ФЕДЕРАЛЬНОЕ АГЕНТСТВО

ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ И МЕТРОЛОГИИ

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСТ Р МЭК 62279— 2016

Железные дороги

СИСТЕМЫ СВЯЗИ, СИГНАЛИЗАЦИИ И ОБРАБОТКИ ДАННЫХ. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ СИСТЕМ УПРАВЛЕНИЯ И ЗАЩИТЫ НА ЖЕЛЕЗНЫХ ДОРОГАХ

(IEC 62279:2015, IDT)

Издание официальное

Москва
Стандартинформ
2017

Предисловие

1 ПОДГОТОВЛЕН Обществом с ограниченной ответственностью «Корпоративные электронные системы» на основе собственного перевода на русский язык англоязычной версии 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 России от 30 ноября 2016 г. № 1881-ст

4 Настоящий стандарт идентичен международному стандарту МЭК 62279:2015 «Железные дороги. Системы связи, сигнализации и обработки данных. Программное обеспечение систем управления и защиты на железных дорогах» (IEC 62279:2015, «Railway applications — Communication, signalling and processing systems — Software for railway control and protection systems», IDT).

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

5 ВВЕДЕН ВПЕРВЫЕ

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

Содержание

Введение

Настоящий стандарт входит в группу связанных стандартов вместе с МЭК 62278:2002 «Железные дороги. Технические условия и демонстрация надежности, эксплуатационной готовности, ремонтопригодности и безопасности» и МЭК 62425:2007 «Железные дороги. Системы связи, сигнализации и обработки данных. Связанные с безопасностью электронные системы сигнализации».

МЭК 62278:2002 рассматривает проблемы крупных систем, а в МЭК 62425:2007 рассмотрен порядок утверждения отдельных систем, которые могут существовать внутри общей системы управления и защиты на железнодорожном транспорте. Настоящий стандарт уделяет внимание практическим методам создания программного обеспечения, удовлетворяющего требованиям полноты безопасности, которые определяются в результате анализа таких крупных систем.

Настоящий стандарт обеспечивает ряд требований, которым должны соответствовать разработка, внедрение и обслуживание любого связанного с безопасностью программного обеспечения, предназначенного для приложений управления и защиты на железных дорогах. Он определяет требования, связанные с организационной структурой, отношением между организациями и разделением ответственности, которые учитываются при разработке, внедрении и в операциях по техобслуживанию. Критерии квалификации и опыта персонала также рассмотрены в настоящем стандарте.

Ключевое понятие настоящего стандарта — это понятие уровней полноты безопасности программного обеспечения. Настоящий стандарт рассматривает пять уровней полноты безопасности программного обеспечения, где УПБ 0 является самым низким, а УПБ 4 — самым высоким связанным с безопасностью уровнем полноты. Чем выше риск, возникающий из-за ошибки в программном обеспечении, тем выше уровень полноты безопасности программного обеспечения будет. Чем более опасны последствия ошибки программного обеспечения, тем будет выше его уровень полноты безопасности.

Настоящий стандарт определяет методы и меры для пяти уровней полноты безопасности программного обеспечения. Требуемые методы и меры для каждого из пяти уровней полноты безопасности программного обеспечения представлены в нормативных таблицах приложения А. В настоящей версии требуемые методы для уровня 1 совпадают с методами для уровня 2, а требуемые методы для уровня 3 совпадают с методами для уровня 4. Настоящий стандарт не дает рекомендаций о том, какой уровень полноты программного обеспечения подходит для данного риска. Это решение будет зависеть от многих факторов, включая природу приложения, насколько другие системы выполняют функции безопасности, также от социально-экономических факторов.

Определение процесса спецификации функций безопасности, выполняемых программным обеспечением, входит в область применения МЭК 62278 и МЭК 62425.

Настоящий стандарт определяет те меры, которые необходимы для достижения таких требований.

МЭК 62278 и МЭК 62425 требуют, чтобы был использован систематический подход:

a) для определения опасностей, оценки рисков и получения решения на основе критериев риска;

b) определения необходимого снижения риска, удовлетворяющего критериям риска;

c) определения спецификации требований для всей системы безопасности, необходимых для гарантированного достижения требуемого снижения риска;

d) выбора подходящей архитектуры системы;

e) планирования, контроля и управления техническими и организационными действиями, необходимыми для реализации спецификации требований к безопасности в системе, связанной с безопасностью, для которой выполняется подтверждение соответствия полноте безопасности.

Поскольку проект декомпозируется и формируется спецификация для отдельных связанных с безопасностью систем и их компонентов, то для них далее выполняется определение уровней полноты безопасности. В конечном счете, это приводит к требуемым уровням полноты безопасности для программного обеспечения.

Современный уровень развития техники таков, что никакое применение методов гарантии качества (так называемых мер предотвращения сбоев и выявления сбоев), ни применение подходов, основанных на отказоустойчивом программном обеспечении, не могут гарантировать абсолютную безопасность системы. Нет никакого известного способа доказать отсутствие сбоев в довольно сложном, связанном с безопасностью программном обеспечении, особенно отсутствие сбоев в проекте и спецификации.

