в том, что заданная безопасность сохранена в наблюдаемой реакции системы.

Литература:

Functional Program Testing and Analysis. W. E. Howden, McGraw-Hill, 1987.

The Art of Software Testing. G. J. Myers, Wiley & Sons, New York, 1979.

Dependability of Critical Computer Systems 3. P. G. Bishop et al., Elsevier Applied Science, 1990, ISBN 1-85166-544-7.

**В.6.9 Испытания в наихудших случаях**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы В.5 и В.6).

Цель. Тестирование ситуаций, специфицированных во время анализа наихудших случаев.

Описание. Эксплуатационные возможности системы и размеры компонентов тестируются для наихудших случаев. При этом для условий окружающей среды задают их допустимые предельные значения. Анализируются и сопоставляются со спецификацией наиболее существенные характеристики системы.

**В.6.10 Испытания с введением неисправностей**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблицы В.5 и В.6).

Цель. Внесение или имитация неисправностей в аппаратные средства системы и документирование реакции системы.

Описание. Представленный метод оценки зависимостей является качественным. Для описания местоположения и типа неисправностей, а также способа их внесения предпочтительно используются детализированные функциональные блоки, схемы и схемные диаграммы: например, питание может не поступать на различные модули; линии питания, линии общей шины или адресные линии могут быть разомкнуты/коротко замкнуты; компоненты или их порты могут быть разомкнуты или закорочены; реле могут быть замкнуты или разомкнуты, либо их действия могут выполняться в несоответствующие моменты времени и т. д. Возникающие в результате отказы системы классифицируются, например, таблицы 1 и 2 в [8]. Обычно вводятся одиночные неисправности в устойчивом состоянии системы. Однако в случае, если неисправность не обнаруживается тестом встроенной диагностики или оказывается не очевидной, она может сохраниться в системе и вызвать следующую неисправность. При этом количество неисправностей может быстро возрасти многократно.

Такие испытания проводятся многопрофильным коллективом специалистов. Поставщик системы должен при этом присутствовать и получать рекомендации. Для отказов, приводящих к опасным последствиям, вычисляют и оценивают среднее время наработки на отказ. Если это время мало, необходима модификация системы.

Литература:

IEC 60812:2006, Analysis techniques for system reliability — Procedure for failure mode and effects analysis (FMEA).

IEC 61069-5:1994, Industrial-process measurement and control — Evaluation of system properties for the purpose of system assessment — Part 5: Assessment of system dependability.

**Приложение С**
**(справочное)**

## Анализ методов и средств достижения полноты безопасности программного обеспечения (см. МЭК 61508-3)

### С.1 Общие положения

Анализ методов, содержащийся в настоящем приложении, не следует рассматривать как полный или исчерпывающий.

### С.2 Требования и детальное проектирование

Примечание — Соответствующие методы и средства приведены в В.2.

#### С.2.1 Структурные методы

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблицы А.2 и А.4).

##### С.2.1.1 Общие положения

Цель. Основная цель методов анализа структуры (структурных методов) состоит в обеспечении качества разработки программного обеспечения. Данные методы в основном используются на ранних стадиях жизненного цикла создаваемой системы. Структурные методы используют как точные, так и интуитивные процедуры и нотации (поддерживаемые компьютерами), а также определяют и позволяют документально оформлять требования и возможности реализации в логической последовательности и структурированным способом.

Описание. Существует достаточно много структурных методов. Некоторые из них созданы для выполнения традиционных функций обработки данных и транзакций, другие в большей степени ориентированы на процессы управления и задачи реального времени (для систем, реализующих такие задачи, характеристика безопасности является более критичной, чем для других систем). UML (см. С.3.12) содержит много примеров структурированных нотаций.

Структурные методы можно считать «интеллектуальными инструментами», предназначенными для обобщенного восприятия и структуризации конкретной проблемы или системы. К их основным свойствам относятся:

- использование логики в рассуждениях и выводах, декомпозиция сложной проблемы на управляемые стадии;
- анализ и документирование всей системы, включая окружающую среду, а также разрабатываемую систему;
- декомпозиция данных и функций в разрабатываемой системе;
- использование контрольных таблиц, то есть списков типов объектов, нуждающихся в анализе;
- малая интеллектуальная перегрузка — простота, интуитивность и практичность при представлении проблемы или системы;
- акцентирование на разработке структурной модели создаваемой системы с поддержкой CASE средств для полноты метода.

Нотации, используемые для анализа и документирования проблем и объектов системы (например на основе процессов и потоков данных), ориентированы на строгость, однако нотации для выражения функций обработки, выполняемых этими объектами, являются более неформальными. В то же время некоторые методы частично используют формальные нотации (например регулярные выражения или конечные состояния автоматов). Увеличение точности нотации не только повышает уровень понимания, но и обеспечивает возможность автоматизированной обработки.

Другим преимуществом структурных нотаций является их наглядность, которая позволяет пользователю интуитивно проверять возможности спецификации или проекта при неполной информации.

Данный краткий обзор описывает несколько структурных методов более подробно.

Литература:

Software Engineering for Real-time Systems. J.E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205.

Software Design. D. Budgen, Pearson Education, 2003, ISBN 0201722194, 9780201722192.

##### С.2.1.2 Управляемое представление требований (CORE)

Цель. Обеспечение требований, определений и формулировок.

Описание. Данный метод должен устранить пробел между потребителем/конечным пользователем и аналитиком. Он не основан на математически строгой теории, а является средством коммуникации. Метод CORE создан для представления требований, а не для спецификаций. Данный метод является структурированным, все его представления проходят через различные уровни уточнений. Метод CORE используется для широкого круга проблем, учитывает сведения об окружающей среде, в которой система функционирует, а также различные точки зрения разных типов пользователей. Метод CORE содержит руководящие материалы и тактические подходы для того, чтобы упростить сложный проект. Такое упрощение может быть скорректировано либо явным образом идентифицировано и документально оформлено. Таким образом, спецификации могут быть неполными, однако выявленные нерешенные проблемы и области высокого риска должны быть рассмотрены при последующем проектировании.

Литература:

Software Engineering for Real-time Systems. J.E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205.

Requirements Engineering. E. Hull, K. Jackson, J. Dick. Springer, 2005, ISBN 1852338792, 9781852338794.

**С.2.1.3 Метод разработки системы по Джексону (JSD)**

Цель. Разработка метода, охватывающего создание программных систем от стадии формирования требований до стадии кодирования, специально для систем реального времени.

Описание. Метод JSD представляет собой поэтапную процедуру разработки, в которой разработчик моделирует поведение реального мира, которое представляется функциями системы, определяет эти функции, вводит их в модель и преобразует образовавшуюся в результате спецификацию, которая реализуема в планируемой среде. Поэтому данный метод охватывает традиционные этапы, такие как создание спецификаций, проектирование и разработка, но несколько отличается от традиционных методов и не является методом нисходящего проектирования.

Данный метод уделяет большое внимание выявлению на ранней стадии сущностей реального мира, относящихся к создаваемой системе, а также моделированию этих сущностей и того, что может с ними произойти. Как только анализ «реального мира» будет выполнен и создана его модель, анализируются функции системы с тем, чтобы определить, как они вписываются в модель «реального мира». Модель результирующей системы дополняется структурным описанием всех процессов модели и затем преобразуется в программы, которые могут работать в заданной программно-аппаратной среде.

Литература:

Systems Analysis and Design. D. Yeates, A. Wakefield. Pearson Education, 2003, ISBN 0273655361, 9780273655367.

An Overview JSD. J.R. Cameron. IEEE Transactions on Software Engineering, SE-12, No. 2, February 1986.

**С.2.1.4 Метод Йордона (Yourdon) для систем реального времени**

Цель. Спецификация и проектирование систем реального времени.

Описание. Данный метод реализует процесс разработки системы, состоящий из трех этапов. На первом этапе происходит создание «сущностной модели», которая описывает поведение системы в целом. На втором этапе строится модель реализации, описывающая структуру и механизмы, которые, будучи реализованными, отражают требуемое поведение системы. На третьем этапе происходит фактическое построение аппаратных и программных средств системы. Три этапа строго соответствуют традиционным спецификации, проектированию и разработке, но главное, что разработчик на каждом этапе должен активно заниматься моделированием.

Сущностная модель состоит из двух частей:

- модели окружающей среды, содержащей описание границ между системой и ее окружением, а также внешних событий, на которые должна реагировать система;
- модели поведения, которая содержит схемы, описывающие преобразования, выполняемые системой в ответ на события, и описание данных, которые система должна содержать для выдачи ответов.

Модель реализации подразделяется на две подмодели, описывающие распределение отдельных процессов в процессорах и декомпозицию процессов на программные модули.

Для создания сущностной модели и модели реализации данный метод использует множество хорошо известных подходов: построение диаграмм потоков данных, преобразование графов, структурированный английский язык, диаграммы переходов состояний и сети Петри. Кроме того, данный метод содержит методики для моделирования представленного из уже сформированных моделей проекта системы или вручную (на бумаге) или автоматически.

Литература:

Real-time Systems Development. R. Williams. Butterworth-Heinemann, 2006, ISBN 0750664711, 9780750664714.

Structured Development for Real-Time Systems (3 Volumes). P.T. Ward and S.J. Mellor. Yourdon Press, 1985.

**С.2.2 Диаграммы потоков данных**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблицы В.5 и В.7).

Цель. Программная поддержка описания потока данных в виде диаграмм.

Описание. Диаграммы потоков данных описывают преобразование входных данных в выходные для каждого компонента схемы, представляющего различные преобразования.

Диаграммы потоков данных состоят из трех компонентов:

- аннотированные стрелки — обозначают поток данных, входящих и исходящих из блоков преобразования, с кратким описанием этих данных;
- аннотированные кружки — обозначают блоки преобразования с кратким описанием преобразований;
- операторы (and, xor) — эти операторы используются для связи аннотированных стрелок.

Каждый аннотированный кружок на диаграмме потока данных может рассматриваться как самостоятельный блок, который при появлении на его входах данных преобразует их в выходные. Одним из основных преимуществ является то, что они показывают преобразования, не предполагая, как они реализуются. Чистая диаграмма потоков данных не включает в себя управляющую информацию или информацию о последовательности процесса, так как управление реализуется в расширениях для реального времени, как в методе Йордона для систем реального времени (см. С.2.1.4).

Создание диаграмм потока данных является наилучшим подходом при анализе систем в направлении от входов к выходам. Каждый кружок на диаграмме должен обозначать разное преобразование — его выходы должны отличаться от его входов. Не существует правил определения общей структуры диаграммы, и создание диаграммы

потока данных является одним из творческих аспектов создания проекта системы в целом. Подобно всем проектам, процедура, уточняющая начальную диаграмму для создания конечной, является итеративной.

Литература:

Software Engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799.

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574.

ISO 5807:1985, Information processing — Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts.

ISO/IEC 8631:1989, Information technology — Program constructs and conventions for their representation.

**С.2.3 Структурные диаграммы**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица В.5).

Цель. Представление структуры программы в виде схемы.

Описание. Структурные диаграммы дополняют диаграммы потоков данных. Они описывают программируемую систему и иерархию ее компонентов, а также отображают их графически в виде дерева. Структурные диаграммы описывают способ реализации элементов диаграммы потоков данных в виде иерархии программных модулей.

Структурная диаграмма показывает взаимоотношения между программными модулями, не указывая при этом порядок активизации программных модулей. Структурные диаграммы изображаются с использованием следующих четырех символов:

- прямоугольника с именем модуля;
- линии, соединяющей эти прямоугольники, формирующие структуру;
- стрелки, отмеченной незаштрихованным кругом, с именем данных, передаваемых в направлении элементов структурной диаграммы и обратно (обычно такая стрелка изображается параллельно линиям, соединяющим прямоугольники схемы);
- стрелки, отмеченной заштрихованным кругом, с именем сигнала управления, проходящего в структурной диаграмме от одного модуля к другому, и эта стрелка также изображается параллельно линии, соединяющей два модуля.

Из любой нетривиальной диаграммы потока данных можно создать множество различных структурных диаграмм.

Диаграммы потоков данных отображают взаимоотношение между информацией и функциями системы. Структурные диаграммы отображают способ реализации элементов системы. Оба метода представляют собой обоснованные, хотя и различные точки зрения на конкретную систему.

Литература:

Software Design & Development. G. Lancaster. Pascal Press, 2001, ISBN 1741251753, 9781741251753.

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799.

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574.

**С.2.4 Формальные методы**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблицы А.1, А.2, А.4 и В.5).

**С.2.4.1 Общие положения**

Цель. Разработка программных средств, основанных на математических принципах. К этим средствам относятся методы формального проектирования и формального кодирования.

Описание. На основе формальных методов разработаны средства описания системы для решения отдельных задач на этапах спецификации, проектирования или реализации. Создаваемое в результате описание представляет собой строгую нотацию, математически анализируемую для обнаружения различных видов несогласованностей или некорректностей. Более того, такое описание может быть в некоторых случаях проанализировано автоматически по аналогии с проверкой компилятором синтаксиса исходной программы или использована анимация в целях показать различные аспекты поведения описываемой системы. Анимация может дать дополнительную уверенность в том, что система соответствует как реальным, так и формально специфицированным требованиям, поскольку это улучшает восприятие человеком специфицированного поведения системы.

Формальный метод обычно предлагает нотацию (как правило, используется один из методов дискретной математики), метод вывода описания в данной нотации и различные методы анализа описания для проверки корректности различных свойств системы.

Ряд формальных методов CCS, CSP, HOL, LOTOS, OBJ, временная логика, VDM и Z описан в подпунктах настоящего пункта. Следует заметить, что другие методы, например, метод конечных автоматов и сети Петри (см. приложение В), в зависимости от корректности использования методами соответствующего строгого математического аппарата, могут рассматриваться как формальные.

Литература:

Formal Specification: Techniques and Applications. N. Nissanke, Springer-Verlag Telos, 1999, ISBN-10: 1852330023.

The Practice of Formal Methods in Safety-Critical Systems. S. Liu, V. Stavridou, B. Dutertre, J. Systems. Software 28, 77—87. Elsevier, 1995.

Formal Methods: Use and Relevance for Development of Safety-Critical Systems. L.M. Barroca, J.A. McDermid, The Computer Journal 35 (6), 579—599, 1992.

How to Produce Correct Software — An Introduction to Formal Specification and Program Development by Transformations. E.A. Boiten et al., The Computer Journal 35 (6), 547—554, 1992.

**C.2.4.2 CCS — расчет взаимодействующих систем**

Цель. Описание и анализ поведения систем, реализующих параллельные коммуникационные процессы.

Описание. CCS — это математический аппарат, описывающий поведение систем. Проект системы моделируется в виде сети независимых процессов, реализующихся последовательно или параллельно. Процессы могут взаимодействовать через порты (аналогичные каналам CSP), и взаимодействие осуществляется только при готовности обоих процессов. Может быть смоделировано отсутствие детерминизма. Начиная с описания всей системы на высоком уровне абстрагирования (трассирование), можно выполнять пошаговое уточнение системы (стратегия сверху вниз) в рамках композиции взаимодействующих процессов, общее поведение которых формирует также поведение всей системы. В равной степени можно выполнять и стратегию снизу вверх, комбинируя процессы и получая в результате необходимые свойства формируемой системы, используя правила вывода композиционного типа.

Литература:

Communication and Concurrency. R. Milner, Prentice-Hall, 1989, ISBN 9780131150072.

**C.2.4.3 CSP — взаимодействующие последовательные процессы**

Цель. Спецификация конкурирующих программных систем, то есть систем, процессы которых реализуются одновременно.

Описание. Метод CSP обеспечивает язык для спецификаций процессов системы и подтверждения соответствия реализации процессов их спецификациям (описанным как трасса, то есть допустимая последовательность событий).

Система моделируется в виде сети независимых процессов, составленных последовательно или параллельно. Каждый независимый процесс описывается в терминах всех его возможных поведений. Независимые процессы могут взаимодействовать (синхронно или обмениваться данными) через каналы, и взаимодействие происходит только при готовности обоих процессов. Может быть промоделирована относительная синхронизация событий.

Теоретические положения метода CSP были непосредственно включены в архитектуру транспьютера INMOS, а язык OCCAM позволил непосредственно реализовывать на сетях транспьютеров системы, специфицированные в языке CSP.

Литература:

Communicating Sequential Processes: The First 25 Years. A. Abdallah, C. Jones, J. Sanders (Eds.). Springer, 2004, ISBN 3540258132, 9783540258131.

**C.2.4.4 HOL — логика высшего порядка**

Цель. Спецификация и верификация аппаратных средств.

Описание. HOL представляет собой разработанную в компьютерной лаборатории Кембриджского университета конкретную логическую нотацию и систему, которая ее автоматически поддерживает. Логическая нотация взята в основном из простой теории типов Черча, а машинная реализация основана на теории LCF (логике вычислимых функции).

Литература:

Higher-Order Computational Logic. J. Lloyd. In Computational Logic: Logic Programming and Beyond, Lecture Notes in Computer Science, Springer Berlin/Heidelberg, 2002, ISBN 978-3-540-43959-2.

**C.2.4.5 LOTOS**

Цель. Описание и анализ поведения систем, реализующих параллельные коммуникационные процессы.

Описание. LOTOS (язык для спецификации процессов, упорядоченных во времени) основан на CCS с дополнительными возможностями из близких алгебраических теорий CSP и CIRCAL (теория цепей). LOTOS преодолевает недостатки CCS в управлении структурами данных и представлении значений выражений, объединяя его с аспектами языка абстрактных типов данных ACT ONE. Процесс описания аспектов в LOTOS может быть однако использован для других формальных методов при описании абстрактных типов данных.

Литература:

Model Checking for Software Architectures. R. Mateescu. In Software Architecture, Lecture Notes in Computer Science, Springer Berlin/Heidelberg, 2004, ISBN 978-3-540-22000-8.

ISO 8807:1989, Information processing systems — Open Systems Interconnection — LOTOS — A formal description technique based on the temporal ordering of observational behavior.

**C.2.4.6 OBJ**

Цель. Обеспечение точной спецификации системы в процессе диалога с пользователем и подтверждение соответствия системы до ее реализации.

Описание. OBJ представляет собой алгебраический язык спецификаций. Пользователи определяют требования в терминах алгебраических выражений. Системные аспекты (поведение или конструктивы) специфицируются в терминах операций, действующих над абстрактными типами данных (ADT). ADT подобен языку ADA, где поведение оператора наблюдаемо, однако подробности реализации скрыты.

Спецификация OBJ и последующая пошаговая реализация подвергаются тем же формальным методам проверки, что и другие формальные методы. Более того, поскольку конструктивные аспекты спецификации OBJ автоматически исполнимы, существует непосредственная возможность подтверждения соответствия системы на основе самой спецификации. Исполнение — это по существу оценка функций системы путем подстановки выражений (перезаписыванием), которая продолжается до тех пор, пока не будут получены конкретные выходные значения. Эта исполнимость позволяет конечным пользователям рассматриваемой системы получать «облик» планируемой

системы на этапе ее спецификации без необходимости знакомства с методами, лежащими в основе формальных спецификаций.

Как и все другие методы ADT, метод OBJ применим только к последовательным системам или к последовательным аспектам параллельных систем. Метод OBJ применяют для спецификации как малых, так и крупных промышленных применений.

Литература:

Software Engineering with OBJ: Algebraic Specification in Action. J. Goguen, G. Malcolm. Springer, 2000, ISBN 0792377575, 9780792377573.

**С.2.4.7 Временна́я логика**

Цель. Непосредственное выражение требований к безопасности и эксплуатации, а также формальное представление сохранения этих качеств на последующих этапах разработки.

Описание. Стандартная предикатная логика первого порядка не содержит концепций времени. Временная логика расширяет логику первого порядка добавлением модальных операторов (например «с этого момента» и «случайно»). Эти операторы могут использоваться для уточнения суждений о системе. Например, свойства безопасности могут потребовать использовать модальный оператор «с этого момента», но может потребоваться, чтобы и другие необходимые состояния системы были достигнуты «случайно» из некоторого другого начального состояния. Временные формулы интерпретируются последовательностями состояний (поведениями). Представление состояния зависит от выбранного уровня описания. Оно может относиться ко всей системе, системным элементам или компьютерной программе.

Квантифицированные временны́е интервалы и ограничения во временной логике явно не обрабатываются. Абсолютное время обрабатывается путем образования дополнительных временных состояний, что является частью описания состояния.

Литература:

Mathematical Logic for Computer Science. M. Ben-Ari. Springer, 2001, ISBN 1852333197, 9781852333195.

**С.2.4.8 VDM, VDM++ — метод разработки Vienna**

Цель. Систематическая спецификация и реализация последовательных (VDM) и параллельных (VDM++) программ реального времени.

Описание. VDM — это математический метод спецификации и уточнения реализаций, который позволяет доказать их корректность относительно спецификации.

В этом основанном на модели методе спецификации состояние системы моделируется в терминах теоретико-множественных структур, в которых описаны инварианты (предикаты), а операции над этим состоянием моделируются путем определения их пред- и постусловий в терминах системных состояний. Операции могут проверяться на сохранение системных инвариантов.

Выполнение спецификаций осуществляется путем реализации состояния системы в терминах структур данных в заданном языке и уточнения операций в терминах программы на заданном языке. Этапы реализации и уточнения позволяют логически вывести свойства, устанавливающие корректность этих этапов. Выполняются или нет эти свойства, определяет разработчик.

В принципе VDM используется на этапе создания спецификации, но может также использоваться на этапах проектирования и реализации исходного кода. VDM может быть также применен к последовательно структурированным программам или к последовательным процессам в параллельных системах.

Объектно ориентированное и параллельное для реального времени расширения VDM, VDM++ представляют собой язык формализованных спецификаций, основанный на языке VDM-SL, созданном в ИСО, и на объектно-ориентированном языке Smalltalk.

VDM++ имеет широкий диапазон конструкций, что позволяет пользователю формально специфицировать параллельные системы реального времени в объектно-ориентированной среде. В VDM++ полная формальная спецификация содержит совокупность спецификаций классов и отдельных характеристик рабочего пространства.

К средствам описания реального времени на языке VDM++ относятся:

- временны́е выражения, предусмотренные для представления как текущего момента, так и момента вызова метода внутри тела метода;
- выражение, описывающее синхронизирующий сигнал, которое может быть добавлено к методу для спецификации верхних (или нижних) пределов времени исполнения для корректности реализаций;
- переменные непрерывного времени, которые должны быть введены. С условными операторами и операторами действия допускается специфицировать отношения (например дифференциальные уравнения) между этими временны́ми функциями, что оказалось очень полезно при спецификации требований к системам, действующим в среде с непрерывным временем. Уточняющие шаги приводят к дискретным программным решениям для программ реального времени.

Литература:

ISO/IEC 13817-1:1996, Information technology — Programming languages, their environments and system software interfaces — Vienna Development Method — Specification Language — Part 1: Base language.

Systematic Software Development using VDM. C. B. Jones. Prentice-Hall. 2nd Edition, 1990.

Conformity Clause for VDM-SL. G.I. Parkin and B.A. Wichmann, Lecture Notes in Computer Science 670, FME'93 Industrial-Strength Formal Methods, First International Symposium of Formal Methods in Europe. Editors: J. C P. Woodcock and P. G. Larsen. Springer Verlag, 501—520.

**C.2.4.9 Z**

Цель. Z — это нотация языка спецификаций для последовательных систем и метод проектирования, позволяющий разработчику выполнять работу, начиная со спецификации на языке Z до исполнительных алгоритмов, обеспечивая при этом доказательство их корректности по отношению к спецификации.

Язык Z в принципе используется на этапе спецификации, однако данный язык был разработан для использования от этапа составления спецификации до проектирования и реализации систем. Более всего он подходит для разработки последовательных систем, ориентированных на данные.

Описание. Как и в VDM, в реализованном в языке Z методе спецификации состояние системы моделируется в терминах теоретико-множественных структур, в которых описаны инварианты (используя предикаты), а операции над этими состояниями моделируются путем определения их пред- и постусловий в терминах системных состояний. Операции допускается проверять на сохранение системных инвариантов для демонстрации их согласованности. Формальная часть спецификации подразделяется на схемы, которые обеспечивают возможность структурирования спецификаций путем их усовершенствования.

Обычно спецификация Z представляет собой сочетание формального текста на языке Z и неформального пояснительного текста на естественном языке. Формальный текст сам по себе может оказаться слишком сжатым для простого восприятия и часто его смысл необходимо пояснять, тогда как неформальный, естественный язык может оказаться неоднозначным и неточным.

В отличие от VDM язык Z представляет собой скорее нотацию, чем завершенный метод. Однако был разработан близкий метод (метод B), который может быть использован в сочетании с языком Z. Метод B основан на принципе пошагового уточнения.

Литература:

Formal Specification using Z, 2nd Edition. D. Lightfoot. Palgrave Macmillan, 2000, ISBN 9780333763278.

The B-Method. S. Schneider. Palgrave Macmillan, 2001, ISBN 9780333792841.

**C.2.5 Программирование с защитой**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица А.4).

Цель. Создание программ, выявляющих во время их исполнения аномальные потоки управления, данных или значения данных и реагирующих на них заранее определенным и приемлемым способом.

Описание. В процессе разработки программ допускается использовать разные методы для проверки аномалий в потоках управления или данных. Эти методы могут применяться систематически в процессе программирования системы для снижения вероятности ошибочной обработки данных.

Существуют два пересекающихся множества методов защиты. Внутренние методы защиты от ошибок проектируются в программном обеспечении для преодоления недостатков в процессе создания этих программных средств. Эти недостатки могут быть обусловлены ошибками при проектировании или кодировании либо ошибочными требованиями. Ниже перечислены некоторые из рекомендаций по защите:

- проверка диапазона значений переменных;
- проверка значений переменных на их достоверность (если возможно);
- проверка типа, размерности и диапазона значений параметров процедур на входе процедур.

Представленные три рекомендации помогают гарантировать допустимость значений, обрабатываемых в программах, как с точки зрения терминов программных функций, так и физических значений переменных.

Параметры «только для чтения» и параметры «для чтения-записи» должны быть разделены, и доступ к ним должен проверяться. Программные функции должны рассматривать все параметры в качестве параметров «только для чтения». Символьные константы не должны быть доступны для записи. Это помогает обнаруживать случайные перезаписи или ошибочное использование переменных.

Устойчивое к ошибкам программное обеспечение проектируется в «предположении», что ошибки существуют в его собственном окружении либо используются выходящие за номиналы значения или предполагаемые условия, но программное обеспечение ведет себя заранее определенным способом. В этом случае применяют следующие проверки:

- проверку на достоверность физических значений входных и промежуточных переменных;
- проверку влияния выходных переменных, предпочтительно путем прямого наблюдения соответствующих изменений состояния системы;
- проверку самим программным обеспечением своей конфигурации, включая наличие и доступность предполагаемых аппаратных средств, а также завершенность самого программного обеспечения, что особенно важно для поддержки полноты в процессе его эксплуатации.

Некоторые из методов защиты программ, например, проверки последовательности потока управления, также справляются и с внешними отказами.

Литература:

Software Engineering for Real-time Systems. J.E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205.

Dependability of Critical Computer Systems: Guidelines Produced by the European Workshop on Industrial Computer Systems, Technical Committee 7 (EWICS TC7, Systems Reliability, Safety, and Security). Elsevier Applied Science, 1989, ISBN 1851663819, 9781851663811.

**С.2.6 Стандарты по проектированию и кодированию**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица А.4).

**С.2.6.1 Общие положения**

Цель. Упрощение верификации, с тем чтобы поддержать групповой объективный подход и установить стандартный метод проектирования.

Описание. В самом начале между участниками проекта должны быть согласованы необходимые правила, охватывающие рассмотренные ниже методы проектирования и разработки (например JSP, сети Петри и т. д.), а также соответствующие стандарты кодирования (см. С.2.6.2).

Данные правила создаются для облегчения разработки, верификации, оценки и эксплуатации. При этом должны учитываться доступные инструментальные средства, в частности, для аналитиков, а также развитие средств проектирования.

Литература:

IEC 60880:2006, Nuclear power plants — Instrumentation and control systems important to safety — Software aspects for computer-based systems performing category A functions.

Verein Deutscher Ingenieure. Software-Zuverlassigkeit — Grundlagen, Konstruktive Mass-nahmen, Nachweisverfahren. VDI-Verlag, 1993, ISBN 3-18-401185-2.

**С.2.6.2 Стандарты кодирования**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица В.1).

Цель. Сократить вероятность ошибок в разрабатываемом коде, связанном с безопасностью, и упростить его верификацию.

Описание. Следующие принципы указывают, как связанные с безопасностью правила кодирования (для любого языка программирования) могут помочь в выполнении нормативных требований МЭК 61508-3 и в достижении информативных «требуемых свойств» (см. приложение F). Необходимо уделить внимание доступным инструментам поддержки.

| Требования и рекомендации МЭК 61508-3 | Указания стандартов кодирования |
|---|---|
| Модульный подход (таблица А.2-7, таблица А.4-4) | Ограничение размера программного модуля (таблица В.9-1) и управление сложностью программного обеспечения (таблица В.9-2). Примеры:<br>- определение «локальных»: размера, метрик сложности и предельных размеров (для модулей);<br>- определение «глобальных» метрик сложности и предельных размеров (для всей структуры модулей);<br>- ограниченное число параметров/фиксированное число параметров подпрограммы (таблица В.9-4).<br>Ограничение доступа/инкапсуляция информации (таблица В.9-3), например, стимулы для того, чтобы использовать определенные функции языка.<br>Полностью определенный интерфейс (таблица В.9-6). Примеры:<br>- точная спецификация сигнатуры функции;<br>- программирование с проверкой ошибок (таблица А.2-3а) и верификация данных (7.9.2.7) с явной спецификацией предварительных условий и постусловий для функций, утверждений, инвариантов типов данных |
| Понятность кода<br>- содействие понятности кода (7.4.4.13);<br>- читаемый, понятный и тестируемый код (7.4.6) | Соглашения о присвоении имен, продвигающие значащие, однозначные имена, например, предотвращение имен, которые можно перепутать (например IO и I0).<br>Символьные имена для числовых значений.<br>Процедуры и руководства для документирования исходного кода (7.4.4.13). Например, в них:<br>- объяснить, почему и зачем (а не только что) кодируется;<br>- описать предостережения;<br>- описать побочные эффекты.<br>Где это возможно, в исходном коде должна содержаться следующая информация (7.4.4.13):<br>- юридическое лицо (например компания, автор(ы) и т. д.);<br>- описание кода;<br>- входные и выходные данные;<br>- история управления конфигурацией.<br>(См. также модульный подход.) |

*Продолжение таблицы*

| Требования и рекомендации МЭК 61508-3 | Указания стандартов кодирования |
|---|---|
| Верифицируемость и тестируемость:<br>- способствовать верификации и тестированию (7.4.4.13);<br>- способствовать обнаружению ошибок при проектировании или программировании (7.4.4.10);<br>- формальная верификация (таблица А.5-9);<br>- формальное доказательство (таблица А.9-1) | - окружения для «критических» библиотечных функций для проверки пред/постусловий;<br>- стимулы для того, чтобы использовать функции языка, которые могут выразить ограничения на использование определенных элементов данных или функций (например констант);<br>- для инструментов, поддерживающих верификацию: правила выполнения ограничений для выбранных инструментов (если это не вредит более существенным целям);<br>- ограниченное использование рекурсии (таблица В.1-6) и другие формы циклических зависимостей.<br>(См. также модульный подход.) |
| Статическая верификация соответствия специфицированному проекту (7.9.2.12) | Указания по кодированию для реализации специфицированных в проекте концепций или ограничений. Например:<br>- указания по кодированию циклов с гарантируемым максимальным значением времени цикла (таблица А.2-13a);<br>- указания по кодированию архитектуры с временным распределением (таблица А.2-13b);<br>- указания по кодированию архитектуры, управляемой событиями, с гарантируемым максимальным временем ответа (таблица А.2-13c);<br>- циклы со статически определенным максимальным числом итераций (за исключением бесконечного цикла, предусмотренного в проекте);<br>- указания по кодированию статического распределения ресурсов (таблица А.2-14) и предотвращение динамических объектов (таблица В.1-2);<br>- указания по кодированию статической синхронизации доступа к совместно используемым ресурсам (таблица А.2-15);<br>- указания по кодированию, по соблюдению ограничений на использование прерываний (таблица В.1-4);<br>- указания по кодированию, чтобы не использовать динамические переменные (таблица В.1-3a);<br>- проверка установки динамических переменных в неавтономном режиме (таблица В.1-3b);<br>указания по кодированию, чтобы гарантировать совместимость с другими используемыми языками программирования (7.4.4.10).<br>Указания, способствующие отслеживаемости проекта |
| Подмножество языков (таблица А.3-3):<br>- запрет опасных функций языка (7.4.4.13);<br>- использование только определенных функций языка (7.4.4.10);<br>- структурное программирование (таблица А.4-6),<br>- строго типизированный язык программирования (таблица А.3-2);<br>- отсутствие автоматического преобразования типов (таблица В.1-8) | Исключение функций языка, приводящих к неструктурированным проектам. Например:<br>- ограниченное использование указателей (таблица В.1-5),<br>- ограниченное использование рекурсии (таблица В.1-6),<br>- ограниченное использование объединений подобных в С;<br>- ограниченное использование исключений, подобных в Ada или С++,<br>- неиспользование неструктурированного потока управления в программах на языках высокого уровня (таблица В.1-7),<br>- одна точка входа/одна точка выхода в подпрограммах и функциях (таблица В.9-5);<br>- неиспользование автоматического преобразования типов;<br>- ограниченное использование побочных эффектов, сигнатур функций (например статических переменных).<br>Не допускать побочные эффекты в оценке условий и во всех формах утверждений.<br>Ограниченное или только документально оформленное использование специфичных для компилятора функций.<br>Ограниченное использование конструкций языка, которые могут ввести в заблуждение.<br>Должны применяться правила, когда эти возможности языка тем не менее используются |

*Окончание таблицы*

| Требования и рекомендации МЭК 61508-3 | Указания стандартов кодирования |
|---|---|
| Хорошая практика программирования (7.4.4.13) | Когда применяются:<br>- указания по кодированию, гарантирующие, что, в случае необходимости, выражения с плавающей точкой оцениваются в правильном порядке (например «a-b+c» не всегда равно «a+c-b»),<br>- в сравнениях с плавающей точкой: использовать только неравенства (меньше чем, меньше или равный, больше чем, больше или равный) вместо строгого равенства; указания, относящиеся к условной компиляции и «препроцессорной обработке»;<br>- систематическая проверка условий возврата (успех/отказ).<br>Документировать и по возможности автоматизировать создание исполняемого кода (make-файлы).<br>Предотвращать побочные эффекты, не очевидные из сигнатур функций. Когда такие побочные эффекты существуют, в соответствии с указаниями их необходимо документально оформить.<br>Заключать в скобки, когда приоритет операторов не абсолютно очевиден.<br>Искать предположительно невозможные ситуации (например ситуация «по умолчанию» в «переключателях» языка С).<br>Использовать «окружения» для критических модулей, в особенности чтобы проверить преди постусловия и условия возврата.<br>Соблюдать указания по кодированию в условиях известных ошибок компилятора и в пределах, установленных оценкой компилятора |

**С.2.6.3 Отказ от динамических переменных или динамических объектов**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблицы А.2 и В.1).

Цель. Исключить:
- нежелательные или необнаруживаемые наложения в памяти;
- узкие места ресурсов в процессе (связанном с безопасностью) выполнения программы.

Описание. В случае применения этого метода динамические переменные и динамические объекты получают определяемые во время выполнения программы определенные и абсолютные адреса в памяти. Объем (размер) распределяемой памяти и ее адреса зависят от состояния системы в момент распределения памяти и не могут быть проверены компилятором или другим автономным инструментом.

Так как число динамических переменных и объектов и существующее свободное пространство памяти для размещения новых динамических переменных или объектов зависит от состояния системы в момент их размещения, то при размещении или при использовании переменных или объектов возможны сбои. Например, если объем свободной памяти, распределяемый системой, недостаточен, то содержимое памяти другой переменной может быть неумышленно стерто. Если динамические переменные или объекты не используются, то появление этих ошибок исключено.

Необходимы ограничения на использование динамических объектов, если динамическое поведение не может быть точно предсказано с помощью статического анализа (то есть перед выполнением программы), и поэтому не может быть гарантировано предсказуемое выполнение программы.

**С.2.6.4 Проверка создания динамических переменных или динамических объектов при выполнении программы**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица В.1).

Цель. Убедиться в том, что память, в которой должны быть размещены динамические переменные и объекты, свободна до ее загрузки, а размещение в ней динамических переменных и объектов во время выполнения программы не повлияет на уже существующие в ней переменные, данные или коды.

Описание. В случае применения этих средств к динамическим переменным относят те переменные, которые имеют свои конкретные и абсолютные адреса в памяти, устанавливаемые во время выполнения программы (в этом смысле динамические переменные являются также атрибутами экземпляров объектов).

Аппаратными либо программными средствами память проверяется на то, что она свободна до размещения в ней динамических переменных или объектов (например для того, чтобы исключить переполнение стека). Если размещение не разрешается (например если памяти по конкретному адресу недостаточно), должны быть предприняты соответствующие действия. После использования динамических переменных или объектов (например после выхода из подпрограммы) вся используемая ими память должна быть освобождена.

Примечание — Альтернативным методом является демонстрация статического распределения памяти.

**С.2.6.5 Ограниченное использование прерываний**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица В.1).

Цель. Сохранение верифицируемости и тестируемости программного обеспечения.

Описание. Использование прерываний должно быть ограничено. Прерывания могут использоваться, если они упрощают систему. Использование программных средств для обработки прерываний должно быть запрещено в критических ситуациях для выполняемых функций (например критичность по времени, критичность изменений данных). Если прерывания используются, то непрерываемые фрагменты должны иметь специфицированное максимальное время вычисления с тем, чтобы можно было вычислить максимальное время, в течение которого прерывание запрещено. Использование прерываний и их маскирование должны подробно документироваться.

**С.2.6.6 Ограниченное использование указателей**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица В.1).

Цель. Исключение проблем, связанных с доступом к данным без предварительной проверки типа и диапазона указателя. Обеспечение модульного тестирования и верификации программных средств. Ограничение последствия отказов.

Описание. В прикладных программных средствах арифметика указателей может быть использована на уровне исходного кода только в случае, если тип и диапазон значений указателя данных (для гарантии того, что ссылка указателя находится внутри корректного адресного пространства) будут проверены перед доступом. Межпроцессное взаимодействие в прикладных программах не должно осуществляться прямым доступом между задачами. Обмен данными должен осуществляться через операционную систему.

**С.2.6.7 Ограниченное использование рекурсий**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица В.1).

Цель. Исключение неверифицируемого и нетестируемого использования вызовов подпрограмм.

Описание. Если используется рекурсия, то должен быть определен четкий критерий, который делает глубину рекурсии предсказуемой.

**С.2.7 Структурное программирование**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица А.4).

Цель. Проектирование и реализация программы с использованием практического анализа программы без ее выполнения. Программа может содержать только абсолютный минимум статистически нетестируемого поведения.

Описание. Для минимизации структурной сложности программы следует применять следующие принципы:

- разделять программу на подходящие, небольшие, минимально связанные программные модули, все взаимодействия между которыми точно специфицированы;
- составлять поток управления программными модулями с использованием таких структурированных конструкций, как последовательности, итерации и выбор;
- обеспечивать небольшое число возможных путей через программные модули и возможно более простые отношения между входными и выходными параметрами;
- исключать сложные ветвления и, в частности, безусловные переходы (goto) при использовании языков высокого уровня;
- по возможности связывать ограничения цикла и ветвление с входными параметрами;
- исключать использование сложных вычислений в ветвлении и цикле.

Следует использовать свойства языков программирования, которые способствуют указанному выше методу, предпочитая их другим свойствам, которые (как утверждают) более эффективны, за исключением случаев, когда эффективность приобретает абсолютный приоритет (например некоторые критичные к безопасности системы).

Литература:

Concepts in Programming Languages. J.C. Mitchell. Cambridge University Press, 2003, ISBN 0521780985, 9780521780988.

A Discipline of Programming. E.W. Dijkstra. Englewood Cliffs NJ, Prentice-Hall, 1976.

**С.2.8 Ограничение доступа/инкапсуляция информации**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица В.9).

Цель. Предотвращение непреднамеренного доступа к данным или процедурам и обеспечение тем самым качественной структуры программных средств.

Описание. Общедоступные для всех программных компонентов данные могут быть случайно или некорректно модифицированы любым из этих компонентов. Любые изменения этих структур данных могут потребовать подробной проверки программного кода и серьезных исправлений.

Ограничение доступа представляет собой общий метод к минимизации указанных выше проблем. Ключевые структуры данных «скрыты», и с ними можно работать только через конкретный набор процедур доступа, это позволяет модифицировать внутренние структуры данных или добавлять новые процедуры и при этом не оказывать влияния на функциональное поведение остальных программных средств. Например, имя директории могут иметь процедуры доступа «вставить», «удалить» и «найти». Процедуры доступа и структуры внутренних данных могут быть изменены (например при использовании различных методов просмотра или запоминании

имен на жестком диске), не оказывая влияния на логическое поведение остальных программных средств, использующих эти процедуры.

В данном случае следует использовать концепцию абстрактных типов данных. Если непосредственная проверка не предусмотрена, может оказаться необходимым проверить, не было ли абстрагирование случайно разрушено.

Литература:

Concepts in Programming Languages. J.C. Mitchell. Cambridge University Press, 2003. ISBN 0521780985, 9780521780988.

On Design and Development of Program Families. D.L. Parnas. IEEE Trans SE-2, March 1976.

**С.2.9 Модульный подход**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблицы А.4 и В.9).

Цель. Декомпозирование программной системы на небольшие законченные модули в целях сокращения сложности системы.

Описание. Модульный подход, или модуляризация, включает в себя несколько различных правил для этапов проектирования, кодирования и эксплуатации проекта программных средств. Эти правила меняются в соответствии с реализуемым методом проектирования. Большинство методов подчиняются следующим правилам:

- программный модуль (или подпрограмма, что одно и тоже) должен выполнять одну четко сформулированную задачу или функцию;
- связи между программными модулями должны быть ограничены и строго определены, уровень связности каждого программного модуля должен быть высоким;
- совокупности подпрограмм должны строиться так, чтобы обеспечивать несколько уровней программных модулей;
- размеры подпрограмм следует ограничить некоторыми конкретными значениями, обычно от двух до четырех размеров экрана;
- подпрограммы должны иметь только один вход и один выход;
- программные модули должны взаимодействовать с другими программными модулями через свои интерфейсы, где используются глобальные или общие переменные, которые должны быть хорошо структурированы; доступ к ним должен находиться под контролем, и их использование в каждом конкретном случае должно быть обосновано;
- все интерфейсы программных модулей должны быть полностью документально оформлены;
- все интерфейсы программных модулей должны содержать только необходимые для их функционирования параметры. Однако эта рекомендация усложнена тем, что язык программирования может иметь параметры по умолчанию или используется объектно-ориентированный подход.

Литература:

The Art of Software Testing, second edition. G.J. Myers, T. Badgett, T.M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124.

Software Engineering for Real-time Systems. J.E. Cooling, Pearson Education, 2003. ISBN 0201596202, 9780201596205.

Concepts in Programming Languages. J.C. Mitchell. Cambridge University Press, 2003. ISBN 0521780985, 9780521780988.

**С.2.10 Использование доверительных/проверенных элементов программного обеспечения**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблицы А.2, С.2, А.4 и С.4).

Цель. Исключение такого проектирования программных модулей и элементов, которое вызывало бы необходимость их интенсивных повторных проверок или перепроектирования для каждого нового применения. Использование преимущества проектов, которые не были формально или строго проверены, но для которых имеется продолжительный опыт эксплуатации. Использовать преимущества уже существующих программных элементов, которые были проверены для различных применений и для которых существует совокупность доказательств подтверждения соответствия.

Описание. Данный метод проверяет наличие в программных элементах систематических ошибок проектирования и/или эксплуатационных отказов.

Строить сложную систему, используя элементарные компоненты, нецелесообразно. Как правило, используют основные узлы («элементы», см. МЭК 61508-4, пункты 3.2.8 и 3.4.5), которые были разработаны ранее для обеспечения некоторых полезных функций и которые могут быть использованы для реализации некоторой части новой системы.

Грамотно спроектированные и структурированные ПЭ системы состоят из ряда программных элементов, которые различаются между собой четким образом и взаимодействуют друг с другом вполне определенными способами. Формирование библиотеки таких общепримениимых программных элементов, которые можно повторно использовать в нескольких применениях, позволяет большую часть ресурсов, необходимых для подтверждения соответствия проектов, распределять по нескольким применениям.

Однако для применений, связанных с безопасностью, важно иметь достаточную уверенность, что новая система, включающая эти уже существующие элементы, имеет необходимую полноту безопасности, а также, что безопасность новой системы не нарушается некоторым некорректным поведением уже существующих элементов.

Существуют два подхода, как обрести уверенность в том, что поведение уже существующих элементов точно известно:

- провести всесторонний анализ опыта эксплуатации элемента, чтобы продемонстрировать, что элемент был «проверен в эксплуатации»;

- оценить совокупность доказательств подтверждения соответствия, которая была собрана для поведения элемента, чтобы определить, соответствует ли этот элемент требованиям настоящего стандарта.

**С.2.10.1 Проверка в эксплуатации**

Только в редких случаях «проверки в эксплуатации» (см. МЭК 61508-4, пункт 3.8.18) будет достаточно в качестве единственного средства, гарантирующего для доверительного элемента программного обеспечения достижение им необходимого уровня полноты безопасности. Для сложных элементов со многими возможными функциями (например операционной системы) важно установить, какая из функций достаточно проверена при ее использовании. Например, процедуру самотестирования для обнаружения ошибок, если в период ее эксплуатации не появилось отказов, нельзя рассматривать как проверенную в эксплуатации.

Программный элемент может быть проверенным в эксплуатации, если он соответствует следующим критериям:

- спецификация не менялась;
- использовался в системах в различных областях применения;
- продолжительность срока его эксплуатации не менее года;
- продолжительность эксплуатации соответствует уровню полноты безопасности или соответствующему числу запросов; для демонстрации частоты отказов, не связанных с безопасностью, менее:
- $10^{-2}$ на один запрос (в год) с 95%-ным уровнем доверия необходимо 300 эксплуатационных прохождений (в год);
- $10^{-5}$ на один запрос (в год) с 99,9%-ным уровнем доверия необходимо 690000 эксплуатационных прохождений (в год).

Примечание — Математический аппарат, обеспечивающий числовые оценки данного метода, приведен в приложении D. Аналогичный метод и статистический подход изложены также в В.5.4;

- весь опыт эксплуатации связан с известным профилем запросов функций программного модуля для гарантии того, что увеличивающийся опыт эксплуатации действительно приводит к увеличению знаний о поведении программного модуля, связанного с соответствующим профилем запроса;
- его отказы не связаны с безопасностью.

Примечание — Отказ, некритичный для безопасности в одном контексте, может быть критичен для безопасности в другом контексте, и наоборот.

Для проверки соответствия критерию программного элемента должно быть документально оформлено:

- точная идентификация каждой системы и ее элементов, включая номера версий (как для программных, так и для аппаратных средств);
- идентификация пользователей и продолжительность их работы;
- продолжительность эксплуатации системы;
- процедура выбора систем, применяемых пользователями, и случаев ее применения;
- процедуры обнаружения и регистрации отказов и устранения сбоев.

**С.2.10.2 Оценка совокупности доказательств подтверждения соответствия**

Уже существующий элемент программного обеспечения (см. МЭК 61508-4, пункт 3.2.8) — это тот, который уже существует и не был разработан специально для текущего проекта или SRS. Уже существующее программное обеспечение может быть коммерческим доступным продуктом, или оно может быть разработано какой-то организацией для предыдущего изделия или системы. Предварительно существующее программное обеспечение может или не может быть разработано в соответствии с требованиями настоящего стандарта.

Для оценки полноты безопасности новой системы, включающей уже существующие программы, необходима совокупность доказательств подтверждения соответствия для определения поведения предварительно существующего элемента. Она может быть получена (1) из собственной документации поставщика элемента и описания процесса разработки элемента или (2) может быть создана или дополнена дополнительными квалифицированными мероприятиями, выполненными разработчиком новой системы, связанной с безопасностью, или третьими лицами. Возможности и ограничения потенциально повторно используемого программного элемента определяются в «Руководстве по безопасности для применяемых изделий».

В любом случае должно существовать (или должно быть создано) руководство по безопасности для применяемых изделий, которое обеспечивает адекватную возможность выполнить оценку полноты безопасности конкретной функции безопасности, которая полностью или частично реализуется повторно используемым элементом. Если его нет, то должен быть сделан консервативный вывод о том, что для элемента не подтверждена возможность его повторного использования в системе, связанной с безопасностью. (Это не означает, что для элемента вообще не подтверждена возможность его повторного использования, просто в данном конкретном случае не было найдено достаточно доказательств.)

Настоящий стандарт предъявляет особые требования к содержанию руководства по безопасности для применяемых изделий, см. МЭК 61508-2, приложение D, МЭК 61508-3, приложение D и МЭК 61508-3, пункты 7.4.2.12 и 7.4.2.13.

В «Руководстве по безопасности для применяемых изделий» будет рассмотрено, что:

- проект элемента известен и документально оформлен;
- элемент был объектом проверки и подтверждения соответствия на основе систематического подхода с документально оформленной проверкой и анализом всех частей проекта элемента и кода;

- неиспользуемые и ненужные функции элемента не помешают новой системе выполнения своих требований к безопасности;
- все вероятные механизмы отказа элемента в новой системе были выявлены и было выполнено их соответствующее ослабление.

Оценка функциональной безопасности новой системы должна установить, что повторно используемый элемент применяется строго в пределах возможностей, которые для этого элемента были обоснованы доказательством и предположениями в «Руководстве по безопасности для применяемых изделий».

Литература:

Component-Based Software Development: Case Studies. Kung-Kiu Lau. World Scientific, 2004, ISBN 9812388281, 9789812388285.

Software Reuse and Reverse Engineering in Practice. P.A. V. Hall (ed.), Chapman & Hall, 1992, ISBN 0-412-39980-6.

Software criticality analysis of COTS/SOUP. P. Bishop, T. Clement, S. Guerra. In Reliability Engineering & System Safety, Volume 81, Issue 3, September 2003, Elsevier Ltd., 2003.

**С.2.11 Прослеживаемость**

Цель. Обеспечить согласованность между этапами жизненного цикла.

Описание. Для того, чтобы гарантировать для программного обеспечения, что результаты действий на этапах жизненного цикла соответствуют требованиям корректной работы системы, связанной с безопасностью, крайне важно гарантировать обеспечение соответствия между этапами жизненного цикла. Ключевым понятием здесь является «прослеживаемость» между действиями. В сущности, это выполнение анализа влияния, проверяющего (1), что решения, принятые на ранней стадии, адекватно реализованы на более поздних стадиях (прямая прослеживаемость), и (2), что решения, принятые на более позднем этапе, действительно необходимы и санкционированы ранее принятыми решениями (обратная прослеживаемость).

Прямая прослеживаемость в основном связана с проверкой адекватности требований на более поздних этапах жизненного цикла. Прямая прослеживаемость важна в нескольких точках жизненного цикла системы безопасности:

- между требованиями к системе безопасности и требованиями к программному обеспечению системы безопасности;
- между спецификациями требований к программному обеспечению системы безопасности и к архитектуре программного обеспечения;
- между спецификациями требований к программному обеспечению системы безопасности и к проекту программного обеспечения;
- между спецификацией проекта программного обеспечения и спецификациями испытаний модуля и интеграции;
- между требованиями к интеграции аппаратных средств/программного обеспечения проекта системы и программного обеспечения и спецификациями испытаний интеграции аппаратных средств/программного обеспечения;
- между спецификацией требований к программному обеспечению системы безопасности и планом подтверждения соответствия программного обеспечения безопасности системы;
- между спецификацией требований к программному обеспечению системы безопасности и планом модификации программного обеспечения (в том числе повторной проверкой и повторного подтверждения соответствия);
- между спецификацией проекта программного обеспечения и планом верификации программного обеспечения (включая верификацию данных);
- между требованиями МЭК 61508-3, раздел 8, и планом оценки функциональной безопасности программного обеспечения.

Обратная прослеживаемость в основном связана с проверкой, насколько корректно любым требованием обосновывается каждое реализационное решение (реализация понимается в широком контексте, а не только реализация кода). Если такое обоснование отсутствует, то реализация будет содержать что-то не столь необходимое, что приведет к увеличению сложности, но не обязательно удовлетворит любому реальному требованию к системе, связанной с безопасностью. Обратная прослеживаемость важна в нескольких точках жизненного цикла системы безопасности:

- между требованиями к системе безопасности и предполагаемыми потребностями безопасности;
- между архитектурой программного обеспечения и спецификацией требований к программному обеспечению системы безопасности;
- между детальным проектом программного обеспечения и архитектурой программного обеспечения;
- между программным кодом и детальным проектом программного обеспечения;
- между планом подтверждения соответствия программного обеспечения безопасности системы и спецификацией требований к программному обеспечению системы безопасности;
- между планом модификации программного обеспечения и спецификацией требований к программному обеспечению системы безопасности;
- между планом верификации программного обеспечения (включая верификацию данных) и спецификацией проекта программного обеспечения.

Литература:

Requirements Engineering. E. Hull, K. Jackson, J. Dick. Springer, 2005, ISBN 1852338792, 9781852338794.

**С.2.12 Проектирование программного обеспечения, не сохраняющего состояние (или проектирование ПО, сохраняющего ограниченное описание состояния)**

Цель. Ограничить сложность поведения программного обеспечения.

Описание. Рассмотрим программу, которая обрабатывает последовательность транзакций: она получает последовательность входных данных и для каждого из них вычисляет выходные данные. Программа может также запоминать некоторые или все состояния в процессе вычисления и может также учитывать эти состояния при вычислении результата для следующих входных данных.

Если выходные данные программы полностью определяются только входными, то говорят, что такая программа работает без запоминания или является программой, не сохраняющей состояние. Каждая транзакция входных/выходных данных полна в том смысле, что на любую транзакцию никак не влияет любая, более ранняя транзакция, и конкретные входные данные всегда приводят к тем же самым связанным с ними выходным данным.

Если программа при вычислении входных данных учитывает, кроме входных данных, также и состояние, которое она запомнила в результате предыдущих вычислений, то такая программа обладает более сложным поведением, потому что в различных случаях она может давать различные выходные данные для одних и тех же входных данных. Результат для конкретных входных данных может зависеть от контекста (то есть от предыдущих входных и выходных данных), в котором они обрабатываются. Необходимо также отметить, что в некоторых приложениях (обычно коммуникационные системы) поведение программы может быть особенно чувствительно к изменениям в сохраненном состоянии, которые могут произойти или непреднамеренно или злонамеренно.

Проектирование с несохраняющимися состояниями (или с сохранением ограниченного описания состояний) является общим подходом, направленным на минимизацию возможной сложности поведения программного обеспечения, исключая или уменьшая использование информации о состоянии при проектировании программного обеспечения.

Литература:

Introduction to Automata Theory, Languages, and Computation (3rd Edition). J. Hopcroft, R. Motwani, J. Ullman, Addison-Wesley Longman Publishing Co, 2006, ISBN:0321462254.