Принципы, применяемые при разработке программного обеспечения с высокими требованиями по безопасности, включают в себя, но не ограничены:

- нисходящие методы разработки;
- модульный принцип;
- проверка каждой стадии жизненного цикла разработки;
- проверенные модули и библиотеки модулей;
- понятная документация и прослеживаемость;
- аудирование документов;
- подтверждение соответствия;
- оценка;
- управление конфигурацией и управление изменениями;
- надлежащее рассмотрение проблем компетентности организации и персонала.

Спецификация требований безопасности к системе идентифицирует все функции безопасности, определенные для программного обеспечения, и определяет их уровни полноты безопасности. В соответствии с настоящим стандартом, как показано на рисунке 1, последовательно выполняются следующие функциональные шаги:

a) определение спецификации требований для программного обеспечения и параллельно рассмотрение архитектуры программного обеспечения. В рамках формирования архитектуры программного обеспечения разрабатывается основная стратегия безопасности для программного обеспечения и его уровня полноты безопасности (см. 7.2 и 7.3);

b) проектирование, разработка и тестирование программного обеспечения выполняется согласно плану обеспечения качества программного обеспечения, уровню полноты безопасности программного обеспечения и жизненному циклу программного обеспечения (см. 7.4 и 7.5);

c) реализация интеграции программного обеспечения и программного обеспечения с аппаратными средствами на целевых аппаратных средствах, а также проверка функциональности (см. 7.6);

d) выполнение приемки и внедрение программного обеспечения (см. 7.7 и 9.1);

e) если в период эксплуатации программного обеспечения требуется его поддержка, то повторно выполняются надлежащие процедуры настоящего стандарта (см. 9.2).

С разработкой программного обеспечения связаны несколько действий таких, как тестирование (см. 6.1), проверка (см. 6.2), подтверждение соответствия (см. 6.3), оценка (см. 6.4), контроль качества (см. 6.5), а также управление модификацией и изменениями (6.6).

Представлены требования к средствам поддержки (см. 6.7) и системам, которые сконфигурированы с помощью данных приложения или алгоритмов (см. 8).

Также даны требования к независимости ролей и компетентности штата, участвующего в разработке программного обеспечения (см. 5.1, 5.2 и приложение B).

Настоящий стандарт не рассматривает использование конкретного жизненного цикла разработки программного обеспечения. Однако в 5.3, рисунки 3 и 4, и 7.1 представлены иллюстративный пример жизненного цикла и необходимые комплекты документации.

В результате оценки различных методов/мер для пяти уровней полноты безопасности программного обеспечения были сформированы таблицы, представленные в приложении A. В этих таблицах есть соответствующие ссылки на библиографию, дающую краткое описание каждого метода/меры со ссылками на дальнейшие источники информации. Библиография методов представлена в приложении D.

Рисунок 1 — Маршрутная карта программного обеспечения

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

Железные дороги

СИСТЕМЫ СВЯЗИ, СИГНАЛИЗАЦИИ И ОБРАБОТКИ ДАННЫХ. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ СИСТЕМ УПРАВЛЕНИЯ И ЗАЩИТЫ НА ЖЕЛЕЗНЫХ ДОРОГАХ

Railway applications. Communication, signalling and processing systems.
Software for railway control and protection systems

Дата введения — 2018—01—01

1 Область применения

1.1 Настоящий стандарт определяет процедуры и технические требования для разработки программируемых электронных систем для использования в приложениях управления и защиты на железных дорогах. Настоящий стандарт нацелен на использование в любой области, где важна безопасность. Такие системы могут быть реализованы, используя специализированные микропроцессоры, программируемые логические контроллеры, многопроцессорные распределенные системы, системы на базе центрального процессора более широкого масштаба или другую архитектуру.

1.2 Настоящий стандарт применим исключительно к программному обеспечению и взаимодействию между программным обеспечением и системой, частью которой оно является.

1.3 Настоящий стандарт не применяется для программного обеспечения, которое было идентифицировано, как не оказывающее влияние на безопасность, т. е. для программного обеспечения, отказы которого не могут влиять ни на какие идентифицированные функции безопасности. Понятие УПБ 0 введено потому, что при оценке риска присутствует неопределенность и даже при идентификации опасностей. По крайней мере, требования УПБ 0 настоящего стандарта выполнены для программного обеспечения, оказывающего влияние на безопасность функций, у которых значение УПБ ниже УПБ 1.

1.4 Настоящий стандарт применяется ко всему связанному с безопасностью программному обеспечению, используемому в системах управления и защиты на железных дорогах, включая:

- прикладное программирование;
- операционные системы;
- инструменты поддержки;
- встроенное микропрограммное обеспечение.

Прикладное программирование включает высокоуровневое программирование, низкоуровневое программирование и специальное программирование (например, лестничная логика программируемого логического контроллера).

1.5 Настоящий стандарт также рассматривает использование существующего ранее программного обеспечения и инструментальных средств. Такое программное обеспечение может использоваться, если выполнены конкретные требования 7.3.4.7 и 6.5.4.16 для уже существующего программного обеспечения и требования 6.7 для инструментальных средств.

1.6 Программное обеспечение, разработанное в соответствии с любой редакцией настоящего стандарта, будет считаться как удовлетворяющее соответствующим требованиям и не должно рассматриваться как ранее существующее программное обеспечение.

Издание официальное

1.7 Настоящий стандарт полагает, что современное проектирование приложений часто использует универсальное программное обеспечение, которое является базовым для различных приложений. Такое универсальное программное обеспечение конфигурируется данными, алгоритмами или тем и другим при создании исполнимого программного обеспечения для приложения. Разделы 1—6 и 9 настоящего стандарта применяются к универсальному программному обеспечению, а также к данным приложения или алгоритмам. Раздел 7 применяется только для универсального программного обеспечения, а раздел 8 содержит конкретные требования для данных приложения или алгоритмов.

1.8 Настоящий стандарт не предназначен для решения коммерческих проблем. Они должны быть основной частью любого договорного соглашения. Но в любой коммерческой ситуации необходимо внимательно рассмотреть все разделы настоящего стандарта.

1.9 Настоящий стандарт не предполагает рассматривать существующие системы. Поэтому он применяется, прежде всего, к новым разработкам и только полностью применяется к существующим системам, если они подвергаются существенным модификациям. Для незначительных изменений применяется только 9.2. Оценщик анализирует доказательства, представленные в документации на программное обеспечение, чтобы подтвердить, соответствуют ли они характеру и области применения изменений программного обеспечения. Однако настоятельно рекомендуется применять настоящий стандарта во время обновлений и сопровождения существующего программного обеспечения.

2 Нормативные ссылки

В настоящем стандарте использованы следующие международные стандарты. Для датированных ссылок применяют только указанное издание ссылочного стандарта. Для недатированных ссылок применяют последнее издание ссылочного стандарта.

IEC 62278:2002, Railway applications — Specification and demonstration of reliability, availability, maintainability and safety (RAMS) (Железные дороги. Технические условия и демонстрация надежности, эксплуатационной готовности, ремонтопригодности и безопасности (RAMS))

ISO/IEC 90003:2014, Software engineering — Guidelines for the application of ISO 9001:2008 to computer software (Разработка программных продуктов. Руководящие указания по применению ИСО 9001:2008 при разработке программных продуктов)

ISO/IEC 25010 series, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models (Информационные технологии. Системная и программная инженерия. Требования и оценка качества систем и программного обеспечения (SQuaRE). Модели качества систем и программных продуктов)

ISO 9000, Quality management systems — Fundamentals and vocabulary (Системы менеджмента качества. Основные положения и словарь)

ISO 9001:2008, Quality management systems — Requirements (Системы менеджмента качества. Требования)

3 Термины, определения и сокращения

3.1 Термины и определения

В настоящем стандарте применены следующие термины с соответствующими определениями.

3.1.1 **оценка** (assessment): Процесс анализа, который выполняется, чтобы определить, удовлетворяет ли конкретным требованиям программное обеспечение, и который может включать анализ процесса, документации, системы, компонентов аппаратных средства и/или программного обеспечения подсистемы, а также, чтобы сформировать мнение о пригодности программного обеспечения своей предназначенной цели.

Примечание — Оценивание безопасности направлено, но не ограничивается, на оценку свойств безопасности системы.

3.1.2 **оценщик** (assessor): Субъект, выполняющий оценку.

3.1.3 **коробочное программное обеспечение** (commercial off-the-shelf (COTS) software): Программное обеспечение, определенное потребностью рынка, коммерчески доступное, пригодность для определенной цели которого было продемонстрировано широким спектром коммерческих пользователей.

3.1.4 **компонент** (component): Составная часть программного обеспечения, которая имеет хорошо определенные интерфейсы и ее поведение соответствует проекту и архитектуре программного обеспечения.

Примечание — Компонент программного обеспечения удовлетворяет следующим критериям:

- он разработан согласно «Компонентам» (см. таблицу А.20);
- он реализует определенное подмножество требований к программному обеспечению;
- он четко идентифицирован и имеет независимую версию в системе управления конфигурацией или является частью набора компонентов (например, подсистемы), у которого есть независимая версия.

3.1.5 **менеджер конфигураций** (configuration manager): Ответственный за реализацию и выполнение процессов управления конфигурацией документов, программного обеспечения и связанных инструментов, включая управление изменениями.

3.1.6 **заказчик** (customer): Субъект, который покупает железнодорожную систему управления и систему защиты включая программное обеспечение.

3.1.7 **проектировщик** (designer): Субъект, который анализирует и преобразовывает конкретные требования в приемлемые проектные решения, имеющие требуемый уровень полноты безопасности.

3.1.8 **субъект** (entity): Человек, группа или организация, которые выполняют роль, как определено в настоящем стандарте.

3.1.9 **сбой** (fault): Аварийное состояние, которое может привести к ошибке в системе.

Примечание — Сбой может быть случайным или систематическим.

3.1.10 **ошибка** (error): Отклонение от намеченного проекта, которое может привести к непреднамеренному поведению системы или отказу.

3.1.11 **отказ** (failure): Недопустимое различие между требуемой и наблюдаемой характеристикой.

3.1.12 **отказоустойчивость** (fault tolerance): Встроенная способность системы обеспечить дальнейшее корректное оказание услуги согласно спецификации, в присутствии ограниченного количества сбоев аппаратных средств или программного обеспечения.

3.1.13 **встроенное микропрограммное обеспечение** (firmware): Программное обеспечение, размещенное в постоянной или в полупостоянной памяти (например такой как флэш-память), так, что оно функционально не зависит от прикладного программного обеспечения.