Stateless connections. T. Aura, P. Nikander. In Proc International Conference on Information And Communications Security (ICICS'97), ed Yongfei Han. Springer, 1997, ISBN 354063696X, 9783540636960.

**С.2.13 Численный анализ в автономном режиме**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица А.9).

Цель. Гарантировать точность числовых вычислений.

Описание. Числовая погрешность может возникнуть при вычислении математической функции как следствие использования конечных представлений идеальных функций и чисел. Ошибка усечения появляется, когда функция аппроксимируется конечным числом членов бесконечного ряда, таким как ряд Фурье. Для представления в реальном компьютере вещественных чисел с конечной точностью, вводится их погрешность округления. Если выполняются какие-либо, кроме самых простейших, вычисления с плавающей точкой, то должна быть проверена обоснованность вычисления, чтобы гарантировать, что точность, требуемая приложением, фактически достигнута.

Литература:

Guide to Scientific Computing. P.R. Turner. CRC Press, 2001, ISBN 0849312426, 9780849312427.

**С.2.14 Диаграммы последовательности сообщений**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблицы В.7 и С.17).

Цель. Помочь получению требований к системе на ранних стадиях проектирования программного обеспечения, включая стадии формирования требований и проектирования архитектуры программного обеспечения. В UML это называется «Диаграмма последовательности системы» (System Sequence Diagram).

Описание. Диаграмма последовательности сообщений — графический механизм для описания поведения системы с точки зрения коммуникаций между агентами системы (агентом может быть человек, компьютерная система или элемент либо объект программного обеспечения, в зависимости от стадии проектирования). Для каждого агента на схеме представлен вертикальный «жизненный путь», а стрелки между ними используются, чтобы представить сообщения. Действия по получении сообщений можно дополнительно показать на схемах в виде прямоугольников. Набор сценариев (описывающих и требуемое и нежелательное поведение) создается как спецификация необходимого поведения системы. Эти сценарии имеют несколько применений. Может быть проведена их анимация, чтобы продемонстрировать поведение системы конечным пользователям. Они могут быть преобразованы в исполнимую реализацию системы. Они могут сформировать основу тестовых данных.

UML содержит расширения обычной концепции диаграммы последовательности сообщений в виде конструкций выбора и итерации, которые позволяют сценариям выполнять условные переходы и циклы, обеспечивая более компактную нотацию. Могут быть также определены подсхемы, на которые можно сослаться из нескольких диаграмм последовательностей более высокого уровня. Также могут быть представлены таймер и внешние события.

Литература:

«Message Sequence charts», D. Harel, P. Thiagarajan. In UML for Real: Design of Embedded Real-Time Systems. ed. L. Lavagno. Springer, 2003, ISBN 1402075014, 9781402075018.

ISO/IEC 19501:2005, Information technology — Open Distributed Processing — Unified Modeling Language (UML) Version 1.4.2.

**С.3 Архитектурное проектирование**

**С.3.1 Обнаружение и диагностика сбоев**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица А.2).

Цель. Обнаружение сбоев в системе, которые могут привести к отказам и тем самым обеспечить основу для контрмер, направленных на минимизацию числа последующих сбоев.

Описание. Обнаружение сбоев представляет собой действие по проверке системы на наличие ошибочных состояний (обусловленных сбоями в проверяемой (под)системе). Основная цель обнаружения сбоев состоит в том, чтобы предотвратить появление неверных результатов. Система, действущая в сочетании с параллельно работающими компонентами, останавливающими управление, в случае, если она обнаруживает, что ее собственные результаты некорректны, называется самопроверяемой.

Обнаружение сбоев основывается на принципах избыточности [в основном при обнаружении сбоев аппаратных средств (см. МЭК 61508-2, приложение А)] и разнообразия (программные ошибки). Необходим один из способов голосования для определения корректности результатов. Применимы специальные методы, к которым относятся программирование утверждений, программирование *N*-версий и различные методы контроля. Для аппаратных средств: введение дополнительных сенсоров; контуров регулирования; кодов, проверяющих ошибки, и др.

Обнаружение сбоев может обеспечиваться проверками в области значений или временной области на различных уровнях, особенно на физическом уровне (температура, напряжение и т. п.), логическом (коды, обнаруживающие ошибки), функциональном (утверждения) или внешнем (проверки достоверности). Результаты этих проверок могут быть сохранены и связаны с данными, на которые повлиял сбой, с тем чтобы обеспечить возможность отслеживания отказов.

Сложные системы состоят из подсистем. Эффективность обнаружения ошибок, диагностики и компенсации ошибок зависит от сложности взаимодействия между подсистемами, влияющими на распространение ошибок.

Диагностику ошибок следует применять на уровне самых малых подсистем, поскольку подсистемы меньших размеров допускают более детальную диагностику ошибок (обнаружение ошибочных состояний).

Интегрированные информационные системы уровня предприятия могут обычным способом передавать состояния безопасности системы, в том числе информацию диагностического тестирования, другим управляющим системам. При обнаружении некорректного поведения оно может быть выделено и использовано для запуска корректирующих действий до возникновения опасной ситуации. При появлении инцидента документирование такого некорректного поведения может способствовать его последующему анализу.

Литература:

Dependability of Critical Computer Systems 1. F. J. Redmill, Elsevier Applied Science, 1988, ISBN 1-85166-203-0.

**С.3.2 Коды обнаружения и исправления ошибок**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица А.2).

Цель. Обнаружение и исправление ошибок в чувствительной к ним информации.

Описание. Для информации, состоящей из *n* битов, генерируется заходированный блок из *k* битов, который позволяет обнаруживать и исправлять *r* ошибок. Примерами могут служить код Хэмминга и полиномиальные коды.

Следует отметить, что в системах, связанных с безопасностью, чаще необходимо уничтожить ошибочные данные, чем пытаться исправлять их, поскольку лишь заранее определенная часть ошибок может быть исправлена.

Литература:

Fundamentals of Error-correcting Codes, W. Huffman, V. Pless. Cambridge University Press, 2003, ISBN 0521782805, 9780521782807

**С.3.3 Программирование с проверкой ошибок**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-2 (таблица А.18) и в МЭК 61508-3 (таблица А.2).

Цель. Обнаружение ошибок, оставшихся при проектировании программных средств, в процессе выполнения программ в целях предотвращения критичных для безопасности отказов систем, и продолжение правильного выполнения программы.

Описание. В методе программирования утверждений уже заложена идея проверки предусловий (до выполнения последовательности операторов начальные условия проверяют на соответствие) и постусловий (проверяют результаты после выполнения последовательности операторов). Если предусловия или постусловия не соблюдаются, то выдается сообщение об ошибке.

***Пример —***
***assert < pre-condition>;***
***action 1;***
***.....***
***action x;***
***assert < post-condition>;***

Литература:

Exploiting Traces in Program Analysis. A. Groce, R. Joshi. Lecture Notes in Computer Science, vol. 3920, Springer Berlin/Heidelberg, 2006, ISBN 978-3-540-33056-1.

Software Development — A Rigorous Approach. C. B. Jones, Prentice-Hall, 1980.

**С.3.4 Методы контроля**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица А.2).

Цель. Защита от необнаруженных на этапах спецификации и реализации ошибок в программных средствах, которые неблагоприятно влияют на их безопасность.

Описание. Различают два подхода реализации контроля: (1) процесс контроля и контролируемая функция реализованы на одном компьютере с некоторой гарантией независимости между ними; и (2) процесс контроля и контролируемая функция реализованы на разных компьютерах.

Метод, в котором процесс контроля и контролируемая функция реализованы на разных компьютерах (имеющих разную спецификацию), называется методом внешнего контроля. Данный метод направлен только на то, чтобы гарантировать, что основным компьютером выполняются безопасные, но не обязательно корректирующие действия. Метод внешнего контроля обеспечивает непрерывный контроль основного компьютера и предотвращает вхождение системы в опасное состояние. Кроме того, если обнаружится, что основной компьютер вошел в потенциально опасное состояние, система должна возвратиться обратно в безопасное состояние с помощью либо средств внешнего контроля, либо основного компьютера.

Аппаратные средства и программное обеспечение средств внешнего контроля следует классифицировать и квалифицировать в соответствии с подходящим УПБ.

Литература:

Requirements based Monitors for Real-Time Systems, D. Peters, D. Parnas. IEEE Transactions on Software Engineering, vol. 28, no. 2, 2002.

**С.3.5 Многовариантное программирование**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица А.2).

Цель. Обнаружение и наложение маски при выполнении программ на невыявленные на этапах проектирования и реализации ошибки программных средств для предотвращения критичных для безопасности отказов системы и продолжения ее правильной работы.

Описание. При многовариантном программировании заданная программная спецификация проектируется и реализуется различными способами $N$ раз. Одни и те же входные значения поступают в $N$ версий и сравниваются результаты, выданные $N$ версиями. Если результат определяется как правильный, он поступает на выходы компьютера.

Важным требованием является то, что в некотором смысле $N$ версий независимы друг от друга, поэтому они не все одновременно перестают правильно работать по общей причине. Независимость версий, являющуюся основой для многовариантного программирования, на практике довольно трудно достичь и продемонстрировать.

$N$ версий могут выполняться параллельно на различных компьютерах, либо все версии могут выполняться на одном компьютере с последующим сравнением полученных результатов на том же компьютере. Для этих $N$ результатов могут быть использованы различные стратегии сравнения, и в зависимости от заданных требований применяются следующие стратегии:

- если система находится в безопасном состоянии, можно потребовать полного согласия (все $N$ результатов одинаковы); в противном случае используется выходное значение, которое заставит систему перейти в безопасное состояние. Для простых пошаговых систем сравнение может обеспечить безопасность. В этом случае безопасное действие может быть разбито по шагам, если какая-либо версия реализует пошаговые операции. Этот подход обычно используется только для двух версий ($N = 2$);
- для систем, находящихся в опасном состоянии, могут быть реализованы стратегии мажоритарного сравнения. В случаях, если отсутствует общее согласие, могут использоваться вероятностные подходы с тем, чтобы максимизировать вероятность выбора правильного значения, например, принять среднее значение, временно зафиксировать выходы, пока не будет достигнуто согласие и т. п.

Данный метод не устраняет ошибок, не выявленных при проектировании программ, а также ошибок в интерпретации спецификации, однако он является средством для обнаружения и маскирования ошибок, прежде чем они смогут повлиять на безопасность.

Литература:

Modelling software design diversity — a review, B. Littlewood, P. Popov, L. Strigini. ACM Computing Surveys, vol 33, No. 2, 2001.

The N-Version Approach to Fault-Tolerant Software, A. Avizienis, IEEE Transactions on Software Engineering, vol. SE-11, No. 12, pp.1491—1501, 1985.

An experimental evaluation of the assumption of independence in multi-version programming, J.C. Knight, N.G. Leveson. IEEE Transactions on Software Engineering, vol. SE-12, No. 1, 1986.

In Search of Effective Diversity: a Six Language Study of Fault-Tolerant Flight Control Software. A. Avizienis, M.R. Lyu and W. Schutz. 18th Symposium on Fault-Tolerant Computing, Tokyo, Japan, 27—30 June 1988, IEEE Computer Society Press, 1988, ISBN 0-8186-0867-6.

**С.3.6 Восстановление предыдущего состояния**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица А.2).

Цель. Обеспечение исправления функциональных операций при наличии одной или нескольких ошибок.

Описание. При обнаружении ошибки система возвращается в первоначальное внутреннее состояние, правильность которого была подтверждена ранее. Данный метод предполагает частое сохранение внутреннего состояния в так называемых четко определенных контрольных точках. Сохранение может быть выполнено глобально (для всей базы данных) или частично (для изменений только между контрольными точками). После этого система

должна устранить изменения, произошедшие за это время путем занесения в журнал (аудиторское отслеживание действий), компенсации (все результаты этих изменений аннулируются) или внешнего (ручного) способа.

Литература:

Looking into Compensable Transactions. Jing Li, Huibiao Zhu, Geguang Pu, Jifeng He. In Software Engineering Workshop, 2007. SEW 2007. IEEE, 2007, ISBN 978-0-7695-2862-5.

Software Fault Tolerance (Trends in Software, No. 3), M.R. Lyu (ed.), John Wiley & Sons, April 1995, ISBN 0471950688.

**С.3.7 Механизмы повторных попыток парирования сбоя**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица А.2).

Цель. Парирование обнаруженного сбоя с помощью механизмов повторных попыток.

Описание. В случае обнаружения сбоя или ошибочного условия предпринимаются попытки парирования сбоя или восстановления ситуации путем повторного выполнения того же кода. Восстановление с помощью повторной попытки может быть полным в виде перезагрузки и повторного пуска процедуры, либо небольшим в виде перепланирования и повторного пуска задачи после выполнения блокировки по времени программы или управляющего действия задачи. Методы повторной попытки широко используются при коммуникационных сбоях или при восстановлении от ошибок, и условия повторной попытки могут быть отделены флажками от ошибки протокола связи (контрольная сумма и т. д.) или от подтверждающего ответа блокировки по времени коммуникации.

Литература:

Reliable Computer Systems: Design and Evaluation, D.P. Siewiorek and R.S. Schwartz, A.K. Peters Ltd., 1998, ISBN 156881092X.

**С.3.8 Постепенное отключение функций**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица А.2).

Цель. Обеспечение пригодности наиболее критичных системных функций, несмотря на отказы, путем игнорирования наименее критичных функций.

Описание. Данный метод устанавливает приоритеты для различных функций, выполняемых системой. Проект создаваемой системы гарантирует, что в случае недостаточности ресурсов для выполнения всех системных функций функции высшего приоритета будут выполнены в предпочтение функциям более низкого приоритета. Например, функции регистрации ошибки и события могут оказаться задачей более низкого приоритета, чем системные функции управления, и в этом случае управление системой будет продолжаться, даже если аппаратные средства из-за регистрации ошибки окажутся неработоспособными. Более того, если аппаратные средства управления системой окажутся неработоспособными, а аппаратные средства регистрации ошибок останутся работоспособными, то аппаратные средства регистрации ошибок возьмут на себя функцию управления.

Данные соображения относятся в основном к аппаратным средствам, но они применимы также и к системе в целом, включая программное обеспечение. Данные соображения должны учитываться, начиная с самых ранних этапов проектирования.

Литература:

Towards the Integration of Fault, Resource, and Power Management, T. Siridakis. In Computer Safety, Reliability, and Security: 23rd International Conference, SAFECOMP 2004. Eds. Maritta Heisel et al. Springer, 2004, ISBN 3540231765, 9783540231769.

Achieving Critical System Survivability Through Software Architectures, E.A. Strunk. Springer Berlin/Heidelberg, 2004, ISBN 978-3-540-23168-4.

The Evolution of Fault-Tolerant Computing. Vol. 1 of Dependable Computing and Fault-Tolerant Systems, Edited by A. Avizienis, H. Kopetz and J. C Laprie, Springer Verlag, 1987, ISBN 3-211-81941-X.

Fault Tolerance, Principle and Practices. T. Anderson and P.A. Lee, Vol. 3 of Dependable Computing and Fault-Tolerant Systems, Springer Verlag, 1987, ISBN 3-211-82077-9.

**С.3.9 Исправление ошибок методами искусственного интеллекта**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица А.2).

Цель. Способность гибко реагировать на возможные угрозы безопасности, используя сочетания методов и моделей процессов, а также некоторые способы безопасности в режиме онлайн и анализа надежности.

Описание. Для различных каналов связи системы прогнозирование (вычисление тенденций), исправление ошибок, обслуживание и контролирующие действия могут достаточно эффективно поддерживаться системами, основанными на методах искусственного интеллекта (AI). Правила для таких систем могут быть созданы непосредственно из спецификаций и проверены на соответствие. С помощью методов искусственного интеллекта некоторые ошибки общего характера, попадающие в спецификации, для устранения которых уже существуют некоторые правила проектирования и реализации, могут быть исключены, особенно при представлении комбинаций моделей и методов функциональным или описательным способом.

Методы выбираются так, чтобы ошибки могли быть устранены и влияние отказов минимизировано для обеспечения требуемой полноты безопасности.

Примечание — Предупреждение об исправлении ошибочных данных см.С.3.2 и об отрицательных рекомендациях применения данного метода — МЭК 61508-3 (таблица А.2, пункт 5).

Литература:

Fault Diagnosis: Models, Artificial Intelligence, Applications. J. Korbicz, J. Koscielny, Z. Kowalczuk, W. Cholewa. Springer, 2004, ISBN 3540407677, 9783540407676.

**С.3.10 Динамическая реконфигурация**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица А.2).

Цель. Обеспечение функциональности системы, несмотря на внутренний отказ.

Описание. Логическая архитектура системы должна быть такой, чтобы ее можно было реализовать на подмножестве доступных средств системы. Логическая архитектура должна быть способна к обнаружению отказа в физических средствах и дальнейшей повторной реализации логической архитектуры на другом подмножестве доступных средств, остающихся функционирующими. Несмотря на то, что данный метод в основном традиционно ограничен только восстановлением отказавших модулей аппаратных средств, он применим также к ошибкам в программных средствах при наличии достаточной «избыточности времени прогона» для повторного выполнения программы или при наличии достаточных избыточных данных, которые обеспечат незначительное влияние отдельного и изолированного отказа.

Данный метод должен рассматриваться на первом этапе проектирования системы.

Литература:

Dynamic Reconfiguration of Software Architectures Through Aspects. C. Costa et al. Lecture Notes in Computer Science, Volume 4758/2007, Springer Berlin/Heidelberg, 2007, ISBN 978-3-540-75131-1.

**С.3.11 Безопасность и работа в жестком реальном времени. Архитектура с временны́м распределением**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица А.2).

Цель. Компонуемость и простая реализация обеспечения отказоустойчивости в критических к безопасности системах жесткого реального времени.

Описание. В архитектуре системы с временным распределением (TTA) все действия системы инициируются и выполняются под управлением глобальной (жесткой) системы синхронизации. Каждому приложению присвоен фиксированный временной слот на шине с временным распределением, во время которого происходит обмен сообщениями с другими приложениями, поэтому каждое приложение может выполнять обмен только согласно жестко определенному расписанию обмена. В управляемых событиями системах системные действия инициируются случайными событиями в непредсказуемые моменты времени. Главные преимущества архитектуры с временны́м распределением (см. Scheidler, Хайнер и др.) следующие:

- компонуемость, которая значительно уменьшает усилия, требуемые для тестирования и сертификации системы;
- простая реализация обеспечения отказоустойчивости, которая делает архитектуру очень востребованной для критических к безопасности приложений;
- применение глобально синхронизируемого времени облегчает проектирование распределенных систем реального времени.

Передача между узлами выполняется в соответствии с протоколом временно́го распределения ресурсов (TTP/C) (см. Kopetz, Hexel и др.) согласно статическому расписанию обмена, в рамках которого решается, когда передать сообщение и является ли полученное сообщение важным для конкретного электронного модуля. Доступ к шине реализуется по схеме с определенным периодическим расписанием в режиме множественного доступа с разделением времени (TDMA), связанной с глобальным временем.

Протокол TTP/C гарантирует (см. Rushby) четыре базовые услуги (базовые службы) в сети узлов TTA системы (см. Kopetz, Bauer):

- Детерминированная передача сообщений в строго определенные моменты времени. Передача сообщений от выходного порта передающего элемента к входным портам получающих элементов в пределах априорно известного временного интервала. Отказоустойчивая транспортная служба, предлагаемая коммуникационной услугой с временны́м распределением, которая доступна через временной интерфейс с сетевым экраном, устраняет передачу ошибок управления проектом и минимизирует связь между элементами. Передача сообщений в строго определенные моменты времени, с минимальной задержкой и с минимальными флуктуациями, крайне важна для достижения устойчивости управления в приложениях реального времени.
- Отказоустойчивая синхронизация. Коммуникационный контроллер (а не центральный сервер) генерирует отказоустойчивый синхронизирующий глобальный временной сигнал (с точностью до нескольких тактов), которым обеспечены подсистемы узлов.
- Контроль целостности данных при сбоях в узлах (служба целостности). Коммуникационный контроллер сообщает каждому SRU («минимальному элементу замены») о состоянии остальных SRU в кластере с временно́й задержкой меньше одного раунда TDMA.
- Строгая изоляция сбоя. Злонамеренно дефектная подсистема узла (включая ее программное обеспечение) может сформировать ошибочные выходные данные, но никогда не сможет вмешаться каким-либо способом в корректную работу остальной части кластера TTP/C. Невоспринимаемость сбоя во времени гарантируется поведением коммуникационного контроллера, реализующего временно́е распределение.

Примечание — Существуют другие протоколы с временны́м распределением: FlexRay и TT — Ethernet (Ethernet с временны́м распределением).

Литература:

Time-Triggered Architecture (TTA). C. Scheidler, G. Heiner, R. Sasse, E. Fuchs, H. Kopetz, C. Temple. In Advances in Information Technologies: The Business Challenge, ed. J-Y. Roger. IOS Press, 1998, ISBN 9051993854, 9789051993851.

A Synchronisation Strategy for a TTP/C Controller. H. Kopetz, R. Hexel, A. Krueger, D. Millinger, A. Schedl. SAE paper 960120, Application of Multiplexing Technology SP 1137, Detroit, SAE Press, Warrendale, 1996.

The Time-Triggered Architecture. H. Kopetz, G. Bauer. Proceedings of the IEEE Special Issue on Modeling and Design of Embedded Software, October 2002.

An Overview of Formal Verification for the Time-Triggered Architecture. J. Rushby: Invited paper, Oldenburg, Germany, September 9—12, 2002. Proceedings FTRTFT 2002, Springer LNCS 2469, 2002, ISBN 978-3-540-44165-6.

**C.3.12 UML**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица В.7).

Цель. Обеспечить исчерпывающий набор нотаций для моделирования требуемого поведения сложных систем.

Описание. UML, в соответствии с названием, — это набор требований и нотаций проектирования, которые предназначены обеспечить всестороннюю поддержку процессу разработки программного обеспечения. Некоторые части UML основаны на нотациях, впервые появившихся в других методах (таких как диаграмма последовательностей и диаграммы переходов), а другие нотации уникальны в UML. UML очень близок к объектно-ориентированному языку, хотя некоторые из нотаций могут не использоваться в объектно-ориентированном программировании. UML поддерживается многими коммерчески доступными CASE инструментами, многие из которых способны автоматически генерировать программные коды из моделей UML.

Нотации UML, которые обычно применяются для спецификации и проектирования систем, связанных с безопасностью, следующие:

- диаграммы классов;
- прецеденты;
- диаграммы действий;
- диаграммы переходов (диаграммы состояний);
- диаграммы последовательностей.

Другие нотации UML относятся к представлению проекта архитектуры программного обеспечения (структуры программного обеспечения), но в данном пункте не рассматриваются.

Диаграмма переходов описана в В.2.3.2 и диаграмма последовательностей — в С.2.14. Остальные нотации описаны в следующих трех подпунктах.

**С.3.12.1 Диаграммы классов**

Диаграммы классов определяют классы объектов, которые должны использоваться в программном обеспечении. Они основаны на более ранних схемах атрибут-связь-сущность, но адаптированы к объектно-ориентированному проектированию. Каждый класс (из которого один или более экземпляров будут использоваться в качестве объектов во время выполнения) представлен прямоугольником, а различные отношения между классами показаны линиями или стрелками. Операции или методы, предлагаемые каждым классом, и атрибуты данных каждого класса могут быть добавлены к схеме. Представляемые отношения состоят как из ссылочных отношений с указанием их кратности (экземпляр класса А может обратиться к одному или нескольким экземплярам класса В), так и из отношений специализации (класс X — уточнение класса Y) с, возможно, дополнительными методами и атрибутами. Может быть изображено множественное наследование.

**С.3.12.2 Прецеденты**

Прецеденты обеспечивают текстовое описание требуемого поведения системы в соответствии с определенным сценарием, обычно с точки зрения внешних агентов, включая пользователей системы и внешних систем. Для представления дополнительного поведения, особенно в случаях ошибочных ответов, внутри данного прецедента могут использоваться альтернативные подсценарии. Чтобы обеспечить достаточно полную спецификацию системных требований, разрабатывается набор прецедентов. Прецеденты могут быть начальной точкой для разработки более строгих моделей, таких как диаграммы последовательностей и диаграммы действий.

Диаграммы прецедентов обеспечивают пиктографическое представление системы и агентов, которые включены в прецеденты, но оно не строгое, так как для спецификации важным является только текстовое описание прецедента.

**С.3.12.3 Диаграммы действий**

Диаграммы действий показывают намеченную последовательность действий, выполняемых элементом программного обеспечения (часто объектом в объектно-ориентированном проекте), включая последовательное и итеративное поведение (некоторые аспекты очень похожи на блок-схему). Диаграммы действий однако позволяют описание параллельных действий для нескольких элементов, указывая взаимодействия между этими элементами стрелками на диаграмме. Точки синхронизации, в которых действие, прежде чем начнет выполняться, должно ожидать один или несколько входных потоков от других действий, обозначаются символом, подобным узлу сети Петри.

Литература:

ISO/IEC 19501:2005, Information technology — Open Distributed Processing — Unified Modeling Language (UML) Version 1.4.2.

### С.4 Инструменты разработки и языки программирования

#### С.4.1 Строго типизированные языки программирования

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица А.3).

Цель. Снижение вероятности ошибок путем использования языка, который компилятором обеспечивает высокий уровень проверки.

Описание. Если скомпилирован строго типизированный язык программирования, то проводится много проверок по использованию типов переменных, например, в вызовах процедур и доступе к внешним данным. Компиляция может оказаться безуспешной и будет выдано сообщение об ошибке при любом использовании типа переменных, которое не соответствует заранее установленным правилам.

Подобные языки обычно позволяют определять установленные пользователем типы данных на основе типов данных базового языка (например целое число, реальное число). Затем эти типы могут быть использованы так же, как и базовый тип. Вводятся строгие проверки, чтобы гарантировать использование правильного типа. Эти проверки проводятся для всей программы, даже если она построена из отдельных скомпилированных модулей. Данные проверки гарантируют также, что число и тип аргументов конкретной процедуры соответствуют числу и типу аргументов в ее вызове, даже если к ней обращаются из отдельно скомпилированных программных модулей.

Строго типизированные языки обычно обеспечивают другие аспекты проверенной на практике техники программного обеспечения, например, легко анализируемые структуры управления (if... then... else..., do... while... и т. п.), которые приводят к четко структурированным программам.

Типичными примерами строго типизированных языков являются Pascal, Ada и Modula 2.

Литература:

Concepts in Programming Languages. J.C. Mitchell. Cambridge University Press, 2003, ISBN 0521780985, 9780521780988.

#### С.4.2 Подмножество языка

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица А.3).

Цель. Снижение вероятности внесения программных ошибок и повышение вероятности обнаружения оставшихся ошибок.

Описание. Язык исследуется для определения программных конструкций, подверженных ошибкам либо сложных для анализа, например, при использовании методов статического анализа. После этого определяется языковое подмножество, которое исключает такие конструкции.

Литература:

Practical Experiences of Safety- and Security-Critical Technologies. P. Amey, A.J. Hilton. Ada User Journal, June, 2004.

Safer C: Developing Software for High-integrity and Safety-critical Systems. L. Hatton, McGraw-Hill, 1994, ISBN 0077076400, 9780077076405.

Requirements for programming languages in safety and security software standard B.A. Wichmann. Computer Standards and Interfaces. Vol. 14, pp. 433—441, 1992.

#### С.4.3 Сертифицированные средства и сертифицированные трансляторы

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица А.3).

Цель. Помощь разработчику на различных этапах разработки программных средств в использовании необходимых инструментальных средств, которые, где это возможно, должны быть сертифицированы с тем, чтобы обеспечить конкретную степень уверенности в корректности результатов.

Описание. Сертификацию инструментальных средств в общем случае допускается проводить независимо, как правило, в национальных органах по сертификации, по независимому набору критериев, находящемуся обычно в национальных или международных стандартах. В идеальном случае инструментальные средства, применяемые на всех стадиях разработки (спецификация, проектирование, кодирование, тестирование и оценка соответствия), и те из них, которые используются в управлении конфигурацией, должны быть сертифицированы.