3.1.14 **универсальное программное обеспечение** (generic software): Программное обеспечение, которое может использоваться для множества установок просто благодаря появлению возможности определять конкретные для применения данные и/или алгоритмы.

3.1.15 **разработчик** (implementer): Субъект, который преобразует конкретные проекты в их физическую реализацию.

3.1.16 **интеграция** (integration): Процесс объединения программного обеспечения и/или элементов аппаратных средств в соответствии со спецификацией архитектуры и проекта, также тестирование интегрированного устройства.

3.1.17 **интегратор** (integrator): Субъект, который выполняет интеграцию программного обеспечения.

3.1.18 **существующее ранее программное обеспечение** (pre-existing software): Программное обеспечение, разработанное до рассматриваемого в настоящее время применения, включающее коммерческое программное обеспечение и открытое программное обеспечение.

3.1.19 **открытое программное обеспечение** (open source software): Доступный широкой публике исходный код с ослабленными или отсутствующими ограничениями на авторские права.

3.1.20 **программируемый логический контроллер** (programmable logic controller): Полупроводниковая система управления с программируемой памятью пользователя для хранения команд, осуществляющих конкретные функции.

3.1.21 **управление проектами** (project management): Административное и/или техническое выполнение проекта, включая вопросы безопасности.

3.1.22 **менеджер проекта** (project manager): Субъект, который выполняет управление проектом.

3.1.23 **надежность** (reliability): Способность элемента выполнить требуемую функцию при заданных условиях в течение установленного периода времени.

3.1.24 **робастность** (robustness): Способность элемента обнаруживать и обрабатывать ненормальные ситуации.

3.1.25 **менеджер требований** (requirements manager): Субъект, выполняющий управление требованиями.

3.1.26 **управление требованиями** (requirements management): Процесс выявления, документального оформления, анализа, формирования приоритетов и достижения соглашения о требованиях, и далее управления изменением и сообщения соответствующим заинтересованным сторонам.

Примечание — Управление требованием — непрерывный процесс в течение выполнения проекта.

3.1.27 **риск** (risk): Сочетание вероятности возникновения несчастных случаев и инцидентов, наносящих ущерб (вызванных опасностью) и тяжести этого ущерба.

3.1.28 **безопасность** (safety): Отсутствие неприемлемого риска опасности для человека.

3.1.29 **уполномоченный орган по безопасности** (safety authority): Организация, ответственная за подтверждение того, что связанные с безопасностью программное обеспечение или услуги удовлетворяют соответствующим установленным требования безопасности.

3.1.30 **функция безопасности** (safety function): Функция, которая реализует часть или все требование безопасности.

3.1.31 **связанное с безопасностью программное обеспечение** (safety-related software): Программное обеспечение, которое выполняет функции безопасности.

3.1.32 **программное обеспечение** (software): Интеллектуальный продукт, включающий программы, процедуры, правила, данные и любую связанную документацию, относящуюся к работе системы.

3.1.33 **базовая конфигурация программного обеспечения** (software baseline): Завершенный и согласованный набор исходных кодов, исполняемых файлов, конфигурационных файлов, инсталляционных подлинников и документации, которые необходимы для выпуска программного обеспечения.

Примечание — Информация о компиляторах, операционных системах, существующем ранее программном обеспечении и зависимых инструментах хранится как часть базовой конфигурации. Она позволяет организации воспроизвести определенные версии и быть входом для будущих версий при улучшениях или при модернизации на стадии обслуживания.

3.1.34 **развертывание программного обеспечения** (software deployment): Передача, установка и активизация поставляемой базовой версии программного обеспечения, которое было уже выпущено и оценено.

3.1.35 **жизненный цикл программного обеспечения** (software life cycle): Действия, происходящие в течение промежутка времени, который начинается, когда программное обеспечение задумано и заканчивается, когда программное обеспечение больше не доступно для использования.

Примечание — Жизненный цикл программного обеспечения, как правило, включает стадии формирования требований, проектирования, испытаний, интеграции, развертывания и сопровождения.

3.1.36 **сопровождаемость программного обеспечения** (software maintainability): Способность программного обеспечения быть модифицируемым для исправления ошибок, улучшать свою производительность или другие характеристики, или адаптироваться к различным внешним условиям.

3.1.37 **сопровождение программного обеспечения** (software maintenance): Действие или набор действий, выполняемых с программным обеспечением после его внедрения с целью усовершенствования или коррекции его функциональности.

3.1.38 **уровень полноты безопасности программного обеспечения** (software safety integrity level): Классификационный индекс, который определяет методы и меры, которые должны быть применены к программному обеспечению.

Примечание — Связанное с безопасностью программное обеспечение было классифицировано на пять уровней полноты безопасности, где 0 является самым низким, а 4 — самым высоким.

3.1.39 **поставщик** (supplier): Субъект, который проектирует и создает систему управления и защиты на железных дорогах, включая программное обеспечение, или ее части.

3.1.40 **уровень полноты безопасности системы** (system safety integrity level); Классификационный индекс, который указывает необходимую степень уверенности, что интегрированная система, включающая аппаратные средства и программное обеспечение, удовлетворяет заданным для нее требованиям безопасности.

3.1.41 **тестировщик** (tester): Субъект, который выполняет тестирование.

3.1.42 **тестирование** (testing): Процесс выполнения программного обеспечения при заданных условиях для установления его поведения и работоспособности при сравнении с соответствующей спецификацией требований.

3.1.43 **класс инструментальных средств Т1** (tool class T1): Созданный инструментом результат не может прямо или косвенно повлиять на исполнимый код (включая данные) программного обеспечения.

Примечание — Примеры класса Т1 включают: редактор текста либо инструментальное средство проектирования без возможностей автоматической генерации объектного кода; средства управления конфигурацией.

3.1.44 **класс инструментальных средств Т2** (tool class T2): Инструментальные средства, поддерживающие тестирование или проверку проекта или исполнимого кода, ошибки в которых не позволяют выявить дефекты, но они не могут непосредственно создавать ошибки в исполнимом программном обеспечении.

Примечание — Примеры класса Т2 включают: генератор испытания ремня безопасности; инструментальное средство измерения тестового охвата; инструментальное средство статического анализа.

3.1.45 **класс инструментальных средств Т3** (tool class T3): Созданный инструментом результат может прямо или косвенно повлиять на исполнимый код (включая данные) связанной с безопасностью системы.

Примечание — Примеры класса Т3 включают: компилятор исходного кода, компилятор данных/алгоритмов, инструмент для изменения уставок во время работы системы; оптимизирующий компилятор, где отношения между исходным кодом программы и сгенерированным объектным кодом не очевидны; компилятор, который включает выполняемый программный пакет времени выполнения в выполнимый код.

3.1.46 **прослеживаемость** (traceability): Степень, с которой может быть установлено отношение между двумя или более результатами процесса разработки, особенно между имеющими предшественника/преемника или отношение основной/зависимый между собой.

3.1.47 **подтверждение соответствия** (validation): Процесс анализа, за которым следует основанное на доказательстве обоснование, при определении того, соответствует ли элемент (например, процесс, документация, программное обеспечение или применение) потребностям пользователя, в особенности относительно безопасности и качества и с учетом пригодности его работы в соответствии с его целью в намеченном для него окружении.

3.1.48 **менеджер по подтверждению соответствия** (validator): Субъект, ответственный за подтверждение соответствия.

3.1.49 **проверка** (verification): Процесс экспертизы, за которым следует основанное на доказательстве обоснование того, что выходные элементы (процесс, документация, программное обеспечение или приложение) конкретной стадии разработки, удовлетворяют требованиям этой стадии, связанным с полнотой, правильностью и согласованностью.

Примечание — Проверка главным образом основана на рецензировании выходных документов (проекта, реализации, документов испытаний и т. д.).

3.1.50 **менеджер по проверке** (verifier): Субъект, ответственный за одно или более действий проверки.

3.2 Сокращения

ASR — оценщик;

COTS — коробочное программное обеспечение;

CGM — менеджер конфигураций;

DES — проектировщик;

HR — настоятельно рекомендуемый;

IMP — разработчик;

INT — интегратор;

JSD — метод JSD;

M — обязательный;

MASCOT — модульный подход к созданию программного обеспечения, выполнению и тестированию;

NR — не рекомендовано;

PM — менеджер проекта;

QAM — менеджер контроля качества;

R — рекомендовано;

RAMS — безотказность, готовность, ремонтопригодность и безопасность;

RQM — менеджер требований;

SDL — язык спецификации и описания;

SFC — последовательные функциональные схемы;

SIL — уровень полноты безопасности;

SOM — моделирование, ориентированное на сервисы;

SSADM — методика структурного анализа и проектирования систем;

TST — тестировщик;

V&V — проверка и подтверждение соответствия;

VAL — менеджер по подтверждению соответствия;

VER — менеджер по проверке.

4 Цели, соответствие и уровни полноты безопасности программного обеспечения

4.1 Распределение связанных с безопасностью системных функций программному обеспечению, а также интерфейсы программного обеспечения, должны быть определены в документации к системе. Для системы, в которую включено программное обеспечение, должно быть полностью определено следующее:

- функции и интерфейсы;
- условия применения;
- конфигурация или архитектура системы;
- опасности, которыми необходимо управлять;
- требования полноты безопасности;
- распределение требований и распределение УПБ для программного обеспечения и аппаратных средств;
- ограничения синхронизации.

Примечание — Распределение требований полноты безопасности может привести к различным значениям УПБ для хорошо разделенных частей программного обеспечения и аппаратных средств подсистемы. Это распределение зависит от вклада частей программного обеспечения и аппаратных средств подсистемы в связанные с безопасностью функции и от механизмов ослабления отказов, включая разделение функций с различными УПБ.

4.2 Полнота программного обеспечения должна быть определена как один из пяти уровней от УПБ 0 (самый низкий) до значений полноты безопасности от УПБ 1 до УПБ 4.

4.3 Необходимый уровень полноты безопасности программного обеспечения должен быть определен и оценен на уровне системы, на основе уровня полноты безопасности системы и уровня риска, связанного с использованием программного обеспечения в системе.

4.4 Требования УПБ 0 в настоящем стандарте должны быть выполнены для части программного обеспечения функций, которые имеют степень неуверенности о влиянии на безопасность, как правило, ниже УПБ 1.