В настоящее время регулярным процедурам сертификации подвергаются только компиляторы (трансляторы); сертификация проводится национальными органами по сертификации и заключается в проверке компиляторов (трансляторов) на соответствие международным стандартам, например, для языков Ada или Pascal.

Важно отметить, что сертифицированные инструментальные средства и сертифицированные трансляторы обычно сертифицируются только на соответствие стандартам на соответствующий язык или процесс. Обычно они никак не сертифицируются на соответствие стандартам по безопасности.

Литература:

The certification of software tools with respect to software standards, P. Bunyakiati et al. In IEEE International Conference on Information Reuse and Integration, IRI 2007, IEEE, 2007, ISBN 1-4244-1500-4.

Certified Testing of C Compilers for Embedded Systems. O. Morgan. In: 3rd Institution of Engineering and Technology Conference on Automotive Electronics. IEEE, 2007, ISBN 978-0-86341-815-0.

The Ada Conformity Assessment Test Suite (ACATS), version 2.5, Ada Conformity Assessment Authority, http:// . 2002.

gost.e-ecolog.ru

**С.4.4 Инструментальные средства и трансляторы. Повышение уверенности на основании опыта использования**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица А.3).

Цель. Исключение любых проблем, обусловленных ошибками транслятора, которые могут появиться во время разработки, верификации и эксплуатации программного пакета.

Описание. Транслятор используется в тех случаях, где не было доказательств ненадлежащего исполнения многих предыдущих проектов. Если отсутствует опыт эксплуатации трансляторов или в них обнаружны любые известные серьезные ошибки, то от таких транслятров следует отказаться, если только нет каких-либо других гарантий корректной работы транслятора (см. С.4.4.1).

Если в трансляторе выявлены небольшие недостатки, то соответствующие языковые конструкции фиксируются и в проектах, связанных с безопасностью, не применяются.

Другим вариантом исключения проблем, обусловленных ошибками транслятора, является ограничение языка до его общеиспользуемых конструкций.

Настоящие рекомендации основаны на опыте построения многих проектов. Доказано, что недоработанные трансляторы служат серьезным препятствием в любой программной разработке. Такие трансляторы в общем случае делают невозможной разработку программного обеспечения, связанного с безопасностью.

В настоящее время не существует методов подтверждения корректности всего транслятора или отдельных его частей.

**С.4.4.1 Сравнение исходных программ и исполнимых кодов**

Цель. Убедиться в том, что инструменты, используемые для создания образа PROM, не вносят в него никаких ошибок.

Описание. Образ PROM обратно преобразуется в совокупность «объектных» модулей. Эти «объектные» модули обратно преобразуются в файлы ассемблера, которые затем, с помощью соответсвующих методов, сравниваются с фактическими исходными файлами, используемыми первоначально для разработки PROM.

Основное преимущество данного метода состоит в том, что инструменты [компиляторы, редакторы связей (компоновщики) и т. п.], используемые для разработки образа PROM, не требуют подтверждения соответствия. Этим методом проверяют правильность преобразования исходного файла, используемого для конкретной системы, связанной с безопасностью.

Литература:

Demonstrating Equivalence of Source Code and PROM Contents. D.J. Pavey and L.A. Winsborrow. The Computer Journal. Vol. 36, No. 7, 1993.

Formal demonstration of equivalence of source code and PROM contents: an industrial example. D.J. Pavey and L.A. Winsborrow. Mathematics of Dependable Systems, Ed. C. Mitchell and V. Stavridou, Clarendon Press, 1995, ISBN 0-198534-91-4.

Assuring Correctness in a Safety Critical Software Application. L.A. Winsborrow and D.J. Pavey. High Integrity Systems. Vol. 1, No. 5, pp. 453—459, 1996.

**С.4.5 Выбор соответствующего языка программирования**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица А.3).

Цель. Обеспечение в максимальной степени требований настоящего стандарта для специального защищающего программирования, строгой типизации, структурного программирования и, возможно, суждений. Выбранный язык программирования должен обеспечить легко верифицируемый код и простые процедуры разработки, верификации и эксплуатации программ.

Описание. Язык программирования должен быть полностью и однозначно определен. Язык должен быть ориентирован на пользователя или проблему, а не на процессор или платформу. Широко используемые языки программирования или их подмножества должны быть предпочтительнее языков специального применения.

Языки программирования также должны обеспечивать:

- блочную структуру организации программ;
- проверку времени трансляции;
- проверку во время работы программы типов и границ массивов.

Язык программирования должен включать в себя:

- использование небольших и управляемых программных модулей;
- ограничение доступа к данным в конкретных программных модулях;
- определение поддиапазонов переменных;
- любые другие типы конструкции, ограничивающие ошибки.

Если операции системы безопасности зависят от ограничений реального времени, то язык программирования должен обеспечивать также обработку исключений или прерываний.

Желательно, чтобы язык программирования обеспечивался соответствующим транслятором, подходящими библиотеками с заранее созданными программными модулями, отладчиком и инструментами как для управления версиями, так и для разработки.

В настоящее время еще не ясно, будут ли объектно-ориентированные языки программирования предпочтительнее других общепринятых языков.

К свойствам, которые усложняют верификацию и поэтому должны быть исключены, относятся:

- безусловные переходы (за исключением вызовов подпрограмм);
- рекурсии;
- указатели, динамически распределяемые области памяти или любые типы динамических переменных или объектов;
- обработка прерываний на уровне исходного кода;
- множество входов или выходов в циклах, блоках или подпрограммах;
- неявная инициализация или объявление переменных;
- вариантные записи и эквивалентность;
- процедурная переменная в качестве параметра.

Языки программирования низкого уровня, в частности ассемблеры, обладают недостатками, связанными с их жесткой ориентацией на процессор машины или на определенную платформу.

Желательным свойством языка программирования является то, что его проектирование и использование должно приводить к созданию программ, выполнение которых предсказуемо. Если используется подходящий конкретный язык программирования, то в нем должно существовать подмножество, которое гарантирует, что выполнение программы предсказуемо. Это подмножество не может быть (в общем случае) статически определено, несмотря на то, что многие статические ограничения помогают гарантировать предсказуемое выполнение. Обычно это может потребовать демонстрации того, что индексы массива находятся в установленных пределах и что числовое переполнение не может возникнуть, и т. п.

Рекомендации по конкретным языкам программирования приведены в таблице С.1.

Литература:

Concepts in Programming Languages. J.C. Mitchell. Cambridge University Press, 2003, ISBN 0521780985, 9780521780988.

IEC 60880:2006, Nuclear power plants — Instrumentation and control systems important to safety — Software aspects for computer-based systems performing category A functions.

IEC 61131-3:2003, Programmable controllers — Part 3: Programming languages.

ISO/IEC 1539-1:2004, Information technology — Programming languages — Fortran — Part 1: Base language.

ISO/IEC 7185:1990, Information technology — Programming languages — Pascal.

ISO/IEC 8652:1995, Information technology — Programming languages — Ada.

ISO/IEC 9899:1999, Programming languages — C.

ISO/IEC 10206:1991, Information technology — Programming languages — Extended Pascal.

ISO/IEC 10514-1:1996, Information technology — Programming languages — Part 1: Modula-2, Base Language.

ISO/IEC 10514-3:1998, Information technology — Programming languages — Part 3: Object Oriented Modula-2.

ISO/IEC 14882:2003, Programming languages — C++.

ISO/IEC/TR 15942:2000, Information technology — Programming languages — Guide for the use of the Ada programming language in high integrity systems.

Таблица С.1 — Рекомендации по конкретным языкам программирования

| Язык программирования | УПБ1 | УПБ2 | УПБ3 | УПБ4 |
|---|---|---|---|---|
| 1 ADA | HR | HR | R | R |
| 2 ADA с подмножеством | HR | HR | HR | HR |
| 3 Java | NR | NR | NR | NR |
| 4 Java с подмножеством (без включения сборки мусора или с включением сборки мусора, которая не будет вызывать остановку прикладной программы в течение значительного промежутка времени). Руководящие указания по использованию объектно-ориентированных средств см. в приложении G. | R | R | NR | NR |
| 5 PASCAL (см. примечание 1) | HR | HR | R | R |
| 6 PASCAL с подмножеством | HR | HR | HR | HR |
| 7 FORTRAN 77 | R | R | R | R |
| 8 FORTRAN 77 с подмножеством | HR | HR | HR | HR |
| 9 C | R | — | NR | NR |
| 10 C с подмножеством и стандартом кодирования, а также использование инструментов статического анализа | HR | HR | HR | HR |

| Язык программирования | УПБ1 | УПБ2 | УПБ3 | УПБ4 |
|---|---|---|---|---|
| 11 C++ (Руководящие указания по использованию объектно-ориентированных средств см. в приложении G.) | R | — | NR | NR |
| 12 C++ с подмножеством и стандартом кодирования, а также использование инструментов статического анализа (Руководящие указания по использованию объектно-ориентированных средств см. в приложении G.) | HR | HR | HR | HR |
| 13 Ассемблер | R | R | — | — |
| 14 Ассемблер с подмножеством и стандартом кодирования | R | R | R | R |
| 15 Многоступенчатые диаграммы | R | R | R | R |
| 16 Многоступенчатая диаграмма с определенным подмножеством языка | HR | HR | HR | HR |
| 17 Диаграмма функциональных блоков | R | R | R | R |
| 18 Диаграмма функциональных блоков с определенным подмножеством языка | HR | HR | HR | HR |
| 19 Структурированный текст | R | R | R | R |
| 20 Структурированный текст с определенным подмножеством языка | HR | HR | HR | HR |
| 21 Последовательная функциональная диаграмма | R | R | R | R |
| 22 Последовательная функциональная диаграмма с определенным подмножеством языка | HR | HR | HR | HR |
| 23 Список команд | R | — | NR | NR |
| 24 Список команд с определенным подмножеством языка | HR | R | R | R |

Примечания

1 Пояснения к рекомендациям R, HR, NR см. в МЭК 61508-3, приложение А.

2 Системное программное обеспечение включает в себя операционную систему, драйверы, встроенные функции и программные модули, являющиеся частью системы. Программные средства обычно обеспечиваются системой безопасности при поставке. Подмножество языка следует выбирать очень внимательно с тем, чтобы исключить сложные структуры, которые могут образоваться в результате ошибок реализации. Следует выполнять проверки для того, чтобы убедиться в правильном использовании подмножества языка программирования.

3 Прикладная программа представляет собой программу, разработанную для конкретного безопасного применения. Во многих случаях такая программа разрабатывается конечным пользователем либо подрядчиком, ориентированным на разработку прикладных программ. В тех случаях, когда ряд языков программирования поддерживает одни и те же рекомендации, разработчику следует выбрать тот, который повсеместно используется персоналом в конкретной промышленности или отрасли. Подмножество языка программирования следует выбирать с особым вниманием, чтобы исключить сложные структуры, которые могут привести к ошибкам реализации.

4 Если конкретный язык программирования не представлен в настоящей таблице, то это не означает, что он исключен. Этот конкретный язык программирования должен соответствовать требованиям настоящего стандарта.

5 Существует ряд расширений языка Паскаль, включая свободно распространяемый Паскаль. Ссылки на Паскаль включают эти расширения.

6 Java имеет сборщик мусора времени выполнения. Подмножество Java может не иметь сборщика мусора. Некоторые реализации Java обеспечивают прогрессивную сборку мусора, которая восстанавливает свободную память в процессе выполнения программы и предотвращает выполнение остановки, когда исчерпана доступная память. Приложения жесткого реального времени не должны использовать любые средства сборки мусора.

7 Если применение языка Java требует использования интерпретатора времени выполнения для промежуточного кода Java, то интерпретатор должен рассматриваться, как часть программного обеспечения, связанного с безопасностью, и удовлетворять требованиям МЭК 61508-3.

8 О пунктах 15—24 см. [7].

**С.4.6 Автоматическая генерация программного обеспечения**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица А.2).

Цель. Автоматизировать наиболее подверженные ошибкам задачи реализации программного обеспечения.

Описание. Проект системы описывается моделью (исполнимой спецификацией) на более высоком уровне абстракции, чем традиционный исполняемый код. Модель автоматически преобразуется генератором кода в исполнимую форму. Цель состоит в том, чтобы улучшить качество программного обеспечения, устраняя подверженные ошибкам ручные задачи кодирования. Дальнейшая возможная выгода в том, что более сложные проекты могут выполняться на более высоком абстрактном уровне.

Литература:

Embedded Software Generation from System Level Design Languages, H. Yu, R. Domer, D. Gajski. In «ASP-DAC 2004: Proceedings of the ASP-Dac 2004 Asia and South Pacific Design Automation Conference, 2004», IEEE Circuits and Systems Society. IEEE, 2004, ISBN 0780381750, 9780780381759.

Transforming Process Algebra Models into UML State Machines: Bridging a Semantic Gap? M. F. van Amstel et al. In Theory and Practice of Model Transformations: First International Conference, ICMT». Ed. A. Vallecillo. Springer, 2008, ISBN 3540699260, 9783540699262.

**С.4.7 Управление тестированием и средства автоматизации**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица А.5).

Цель. Обеспечить систематический и всесторонний подход к тестированию системы и программного обеспечения.

Описание. Использование подходящих средств поддержки автоматизирует более трудоемкие и подверженные ошибкам задачи при разработке системы и дает возможность систематического подхода к управлению тестированием. Доступность поддержки обеспечивает более всесторонний подход и к обычному и регрессионному тестированию.

Литература:

Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing. R. Black, John Wiley and Sons, 2002, ISBN 0471223980, 9780471223986.

**С.5 Верификация и модификация**

**С.5.1 Вероятностное тестирование**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблицы А.5, С.15, А.7 и С.17).

Цель. Получение количественных показателей надежности исследуемой программы.

Описание. Количественные показатели могут быть получены с учетом относительных уровней доверия и значимости и должны иметь следующий вид:

- вероятность ошибки при запросе;
- вероятность ошибки в течение определенного периода времени;
- вероятность последствий ошибки.

Из этих показателей могут быть получены другие показатели, например:

- вероятность безошибочной работы;
- вероятность живучести;
- доступность;
- MTBF или частота отказов;
- вероятность безопасного исполнения.

Вероятностные соображения основываются либо на статистических испытаниях, либо на опыте эксплуатации. Обычно количество тестовых примеров или наблюдаемых практических примеров очень велико. Обычно тестирование в режиме запросов занимает значительно меньше времени, чем в непрерывном режиме работы.

Для формирования входных данных тестирования и управления выходными данными тестирования обычно используются инструменты автоматического тестирования. Крупные тесты прогоняются на больших центральных компьютерах с имитацией соответствующей периферии. Тестируемые данные выбираются с учетом как систематических, так и случайных ошибок аппаратных средств. Например, общее управление тестированием гарантирует профиль тестируемых данных, тогда как случайный выбор тестируемых данных может управлять отдельными тестовыми примерами более детально.

Как указано выше, индивидуальные средства для тестирования, выполнение тестирования и управление тестированием определяются детализированными целями тестирования. Другие важные условия задаются математическими предпосылками, которые должны быть соблюдены, если оценка тестирования удовлетворяет заданным целям тестирования.

Из опыта эксплуатации также могут быть получены вероятностные характеристики поведения любого тестируемого объекта. Если соблюдаются одинаковые условия, то к оценкам результатов тестирования может быть применен одинаковый математический аппарат.

Используя эти методы, достаточно сложно продемонстрировать на практике сверхвысокие уровни надежности.

Литература:

A discussion of statistical testing on a safety-related application. S. Kuball, J. H. R. May, Proc IMechE. Vol. 221. Part O: J. Risk and Reliability, Institution of Mechanical Engineers, 2007.

Estimating the Probability of Failure when Testing Reveals No Failures, W.K. Miller, L.J. Morell et al. IEEE Transactions on Software Engineering, Vol. 18, No.1, pp. 33–43, January 1992.

Reliability estimation from appropriate testing of plant protection software. J. May, G. Hughes, A.D. Lunn. IEE Software Engineering Journal. Vol.10. No. 6. pp. 206—218, Nov., 1995 (ISSN: 0268—6961).

Validation of ultra high dependability for software based systems, B. Littlewood and L. Strigini. Comm. ACM 36 (11), 69—80, 1993.

**С.5.2 Регистрация и анализ данных**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблицы А.5 и А.8).

Цель. Документирование всех данных, решений и разумного обоснования программных проектов в целях обеспечения верификации, подтверждения соответствия, оценки и эксплуатации.

Описание. В процессе всего проектирования разрабатывается подробная документация, в которую входят:

- тестирование, выполняемое на каждом программном модуле;
- решения и их разумные обоснования;
- проблемы и их решения.

В процессе и по завершении проекта эта документация может быть проанализирована на наличие широкого набора информации. В частности, такая информация, использовавшаяся в качестве обоснования при принятии конкретных решений в процессе разработки проекта и очень важная для обслуживания вычислительных систем, не всегда известна инженерам по эксплуатации.

Литература:

Dependability of Critical Computer Systems 2. F.J. Redmill, Elsevier Applied Science, 1989, ISBN 1851663819, 9781851663811.

**С.5.3 Тестирование интерфейса**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица А.5).

Цель. Обнаружение ошибок в интерфейсах подпрограмм.

Описание. Возможны несколько уровней детализации или полноты тестирования. К наиболее важным уровням относится тестирование:

- всех интерфейсных переменных с их предельными значениями;
- всех отдельных интерфейсных переменных с их предельными значениями с другими интерфейсными переменными с их нормальными значениями;
- всех значений предметной области каждой интерфейсной переменной с другими интерфейсными переменными с их нормальными значениями;
- всех значений всех переменных в разных комбинациях (возможно только для небольших интерфейсов);
- при специфицированных условиях тестирования, уместных для каждого вызова каждой подпрограммы.

Эти тестирования особенно важны, если интерфейсы не имеют возможности обнаруживать неправильные значения параметров. Такие тестирования также важны при генерации новых конфигураций ранее существовавших подпрограмм.

**С.5.4 Анализ граничных значений**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблицы В.2, В.3 и В.8).

Цель. Обнаружение программных ошибок при предельных и граничных значениях параметров.

Описание. Предметная входная область программы разделяется на множество входных классов в соответствии с отношениями эквивалентности (см. С.5.7). Тестирование должно охватывать границы и экстремальные значения классов. Данное тестирование проверяет совпадение границы предметной входной области в спецификации с границами, установленными программой. Использование нулевого значения как в непосредственных, так и в косвенных преобразованиях часто приводит к ошибкам. Особого внимания требуют:

- нулевой делитель;
- знаки пробела ASCII;
- пустой стек или элемент списка;
- заполненная матрица;
- ввод нулевой таблицы.

Обычно границы входных значений напрямую соотносятся с границами выходных значений. Для установления выходных параметров в их предельные значения необходимо записывать специальные тестовые примеры. Следует также по возможности рассмотреть спецификацию такого тестового примера, который побуждает выходное значение превысить установленные спецификацией граничные значения.

Если выходные значения являются последовательностью данных, например, таблица, то особое внимание следует уделить первому и последнему элементам, а также спискам, содержащим либо ни одного, либо один, либо два элемента.

Литература:

The Art of Software Testing, second edition. G.J. Myers, T. Badgett, T.M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124.

**С.5.5 Предположение ошибок**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблицы В.2 и В.8).

Цель. Исключение ошибки программирования.

Описание. Опыт тестирования и интуиция в сочетании со сведениями и заинтересованностью относительно тестируемой системы могут добавить некоторые неклассифицированные тестовые примеры к набору заданных тестовых примеров.

Специальные значения или комбинации значений могут быть подвержены ошибкам. Некоторые вызывающие интерес тестовые примеры могут быть получены из анализа контрольных списков. Следует также рассмотреть, является ли система достаточно устойчивой. Например, следует ли нажимать клавиши на передней панели слишком быстро или слишком часто. Что произойдет, если две клавиши нажать одновременно.

Литература:

The Art of Software Testing, second edition. G.J. Myers, T. Badgett, T.M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124.

**С.5.6 Введение ошибок**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица В.2).

Цель. Подтверждение адекватности набора тестовых примеров.

Описание. Некоторые известные типы ошибок вводятся (подсеиваются) в программу, и программа выполняется с тестовыми примерами в режиме тестирования. При обнаружении только некоторых подсеянных ошибок тестовый пример становится неадекватным. Отношение числа найденных подсеянных ошибок к общему числу подсеянных ошибок оценивается как отношение числа найденных реальных ошибок к общему числу реальных ошибок. Это дает возможность оценить количество остаточных ошибок, и тем самым, остальную работу по тестированию.

$$\frac{\text{Найденные подсеянные ошибки}}{\text{Общее число подсеянных ошибок}} = \frac{\text{Найденные реальные ошибки}}{\text{Общее число реальных ошибок}}.$$

Обнаружение всех подсеянных ошибок может указывать либо на адекватность тестового примера, либо на то, что подсеянные ошибки было слишком легко найти. Ограничениями данного метода являются: порядок получения любых полезных результатов, типы ошибок. Также необходимо, чтобы позиции подсеивания отражали статистическое распределение реальных ошибок.

Литература:

Software Fault Injection: Inoculating Programs Against Errors. J. Voas, G. McGraw. Wiley Computer Pub., 1998, ISBN 0471183814, 9780471183815.

Faults, Injection Methods, and Fault Attacks. Chong Hee Kim, Jean-Jacques Quisquater, IEEE Design and Test of Computers, vol. 24, No. 6, pp. 544—545, Nov., 2007.

Fault seeding for software reliability model validation. A. Pasquini, E. De Agostino. Control Engineering Practice, Volume 3, Issue 7, July 1995. Elsevier Science Ltd.

**С.5.7 Разделение входных данных на классы эквивалентности**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблицы В.2 и В.3).

Цель. Адекватное тестирование программных средств с использованием минимума тестируемых данных. Тестируемые данные образуются путем выбора частей входных данных предметной области, требующихся для анализа программных средств.

Описание. Данный метод тестирования основывается на отношении эквивалентности входных данных, определяющем разбиение входных данных предметной области.

Тестовые примеры выбираются в целях охвата всех предварительно специфицированных разбиений. Из каждого класса эквивалентности выбирается по меньшей мере один тестовый пример.

Существуют следующие основные возможности разбиения входных данных:

- классы эквивалентности, образованные из спецификации, — интерпретация спецификации может быть ориентирована либо на входные значения, например, выбранные значения считаются одинаковыми, либо ориентирована на выходные значения, например, набор значений приводит к одному и тому же функциональному результату;
- классы эквивалентности, образованные в соответствии с внутренней структурой программы, — результаты класса эквивалентности определяются из статического анализа программ, например, набор значений обрабатывается одним и тем же способом.

Литература:

The Art of Software Testing, second edition. G.J. Myers, T. Badgett, T.M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124.

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799.

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574.

Static Analysis and Software Assurance. D. Wagner, Lecture Notes in Computer Science, Volume 2126/2001, Springer, 2001, ISBN 978-3-540-42314-09.

**С.5.8 Структурное тестирование**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица В.2).

Цель. Применение тестов, анализирующих определенные подмножества структуры программы.

Описание. На основе анализа программы выбирается набор входных данных так, чтобы мог быть проанализирован достаточно большой (часто с заранее заданным значением) процент программных кодов. Меры охвата программы, в зависимости от степени требуемой строгости, могут быть различными. В любом случае целью должны быть 100 % выбранной метрики охвата; если невозможно достигнуть 100%-ного охвата, то причины, почему 100%-ный охват не может быть достигнут, должны быть документально оформлены в отчете о тестировании (например если возникает аппаратная проблема, то может быть введен только код защиты). Первые четыре метода в следующем списке специально упомянуты в рекомендациях в таблице В.2 МЭК 61508-3 и широко поддерживаются инструментами тестирования; оставшиеся методы могут быть также рассмотрены:

- охват точек входа (граф вызовов) — гарантирует, что каждая подпрограмма (стандартная подпрограмма или функция) по крайней мере однажды должна быть вызвана (это — наименее строгое структурное измерение охвата).

Примечание — В объектно-ориентированных языках может быть несколько подпрограмм с одним и тем же именем, которые применяются к различным вариантам полиморфизма (переопределяющие подпрограммы), которые могут вызываться динамической диспетчеризацией. В таких случаях должна быть протестирована каждая такая переопределенная подпрограмма;

- операторы — гарантирует, что все операторы в коде были выполнены по крайней мере однажды;
- условные переходы — должны быть проверены обе ветви каждого условного перехода. Это может оказаться нецелесообразным для некоторых типов кодов защиты;
- составные условия — анализируется каждое условие в составном условном переходе (связанное оператором И/ИЛИ). См. MCDC (охват условного модифицированного решения, документ DO178B);
- LCSAJ — последовательность линейного кода и переходов представляет собой любую линейную последовательность закодированных утверждений, включая условные утверждения, заканчивающиеся переходом. Многие потенциальные подпоследовательности могут оказаться невыполнимыми из-за ограничений, которые налагаются на входные данные в результате выполнения предыдущего кода;
- поток данных — выполняющиеся последовательности выбираются на основе используемых данных; например, последовательность, где одна и та же переменная и записывается, и считывается;
- базовая последовательность — одна из минимального набора конечных последовательностей от начала до конца, когда все дуги охвачены (перекрывающиеся комбинации последовательностей в этом базовом наборе могут сформировать любую последовательность через эту часть программы). Тесты всех базовых последовательностей показали свою эффективность при обнаружении ошибок.

Литература:

The Art of Software Testing, second edition. G. J. Myers, T. Badgett, T.M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124.

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799.

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574.

RTCA, Inc. document DO-178B and EUROCAE document ED-12B, Software Considerations in Airborne Systems and Equipment Certification, dated December 1, 1992.

**С.5.9 Анализ потоков управления**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица В.8).

Цель. Обнаружение низкокачественных и потенциально некорректных структур программ.

Описание. Анализ потока управления представляет собой метод статического тестирования для нахождения подозреваемых областей программы, которые не соответствуют оправдавшей себя практике программирования. Программа анализируется, формируя направленный граф, который может быть проанализирован на наличие:

- недоступных фрагментов программы, например, безусловных переходов, которые делают фрагменты программы недостижимыми;
- запутанных кодов. Хорошо структурированный код имеет управляющий граф, допускающий сокращение путем последовательного сокращения графа до одного узла. В отличие от этого плохо структурированный код может быть сокращен только до группы, состоящей из нескольких узлов.

Литература:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799.

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574.

**С.5.10 Анализ потока данных**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица В.8).

Цель. Обнаружение низкокачественных и потенциально некорректных структур программ.

Описание. Анализ потока данных представляет собой метод статического тестирования, объединяющий информацию, полученную из анализа потока управления, с информацией о том, какие переменные считываются или записываются в различных частях кода. Данный метод может проверять:

- переменные, которые могут быть считаны до присвоения им значений. Такую ситуацию можно исключить, если всегда присваивать значение при объявлении новой переменной;
- переменные, записанные несколько раз, но не считанные. Такая ситуация может указывать на пропущенный код;
- переменные, которые записаны, но никогда не считываются. Такая ситуация может указывать избыточный код.

Аномальный поток данных не всегда непосредственно соответствует программным ошибкам, но если аномалии исключены, то маловероятно, что код будет содержать ошибки.

Литература:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799.

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574.

**С.5.11 Тестирование на символьном уровне**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица В.8).

Цель. Показать соответствие между исходным кодом и спецификацией.

Описание. Переменные программы оцениваются после замены во всех операторах присваивания левой его части на правую. Условные ветви и циклы преобразуются в булевские выражения. Окончательный результат представляет собой символьное выражение для каждой переменной программы. Это выражение является формулой для значения, которое программа вычислила бы, если задать реальные данные. Оно может быть проверено относительно предполагаемого выражения.