4.5 Чтобы соответствовать настоящему стандарту, нужно показать, что удовлетворены определенные в каждом подразделе требования по отношению к уровню полноты безопасности программного обеспечения и методам и мерам, определенным в приложении А.

4.6 Если требование определено словами «до степени, требуемой уровнем полноты безопасности программного обеспечения», то это указывает, что должен использоваться целый ряд методов и мер, чтобы удовлетворить это требование.

4.7 Если применен 4.6, то для помощи в выборе методов и мер, соответствующих уровню полноты безопасности программного обеспечения, должны использоваться таблицы из приложения А. Выбор должен быть зарегистрирован в плане обеспечения качества программного обеспечения или в другом документе, на который ссылается план обеспечения качества программного обеспечения. Указания к этим методам даются в приложении D.

4.8 Если метод или мера, которая оценивается в таблицах как настоятельно рекомендуемая (HR), не используется, то должно быть дано подробное объяснение для использования альтернативных методов и сделана запись или в плане обеспечения качества программного обеспечения или в другом документе, на который ссылается план обеспечения качества программного обеспечения. В этом нет необходимости, если используется одобренная комбинация методов, данная в соответствующей таблице. Выбранные методы должны быть продемонстрированы для их правильного применения.

4.9 Если предложенный для использования метод или мера не содержится в таблицах, то его эффективность и пригодность соответствовать определенному требованию и главной цели подраздела должны быть обоснованы и зарегистрированы или в плане обеспечения качества программного обеспечения или в другом документе, на который ссылается план обеспечения качества программного обеспечения.

4.10 Соответствие требованиям определенного подраздела и их соответствующим методам и мерам, подробно описанным в таблицах, должно быть проверено с помощью контроля документов, требуемого настоящим стандартом. Где это необходимо, должны также быть учтены другие объективные доказательства, проверка работы и наблюдение за проведением испытаний.

5 Организация и управление программным обеспечением

5.1 Организация, роли и обязанности

5.1.1 Цель

5.1.1.1 Гарантировать, что весь персонал, у кого есть обязанности в области программного обеспечения, был организован, имел полномочия и был способен к выполнению своих обязанностей.

Примечание — Независимость между ролями может потребоваться для снижения возможности людей в различных ролях, страдающих от тех же самых неправильных представлений или делающих те же самые ошибки. Эта форма независимости может быть достигнута, используя различных людей в различных ролях, но обычно не требует, чтобы роли были расположены в различных частях организации или в различных компаниях (если специально не требуется). Также важно, что люди в ролях, которые включают формирование суждения о приемлемости изделия или процесса с точки зрения безопасности, не должны быть под влиянием их коллег или непосредственных руководителей, или коммерческой выгоды. Эта форма независимости, более вероятно, потребует, чтобы различные роли были расположены в различных частях организации или были расположены в другой компании. В целом большая степень безопасности требует большей степени независимости для различных ролей и организации.

5.1.2 Требования

5.1.2.1 Как минимум, поставщик должен реализовать разделы ИСО 9001:2008, посвященные организации и управлению персоналом и обязанностям.

5.1.2.2 Обязанности должны быть совместимы с требованиями, определенными в приложении В.

5.1.2.3 Персонал, назначенный на роли, включенные в разработку или поддержку программного обеспечения, должен быть определен и зарегистрирован.

5.1.2.4 Оценщик должен быть назначен поставщиком, клиентом или полномочным органом по безопасности.

5.1.2.5 Оценщик должен быть независим от поставщика. Однако на усмотрение полномочного органа по безопасности оценщик может быть из организации поставщика или организации заказчика, но не должен участвовать в разработке проекта.

5.1.2.6 Оценщик должен быть независим от проекта.

5.1.2.7 Оценщику необходимо обладать полномочиями для выполнения оценки программного обеспечения.

5.1.2.8 Менеджер по проверке должен дать/не дать разрешение на выпуск программного обеспечения.

5.1.2.9 На всем жизненном цикле программного обеспечения назначение ролей персоналу должно быть выполнено в соответствии с 5.1.2.10—5.1.2.14 в объеме требований УПБ программного обеспечения.

5.1.2.10 Предпочтительная организационная структура для УПБ 3 и УПБ 4:

a) Менеджер требований, проектировщик и разработчик компонента программного обеспечения могут быть одним и тем же человеком.

b) Менеджер требований, проектировщик и разработчик компонента программного обеспечения должны предоставлять отчет менеджеру проектов.

c) Интегратор и тестировщик компонента программного обеспечения могут быть одним и тем же человеком.

d) Интегратор и тестировщик компонента программного обеспечения могут предоставлять отчет менеджеру проектов или менеджеру по подтверждению соответствия.

e) Менеджер по проверке или менеджер контроля качества могут предоставлять отчет менеджеру проектов или менеджеру по подтверждению соответствия.

f) Менеджер по подтверждению соответствия не должен предоставлять отчет менеджеру проектов, т. е. менеджер проектов не должен иметь никакого влияния на решения менеджера по подтверждению соответствия, но менеджер по проверке сообщает менеджеру проектов о своих решениях.

g) Лицо, являющееся менеджером требований, проектировщиком или разработчиком компонента программного обеспечения, не должно быть ни тестировщиком, ни интегратором для того же самого компонента программного обеспечения.

h) Лицо, являющееся интегратором или тестировщиком компонента программного обеспечения, не должно быть ни менеджером требований, ни проектировщиком, ни разработчиком для того же самого компонента программного обеспечения.

i) Лицо, являющееся менеджером по проверке или менеджером контроля качества, не должно быть ни менеджером требований, ни проектировщиком, ни разработчиком, ни интегратором, ни тестировщиком, ни менеджером по подтверждению соответствия.

j) Лицо, являющееся менеджером по подтверждению соответствия, не должно ни быть менеджером требований, ни менеджером контроля качества, ни проектировщиком, ни разработчиком, ни интегратором, ни тестировщиком, ни менеджером по проверке.