Такое использование символьного выполнения является систематическим способом генерирации тестовых данных для ветвей логики программы. Символьное средство выполнения может быть включено в интегрированный комплекс инструментальных средств для выполнения тестирования и анализа кода элемента программного обеспечения.

Литература:

Using symbolic execution for verifying safety-critical systems. A. Coen-Porisini, G. Denaro, C. Ghezzi, M. Pezzé. Proceedings of the 8th European software engineering conference, and 9th ACM SIGSOFT international symposium on Foundations of software engineering. ACM, 2001, ISBN:1-58113-390-1.

Using symbolic execution to guide test generation. G. Lee, J. Morris, K. Parker, G. Bundell, P.Lam. In Software Testing, Verification and Reliability, vol. 15, No. 1, 2005. John Wiley & Sons, Ltd.

**С.5.12 Формальное доказательство (верификация)**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблицы А.5 и А.9).

Цель. Доказать правильность программы относительно некоторой абстрактной модели программы, используя теоретические и математические модели и правила.

Описание. Тестирование — распространенный способ исследовать правильность программы. Однако исчерпывающее тестирование обычно недостижимо ввиду сложности программ, имеющих практическое значение, поэтому таким способом может быть исследована только часть возможного поведения программы. Напротив, формальная верификация применяет математические операции к математическому представлению программы, чтобы установить, что программа ведет себя, как определено для всех возможных входных данных.

Для формальной верификации системы требуется абстрактная модель программы и ее заданное поведение (то есть спецификация), представленные на формальном языке. Спецификация может быть полной или она может быть ограничена определенными свойствами программы:

- свойствами функциональной корректности, то есть программа должна продемонстрировать определенную функциональность;
- свойствами безопасности (то есть неправильное поведение никогда не будет происходить), и живучести (то есть в конечном счете будет вести себя правильно);
- синхронизирующими свойствами, то есть события, реализующие поведение, произойдут в определенное время.

Результатом формальной верификации является строгий вывод о том, что абстрактная модель программы корректна относительно спецификации для всех возможных входных данных, то есть модель удовлетворяет заданным свойствам.

Однако правильность модели непосредственно не доказывает правильность фактической программы, поэтому далее необходим шаг, который должен показать, что модель — точная абстракция фактической программы для моделируемых свойств. Некоторые свойства программы, представляющие практический интерес, не могут быть формально описаны (например большинство проблем синхронизации и планирования или субъективные свойства, такие как «ясный и простой» пользовательский интерфейс, или, на самом деле, любое свойство или цель проекта, которые не могут быть легко представлены на формальном языке). Поэтому формальная верификация не полностью заменяет моделирование и тестирование, но зато она дополняет эти методы, обеспечивая их доказательством корректности работы программы для всех входных данных. Формальная верификация может гарантировать корректность абстрактной модели программы, а тестирование гарантирует, что фактическая программа ведет себя, как ожидалось.

Использование формальной верификации на стадии проектирования может значительно уменьшить время разработки, обнаруживая существенные ошибки и упущения на ранних стадиях проектирования, и таким образом сократить время, необходимое на итерации между проектированием и тестированием.

Ряд формальных методов, применяющихся на практике, описаны в С.2.4: например, CCS, CSP, HOL, LOTOS, OBJ, временная логика, VDM и Z.

**С.5.12.1 Проверка модели**

Проверка модели — метод для формальной верификации реактивных и конкурентных систем. Задавая конечное состояние структуры, которая описывает поведение системы, проверяется свойство, записанное в виде формулы во временно́й логике, удовлетворяет оно или нет структуре. Для автоматического и исчерпывающего прохода всех состояний структуры используются эффективные алгоритмы (например SPIN, SMV и UPPAAL). Если свойство не удовлетворяется, то генерируется контрпример. Эта процедура показывает, как свойства нарушаются в структуре, и содержит очень полезную информацию для исследования системы. Метод проверки модели может обнаружить «глубокие ошибки», которые могут быть не обнаружены при обычном контроле и тестировании.

Необходимо отметить, что метод проверки модели полезен при анализе нюансов сложной структуры. Это может быть полезно для некоторых приложений с низким УПБ, но необходимо быть очень внимательным, если нюансы сложной структуры существуют в приложениях с высоким УПБ.

Литература:

Is Proof More Cost-Effective Than Testing? S. King, R. Chapman, J. Hammond, A. Pryor. IEEE Transactions on Software Engineering, vol. 26, No. 8, August 2000.

Model Checking. E.M. Clarke, O. Grumberg, and D.A. Peled. MIT Press, 1999, ISBN 0262032708, 9780262032704.

Systems and Software Verification: Model-Checking Techniques and Tools. B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, Ph. Schnoebelen, and P. Mckenzie, Springer, 2001, ISBN 3-540-41523-8.

Logic in Computer Science: Modelling and Reasoning about Systems. M. Huth and M. Ryan. Cambridge University Press, 2000, ISBN 0521652006, 0521656028.

The Spin Model Checker: Primer and Reference Manual. G.J. Holzmann. Addison-Wesley, 2003, ISBN 0321228626, 9780321228628.

**С.5.12.2** (Не используется.)

**С.5.13 Метрики сложности программного обеспечения**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблицы А.9 и С.19).

Цель. Прогнозирование характеристик программ исходя из свойств самих программ или их разработки, либо предысторий тестирования.

Описание. Данные методы оценивают некоторые структурные свойства программных средств и их отношения к требуемым характеристикам, например, надежность или сложность. Для оценки большинства средств требуются программные инструменты. Некоторые применяющиеся метрики перечислены ниже:

- теоретическая сложность графа. Эта метрика может быть применена на раннем этапе жизненного цикла для оценки компромиссных решений и основана на величине сложности графа управления программы, представленной ее цикломатическим числом;
- число способов активизации некоторых программных модулей (доступность) — чем больше программных модулей может быть доступно, тем должна быть большая вероятность их отладки;
- теория метрик Холстеда. При помощи этих средств вычисляют длину программы путем подсчета количества операторов и операндов; данная метрика дает меру сложности и размеры, которые формируют основу для сравнений при оценке будущих разрабатываемых ресурсов;
- число входов и выходов на программный модуль. Сведе́ние к минимуму числа точек входов/выходов является ключевой особенностью методов структурного проектирования и программирования.

Литература:

Metrics and Models in Software Quality Engineering. S.H. Kan. Addison-Wesley, 2003, ISBN 0201729156, 9780201729153.

**С.5.14 Формальные проверки**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица В.8).

Цель. Обнаружение ошибок в элементе программного обеспечения.

Описание. Формальный контроль — структурированный процесс проверки материалов о программном обеспечении, который выполняется коллегами разработчика этого материала в целях найти ошибки и помочь разработчику улучшить материал. Разработчик не должен принимать участие в процессе контроля, кроме информирования проверяющих на этапе их ознакомления с материалами. Формальные проверки могут быть выполнены для конкретных элементов программного обеспечения, созданных на любой стадии жизненного цикла разработки программного обеспечения.

Проверяющие должны ознакомиться с проверяемыми материалами до выполнения проверки. Роли проверяющих в процессе проверки должны быть ясно определены. Должна быть подготовлена программа проверки. Должны быть определены входные и выходные критерии, основанные на требуемых характеристиках элемента программного обеспечения. Входные критерии — это такие критерии или требования, которые должны быть удовлетворены до выполнения проверки. Выходные критерии — это такие критерии или требования, которые должны быть удовлетворены, чтобы считать процесс проверки завершенным успешно.

В процессе проверки ее результаты должны быть формально зафиксированы модератором, роль которого должна упростить проверку. По результатам всеми проверяющими должно быть достигнуто согласие. Ошибки должны быть разделены на: a) требующие исправления до принятия элемента программного обеспечения и b) требующие исправления к заданному моменту времени или этапу. После завершения проверки выявленные ошибки должны быть переданы разработчику для последующего исправления. В зависимости от количества и контекста выявленных ошибок модератор может решить вопрос о необходимости повторной проверки материалов о программном обеспечении.

Литература:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799.

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574.

The Art of Software Testing, second edition. G. J. Myers, T. Badgett, T. M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124.

Fagan, M. Design and Code Inspections to Reduce Errors in Program Development. IBM Systems Journal 15, 3 (1976): 182—211.

**C.5.15 Сквозной контроль (программного обеспечения)**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица В.8).

Цель. Обнаружение несоответствий между спецификацией и реализацией.

Описание. Сквозной контроль является неформальным методом, выполняемым разработчиком элемента программного обеспечения в присутствии его коллег в целях обнаружения ошибок в элементе программного обеспечения. Он может быть выполнен для конкретных элементов программного обеспечения, созданных на любой стадии жизненного цикла разработки программного обеспечения.

Чтобы гарантировать, что система, связанная с безопасностью, соответствует требованиям, заданным в спецификации, необходимо исследовать и оценить заданные в спецификации функции системы, связанной с безопасностью. Любые сомнения, связанные с реализацией и использованием создаваемой системы, документально оформляются в целях их дальнейшего решения. В отличие от формальной проверки в процедуре сквозного контроля разработчик принимает активное участие.

Литература:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799.

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574.

The Art of Software Testing, second edition. G. J. Myers, T. Badgett, T. M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124.

**C.5.16 Анализ проекта**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица В.8).

Цель. Выявление дефектов в проекте программного обеспечения.

Описание. Под анализом проекта понимается формальное, документально оформленное, всестороннее и систематическое исследование проекта программного обеспечения в целях оценки требований к проекту и возможности проекта удовлетворить этим требованиям, а также для определения проблем и предложений по их решению.

Анализ проекта является средством оценки соответствия состояния проекта входным требованиям, а также средством идентификации возможностей для его дальнейшего усовершенствования. Даже если разработка проекта на этапах жизненного цикла выполняется успешно и основные конкретные проектные требования удовлетворены, то анализ проекта должен быть выполнен, чтобы исследовать все интерфейсные аспекты; гарантировать, что проект может быть верифицирован, чтобы быть уверенным, что он удовлетворяет проектным требованиям; и гарантировать, что проект в наибольшей степени удовлетворяет требованиям безопасности. Такой анализ, прежде всего, предназначен для проверки результатов работы разработчиков и должен рассматриваться как действия по подтверждению и совершенствованию этих результатов.

Чтобы обнаружить неправильное поведение программного обеспечения, такое как непредвиденные последовательности выполнения операторов или логики, непреднамеренные выходы, неправильная синхронизация, нежелательные действия, может использоваться строгий метод проверки, такой как «анализ скрытых схем».

Литература:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799.

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574.

The Art of Software Testing, second edition. G. J. Myers, T. Badgett, T. M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124.

IEC 61160:2005, Design review.

Space Product Assurance, Sneak analysis — Part 2: Clue list. ECSS-Q-40-04A Part 2. ESA Publications Division, Noordwijk, 1997, ISSN 1028-396X.

gost.e-ecolog.ru

**C.5.17 Макетирование/анимация**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблицы В.3 и В.5).

Цель. Проверка возможности реализации системы при наличии заданных ограничений. Увязка интерпретации разработчика спецификации системы с ее потребителем для исключения непонимания между ними.

Описание. Выделяются подмножество системных функций, ограничения и требования к рабочим параметрам. С помощью инструментов высокого уровня строится макет. На данном этапе не требуется рассматривать ограничения, например, используемый компьютер, язык реализации, объем программ, обслуживание, надежность и доступность. Макет оценивается по критериям потребителя, и системные требования могут быть модифицированы в свете этой оценки.

Литература:

Software Engineering for Real-time Systems. J.E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205.

**С.5.18 Моделирование процесса**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблицы А.7, С.7, В.3 и С.13).

Цель. Тестирование функции программной системы вместе с ее интерфейсами во внешнем окружении, не допуская модификации реального окружения.

Описание. Создание системы только для целей тестирования, имитирующей поведение управляемого оборудования (УО).

Имитация может осуществляться только программными средствами либо сочетанием программных и аппаратных средств. Она должна:

- обеспечить входные данные, эквивалентные тем, которые реализуются на реальной установке УО;
- реагировать на выходные результаты тестирования программных средств способом, точно отражающим объект управления;
- иметь возможность для оператора вводить входные данные, чтобы обеспечить любые возмущения, с которыми должна справиться тестируемая система.

Когда тестируется программное обеспечение, то моделируются заданные аппаратные средства с их входными и выходными данными.

Литература:

EmStar: An Environment for Developing Wireless Embedded Systems Software. J. Elson et al. gost.e-ecolog.ru
edu/TechReports/9_emstar.pdf.

A hardware-software co-simulator for embedded system design and debugging. A. Ghosh et al. In Proceedings of the IFIP International Conference on Computer Hardware Description Languages and Their Applications, IFIP International Conference on Very Large Scale Integration, 1995. IEEE, 1995, ISBN 4930813670, 9784930813671.

**С.5.19 Требования к реализации**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица В.6).

Цель. Установление демонстрируемых требований к функционированию системы программных средств.

Описание. Выполняется анализ как системы, так и спецификаций требований программного обеспечения в целях спецификации всех общих и конкретных, явных и неявных требований к функционированию.

Каждое требование к функционированию анализируется по очереди для того, чтобы определить:

- критерии успешности результата, который следует получить;
- возможность получения меры критерия успешности;
- потенциальную точность таких результатов измерения;
- этапы проектирования, на которых эти результаты измерения могут быть оценены, и
- этапы проектирования, на которых могут быть получены эти результаты измерений.

Затем анализируется целесообразность каждого требования к функционированию для получения списка требований к функционированию, критериев успешности результата и возможных результатов измерений. Основными целями являются:

- связь каждого требования к функционированию по крайней мере с одной мерой;
- выбор (где это возможно) точных и эффективных мер, которые могут быть использованы на самых ранних стадиях разработки;
- спецификация важных и факультативных требований к функционированию и критериев успешности результата;
- использование (по возможности) преимуществ применения одной меры для нескольких требований к функционированию.

Литература:

Software Engineering for Real-time Systems. J.E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205.

**С.5.20 Моделирование реализации**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблицы В.2 и В.5).

Цель. Гарантировать, что рабочая производительность системы достаточна для удовлетворения специфицированных требований.

Описание. Спецификация требований включает в себя требования к пропускной способности и реакции конкретных функций, возможно, объединенных с ограничениями на использование общих системных ресурсов. Предложенный проект системы сравнивается с установленными требованиями путем:

- создания модели процессов системы и их взаимодействий;
- определения используемых каждым процессом ресурсов (время процессора, полоса пропускания канала связи, объем памяти и т. п.);
- определения распределения запросов, выдаваемых системе при средних и наихудших условиях;
- вычисления средних и для наихудших случаев значений величин пропускной способности и времени ответа для конкретных функций системы.

Для простых систем может оказаться достаточным аналитическое решение, тогда как для более сложных систем более подходящим для получения точных результатов является создание модели системы.

Перед детальным моделированием может быть использована более простая проверка «бюджета ресурсов», которая суммирует требования к ресурсам всех процессов. Если сумма этих требований к системе превышает возможности спроектированной системы, проект считается нереализуемым. Даже в случае, если проект проходит эту простую проверку, моделирование выполнения может показать, что слишком большие задержки и времена ответов происходят из-за недостатка ресурсов. Для исключения такой ситуации инженеры часто проектируют системы, использующие только часть (например 50 %) общих ресурсов для уменьшения вероятности нехватки ресурсов.

Литература:

Software Engineering for Real-time Systems. J.E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205.

**С.5.21 Проверка на критические нагрузки/стресс-тестирование**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица В.6).

Цель. Подвергнуть тестируемый объект исключительно высокой нагрузке, чтобы показать, что тестируемый объект будет легко выдерживать нормальную рабочую нагрузку.

Описание. Существует множество тестов для проверки на критические нагрузки или стресс-тестирование, например:

- если работа объекта происходит в режиме упорядоченного опроса, то объект тестирования подвергается в единицу времени гораздо большим входным изменениям, чем при нормальных условиях;
- если работа объекта происходит по запросам, то число запросов в единицу времени для тестируемого объекта увеличивается относительно нормальных условий;
- если объем базы данных играет важную роль, то этот объем увеличивается относительно ее объема при нормальных условиях;
- имеющие решающее влияние устройства настраиваются на свои максимальные скорости или самые малые скорости соответственно;
- для экстремальных тестов все факторы, имеющие решающее влияние, по возможности вводятся одновременно в граничные условия.

Для указанных выше тестов может быть оценено поведение во времени тестируемого объекта. Можно также исследовать изменения нагрузки и проверить размер внутренних буферов или динамических переменных, стеков и т. п.

**С.5.22 Ограничения на время ответа и объем памяти**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица В.6).

Цель. Обеспечение соответствия системы требованиям к параметрам времени и памяти.

Описание. Спецификация требований к системе и программному обеспечению включает в себя требования к памяти и времени выполнения системой конкретных функций, возможно, объединенных с ограничениями на использование общих системных ресурсов.

Данный метод выполняется для определения распределения запросов при средних и наихудших условиях. Рассматриваемый метод требует оценки используемых ресурсов и затраченного времени каждой функцией системы. Такие оценки могут быть получены различными способами, например, сравнением с существующей системой или макетированием и дальнейшим сравнением времени реакции с критическими системами.

**С.5.23 Анализ влияния**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица А.8).

Цель. Определение влияния, изменяющего или расширяющего программную систему, которому могут подвергаться также и другие программные модули в данной программной системе, а также другие системы.

Описание. Перед выполнением модификации или расширением программного обеспечения следует определить влияние модификаций или расширений на программное обеспечение, а также определить, на какие программные системы и программные модули это повлияет.

Далее принимается решение о повторной верификации программной системы. Это зависит от числа подвергнувшихся воздействию программных модулей, их критичности и характера изменений. Возможными решениями могут быть:

- повторная проверка только изменений программного модуля;
- повторная проверка всех подвергнувшихся воздействию программных модулей;

- повторная проверка всей системы.

Литература:

Requirements Engineering. E. Hull, K. Jackson, J. Dick. Springer, 2005, ISBN 1852338792, 9781852338794.

**С.5.24 Управление конфигурацией программного обеспечения**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица А.8).

Цель. Обеспечение согласованности результатов работы групп поставщиков проекта, а также изменений в этих поставках. В общем случае управление конфигурацией применимо к разработке как аппаратных, так и программных средств.

Описание. Управление конфигурацией программных средств представляет собой метод, используемый в течение всей разработки (см. МЭК 61508-3, пункт 6.2.3). В сущности он требует документального оформления разработки каждой версии, каждой значимой ее поставки и каждой взаимосвязи между различными версиями разработки различных поставщиков. Полученная документация позволяет разработчику определять, как влияет на другие поставки изменение в первой поставке (особенно одного из его элементов). В частности, системы или подсистемы могут надежно компоноваться (конфигурироваться) из согласованных наборов версий элементов.

Литература:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006. ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574.

Software Configuration Management: Coordination for Team Productivity. W.A. Babich. Addison-Wesley, 1986, ISBN 0201101610, 9780201101614.

CMMI: guidelines for process integration and product improvement, Mary Beth Chrissis, Mike Konrad, Sandy Shrum, Addison-Wesley, 2003, ISBN 0321154967, 9780321154965.

**С.5.25 Регрессионное подтверждение соответствия**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица А.8).

Цель. Гарантировать, что обоснованные выводы сделаны из регрессионного тестирования.

Описание. Полное регрессионное тестирование большой или сложной системы обычно требует больших усилий и ресурсов. По возможности желательно ограничить регрессионное тестирование, охватив только системные аспекты, представляющие в данный момент основной интерес для разрабатываемой системы. В таком частичном регрессионном тестировании важно иметь четкое понимание области применения этого частичного тестирования и сделать строго обоснованные выводы о тестируемом состоянии системы.

Литература:

Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing. R. Black, John Wiley and Sons, 2002, ISBN 0471223980, 9780471223986.

**С.5.26 Анимация спецификации и проектирования**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица А.9).

Цель. Проводить верификацию программного обеспечения посредством систематической проверки спецификации.

Описание. Проверяется представление программного обеспечения (спецификация или описание проекта), являющееся более абстрактным, чем исполняемый код, чтобы определить поведение возможного исполнимого программного обеспечения. Проверка до некоторой степени автоматизирована (в зависимости от возможностей, по своей природе и уровнем абстракции представления), чтобы промоделировать поведение и получить результаты исполнимого программного обеспечения. Одним из результатов этого подхода являются сгенерированные тесты (или «оракулы»), которые могут быть позже применены к исполнимому программному обеспечению, таким образом, автоматизируя, в некоторой степени, процесс тестирования. Другим результатом является анимация пользовательского интерфейса для того, чтобы конечные пользователи, не являющиеся техническими специалистами, смогли подробно разобраться в спецификации, с которой будут работать разработчики программного обеспечения, что обеспечивает ценный метод взаимодействия между этими двумя группами.

Литература:

Supporting the Software Testing Process through Specification Animation. T. Miller, P. Strooper. In Proceedings of the First International Conference on Software Engineering and Formal Methods (SEFM'03), ed. P. Lindsay. IEEE Computer Society, IEEE Computer Society, 2003, ISBN 0769519490, 9780769519494.

B model animation for external verification. H. Waeselynck, S. Behnia, In Proceedings of the Second International Conference on Formal Engineering Methods, 1998. IEEE Computer Society, 1998, ISBN 0-8186-9198-0.

**С.5.27 Тестирование, основанное на модели (генерация тестов)**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица А.5).

Цель. Обеспечить эффективную автоматическую генерацию тестовых примеров из моделей системы и создавать наборы тестов с высокой воспризводимостью.

Описание. Метод тестирования, основанный на модели (MBT), использует подход «черного ящика», в котором общие задачи тестирования, такие как генерация тестовой комбинации (TCG) и оценка результатов тестирова-

ния, основаны на модели тестируемой (прикладной) системы (SUT). Как правило, но не всегда, данные о системе и поведение пользователя смоделированы с использованием методов конечных автоматов, марковских процессов, таблиц решений и т. п. (El-Far, 2001). Кроме того, тестирование, основанное на модели, может быть объединено с измерением тестового охвата на уровне исходного кода, а функциональные модели могут быть основаны на существующем исходном коде.

Тестирование, основанное на модели, обеспечивает автоматическую генерацию эффективных контрольных примеров/процедур, используя модели системных требований и заданной функциональности (SoftwareTech, 2009).

Так как тестирование — очень дорогой процесс, существует большой спрос на автоматические средства генерации тестов. Поэтому направление тестирования, основанное на модели, в настоящий момент очень активно исследуется и создается большое число доступных средств генерации тестов (TCG). Эти средства обычно формируют тестовый набор из модели поведения системы, гарантируя при этом, что требования диагностического охвата будут удовлетворены.

Модель является абстрактным частичным представлением требуемого поведения тестируемой системы. Из этой модели формируются модели тестов, создавая абстрактный тестовый набор. Из этого абстрактного тестового набора выводятся контрольные примеры и проверяют систему, кроме того, эти контрольные примеры могут проверять также и модель системы. Метод тестирования, основанный на модели, с генерацией тестов базируется и тесно связан с использованием формальных методов, поэтому понятны рекомендации для уровней полноты безопасности (УПБ): КР (крайне рекомендованный) для более высоких УПБ и не требуется для низких УПБ.

В общем случае метод состоит из набора следующих действий:

- создание модели (из системных требований);
- генерация ожидаемых входов;
- генерация ожидаемых выходов;
- выполнение тестов;
- сравнение фактических результатов с ожидаемыми выходами;
- выбор дальнейших действий (изменение модели, генерация дальнейших тестов, оценка надежности/качества программного обеспечения).

Для получения тестов могут быть использованы различные методы и средства представления моделей поведения пользователя/системы, например:

- таблицы решений;
- конечные автоматы;
- формальные грамматики;
- цепи Маркова;
- диаграммы состояний;
- доказательство теорем;
- логическое программирование с ограничениями;
- модель проверки;
- моделирование на символьном уровне;
- использование модели потока событий;
- параллельные иерархические конечные автоматы для тестирования реактивных систем и т. д.

Тестирование, основанное на модели, с недавних пор целенаправленно используется в областях, критических к безопасности. Оно позволяет на ранних стадиях выявить неоднозначности в спецификации и проекте, обеспечивает возможность автоматически генерировать много неповторяемых эффективных тестов, оценить регрессионный набор тестов и оценить надежность и качество программного обеспечения, а также облегчает обновление наборов тестов.

Полный обзор дан в El-Far (2001) и SoftwareTech (2009), другие подробности, а также проблемы, зависящие от предметной области, рассмотрены в других источниках.

Литература:

T. Bauer, F. Böhr, D. Landmann, T. Beletski, R. Eschbach, Robert and J.H. Poore, From Requirements to Statistical Testing of Embedded Systems Software Engineering for Automotive Systems — SEAS 2007, ICSE Workshops, Minneapolis, USA.

Eckard Bringmann, Andreas Krämer; Model-based Testing of Automotive Systems In: ICST, pp. 485—493, 2008 International Conference on Software Testing, Verification, and Validation, 2008.

Broy M., Challenges in automotive software engineering, International conference on Software engineering (ICSE '06),Shanghai, China, 2006.

I.K. El-Far and J.A. Whittaker, Model-Based Software Testing. Encyclopedia of Software Engineering (edited by J. J. Marciniak). Wiley, 2001.

Heimdahl, M.P. E.: Model-based testing: challenges ahead, Computer Software and Applications Conference (COMPSAC 2005), 25—28 July 2005, Edinburgh, Scotland, UK, 2005.

Jonathan Jacky, Margus Veanes, Colin Campbell, and Wolfram Schulte, Model-Based Software Testing and Analysis with C#, ISBN 978-0-521-68761-4, Cambridge University Press, 2008.

A. Paradkar, Case Studies on Fault Detection Effectiveness of Model-based Test Generation Techniques, in ACM SIGSOFT SW Engineering Notes, Proc. of the first int. workshop on Advances in model-based testing A-MOST '05, Vol. 30, Issue 4. ACM Press, 2005.

S.J. Prowell, Using Markov Chain Usage Models to Test Complex Systems, HICSS '05: 38th Annual Hawaii, International Conference on System Sciences, 2005.

Mark Utting and Bruno Legeard, Practical Model-Based Testing: A Tools Approach, ISBN 978-0-12-372501-1, Morgan-Kaufmann, 2007.

Hong Zhu et al. (2008). AST '08: Proceedings of the 3rd International Workshop on Automation of Software Test. ACM Press. ISBN 978-1-60558-030-2.

Model-Based Testing of Reactive Systems Advanced Lecture Series, LNCS 3472, Springer-Verlag, 2005, ISBN 978-3-540-26278-7.

Model-based Testing, SoftwareTech, July 2009, Vol. 12, No. 2, Software Testing: A Life Cycle Perspective, http://www.goldpractices.com/practices/mbt/.

**С.6 Оценка функциональной безопасности**

Примечание — Соответствующие методы и средства см. также в В.6.

**С.6.1 Таблицы решений (таблицы истинности)**

Примечание — Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблицы А.10 и В.7).

Цель. Обеспечение ясных и согласующихся спецификаций и анализа сложных логических комбинаций и их отношений.

Описание. Данный метод использует бинарные таблицы для точного описания логических отношений между булевыми переменными программы.

Использование таблиц и точность метода позволили применить его в качестве средства анализа сложных логических комбинаций, выраженных в бинарных кодах.

Рассматриваемый метод достаточно легко автоматизируется, поэтому его можно использовать в качестве средства спецификации систем.

**С.6.2 Исследование опасности и работоспособности программного обеспечения (CHAZOP, FMEA)**

Цель. Определение угроз безопасности в предлагаемой или существующей системе, их возможных причин и последствий и рекомендуемых действий по минимизации вероятности их появления.