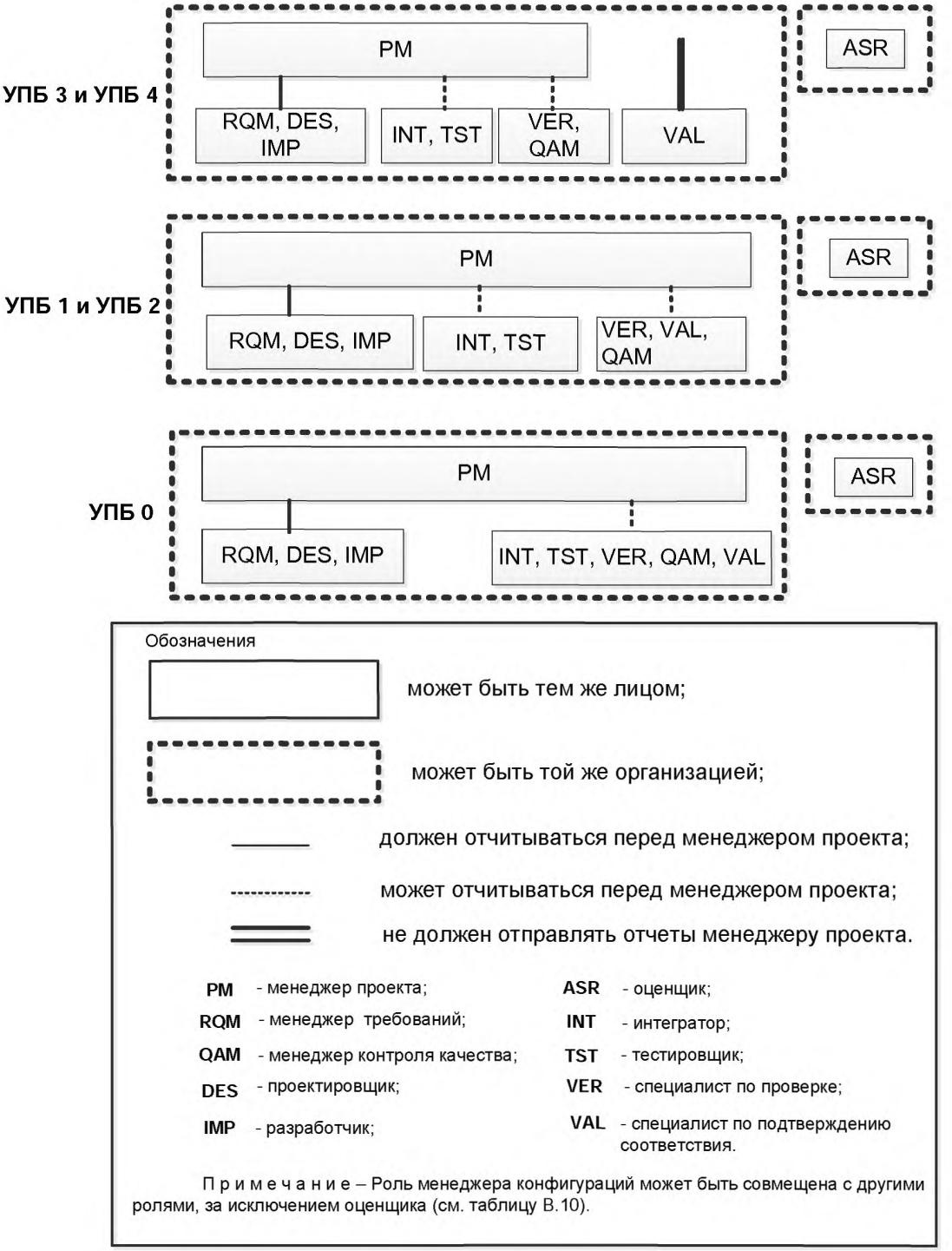

Рисунок 2 — Иллюстрация предпочтительной организационной структуры

Примечание — Рисунок 2 только иллюстрирует предпочтительную организационную структуру.

k) Лицо, являющееся менеджером проектов, может дополнительно выполнить роли менеджера требований, менеджера контроля качества, проектировщика, разработчика, интегратора, тестировщика или менеджера по проверке, если требования независимости между этими дополнительными ролями соблюдаются.

l) Менеджер проектов, менеджер требований, менеджер контроля качества, проектировщик, разработчик, интегратор, тестировщик, менеджер по проверке и менеджер по подтверждению соответствия могут быть из одной организации.

m) Оценщик должен быть независим и организационно независим от ролей менеджера проектов, менеджера требований, менеджера контроля качества, проектировщика, разработчика, интегратора, тестировщика, менеджера по проверке и менеджера по подтверждению соответствия.