Описание. Группа специалистов в области создаваемой системы принимает участие в структурном анализе проекта системы путем ряда запланированных совещаний. Они рассматривают как реализацию функций проекта системы, так и способы работы системы на практике (включая действия персонала и процедуры эксплуатации системы). Руководитель группы специалистов инициирует ее участников создавать потенциальные опасности и управляет этой процедурой, описывая каждую часть системы в сочетании с отдельными ключевыми словами: «отсутствует», «более», «менее», «часть целого», «больше чем» (или «так же, как и») и «иначе чем». Каждое применимое условие или режим отказа рассматривается с точки зрения реализуемости, причин возникновения, возможных последствий (появляется ли опасность), способа устранения и, в случае устранения, выбора наиболее целесообразного метода.

Исследование опасностей может выполняться на разных стадиях разработки проекта, однако наиболее эффективным такое исследование может быть на начальных стадиях с тем, чтобы как можно раньше повлиять на основные решения по проектированию и работоспособности.

Метод HAZOP создавался для производственных процессов и требует модификации при его применении к программному обеспечению. Были предложены различные производные методы (Computer HAZOPs — «CHAZOPs»), которые сопровождались новыми руководящими материалами и/или реализовывали способы систематического охвата системной и программной архитектур.

Литература:

OF-FMEA: an approach to safety analysis of object-oriented software intensive systems, T. Cichocki, J. Gorski. In Artificial Intelligence and Security in Computing Systems: 9th International Conference, ACS '2002. Ed. J. Soldek. Springer, 2003, ISBN 1402073968, 9781402073960.

Software FMEA techniques. P.L. Goddard. In Proc Annual 2000 Reliability and Maintainability Symposium, IEEE, 2000, ISBN: 0-7803-5848-1.

Software criticality analysis of COTS/SOUP. P. Bishop, T. Clement, S. Guerra. In Reliability Engineering & System Safety, Volume 81, Issue 3, September 2003, Elsevier Ltd., 2003.

**С.6.3 Анализ отказов по общей причине**

Примечания

1 Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица А.10).

2 См. также МЭК 61508-6 (приложение D).

Цель. Определение возможных отказов в нескольких системах или нескольких подсистемах, которые могут свести к нулю преимущества избыточности из-за одновременного появления одних и тех же отказов во многих частях системы.

Описание. Системы, ориентированные на безопасность объекта, часто используют избыточность аппаратных средств и мажоритарный принцип голосования. Этот подход исключает случайные отказы в компонентах или подсистемах аппаратных средств, которые могут помешать корректной обработке данных.

Однако некоторые отказы могут оказаться общими для нескольких компонентов или подсистем. Например, если система установлена в одном помещении, то недостатки вентиляции могут снизить преимущества избыточности. Это может оказаться верным и для других внешних влияний на систему (например пожар, затопление, электромагнитные влияния, трещины в панелях и землетрясение). Система может быть также подвержена воздействиям, относящимся к ее функционированию и эксплуатации. Поэтому важно, чтобы в рабочих инструкциях были предусмотрены адекватные и хорошо задокументированные процедуры по функционированию и эксплуатации системы, а обслуживающий персонал был хорошо обучен.

Внутренние причины также вносят большой вклад в общее число отказов. Их основой могут являться ошибки проектирования общих или идентичных компонентов и их интерфейсов, в том числе и устаревших компонентов. Анализ отказов по общей причине должен отыскивать также общие дефекты в системе. К методам анализа отказов по общей причине относятся: общее управление качеством; анализ проектов; верификация и тестирование независимой группой; анализ реальных ситуаций, полученных из опыта работы аналогичных систем. Однако область применения такого анализа выходит за рамки только аппаратных средств. Даже если разные программы используются в разных каналах избыточных систем, возможна некоторая общность в программных подходах, которая может привести к росту отказов по общей причине (например ошибки в общей спецификации).

Если отказы по общей причине не появляются точно в одно и то же время, то должны быть предприняты меры предосторожности путем сравнения методов, применяемых в различных каналах. При этом применение каждого метода должно обнаруживать отказ до того, как он окажется общим для всех каналов. Анализ отказов по общей причине должен использовать этот подход.

Литература:

Reliability analysis of hierarchical computer-based systems subject to common-cause failures L. Xing, L. Meshkat, S. Donohue. Reliability Engineering & System Safety Volume 92, Issue 3, March 2007.

**С.6.4 Структурные схемы надежности**

Примечания

1 Ссылка на данный метод/средство приведена в МЭК 61508-3 (таблица А.10) и метод также использован в МЭК 61508-6 (приложение В).

2 См. также В.6.6.7 «Структурные схемы надежности».

Цель. Моделирование в форме диаграмм набора событий, которые должны происходить, и условий, которые должны быть удовлетворены для успешного выполнения операций системы или задач.

Описание. Данный метод позволяет сформировать успешный маршрут, состоящий из блоков, линий и логических переходов. Такой успешный маршрут начинается от одной стороны диаграммы и проходит через блоки и логические переходы до другой стороны диаграммы. Блок представляет собой условие или событие, маршрут проходит через него, если условие истинно или событие произошло. Когда маршрут подходит к логическому переходу, то он продолжается, если критерий логического перехода выполняется. Если маршрут достигает какой-либо вершины, то он может продолжаться по всем исходящим из нее путям. Если существует по меньшей мере один успешный маршрут через всю диаграмму, то цель анализа считается достигнутой.

Литература:

IEC 61025:2006, Fault tree analysis (FTA).

From safety analysis to software requirements. K. M. Hansen, A. P. Ravn, A. P, V Stavridou. IEEE Trans Software Engineering, Volume 24, Issue 7, Jul 1998.

IEC 61078:2006, Analysis techniques for dependability — Reliability block diagram and boolean methods.

**Приложение D**
**(справочное)**

## Вероятностный подход определения полноты безопасности предварительно разработанных программных средств

### D.1 Общие положения

Настоящее приложение содержит исходные руководящие материалы по использованию вероятностного подхода к определению полноты безопасности программных средств для предварительно разработанных программ на основе их опыта эксплуатации. Вероятностный подход является наиболее подходящим для оценки операционных систем, компонентов библиотек, компиляторов и других программных систем. Настоящее приложение также содержит описание возможностей вероятностного подхода, однако его следует использовать только тем специалистам, кто компетентен в статистическом анализе.

Примечание — В настоящем приложении используется термин «уровень доверия», который описан в [9]. Эквивалентный термин «уровень значимости» приведен в [10].

Предложенные в настоящем приложении методы могут быть также использованы для демонстрации роста уровня полноты безопасности программных средств, которые некоторое время успешно эксплуатировались. Например, программные средства, созданные в соответствии с требованиями МЭК 61508-3 для УПБ1, после соответствующего периода успешной работы в большом числе применений могут продемонстрировать соответствие уровню полноты безопасности УПБ2.

Число запросов без отказов при испытании или число часов, необходимое для работы без отказов, для определения конкретного уровня полноты безопасности представлено в таблице D.1. В таблице D.1 также обобщены результаты, приведенные в D.2.1 и D.2.3.

Опыт эксплуатации может быть выражен математически, как показано в D.2, для дополнения или замены статистического тестирования, а опыт эксплуатации, полученный из нескольких мест эксплуатации, может быть объединен (путем добавления конкретного числа обработанных запросов или часов работы в течение эксплуатации), но только в случае, если:

- программная версия, подлежащая использованию в Э/Э/ПЭ системе, связанной с безопасностью, будет идентична версии, для которой предъявлен результат опыта ее эксплуатации;
- эксплуатационный профиль входного пространства очень близок друг другу;
- существует эффективная система уведомлений и документирования отказов;
- справедливы принятые в D.2 предположения.

Таблица D.1 — Необходимая предыстория для определения уровня полноты безопасности

| УПБ | Режим работы с низкой интенсивностью запросов (вероятность отказа при выполнении планируемых функций по запросу) | Количество реальных запросов | | Режим с высокой интенсивностью запросов или в режиме с непрерывным запросом (вероятность опасного отказа в час) | Общее количество часов эксплуатации | |
|---|---|---|---|---|---|---|
| | | $1-\alpha = 0{,}99$ | $1-\alpha = 0{,}95$ | | $1-\alpha = 0{,}99$ | $1-\alpha = 0{,}95$ |
| 4 | $\geq 10^{-5}$ до $< 10^{-4}$ | $4{,}6 \cdot 10^5$ | $3 \cdot 10^5$ | $\geq 10^{-9}$ до $< 10^{-8}$ | $4{,}6 \cdot 10^9$ | $3 \cdot 10^9$ |
| 3 | $\geq 10^{-4}$ до $< 10^{-3}$ | $4{,}6 \cdot 10^4$ | $3 \cdot 10^4$ | $\geq 10^{-8}$ до $< 10^{-7}$ | $4{,}6 \cdot 10^8$ | $3 \cdot 10^8$ |
| 2 | $\geq 10^{-3}$ до $< 10^{-2}$ | $4{,}6 \cdot 10^3$ | $3 \cdot 10^3$ | $\geq 10^{-7}$ до $< 10^{-6}$ | $4{,}6 \cdot 10^7$ | $3 \cdot 10^7$ |
| 1 | $\geq 10^{-2}$ до $< 10^{-1}$ | $4{,}6 \cdot 10^2$ | $3 \cdot 10^2$ | $\geq 10^{-6}$ до $< 10^{-5}$ | $4{,}6 \cdot 10^6$ | $3 \cdot 10^6$ |

Примечания

1 Величина $1-\alpha$ представляет собой уровень доверия.

2 Предпосылки и описание процедур получения числовых значений в настоящей таблице см. в D.2.1 и D.2.3.

### D.2 Формулы статистического тестирования и примеры их использования

#### D.2.1 Простой статистический тест для режима работы с низкой интенсивностью запросов

D.2.1.1 Исходные предпосылки

a) Распределение тестовых данных равно распределению запросов при выполнении операций в режиме онлайн.

b) Прохождения тестов статистически не зависят друг от друга в отношении причины отказа.

c) Для обнаружения любых отказов, которые могут появиться, существует адекватный механизм.

d) Число тестовых примеров $n > 100$.

e) Во время прогона $n$ тестовых примеров отказы отсутствуют.

D.2.1.2 Результаты

Вероятность отказа $p$ (на один запрос) при уровне доверия $1-\alpha$ определяется из выражения

$$p \le 1-\sqrt[n]{\alpha} \text{ или } n \ge -\frac{\ln\alpha}{p}.$$

D.2.1.3 Пример

Таблица D.2 — Вероятности отказа режима работы с низкой интенсивностью запросов

| $1-\alpha$ | $P$ |
|---|---|
| 0,95 | $3/n$ |
| 0,99 | $4,6/n$ |

Для вероятности отказа при запросе для уровня полноты безопасности УПБ 3 при 95%-ном уровне доверия применение указанной формулы дает 30000 тестовых примеров при выполнении условий принятых предпосылок. Результаты для каждого уровня полноты безопасности объединены в таблице D.1.

**D.2.2 Тестирование входного массива (предметной области) для режима работы с низкой интенсивностью запросов**

D.2.2.1 Исходные предпосылки

Единственная исходная предпосылка состоит в том, что тестируемые данные выбираются так, чтобы обеспечить случайное унифицированное распределение по входному массиву (предметной области).

D.2.2.2 Результаты

Неодходимо определить количество тестов $n$, которые требуются, исходя из порога точности $\delta$ входов для тестируемой функции с низкой интенсивностью запросов (например безопасное отключение).

Таблица D.3 — Средние расстояния между двумя точками тестирования

| Размер предметного пространства | Среднее расстояние между двумя точками тестирования в произвольном направлении |
|---|---|
| 1 | $\delta = 1/n$ |
| 2 | $\delta = \sqrt{1/n}$ |
| 3 | $\delta = \sqrt[3]{1/n}$ |
| $K$ | $\delta = \sqrt[K]{1/n}$ |
| Примечание — $K$ может быть любым положительным целым числом. Значения 1, 2 и 3 приведены только в качестве примеров. | |

D.2.2.3 Пример

Рассмотрим безопасное отключение, которое зависит только от двух переменных A и B. Если проверкой было установлено, что пороговые значения, которые разделяют входную пару переменных A и B, определены с точностью до 1 % от диапазона измерения A или B, то число равномерно распределенных тестовых примеров, требуемое в области A и B, будет равно

$$n = 1/\delta^2 = 10^4.$$

**D.2.3 Простой статистический тест для режима с высокой интенсивностью запросов или в режиме с непрерывным запросом**

D.2.3.1 Исходные предпосылки

a) Распределение данных такое же, как и распределение при выполнении операций в режиме онлайн.

b) Относительное уменьшение вероятности отсутствия отказа пропорционально длительности рассматриваемого интервала времени и постоянно в противном случае.

c) Для обнаружения любых отказов, которые могут появиться, существует адекватный механизм.

d) Тест выполняется в течение времени тестирования $t$.

e) Во время тестирования $t$ никаких отказов не происходит.

D.2.3.2 Результаты

Соотношение между интенсивностью отказов $\lambda$, уровнем доверия $1-\alpha$ и временем тестирования $t$ имеет вид

$$\lambda = -\frac{\ln\alpha}{t}.$$

Интенсивность отказов обратно пропорциональна среднему времени наработки на отказ

$$\lambda = \frac{1}{MTBF}.$$

Примечание — Настоящий стандарт не делает различий между интенсивностью отказов в час и частотой отказов в час. Строго говоря, вероятность отказа $F$ связана с частотой отказов $f$ выражением $F = 1 - e^{-ft}$, однако область применения настоящего стандарта охватывает частоту отказов менее $10^{-5}$ 1/ч, а для небольших значений частоты справедливо $F \cong f \cdot t$.

D.2.3.3 Пример

Таблица D.4 — Вероятности отказа для режима с высокой интенсивностью запросов или для режима с непрерывным запросом

| $1-\alpha$ | $\gamma$ |
|---|---|
| 0,95 | $3t$ |
| 0,99 | $4{,}6t$ |

Для подтверждения того, что среднее время наработки на отказ составляет по меньшей мере 108 ч с уровнем доверия 95 %, требуется время тестирования $3 \cdot 10^8$ ч и должны быть соблюдены исходные предпосылки. Число тестов, необходимое для каждого уровня полноты безопасности, — в соответствии с таблицей D.1.

**D.2.4 Полное тестирование**

Программу можно рассматривать как урну, содержащую $N$ шаров. Каждый шар представляет собой конкретное свойство программы. Шары извлекаются случайно и заменяются после проверки. Полное тестирование достигается, если все шары извлечены.

D.2.4.1 Исходные предпосылки

a) Распределение тестируемых данных таково, что каждое из $N$ свойств программы тестируется с равной вероятностью.

b) Тесты проводятся независимо друг от друга.

c) Каждый появляющийся отказ обнаруживается.

d) Число тестовых примеров $n \gg N$.

e) Во время прогона $n$ тестовых примеров отказы не появляются.

f) Каждый прогон теста контролирует одно свойство программы (свойство программы — это то, что может быть протестировано во время одного прогона теста).

D.2.4.2 Результаты

Вероятность тестирования всех свойств программы $p$ определяется выражением

$$p = \sum_{j=0}^{N-1} (-1)^j \binom{N}{j} \left(\frac{N-j}{N}\right)^n \text{ или } p = 1 + \sum_{j=1}^{N} (-1)^j C_{j,N} \left(\frac{N-j}{N}\right)^n,$$

где $C_{j,N} = \dfrac{N(N-1)\ldots(N-j+1)}{j!}$.

При оценке этого выражения обычно только первые его члены имеют значение, поскольку в реальных условиях выполняется соотношение $n \gg N$, что делает все члены этого выражения при большом $j$ несущественными. Это видно из таблицы D.5.

D.2.4.3 Пример

Рассмотрим программу, которая имела несколько инсталляций в течение нескольких лет. За это время она выполнялась по меньшей мере $7{,}5 \cdot 10^6$ раз. Предположим, что каждое сотое выполнение программы соответствует перечисленным выше исходным предпосылкам (см. D.2.4.1). Поэтому для статистической оценки могут быть приняты $7{,}5 \cdot 10^4$ выполнений программы. Если предположить, что 4000 тестовых прохождений программы могут вы-

полнить исчерпывающее тестирование, считая такую оценку консервативной, то в соответствии с таблицей D.5, вероятность того, что не все будет протестировано, составляет $2,87 \cdot 10^{-5}$.

При $N = 4000$ значения первых членов в зависимости от $n$ представлены в таблице D.5.

Таблица D.5 — Вероятность тестирования всех свойств программы

| $n$ | $p$ |
|---|---|
| $5 \cdot 10^4$ | $1 - 1,9 \cdot 10^{-2} + 1,10 \cdot 10^{-4} - \ldots$ |
| $7,5 \cdot 10^4$ | $1 - 2,87 \cdot 10^{-5} + 4 \cdot 10^{-10} - \ldots$ |
| $1 \cdot 10^5$ | $1 - 5,54 \cdot 10^{-8} + 1,52 \cdot 10^{-15} - \ldots$ |
| $2 \cdot 10^5$ | $1 - 7,67 \cdot 10^{-19} + 2,9 \cdot 10^{-37} - \ldots$ |

На практике такие оценки должны быть консервативными.

**D.3 Литература**

Более подробную информацию по указанным выше методам можно найти в [9] — [13].

**Приложение Е**
**(справочное)**

## Краткий обзор методов и мер для проектирования СИС

Примечание — В данном приложении дан обзор методов и мер, на которые ссылается МЭК 61508-2. Данное приложение не должно рассматриваться как полное или исчерпывающее.

### Е.1 Описание проекта на языке (V)HDL

Цель. Функциональное описание на высокоуровневом языке описания технических средств, например, VHDL или Verilog.

Описание. Высокоуровневое абстрактное функциональное описание проекта на языке описания аппаратных средств, например, VHDL или Verilog. Используемый язык описания аппаратных средств должен позволить функциональное и/или ориентированное на применение описание и должен быть абстрагирован от последующих деталей реализации. Потоки данных, условные переходы, арифметические и/или логические операции должны быть реализованы присваиванием и операторами языка описания аппаратных средств без ручного преобразования в логические вентили из прикладной библиотеки.

Примечание — Для простоты «функциональное описание на высокоуровневом языке описания технических средств» будем далее обозначать (V)HDL.

Литература:
IEEE VHDL, Verilog + Standard VHDL Design guide.

### Е.2 Ввод описаний схем

Цель. Функциональное описание компонуемой схемы при ее представлении на уровне логических элементов и/или макросов библиотеки поставщика.

Описание. Описание функциональности компонуемой схемы формируется при вводе описаний схем и связей между ними. Функция, которая должна быть реализована, компонуется (собирается) из элементарных логических схем, таких как И, ИЛИ, НЕ, а также макросов, состоящих из сложных арифметических и логических функций, которые затем соединяются между собой. Сложные функциональные схемы должны быть иерархически декомпозированы и могут быть представлены на различных иерархически связанных рисунках. Межсоединения должны быть уникально определены и иметь конкретные имена сигналов по всей иерархии. Насколько это возможно, необходимо избегать использования глобальных сигналов (меток).

### Е.3 Структурное описание

Примечание — См. также С.2.7 «Структурное программирование» и Е.12 «Разбиение на модули».

Цель. Описание функциональности схемы должно быть структурировано таким способом, чтобы оно было легко воспринимаемо, то есть функция схемы должна быть интуитивно понята из описания без привлечения моделирования.

Описание. Описание функциональности схемы формируется на языке (V)HDL или вводом описания схем. Рекомендуется легко узнаваемая модульная структура. Каждый модуль также должен быть реализован одним и тем же способом и должен быть описан так, чтобы можно было легко понять все его подфункции. Рекомендуется строго различать описание реализуемой функции от описания структуры на уровне межсоединений, то есть модуль, реализуемый из нескольких других подмодулей, должен содержать только описание межсоединений этих подмодулей и не должен содержать описание логики схемы.

### Е.4 Средства, проверенные в эксплуатации

Цель. Применение средств, проверенных на практике, для предотвращения систематических отказов при условии, что эти средства достаточно долго и успешно применялись в различных проектах.

Описание. Большинство используемых средств для разработки ASIC и FPGA включает сложное программное обеспечение, которое не может считаться работающим без каких либо ошибок, а также довольно часто ошибки могут возникнуть вследствие неправильной эксплуатации таких средств. Поэтому для разработки ASIC и FPGA необходимо ипользовать только средства с атрибутом «проверено на практике». Это подразумевает:

- применение средств, которые использовались (в сопоставимой версии программного обеспечения) в течение длительного периода времени или большим числом пользователей в различных проектах такой же сложности;
- наличие у каждого разработчика ASIC/FPGA опыта работы с этими средствами в течение длительного периода времени;
- применение общеиспользуемых средств (соответствующим числом пользователей) так, чтобы была доступна информация об известных отказах от всех пользователей (в процессе управления версиями со «Списком ошибок»). Эта информация должна быть быстро интегрирована в процесс проектирования, чтобы помочь избежать систематических отказов;

- проверку непротиворечивости внутренних средств базы данных и проверку достоверности данных, что поможет избежать ошибок данных, получаемых из базы данных. Непротиворечивость данных внутри базы данных должны проверять стандартные средства, которые, чтобы работать с уникальными данными, проверяют непротиворечивость данных, полученных в результате работы средств синтеза, размещения и трассировки.

Примечание — Проверка непротиворечивости — обязательная функция используемого средства, и разработчик на нее влияет слабо. Поэтому, если существует возможность ручной проверки непротиворечивости, разработчик должен адекватно ее использовать.

**Е.5 Моделирование на языке (V)HDL**

Примечание — См. также Е.6 «Функциональное тестирование на уровне модулей».

Цель. Функциональная проверка схемы, описанной на языке (V)HDL, средствами моделирования.

Описание. Проверка функции с помощью моделирования всей схемы или каждого подмодуля. Средство моделирования (V)HDL позволяет получить последовательность выходных сигналов, отражающих внутренние изменения состояний схемы, вызванные поданными входными сигналами. Проверка полученной выходной последовательности может быть выполнена или по предварительно известной последовательности выходных сигналов («форме сигнала»), или на специальном испытательном стенде, который создается во время процесса проектирования. Выбранное средство моделирования должно иметь статус «проверено в эксплуатации», чтобы обеспечить корректные результаты и заблокировать неправильное поведение сигналов синхронизации (импульсные помехи, отслеживание троичных состояний), которые могут быть вызваны самим средством моделирования или ошибкой моделирования.

**Е.6 Функциональное тестирование на уровне модулей**

Примечание — См. также Е.5 «Моделирование на языке (V)HDL», Е.13 «Охват сценариями верификации (испытательный стенд)».

Цель. Функциональная проверка «Снизу вверх»

Описание. Проверка реализуемой функции — например моделированием — на уровне модуля. Тестируемый модуль будет загружаться в типичную виртуальную среду тестирования, называемую «испытательный стенд», и тестируется содержащимися в среде наборами тестов. Для указанной функции требуется по крайней мере достаточно высокий охват тестами, включая все особые случаи, если они существуют. Автоматическая проверка выходных сигналов средствами «испытательного стенда» должна быть более приоритетна по сравнению с их ручной проверкой.

**Е.7 Функциональное тестирование на верхнем уровне**

Примечание — См. также Е.8 «Функциональное тестирование во внешней системной среде».

Цель. Проверка СИС (всей схемы).

Описание. Задача этого теста — проверка всей схемы (СИС).

**Е.8 Функциональное тестирование во внешней системной среде**

Примечание — См. также Е.7 «Функциональное тестирование на верхнем уровне».

Цель. Проверка заданной функции, встроенной в системное окружение.

Описание. Этот тест проверяет всю функциональность схемы (СИС) в ее системной среде, например со всеми другими компонентами, которые расположены на печатных платах или в другом месте. Моделирование всех соответствующих компонентов на печатной плате и моделирование СИС, на основе ее модели, для проверки корректной функциональности рекомендуется проводить с учетом сигналов синхронизации. Полное функциональное тестирование включает также тестирование модулей, которые становятся активными только во время отказа.

**Е.9 Ограниченное использование асинхронных структур**

Цель. Предотвращение типичных проблем синхронизации в процессе синтеза, предотвращение неоднозначности в процессе моделирования и синтеза, связанных с трудностями создаваемой модели, проектирование тестируемости.

Описание. Асинхронные структуры, формирующие такие сигналы, как УСТАНОВКА и СБРОС, построенные на комбинационнной логике, чувствительны к процедуре синтеза. В результате могут формироваться схемы, создающие импульсные помехи или срабатывающие от инверсной синхронизации, и поэтому их необходимо избегать. Такие проблемы при создании модели не могут быть интерпретированы должным образом средствами синтеза, что приводит к неоднозначным результатам при моделировании асинхронных структур. Так как асинхронные структуры являются плохо тестируемыми или нетестируемыми вовсе, то эффективность охвата тестами при заводских испытаниях или тестирования в режиме онлайн снижается. Поэтому рекомендуется реализовывать полностью синхронную систему с ограниченным числом синхросигналов. В системах с многофазной синхронизацией все синхросигналы должны формироваться из одного центрального генератора синхросигналов. Синхронизация на входе последовательностной логики всегда должна осуществляться только синхросигналом, который не содержит комбинационной логики. Входы последовательностных ячеек асинхронных структур УСТАНОВКА и СБРОС всегда должны обеспечи-

ваться синхросигналами, которые не содержат комбинационной логики. Устройства, задающие сигналы УСТАНОВКА и СБРОС, должны синхронизироваться с помощью двух триггеров.

**Е.10 Синхронизация основных входов и управление метастабильностью**

Цель. Предотвращение неоднозначного поведения схемы в результате нарушения времен установки и промежуточного хранения.

Описание. Входные сигналы от внешних периферийных устройств являются обычно асинхронными и могут произвольно изменять свое состояние. Непосредственная обработка таких сигналов выполняется синхронными элементами последовательностных схем ASIC/FPGA. Например, использование триггеров ведет к нарушению времени установки и промежуточного хранения, что приводит к непредсказуемым нарушениям в синхронизации и функциональном поведении ASIC/FPGA. Это может привести к метастабильности элемента памяти. Поэтому, чтобы избежать функциональной неоднозначности, каждый асинхронный входной сигнал должен синхронизироваться системой синхронизации ASIC. Рекомендуются следующие меры:

- входные сигналы должны синхронизироваться двумя последовательными элементами памяти (триггерами) или некоторой эквивалентной схемой, чтобы достигнуть предсказуемого функционального поведения;
- каждый асинхронный входной сигнал должен обязательно синхронизироваться способом, определенным выше, то есть каждый асинхронный сигнал соединяется только с одной такой схемой синхронизации. В случае необходимости выход схемы синхронизации может использоваться для множественного доступа;
- такая схема синхронизации должна использоваться для проверки устойчивости сигналов параллельной шины и управлять согласованностью данных в точке выборки.

**Е.11. Проектирование тестируемости**

Примечание — См. также Е.31 «Реализация тестовых структур»

Цель. Предотвращение нетестируемых или плохо тестируемых структур, чтобы обеспечить высокий уровень тестового охвата при заводских испытаниях или тестирования в режиме онлайн.

Описание. Проектом по тестированию необходимо управлять в целях предотвращения:

- асинхронных структур;
- защелок и сигналов с тремя состояниями на микросхеме;
- монтажного И/монтажного ИЛИ и избыточной логики.

Комбинационная глубина подсхем играет важную роль в процессе тестирования. Тестовый пример, необходимый для полного тестирования, экспоненциально возрастает с глубиной комбинационной схемы. Поэтому схемы с высокой комбинационной глубиной довольно плохо тестируются существующими средствами.

Подход, ориентируемый на проектирование тестируемости, гарантирует, что требуемый тестовый охват достигнут. Поскольку фактический тестовый охват может быть определен на очень поздней стадии процесса проектирования, недостаточное внимание проблемам «проектирования тестируемости» может существенно уменьшить достижимый тестовый охват, приводя к дополнительному объему работ.

Примечание — Тестовый охват обычно определяется процентом обнаруженных константных отказов.

**Е.12 Разбиение на модули**

Примечание — См. также С.2.8 «Ограничение доступа/инкапсуляция информации», С.2.9 «Модульный подход» и Е.3 «Структурное описание».

Цель. Описание модулей, реализующих функции схемы.

Описание. Строгое разделение полной функциональности по различным модулям с ограниченными функциями. Таким способом устанавливается прозрачность модулей с точно определенным интерфейсом. Каждая подсистема на всех уровнях проекта явно определена и ограничена по размеру (только несколько функций). Интерфейсы между подсистемами сохранены настолько простыми, насколько это возможно, и пересечение модулей (то есть совместно используемые данные, обмен информацией) минимизировано. Также ограничена сложность отдельных подсистем.

**Е.13 Охват сценариями верификации (испытательные стенды)**

Цель. Количественная и качественная оценка применяемых сценариев верификации в процессе функционального тестирования.

Описание. Качество сценариев верификации определяется в процессе функционального тестирования, то есть применяемый тестовый набор для верификации конкретной функции, включая все особые случаи, если они существуют, должен быть качественно и/или количественно документально оформлен. При количественном подходе должны быть документально оформлены достигнутый тестовый охват и глубина примененных функциональных тестов. Получающийся охват должен удовлетворять уровням, установленным для каждой из метрик охвата. Любое исключение должно быть документально оформлено. В случае качественного подхода должно быть оценено число проверенных строк кода, инструкций или путей («Охват кода») проверяемого кода схемы.

Примечание — Особый метод анализа «Охват кода» имеет ограниченное применение из-за высокого параллелизма описания аппаратных средств и необходимости обоснования исчерпывающими проверками. «Охват кода» обычно служит для демонстрации неохваченного функционального кода.

**Е.14 Соблюдение руководств по кодированию**

Цель. Строгое соблюдение стиля кодирования приводит к синтаксически и семантически корректному коду схемы.

Описание. Синтаксические правила кодирования помогают создать легко читаемый код и лучшую документацию, включая управление версиями. Обычно руководства содержат правила по организации и комментированию блоков или модулей схемы.

Семантические правила кодирования помогают предотвратить типичные проблемы реализации с помощью устранения структур, которые приводят к ошибке синтеза — неоднозначной реализации функции схемы. Типичными правилами являются, например, предотвращение асинхронных структур или структур, которые производят непредсказуемую последовательность синхросигналов. Обычно к таким неоднозначностям приводит использование защелок или взаимодействие данных с синхросигналами.

Руководства по проектированию рекомендуют избегать систематических отказов проекта во время процесса разработки СИС. Стиль кодирования в определенном смысле ограничивает эффективность проекта, однако, в свою очередь, дает преимущество в предотвращении отказов во время процесса разработки СИС. Он в частности:

- предотвращает типичные недостатки кодирования или отказы;
- ограниченно использует проблемные структуры, которые приводят к неоднозначным результатам синтеза;
- применяется для проектирования тестируемости;
- обеспечивает прозрачный и удобный код.

Пример стиля кодирования.

1 Код должен содержать столько комментариев, сколько это необходимо для понимания деталей реализации и функции. Используемые соглашения должны быть определены перед началом проекта. В течение стадии проектирования должно быть проверено соответствие определенных соглашений.

1.1 Стандартные заголовки включают историю, перекрестные ссылки на спецификацию, информацию об ответственности и данные, сопровождающие проект, такие как номер версии, запросы на изменение и т. д.

1.2 Легко читаемые шаблоны: эквивалентные процессы должны быть описаны одинаковой процедурой, то есть использование предопределенных шаблонов для повторяющихся процессов (если-то-иначе, для и т. д.).

1.3 Точное и читаемое соглашение о присвоении имен, например, прописная/строчная буква, префикс и постфикс, точное дифференцирование между именем порта, внутренними сигналами, константами, переменными, низкий активный уровень (xxx_n) и т. д.

1.4 Должны быть введены ограничения на размер модуля и число портов на модуле для увеличения читаемости кода.

1.5 Структурированная и защищенная разработка кода, например, информация о состоянии должна инкапсулироваться в FSM (сокрытие информации), чтобы обеспечить легкое изменение кода.

1.6 Должны быть реализованы проверки достоверности, такие как проверка диапазона и т. д.

1.7 Предотвращение следующих структур/команд:

- использование просмотра диапазона по индексу в порядке возрастания ключа (от *x* к *y*) для сигналов шины;
- команды «Отключить» в Verilog (соответствует команде goto);
- многомерных массивов (> 2), записей;
- сочетания типов данных без знака и со знаком.

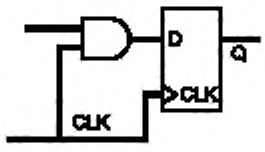
2 Завершенный проект синхронизации (допускается получение тактовых импульсов только от центральных тактовых генераторов).

2.1 Выходы модуля должны быть синхронизированы, это также обеспечит тестируемость и статический анализ временны́х диаграмм.

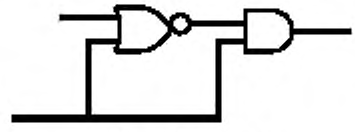
2.2 Управляющие импульсы должны быть обработаны со специальной предосторожностью.

3 Предотвращение связи сигналов данных с синхросигналами повышает тестируемость, воспроизводимость между данными до и после компоновки и уровень соответствия поведения на уровне межрегистровых пересылок.

4 Избыточная логика не является тестируемой и ее необходимо избегать:




Связь сигналов данных с синхросигналами



Избыточная логика

5 Необходимо избегать обратные связи в комбинационной логике, потому что это ведет к нестабильности проекта, и он не будет тестируемым.

6 Рекомендуется, чтобы проект был полностью просматриваемым (прогоняемым при моделировании).

7 Предотвращение защелок увеличивает тестируемость и сокращает ограничения синхронизации в процессе синтеза.

8 Сигнал основного сброса и все асинхронные входные сигналы должны синхронизироваться двумя последовательными элементами памяти (триггерами) или эквивалентной(ыми) схемой(ами) (метастабильность).

9 Рекомендуется избегать асинхронных сигналов установки/сброса, за исключением сигнала основного сброса.

10 Сигналы на уровне порта модуля должны иметь тип std_logic или std_logic_vector.

**Е.15 Применение средств проверки кода**

Цель. Автоматическая проверка правил кодирования («Стиль кодирования») инструментальными средствами проверки кода.

Описание. Применение средств проверки кода помогает в значительной степени автоматически соблюдать стиль кодирования и генерирует документацию в режиме онлайн. Однако автоматическое средство проверки кода может проверить синтаксис и семантику кода в целом. Поэтому применение таких инструментов должно сопровождаться расширением общих правил кодирования («конкретный инструмент») с помощью проектирования специальных правил кодирования, которые разработчик должен реализовывать и оценивать отдельно.

**Е.16 Программирование с защитой**

Примечание — См. С.2.5.

**Е.17 Документальное оформление результатов моделирования**

Цель. Документальное оформление всех данных, необходимых для успешного моделирования, чтобы проверить указанную функцию схемы.

Описание. Все данные, необходимые для функционального моделирования на уровне модуля, микросхемы или системы должны быть правильно документально оформлены и заархивированы для того, чтобы:

- повторить моделирование на любой более поздней стадии, поочередно изменяя параметры модели;
- продемонстрировать правильность и полноту всех определенных функций.

Должна быть заархивирована база данных, хранящая:

- средства моделирования, включающие полное программное обеспечение используемых инструментов, например, средства моделирования, синтеза с указанием версий и необходимой библиотеки моделирования;
- файл с журналом моделирования, который включает полную информацию о времени моделирования, примененных инструментов с указанием версий и полный текст отчета о всей работе, если это необходимо;
- все соответствующие результаты моделирования, включая последовательности сигналов, особенно в случае ручной проверки, и документацию полученных результатов.

**Е.18 Проверка кода**

Примечание — См. С.5.14 «Формальные проверки».

Цель. Анализ описания схемы.

Описание. Анализ описания схемы должен быть выполнен в целях проверки:

- стиля кодирования;
- соответствия со спецификацией описанной функциональности;
- кодирования с защитой, обработки ошибок и обработки исключений.

Примечание — Если моделирование на языке (V)HDL не выполнено, у полноты проверки кода и достигнутых результатов должно быть эквивалентное качество, которое может быть достигнуто моделированием на языке (V)HDL.

**Е.19 Сквозной контроль**

Примечание — См. С.5.15 «Сквозной контроль».

Цель. Проверка описания схемы с помощью сквозного контроля.

Описание. Сквозной контроль выполняется следующим образом: команда сквозного контроля выбирает небольшой набор тестовых примеров — представительные наборы входных и соответствующих ожидаемых выходных данных для программы. Затем вручную прослеживается прохождение тестовых данных через логику программы.

Примечание — Как самостоятельное средство оно должно применяться только к схемам с очень низкой сложностью. В случае сбоя при моделировании на языке (V)HDL у полноты сквозного контроля и качества достигнутых результатов должно быть эквивалентное качество, которое будет достигнуто моделированием на языке (V)HDL.

Литература:

IEC 61160:2005, Design review.

**Е.20 Применение прошедших подтверждение соответствия программных блоков, специфицированных на языке описания аппаратных средств, для проектирования микросхем**

Цель. Предотвращение отказа в процессе эксплуатации программного блока применением прошедшего подтверждение соответствия программного блока.

Описание. Если поставщик подтверждает соответствие программного блока, то должны быть выполнены следующие требования:

- должно быть выполнено подтверждение соответствия программного блока для его использования в системе, связанной с безопасностью, и оно должно иметь по крайней мере эквивалентный или более высокий уровень полноты безопасности, чем планируемая система;
- должны быть выполнены все предположения и ограничения, которые необходимы для подтверждения соответствия программного блока;
- должны быть легко доступны все необходимые документы для подтверждения соответствия программного блока, см. также Е.17 «Документальное оформление результатов моделирования»;
- должна строго соблюдаться каждая спецификация поставщика и доказательство соответствия должно быть документально оформлено.

**Е.21 Подтверждение соответствия программных блоков, специфицированных на языке описания аппаратных средств, для проектирования микросхем**

Примечание — См. Е.6 «Функциональное тестирование на уровне модулей».

Цель. Предотвращение отказа в процессе эксплуатации программного блока подтверждением соответствия программного блока в процессе его создания.

Описание. Если программный блок не разработан конкретно для эксплуатации в системе, связанной с безопасностью, то для сгенерированного кода должно быть проведено подтверждение соответствия в тех же самых условиях, которые применяются для проведения подтверждения соответствия любого исходного кода. Это означает, что должны быть определены и выполнены все возможные тестовые примеры. Затем с помощью моделирования должна быть выполнена функциональная проверка.

**Е.22 Моделирование логической схемы на основе списка соединений для проверки ограничений синхронизации**

Цель. Независимая проверка достигаемого ограничения синхронизации во время синтеза.

Описание. Моделирование логической схемы на основе списка соединений, созданного на этапе синтеза, с учётом реальной длины соединений при расчёте времени задержки в линиях связи и задержек в логических элементах. Должны быть сформированы такие входные сигналы, с помощью которых будет охвачен высокий процент ограничений синхронизации и будут включены все наихудшие случаи для синхронизации. Вообще входные сигналы должны быть такими, чтобы выполнялся «Функциональный тест на уровне модуля» (Е.6) или «Функциональный тест на верхнем уровне» (Е.7), подходящие критерии для выбора которого обеспечены при условии, что во время функционального теста может требоваться достаточный тестовый охват. Схема должна быть протестирована при наилучших и наихудших условиях при указанной максимальной тактовой частоте.

Проверка синхронизации может быть выполнена с помощью автоматической проверки времени установки и времени промежуточного хранения для элементов памяти (триггеров) целевой библиотеки, а также с помощью функциональной проверки схемы. Функциональная проверка в основном должна осуществляться с помощью анализа выходных сигналов микросхемы. Она может быть выполнена автоматически, путем сравнения выходных сигналов схемы с соответствующей эталонной моделью или с исходным кодом схемы на языке (V)HDL. Этот тест известен как «регрессионный тест» и его выполнение должно быть более предпочтительным по сравнению с ручной проверкой выходных сигналов.

Примечание — Данный метод позволяет проверить работу системы синхронизации только для тех путей списка соединений логических элементов, которые фактически активны во время моделирования и поэтому такой специально формируемый метод не может обеспечить полный временной анализ схемы в общем случае.

**Е.23 Статический анализ задержки распространения сигнала (STA)**

Цель. Независимая проверка ограничений синхронизации, осуществляемая во время синтеза.

Описание. Статический временной анализ (STA) анализирует все пути списка соединений (схемы), полученного средствами синтеза, с учётом реальной длины соединений при расчёте времени задержки в линиях связи и задержек логических элементов, не выполняя реальное моделирование. Это позволяет в общем случае выполнить полный анализ ограничений синхронизации для всей схемы. Тестируемая схема должна быть проанализирована в наилучших и наихудшие условиях эксплуатации, на максимально задаваемой тактовой частоте, с учетом возможной неустойчивости синхронизации и расфазировки ее рабочего цикла. Число путей, не соответствующих синхронизации, может быть ограничено определенным минимумом, в зависимости от испльзуемого метода проектирования. Перед началом проектирования рекомендуется исследовать, анализировать и определять применяемый метод, который должен обеспечить легко читаемые результаты.

Примечание — Можно предположить, что STA явно охватывает все существующие синхронизируемые пути, если:

а) ограничения синхронизации должным образом определены;

b) тестируемая схема содержит только такие синхронизируемые пути, которые могут быть проанализированы средствами STA, то есть в общем случае полностью синхронные схемы.

**Е.24 Проверочное сравнение списка соединений логических элементов с эталонной моделью средствами моделирования**

Цель. Проверка функциональной эквивалентности синтезированного списка соединений логических элементов.

Описание. Моделирование списка соединений логических элементов, сгенерированного средствами синтеза. Используемые входные сигналы для проверки схемы моделированием точно соответствуют входным сигналам, примененным в процессе выпополнения «Функционального тестирования на уровне модуля» (Е.6) и «Функционального тестирования на верхнем уровне» (Е.7) для функциональной проверки на уровне модуля и на верхнем уровне соответственно. Функциональная проверка в основном должна осуществляться с помощью анализа выходных сигналов микросхемы. Она может быть выполнена автоматически, путем сравнения выходных сигналов схемы с соответствующей эталонной моделью или с исходным кодом схемы на языке (V)HDL. Этот тест известен как «регрессионный тест» и его выполнение должно быть более предпочтительным по сравнению с ручной проверкой выходных сигналов.

Примечание — Данный метод позволяет проверить функциональное поведение только для тех путей списка соединений логических элементов, которые фактически активны во время моделирования. Поэтому тестовый охват может быть только таким же, как и во время исходного функционального тестирования на уровне модуля или на верхнем уровене соответственно. Можно дополнить этот метод формальным тестом на эквивалентность. Во всех случаях функциональная проверка исходного кода на языке (V)HDL должна выполняться с окончательным списком соединений, сгенерированным средствами синтеза.

**Е.25 Сравнение списка соединений логических элементов с эталонной моделью (формальный тест на эквивалентность)**

Цель. Независимая от моделирования проверка функциональной эквивалентности.

Описание. Сравнение функциональности схемы, описанной в исходных кодах языка (V)HDL с функциональностью схемы, описанной списком соединений логических элементов, сгенерированным средствами синтеза. Средства, в основе которых лежит принцип формальной эквивалентности, могут проверять функциональную эквивалентность схемы, описанной различными формами ее представления, например, на языке (V)HDL или в виде ее списка соединений. Если применяется этот метод, то нет необходимости в функциональном моделировании, но независимая функциональная проверка возможна. Успешное применение этого метода может быть гарантировано, если только используемый инструмент будет способен доказывать полную эквивалентность, а все сообщения о несоответствиях оцениваются автоматически или проверяются вручную.

Примечание — Данный метод выгодно объединить с методом Е.24 «Проверочное сравнение списка соединений логических элементов с эталонной моделью средствами моделирования». В любом случае функциональная проверка исходного кода на языке (V)HDL должна выполняться с окончательным списком соединений, сгенерированным средствами синтеза.

**Е.26 Проверка требований и ограничений поставщика СИС**

Цель. Предотвращение отказов в процессе разработки проверкой требований поставщика.

Описание. Тщательная проверка требований и ограничений поставщика (например минимальное и максимальное разветвление на входе и разветвление на выходе, максимальная длина соединения (задержка линии связи), максимальная скорость фронта выходных сигналов, расфазировка тактовых сигналов и т. д.) средствами синтеза улучшает надежность изделия. Требования поставщика к процессу разработки очень важны, поэтому их нарушение оказывает огромное влияние на обоснованность применяемых моделей, использующихся для моделирования. Поэтому любое нарушение требований и ограничений поставщика ведет к некорректным результатам моделирования, дающим нежелательную функциональность.

**Е.27 Документальное оформление ограничений, результатов и средств синтеза**

Цель. Документальное оформление всех сформированных ограничений, которые необходимы для оптимального синтеза при генерации окочательного списка соединений логических элементов.

Описание. Документальное оформление всех ограничений и результатов синтеза необходимо, чтобы:

- воспроизвести синтез на любой более поздней стадии;
- независимо генерировать результаты синтеза для проверки.

Важными документами являются:

- описание настроек синтеза, включая применяемые инструментальные средства и программное обеспечение синтеза с указанием фактических версий, используемые библиотеки синтеза и заданные ограничения и сценарии;
- журнал выполнения синтеза (лог-файл) с указанием времени, используемого средства с указанием версии и полную документацию для синтеза;
- сгенерированный список соединений с предполагаемыми задержками (файл в формате SDF).

**Е.28 Применение проверенных в эксплуатации средств синтеза**

Цель. Применить средство, выполняющее преобразование описания схемы на языке (V)HDL в список соединений логических элементов.

Описание. Средство, отображающее функциональность схемы, описанной на исходном коде языка (V)HDL, в соединения соответствущих логических элементов и примитивы схем из целевой библиотеки СИС. Выбор реализации из множества возможных реализаций, выполняющей требуемую функциональность, зависит от самого оптимального результата, который получен для ограничений синтеза, таких как синхронизация (тактовая частота) и площадь кристалла.

**Е.29 Применение проверенной в эксплуатации целевой библиотеки**

Примечание — См. также Е.4 «Средства, проверенные в эксплуатации».

Цель. Предотвращение систематических отказов, вызванных ошибками в целевой библиотеке.

Описание. Целевые библиотеки синтеза и моделирования для разработки СИС формируются из общей базы данных и поэтому не являются независимыми. Типичными примерами систематических отказов являются:

- неоднозначность между реальным и промоделированным поведением элементов схемы;
- некорректное моделирование, например, времени установки и времени хранения.

Поэтому при проектировании СИС, выполняющих функции безопасности, должны использоваться только «проверенные в эксплуатации» технологии и целевые библиотеки. Это означает:

- применение целевых библиотек, использующихся в течение достаточно длительного времени в проектах с сопоставимой сложностью и тактовой частотой;
- доступность технологии и соответствующей целевой библиотеки в течение достаточно длительного периода, поэтому можно считать, что библиотека обеспечивает достаточную точность моделирования.

**Е.30 Процедуры, основанные на сценарии**

Цель. Воспроизводимость результатов и автоматизация циклов синтеза.

Описание. Автоматическое и основанное на сценарии управление циклами синтеза, включая определение применяемых ограничений. Помимо точной документации на все ограничения синтеза, это помогает воспроизвести список соединений после изменения исходного кода на языке (V)HDL при идентичных условиях.

**Е.31 Реализация тестовых структур**

Цель. Проектирование тестируемых СИС, гарантирующих заключительное испытание.

Описание. Проектирование тестируемости с помощью создания различных тестовых структур обеспечивает создание схем, которые легко тестируются, например:

- сканирование пути. В этом методе либо все (полное сканирование проекта) или часть триггеров (частичное сканирование проекта) соединены в единую цепь или несколько цепей, создающих цепочку сдвиговых регистров. Метод сканирования пути автоматически генерирует тестовый пример для всей логики схемы. Инструментальное средство, генерирующее тестовый пример, называют «Автоматическим генератором тестового примера» (ATPG). Реализация метода сканирования пути существенно улучшает тестируемость схемы и позволяет получить более 98 % тестового охвата при разумных усилиях. Поэтому рекомендуется реализовывать полное сканирование, если это возможно;
- НЕ—И-дерево. В НЕ—И-дереве все основные входы схемы соединены каскадно и создают цепочку. Применяя подходящий тестовый пример («блуждающий бит») можно протестировать поведение схемы при переключении (синхронизацию и уровень запуска). НЕ—И-дерево является средством, непосредственно характеризующим первичные входы. Его рекомендуется применять, если поведение схемы при переключении не может быть протестировано иначе;
- встроенное самотестирование (BIST): Самотестирование схемы и в особенности самотестирование встроенной памяти может быть выполнено очень эффективно, если на кристалле реализовать генератор тестовых примеров. BIST выполняет автоматическую проверку структуры схемы, применяя псевдослучайные тестовые примеры и оценивая сигнатуру реализованной структуры схемы. BIST рекомендуется как дополнительная мера особенно для тестирования памяти. Тест «Сканирующий путь» может быть заменен на BIST;
- тестирование статического тока потребления (IDDQ-тест). Статическая КМОП-схема потребляет ток, главным образом, в процессе переключения. Поэтому абсолютно исправная схема потребляет очень небольшую величину тока (ток утечки < 1 мА) до тех пор, пока не запускается тестовый пример. IDDQ-тест очень эффективен и обеспечивает более, чем 50%-ный тестовый охват сразу же после применения нескольких тестов. IDDQ-тест может быть применен на функциональных тестовых примерах, так же, как и на синтезированных тестовых примерах, сгенерированных средствами ATPG. Этот метод тестирования на практике оказался самым полезным и обнаруживал отказы, которые другие тесты редко или даже совсем не могли выявить. Поэтому данный метод должен применяться дополнительно, вместе с регулярными испытаниями проекта;
- периферийное сканирование. Тестовая архитектура для проверки соединений компонентов на печатной плате реализуется в соответствии со стандартом JTAG. Тот же самый подход может быть применен для проверки соединений модулей на уровне кристалла. Периферийное сканирование в основном рекомендуется для улучшения тестируемости печатных плат.

**Е.32 Оценка тестового охвата моделированием**

Цель. Определение достигаемого тестового охвата, реализованного архитектурой тестов во время испытаний проекта.

Описание. Тестовый охват, достигаемый тестом «Сканирование пути», BIST, функциональным тестовым примером или любыми другими методами, может быть определен моделированием отказа. Во время моделирования отказа тестовый пример применяется к схеме, в которую включены отказы. Реакция схемы на отказ при запуске тестового примера соответствует включенным отказам и, таким образом, вносит вклад в тестовый охват. Моделирование отказа позволяет обнаружить константные отказы, «константный 1» и «константный 0», а достигаемый тестовый охват представляет качество примененного тестового примера. Вообще моделирование отказа может использоваться очень эффективно для обнаружения отказов, связанных с логикой, которая не охватывается тестом «Сканирование пути», например, в случае частичного сканирования.

**Е.33 Оценка тестового охвата средствами ATPG**

Цель. Определение ожидаемого тестового охвата для синтезируемых тестовых примеров («Сканирование пути», BIST) в процессе испытаний проекта.

Описание. В настоящее время существуют разные процедуры, которые генерируют псевдослучайные или алгоритмические тестовые примеры для схемы, реализованные в методе «Сканирование пути». Средства синтеза, такие как ATPG, создают в процессе синтеза каталог невыявленных отказов. Таким способом можно оценить тестовый охват и определить нижний предел достигаемого тестового охвата для применяемого тестового примера. Важно заметить, что тестовый охват ограничен логикой схемы, которая охвачена методом «Сканирование пути». Модули, такие как память, BIST или часть схем, которые оказались не охваченными методом «Сканирование пути», не рассматривают при оценке тестового охвата.

**Е.34 Обоснование проверкой в эксплуатации применяемых блоков СИС на физическом уровне реализации**

Цель. Предотвращение систематических отказов в применяемых блоках СИС на физическом уровне реализации.

Описание. Блок СИС на физическом уровне реализации обычно рассматривается как «черный ящик», который обеспечивает требуемую функциональность и составляет основные данные по размещению в заданной технологии, и который поддерживает необходимый компонент схемы. Возможный функциональный отказ может трактоваться по аналогии с дискретными компонентами, такими как стандартные микропроцессоры, память и т. д. Эксплуатация таких блоков без проверки корректности функционирования возможна, если для применяемой заданной технологии использование блока можно рассматривать как проверенный в эксплуатации компонент. В таком случае остальная часть схемы должна быть обязательно проверена.

**Е.35 Применение блоков СИС (на физическом уровне реализации), прошедших подтверждение соответствия**

Примечание — См. также Е.6 «Функциональное тестирование на уровне модуля».

Цель. Предотвращение систематических отказов в применяемых блоках СИС на физическом уровне реализации.

Описание. Ввиду сложности блока СИС и принятых ограничений подтверждение соответствия блока СИС должно выполняться поставщиком на стадии проектирования с использованием исходного кода на языке (V)HDL. Подтверждение соответствия может быть обосновано только для конкретной конфигурации и заданной технологии применяемого компонента.

**Е.36 Тестирование блоков СИС (на физическом уровне реализации) в неавтономном режиме**

Примечание — См. также Е.13 «Охват сценариями верификации (испытательные стенды)».

Цель. Предотвращение систематических отказов в применяемых блоках СИС на физическом уровне реализации.

Описание. Проверка тестами в режиме онлайн корректности функционирования и реализации используемых блоков СИС. Для применения этого средства требуется эффективная концепция испытания, а оценка применяемой концепции должна быть документально оформлена.

**Е.37 Проверка правила проектирования (DRC)**

Цель. Проверка правил проектирования поставщика.

Оисание: Проверка сгенерированного размещения в соответствии с правилами проектирования поставщика, например, минимальные длины проводников, максимальные длины проводников и ряд правил, связанных с компоновкой размещаемых структур. Полное и корректное выполнение DRC должно быть подробно документально оформлено.

**Е.38 Проверка соответствия топологии схеме (LVS)**

Цель. Независимая проверка размещения.

Описание. LVS формирует функциональность схемы из данных о размещении и сравнивает полученные элементы схемы, включая соединения с входным списком соединений. Это гарантирует эквивалентность размещения схемы со списком соединений, определяющим функциональность схемы. Полное и корректное выполнение LVS должно быть подробно документально оформлено.

**Е.39 Дополнительный резерв времени (> 20 %) для технологических процессов, которые применяются менее трех лет**

Цель. Обеспечение устойчивости реализуемой функциональности схемы даже при серьезной флуктуации процесса и параметров.

Описание. Реальное поведение схемы определено рядом одновременно влияющих на нее физических факторов, особенно на ее малые структуры (например менее 0,5 мк). Фактически, из-за отсутствия подробных знаний и необходимых упрощений, точную модель элементов схемы построить нельзя. С уменьшением геометрических размеров структур схемы задержки в соединениях играют все более важную роль. Задержки сигнала в соединениях между элементами и емкости перекрестных связей между проводниками растут нелинейно. Задержки сигнала в проводниках становятся сравнимыми с задержками в логических элементах. Оценки задержек в соединениях при уменьшении геометрических размеров структур указывают на увеличивающийся риск появления сбоев.

Поэтому рекомендуется запланировать необходимый резерв времени (> 20 %) относительно минимальных и максимальных ограничений синхронизации для схем, в разработке которых использовались процессы, опыт применения которых составляет менее трех лет, чтобы гарантировать корректное выполнение функциональности схемы в условиях серьезно изменяющихся производственных параметров или из-за отсутствия точного моделирования.

**Е.40 Отбраковочное испытание**

Цель. Обеспечение живучести изготавливаемой микросхемы. Устранение отказов на ранних стадиях. Изделия с дефектами на кристалле не должны доказывать свою живучесть выжиганием дефектов, а должны, например, использовать стресс-методы на уровне пластины.

Описание. Отбраковочное испытание должно быть выполнено при самой высокой приемлемой рабочей температуре (обычно 125 °С). Продолжительность тестирования зависит от целевого УПБ или от заданных рекомендаций по выжиганию дефектов, например, производителем СИС. Выжигание дефектов может быть использовано, чтобы:

- устранить отказы на ранних стадиях (уменьшение интенсивности отказов в начале ваннообразной кривой распределения отказов);
- доказать, что отказы на ранних стадиях уже устранены в процессе производства и тестирования (то есть что устройства, вышедшие из производства, уже находятся в области постоянной интенсивности отказов ваннообразной кривой).

**Е.41 Применение серийных устройств, проверенных в эксплуатации**

Цель. Обеспечение безотказности изготовленных кристаллов.

Описание. У производителя системы безопасности должен быть достаточный опыт применения технологии программируемых устройств, а также средств разработки.

**Е.42 Процесс производства, проверенный в эксплуатации**

Цель. Обеспечение безотказности изготовленных кристаллов.

Описание. Проверенный в эксплуатации процесс производства характеризуется достаточно высоким качеством серийного производства.

**Е.43 Контроль качества производственного процесса**

Меры качества и механизмы управления производственного процесса устройства гарантируют непрерывное управление процессом. Например, оптическое или электрическое управление тестовыми структурами, температурные испытания с изменением параметров влажности или циклический температурный тест (см. [14], [15] и т. д.).

**Е.44 Передача качественно изготовленного устройства**

Качество устройства обеспечивается выполнением выбранной группы стресс-тестов, например, температурными испытаниями с изменением параметров влажности или тестами с изменением температуры (см. [14], [15] и т. д.). Производитель устройства предоставит такие доказательства.

**Е.45 Передача качественно функционирующего устройства**

Все устройства будут функционально протестированы. Производитель устройства предоставит такие доказательства.

**Е.46 Стандарты качества**

Производитель СИС должен предусмотреть достаточный уровень управления качеством, например, иметь оформленное «Руководство по качеству и надежности», в котором документально подтверждено выполнение сертификации по ISO 9000 или оценки качества стандартного поставщика (SSQA — Standard Supplier Quality Assessment).

**Приложение F**
**(справочное)**

## Определение свойств стадий жизненного цикла программного обеспечения

Таблица F.1 — Спецификация требований к программному обеспечению системы безопасности (см. МЭК 61508-3, подраздел 7.2 и таблица С.1)

| | Свойство | Определение |
|---|---|---|
| 1.1 | Полнота охвата потребностей безопасности программным обеспечением | Спецификация требований к программному обеспечению системы безопасности охватывает все потребности и ограничения системы безопасности, появившиеся на более ранних стадиях жизненного цикла системы безопасности и определенные для программного обеспечения.<br>Потребности и ограничения системы безопасности обычно устанавливаются перед началом разработки спецификации требований к программному обеспечению системы безопасности. Они могут включать спецификацию того, что программное обеспечение не должно выполнять или должно избегать |
| 1.2 | Корректность охвата потребностей безопасности программным обеспечением | Спецификация требований к программному обеспечению системы безопасности адекватно отвечает потребностям и ограничениям системы безопасности, которые были определены для программного обеспечения.<br>Целью является уверенность в том, что все, что определено в спецификации, действительно гарантирует безопасность для всех необходимых условий |
| 1.3 | Отсутствие ошибок в самой спецификации, включая отсутствие неоднозначности | Внутренняя полнота и согласованность спецификации требований к программному обеспечению системы безопасности: предоставление всей необходимой информации для всех функций и ситуаций, которая должна быть получена из спецификации; отсутствие в ней противоречивых или несогласованных положений.<br>Противоречие полноте и согласованности потребностям системы безопасности, внутренней полноте и согласованности может быть оценено только на основе спецификации требований к программному обеспечению системы безопасности |
| 1.4 | Ясность требований к безопасности | Спецификация требований к программному обеспечению системы безопасности должна быть полностью понятна без излишних усилий тем специалистам, которые должны ее читать, даже если они не принимали участие в ее разработке, при условии, что эти специалисты обладают необходимыми знаниями.<br>Одна важная цель состоит в том, чтобы облегчить проверку и, возможно, модификации |
| 1.5 | Отсутствие неблагоприятного взаимовлияния функций, не связанных с безопасностью, и функций безопасности, реализуемых программным обеспечением системы безопасности | Спецификация требований к программному обеспечению системы безопасности не содержит требований, которые не нужны для обеспечения безопасности УО.<br>Цель состоит в том, чтобы не усложнять разработку и реализацию программного обеспечения и сократить риск отказов и функций, не связанных с безопасностью, но влияющих или создающих опасные условия для отказов и функций, важных для безопасности |
| 1.6 | Способность обеспечения проведения оценки и подтверждения соответствия | Спецификация требований к программному обеспечению системы безопасности является первоисточником необходимой информации для выполнения тестов и исследований, которые в результате их выполнения формируют объективное подтверждение о том, что программное обеспечение удовлетворяет спецификации требований к программному обеспечению системы безопасности |

Таблица F.2 — Проектирование и разработка программного обеспечения — проектирование архитектуры программ (см.МЭК 61508-3, пункт7.4.3 и таблица С.2)

| | Свойство | Определение |
|---|---|---|
| 2.1 | Полнота спецификации требований к программному обеспечению системы безопасности | Проект архитектуры программного обеспечения охватывает все потребности и ограничения системы безопасности, появившиеся в спецификации требований к программному обеспечению системы безопасности |
| 2.2 | Корректность спецификации требований к программному обеспечению системы безопасности | Проект архитектуры программного обеспечения адекватно соответствует спецификации требований к программному обеспечению системы безопасности |
| 2.3 | Отсутствие собственных ошибок проекта | Проект архитектуры программного обеспечения и проектная документация не имеют ошибок, которые могут быть идентифицированы, независимо от любого указанного требования к программному обеспечению системы безопасности.<br>Примеры: зависания, доступ к несанкционированным ресурсам, утечки ресурсов, внутренняя неполнота (то есть, ошибки при обращении ко всем ситуациям, которые устанавливаются непосредственно в проекте) |
| 2.4 | Простота и понятность<br><br>Предсказуемость поведения | Проект архитектуры программного обеспечения, обеспечивающий корректный и точный прогноз функционирования программного обеспечения для всех указанных ситуаций.<br>В частности, эти ситуации включают ситуации с ошибками и с отказами.<br>Предсказуемость подразумевает, в частности, что функционирование не зависит от элементов, неконтролируемых разработчиками или пользователями |
| 2.5 | Верифицируемость и тестируемость проекта | Проект архитектуры программного обеспечения и проектная документация обеспечивают и облегчают формирование убедительного доказательства, что все заданные требования к программному обеспечению системы безопасности правильно учтены в проекте и что проект лишен внутренних ошибок.<br>Верифицируемость может подразумевать получение таких свойств, как простота, модульность, ясность, тестируемость, доказуемость и т. д., в зависимости от используемых методов верификации |
| 2.6 | Отказоустойчивость | Проект архитектуры программного обеспечения дает уверенность, что программное обеспечение будет вести себя безопасно в присутствии ошибок (внутренних ошибок, ошибок операторов или внешних систем).<br>Можно спроектировать активную или пассивную защиту. Проекты активной защиты могут включать такие функции, как обнаружение, создание сообщений и локализацию ошибок, постепенный вывод из эксплуатации и освобождение от любых нежелательных побочных эффектов до возобновления нормального функционирования. Проекты пассивной защиты включают функции, которые гарантируют непроникновение определенных типов ошибок или определенных условий (лавинообразных потоков на входах, особенно дат и времени) без привлечения программного обеспечения |
| 2.7 | Защита от отказов по общей причине, вызванной внешним событием | Проект архитектуры программного обеспечения облегчает идентификацию видов отказов по общей причине и эффективных средств предостережения от отказов |

Таблица F.3 — Проектирование и разработка программного обеспечения: инструментальные средства поддержки и языки программирования (см.МЭК 61508-3, пункт 7.4.4 и таблица С.3)

| | Свойство | Определение |
|---|---|---|
| 3.1 | Поддержка разработки программного обеспечения с требуемыми свойствами программного обеспечения | Средства, обеспечивающие обнаружение ошибок или устранение подверженных ошибкам структур |
| 3.2 | Четкость работы и функциональность инструментальных средств | Обеспечение всестороннего охвата и ответной реакции для всех аспектов работы инструментальных средств |
| 3.3 | Корректность и воспроизводимость результата | Непротиворечивость и точность результата работы инструментального средства для любого входного задания |

Таблица F.4 — Проектирование и разработка программного обеспечения: детальное проектирование (см.МЭК 61508-3, пункт 7.4.5 и таблица С.4)

| | Свойство | Определение |
|---|---|---|
| 4.1 | Полнота спецификации требований к программному обеспечению системы безопасности | Приняты методы детального проектирования и разработки программного обеспечения, которые гарантируют, что получающееся программное обеспечение охватывает все потребности и ограничения системы безопасности, сформированные для программного обеспечения |
| 4.2 | Корректность спецификации требований к программному обеспечению системы безопасности | Существует конкретное доказательство, позволяющее утверждать, что требования к системе безопасности, сформированные для программного обеспечения, выполняются разработанным программным обеспечением |
| 4.3 | Отсутствие собственных ошибок проекта | Разработанное программное обеспечение лишено внутренних отказов. Примеры: зависания, доступ к несанкционированным ресурсам, утечки ресурсов |
| 4.4 | Простота и понятность. Предсказуемость поведения | Поведение разрабатываемого программного обеспечения предсказуемо объективным и убедительным тестированием и анализом |
| 4.5 | Верифицируемость и тестируемость проекта | Разработанное программное обеспечение является поддающимся проверке и тестируемым |
| 4.6 | Отказоустойчивость/Обнаружение неисправностей | Методы и разработки дают гарантию, что разработанное программное обеспечение будет вести себя безопасно в присутствии ошибок |
| 4.7 | Отсутствие отказов по общей причине | Методы и проекты идентифицируют виды отказа по общей причине и обеспечивают эффективные средства предостережения от отказов программного обеспечения |

Таблица F.5 — Проектирование и разработка программного обеспечения: тестирование и интеграция программных модулей (см.МЭК 61508-3, пункты 7.4.7, 7.4.8 и таблица С.5)

| | Свойство | Определение |
|---|---|---|
| 5.1 | Полнота тестирования и интеграции в соответствии со спецификациями проекта программного обеспечения | Тестирование программного обеспечения исследует поведение программного обеспечения с достаточной полнотой, чтобы гарантировать, что все требования спецификации проектирования программного обеспечения были удовлетворены |
| 5.2 | Корректность тестирования и интеграции в соответствии со спецификациями проекта программного обеспечения (успешное выполнение) | Если задача тестирования модуля завершена, то существует конкретное доказательство, позволяющее утверждать, что требования к системе безопасности были удовлетворены |

| | Свойство | Определение |
|---|---|---|
| 5.3 | Воспроизводимость | К согласованным результатам приводит повторение отдельных оценок, выполняемых как часть тестирования и интеграции модуля |
| 5.4 | Точно определенная тестируемая конфигурация | Для надлежащих версий элементов и программного обеспечения выполняют тестирование и интеграцию модуля, полученный требуемый результат связывают с конкретной конфигурацией «как — построено» программного обеспечения |

Таблица F.6 — Интеграция программируемых электронных устройств (программное обеспечение и аппаратные средства) (см.МЭК 61508-3, подраздел 7.5 и таблица С.6)

| | Свойство | Определение |
|---|---|---|
| 6.1 | Полнота интеграции в соответствии со спецификациями проекта | Интеграция обеспечивает соответствующую глубину и охват элементов системы, чтобы продемонстрировать, что система может выполнить функции, для которых она предназначена, и не выполняет непредусмотренные функции при всех возможных условиях эксплуатации и при отказе системы. Интеграция включает принципы, используемые для проверки, целевые уровни проекта и аспекты интеграции (например проверку полноты взаимодействия между модулями) |
| 6.2 | Корректность интеграции в соответствии со спецификациями проекта (успешное выполнение) | Интеграция основана на корректных предположениях.<br>Например, правильность ожидаемых результатов, рассматриваемых условий использования, а также репрезентативность тестовых сред.<br>Если задача интеграции завершена, то существует конкретное доказательство, позволяющее утверждать, что требования к безопасности были удовлетворены |
| 6.3 | Воспроизводимость | К согласованным результатам приводит повторение отдельных оценок, выполняемых, как часть интеграции модуля |
| 6.4 | Точно определенная конфигурация интеграции | Интеграция дает соответствующую гарантию, что она была эффективно применена в соответствии с документами к правильной версии элементов и программного обеспечения, а полученный требуемый результат связывают с конкретной конфигурацией «как — построено» программного обеспечения |

Таблица F.7 — Подтверждение соответствия для аспектов программного обеспечения системы безопасности (см.МЭК 61508-3, подраздел 7.7 и таблица С.7)

| | Свойство | Определение |
|---|---|---|
| 7.1 | Полнота подтверждения соответствия в соответствии со спецификацией проекта программного обеспечения | Подтверждение соответствия программного обеспечения охватывает все требования спецификации проектирования программного обеспечения |
| 7.2 | Корректность подтверждения соответствия в соответствии со спецификацией проекта программного обеспечения (успешное выполнение) | Если задача подтверждения соответствия программного обеспечения выполнена, то существует конкретное доказательство, позволяющее утверждать, что требования к безопасности были удовлетворены |
| 7.3 | Воспроизводимость | К согласованным результатам приводит повторение отдельных оценок, выполняемых как часть подтверждения соответствия программного обеспечения |
| 7.4 | Подтверждение соответствия точно определенной конфигурации | Четкое и краткое определение:<br>- системы,<br>- требований,<br>- окружающей среды |

Таблица F.8 — Модификация программного обеспечения (см.МЭК 61508-3, подраздел 7.8 и таблица С.8)

| | Свойство | Определение |
|---|---|---|
| 8.1 | Полнота модификации в соответствии с требованиями к модификации | Модификация была должным образом одобрена авторизованным персоналом, с соответствующим пониманием ее функционала, последствий для системы безопасности, а также технических и эксплуатационных последствий |
| 8.2 | Корректность модификации в соответствии с требованиями к модификации | Модификация достигает своих заданных целей |
| 8.3 | Отсутствие собственных ошибок проекта | Модификация не вносит новые систематические ошибки. Примеры: деление на ноль, выход индексов или указателей за границы своих значений, использование не инициализированных переменных |
| 8.4 | Предотвращение нежелательного поведения | Модификация не вносит какое-либо поведение, которое, согласно ограничениям, установленным в спецификации требований к программному обеспечению системы безопасности, должно быть предотвращено |
| 8.5 | Верифицируемость и тестируемость проекта | Проект программного обеспечения является таким, что влияние модификации полностью и всесторонне оценивается |
| 8.6 | Регрессионное тестирование и охват проверкой | Проект программного обеспечения является таким, что эффективное и полное регрессионное тестирование позволяет продемонстрировать, что программное обеспечение после модификации продолжает удовлетворять спецификации требований к программному обеспечению системы безопасности |

Таблица F.9 — Верификация программного обеспечения системы (см.МЭК 61508-3, подраздел 7.9 и таблица С.9)

| | Свойство | Определение |
|---|---|---|
| 9.1 | Полнота верификации в соответствии с предыдущей стадией | Верификация способна установить, что программное обеспечение удовлетворяет всем соответствующим требованиям спецификации требований к программному обеспечению системы безопасности |
| 9.2 | Корректность верификации в соответствии с предыдущей стадией (успешное выполнение) | Если задача верификации программного обеспечения завершена, то существует конкретное доказательство, позволяющее утверждать, что требования к системе безопасности были удовлетворены |
| 9.3 | Воспроизводимость | К согласованным результатам приводит повторение отдельных оценок, выполняемых как часть верификации |
| 9.4 | Верификация точно определенной конфигурации | Для надлежащих версий элементов и программного обеспечения выполняют верификацию, полученный требуемый результат связывают с конкретной конфигурацией «как — построено» программного обеспечения |

Таблица F.10 — Оценка функциональной безопасности (см.МЭК 61508-3, раздел 8 и таблица С.10)

| | Свойство | Определение |
|---|---|---|
| 10.1 | Полнота оценки функциональной безопасности в соответствии с настоящим стандартом | Оценка функциональной безопасности программного обеспечения формирует ясное утверждение о степени найденного соответствия, сделанных обоснованиях, мерах по устранению недостатков с рекомендуемыми сроками их устранения, полученные выводы и рекомендации по их принятию, квалифицированному принятию, или отклонению с указанием любых временных ограничений для этих рекомендаций |
| 10.2 | Корректность оценки функциональной безопасности в соответствии с проектными спецификациями (успешное выполнение) | Задача оценки функциональной безопасности программного обеспечения завершена, и существует конкретное доказательство, позволяющее утверждать, что требования к системе безопасности были удовлетворены |
| 10.3 | Доступное для анализа решение всех выявленных проблем | Существует ясное утверждение о том, в каком объеме были рассмотрены проблемы, возникающие во время оценки функциональной безопасности программного обеспечения. |
| 10.4 | Возможность модификации оценки функциональной безопасности после изменения проекта без необходимости проведения серьезной переработки оценки | Результаты оценки функциональной безопасности программного обеспечения могут быть перерассчитаны, причем при повторной оценке функциональной безопасности частей программного обеспечения после их изменения не выполняется повторная оценка функциональной безопасности всего программного обеспечения, она лишь корректируется с учетом измененных частей |
| 10.5 | Воспроизводимость | Оценка функциональной безопасности выполняется как согласованный, запланированный и открытый процесс с конкретными персоналом и документами, который реализует рассмотрение основания для выполнения оценок и решения, которые выполняют все те, на которых влияют эти решения, включая поставщиков системы, пользователей, специалистов по обслуживанию и руководство<br>Оценка функциональной безопасности позволяет независимому компетентному персоналу повторять отдельные оценки, выполненные как часть всей оценки |
| 10.6 | Своевременность | Оценка функциональной безопасности выполняется с соответствующей частотой, связанной со стадиями жизненного цикла программного обеспечения системы безопасности и по крайней мере до определения существующих опасностей. Также она обеспечивает своевременное создание отчетов о несоответствиях<br>Результаты тестов, проверок, исследований и т. д. фактически доступны, если они требуются в качестве входной информации для формирования решения об оценке |
| 10.7 | Точно определенная конфигурация | Результаты оценки функциональной безопасности программного обеспечения связывают с конкретной конфигурацией системы, для которой результаты оценки функциональной безопасности должны быть обоснованы |

**Приложение G**
**(справочное)**

## Руководство по разработке связанного с безопасностью объектно-ориентированного программного обеспечения

Все рекомендации настоящего стандарта по проектированию программного обеспечения применяются к объектно-ориентированному программному обеспечению. Поскольку объектно-ориентированный подход представляет информацию в отличие от процедурных или функциональных подходов по-другому, то далее даны рекомендации, для конкретного рассмотрения которых необходимо:

- понимание иерархий классов и идентификации функции(й) программного обеспечения, которые будут выполняться при вызове заданного метода (включая существующую библиотеку классов, если она используется);
- структурное тестирование (МЭК 61508-3, таблица В.2 и МЭК 61508-7, подраздел С.5.8).

Таблицы G.1 и G.2 содержат справочные руководящие указания по использованию объектно-ориентированного программного обеспечения, которые дополняют более общие нормативные руководящие указания, представленные в МЭК 61508-3, таблицы 2 и 4.

Таблица G.1 — Архитектура объектно-ориентированного программного обеспечения

| | Рекомендации | Подробности | УПБ1 | УПБ2 | УПБ3 | УПБ4 |
|---|---|---|---|---|---|---|
| G1.1 | Прослеживаемость понятия прикладной области с классами архитектуры | Примечание 1 | R | HR | HR | HR |
| G1.2 | Использование подходящих фреймов, общеиспользуемых комбинаций классов и шаблонов проектирования<br>Примечание — При использовании существующих фреймов и шаблонов проектирования к ним применяются требования, как и к предварительно разработанному программному обеспечению | Примечание 2 | R | HR | HR | HR |

Примечания:

1 Прослеживаемость прикладной области с архитектурой класса менее важна.

2 ***Примеры***

***1 Для некоторой части, предназначенной для проекта, связанного с безопасностью, может существовать фрейм из проекта, не связанного с безопасностью, который успешно решает подобную задачу, и это хорошо известно участникам проекта. В этом случае рекомендуется использовать такой фрейм.***

***2 Могут понадобиться различные алгоритмы для решения тесно связанных подзадач проекта, связанного с безопасностью. Поэтому может быть выбран шаблон стратегии доступа к алгоритмам.***

***3 Часть проекта, связанного с безопасностью, может состоять из выдачи надлежащих предупреждений внутренним и внешним заинтересованным сторонам. Для организации этих предупреждений может быть выбран шаблон наблюдателя. Это требование не применяется для библиотек.***

3 Как правило, абстрактный базовый класс обеспечивает доступ к производным конкретным классам.

Таблица G.2 — Объектно-ориентированный рабочий проект

| | Рекомендации | УПБ1 | УПБ2 | УПБ3 | УПБ4 |
|---|---|---|---|---|---|
| G2.1 | Классы должны иметь только одну цель | R | R | HR | HR |
| G2.2 | Наследование используется, только если производный класс является уточнением своего основного класса | HR | HR | HR | HR |
| G2.3 | Глубина наследования ограничена стандартами кодирования | R | R | HR | HR |
| G2.4 | Переопределение операций (методов) строго контролируется | R | HR | HR | HR |
| G2.5 | Множественное наследование используется только для интерфейсных классов | HR | HR | HR | HR |
| G2.6 | Наследование от неизвестных классов | — | — | NR | NR |

*Окончание таблицы G.2*

| | Рекомендации | УПБ1 | УПБ2 | УПБ3 | УПБ4 |
|---|---|---|---|---|---|
| G2.7 | Подтверждение того, что повторно используемые объектно-ориентированные библиотеки отвечают рекомендациям данной таблицы | HR | HR | HR | HR |

Примечания:

1 Один класс характеризуется наличием одной ответственности — необходимо быть внимательным при описании тесно связанных данных и операций над этими данными.

2 Необходимо внимательно следить за тем, чтобы не допустить циклических зависимостей между объектами.

Далее неформально определены термины, использованные выше.

Таблица G.3 — Некоторые объектно-ориентированные термины

| Термин | Неформальное определение |
|---|---|
| Основной класс | Класс, у которого есть производные классы. Основной класс иногда называют надклассом, или родительским классом |
| Производный класс | Класс (совокупность атрибутов и операций), который наследовал атрибуты и/или операции от другого класса (основного класса). Производный класс иногда называют подклассом, или дочерним классом |
| Фрэйм | Структура программы, как правило, предварительно разработана для заполнения данными конкретного применения |
| Переопределение | Замена операции (метода, подпрограммы) другой операцией (методом, подпрограммой) с той же самой сигнатурой и иерархией наследования во время выполнения; свойство объектно-ориентированных языков или программ, реализующее полиморфизм |
| Сигнатура операции | Имя операции (подпрограммы, метода) вместе с ее параметрами (аргументами) и их типы, иногда также их возвращаемые типы. Две сигнатуры равны, если у них одни и те же имена, число и типы параметров; в некоторых языках должны быть одинаковыми также возвращаемые типы |

**Приложение ДА**
**(справочное)**

## Сведения о соответствии ссылочных международных стандартов ссылочным национальным стандартам Российской Федерации

Таблица ДА.1

| Обозначение ссылочного международного стандарта | Степень соответствия | Обозначение и наименование соответствующего национального стандарта Российской Федерации |
|---|---|---|
| ИСО/МЭК Руководство 51:1990 | IDT | ГОСТ Р 51898—2002 «Аспекты безопасности. Правила включения в стандарты» |
| МЭК Руководство 104:1997 | — | * |
| МЭК 61508-1:2010 | IDT | ГОСТ Р МЭК 61508-1—2012 «Функциональная безопасность систем электрических, электронных, программируемых электронных, связанных с безопасностью. Часть 1. Общие требования» |
| МЭК 61508-2:2010 | IDT | ГОСТ Р МЭК 61508-2—2012 «Функциональная безопасность систем электрических, электронных, программируемых электронных, связанных с безопасностью. Часть 2. Требования к системам» |
| МЭК 61508-3:2010 | IDT | ГОСТ Р МЭК 61508-3—2012 «Функциональная безопасность систем электрических, электронных, программируемых электронных, связанных с безопасностью. Часть 3. Требования к программному обеспечению» |
| МЭК 61508-4:2010 | IDT | ГОСТ Р МЭК 61508-4—2012 «Функциональная безопасность систем электрических, электронных, программируемых электронных, связанных с безопасностью. Часть 4. Термины и определения» |

* Соответствующий национальный стандарт отсутствует. До его утверждения рекомендуется использовать перевод на русский язык данного международного стандарта. Перевод данного международного стандарта находится в Федеральном информационном фонде технических регламентов и стандартов.

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

- IDT — идентичные стандарты.

## Библиография


[1] IEC 61511 (all parts), Functional safety — Safety instrumented systems for the process industry sector

[2] IEC 62061:2005, Safety of machinery — Functional safety of safety-related electrical, electronic and programmable electronic control systems

[3] IEC 61800-5-2, Electromagnetic compatibility (EMC) — Part 5: Installation and mitigation guidelines — Section 2: Earthing and cabling

[4] IEC 60601 (all parts)Medical electrical equipment

[5] IEC 61326-3-1:2008, Electrical equipment for measurement, control and laboratory use — EMC requirements — Part 3-1: Immunity requirements for safety-related systems and for equipment intended to perform safety-related functions (functional safety) — General industrial applications

[6] IEC 61326-3-2:2008, Electrical equipment for measurement, control and laboratory use — EMC requirements — Part 3-2: Immunity requirements for safety-related systems and for equipment intended to perform safety-related functions (functional safety) — Industrial applications with specified electromagnetic environment

[7] IEC 61131-3:2003, Programmable controllers — Part 3: Programming languages

[8] IEC 60812:2006, Analysis techniques for system reliability — Procedure for failure mode and effects analysis (FMEA)

[9] IEEE 352:1987, IEEE Guide for general principles of reliability analysis of nuclear power generating station safety systems

[10] IEC 61164:2004, Reliability growth — Statistical test and estimation methods

[11] Verification and Validation of Real-Time Software, Chapter 5. W.J. Quirk (ed.). Springer Verlag, 1985, ISBN 3-540-15102-8.

[12] Combining Probabilistic and Deterministic Verification Efforts. W.D. Ehrenberger, SAFECOMP 92, Pergamon Press, ISBN 0-08-041893-7.

[13] Ingenieurstatistik. Heinhold/Gaede, Oldenburg, 1972, ISBN 3-486-31743-1.

[14] IEC 60068-2-1, Environmental testing — Part 2-1: Tests — Test A: Cold

[15] IEC 60068-2-2, Environmental testing — Part 2-2: Tests — Test B: Dry heat

УДК 62-783:614.8:331.454:006.354 ОКС 25.040.40, 35.240.50 Т51

Ключевые слова: функциональная безопасность, жизненный цикл систем, электрические компоненты, электронные компоненты, программируемые электронные компоненты и системы, системы, связанные с безопасностью, случайные отказы оборудования, систематические отказы, планирование функциональной безопасности, методы и средства, полнота безопасности

Редактор *Т.С. Никифорова*
Технический редактор *А.И. Белов*
Корректор *Г.Н. Старкова*
Компьютерная верстка *А.С. Шаповаловой*

Сдано в набор 08.04.2014. Подписано в печать 06.05 2014 Формат 60×84⅛. Гарнитура Ариал.
Усл. печ. л. 11,63. Уч.-изд. л. 9,30. Тираж 66 экз. Зак. 1253.

Набрано в Издательском доме «Вебстер»
www.idvebster.ru project@idvebster.ru
Отпечатано в Калужской типографии стандартов, 248021 Калуга, ул. Московская, 256.

gost.e-ecolog.ru