Однако возможны следующие варианты:

n) лицо, являющееся менеджером по подтверждению соответствия, может также выполнять роль менеджера по проверке, но поддерживать независимость от менеджера проектов. В этом случае документы, подготовленные менеджером по проверке, должны быть рассмотрены другим компетентным лицом с тем же самым уровнем независимости как и менеджер по проверке. Этот организационный вариант должен быть одобрен оценщиком;

o) лицо, являющееся менеджером по проверке или менеджером контроля качества, может также выполнять роль интегратора и тестировщика в том случае, если роль менеджера по подтверждению соответствия заключается в проверке соответствия документально оформленных доказательств интеграции и тестирования с конкретными целями проверки.

5.1.2.11 Предпочтительная организационная структуры для УПБ 1 и УПБ 2

a) Менеджер требований, проектировщик, разработчик компонента программного обеспечения могут быть одним и тем же человеком и должны предоставлять отчет менеджеру проектов.

b) Интегратор и тестировщик компонента программного обеспечения могут быть одним и тем же человеком.

c) Интегратор и тестировщик компонента программного обеспечения могут предоставлять отчет менеджеру проектов или менеджеру по подтверждению соответствия.

d) Менеджер по проверке, менеджер контроля качества и менеджер по подтверждению соответствия могут быть одним и тем же человеком.

e) Менеджер по проверке, менеджер контроля качества и менеджер по подтверждению соответствия могут предоставлять отчет менеджеру проектов.

f) Лицо, являющееся менеджером требований, менеджером контроля качества, проектировщиком или разработчиком компонента программного обеспечения, не должно быть ни тестировщиком, ни интегратором для того же самого компонента программного обеспечения.

g) Лицо, являющееся интегратором или тестировщиком для компонента программного обеспечения, не должно быть ни менеджером требований, ни проектировщиком, ни разработчиком для того же самого компонента программного обеспечения.

h) Лицо, являющееся менеджером по проверке, менеджером контроля качества или менеджером по подтверждению соответствия, не должно быть ни менеджером требований, ни проектировщиком, ни разработчиком, ни интегратором, ни тестировщиком.

i) Лицо, являющееся менеджером проектов, может дополнительно выполнять роли менеджера требований, менеджера контроля качества, проектировщика, разработчика, интегратора, тестировщика, менеджера по проверке или менеджера по подтверждению соответствия, если требования независимости между этими дополнительными ролями соблюдаются.

j) Менеджер проектов, менеджер требований, менеджер контроля качества, проектировщик, разработчик, интегратор, тестировщик, менеджер по проверке и менеджер по подтверждению соответствия могут быть из одной организации.

k) Оценщик должен быть независим и организационно независим от ролей менеджера проектов, менеджера требований, менеджера контроля качества, проектировщика, разработчика, интегратора, тестировщика, менеджера по проверке и менеджера по подтверждению соответствия.

Однако возможны следующие варианты:

l) лицо, являющееся менеджером по проверке или менеджером контроля качества, может также выполнять роль интегратора и тестировщика в том случае, если роль менеджера по подтверждению соответствия должна включать рассмотрение документов, подготовленных менеджером по проверке или менеджером контроля качества, следовательно, поддерживающих два уровня проверки в проектной организации;

m) лицо, являющееся менеджером по подтверждению соответствия, может также выполнять роль менеджера по проверке, менеджера контроля качества, интегратора и тестировщика. В этом случае документы, подготовленные менеджером по проверке или менеджером контроля качества должны быть

рассмотрены другим компетентным лицом с тем же самым уровнем независимости как и менеджер по подтверждению соответствия. Этот организационный вариант должен быть одобрен оценщиком.

5.1.2.12 Предпочтительная организационная структура для УПБ 0

a) Менеджер требований, проектировщик и разработчик компонента программного обеспечения могут быть одним и тем же человеком и должны управляться менеджером проектов.

b) Интегратор, тестировщик, менеджер по проверке, менеджер контроля качества и менеджер по подтверждению соответствия компонента программного обеспечения могут быть одним и тем же человеком.

c) Интегратором, тестировщиком, менеджером по проверке, менеджером контроля качества и менеджером по подтверждению соответствия может руководить менеджер проектов.

d) Лицо, являющееся менеджером требований, проектировщиком или разработчиком компонента программного обеспечения, не должно быть ни тестировщиком, ни интегратором того же самого компонента программного обеспечения.

e) Лицо, являющееся менеджером по проверке, менеджером контроля качества или менеджером по подтверждению соответствия, не должно быть ни менеджером требований, ни проектировщиком, ни разработчиком.

f) Лицо, являющееся менеджером проектов, может дополнительно выполнять роли менеджера требований, менеджера контроля качества, проектировщика, разработчика, интегратора, тестировщика, менеджера по проверке или менеджера по подтверждению соответствия, если требования независимости между этими дополнительными ролями соблюдаются.

g) Менеджер проектов, менеджер требований, менеджер контроля качества, проектировщик, разработчик, интегратор, тестировщик, менеджер по проверке и менеджер по подтверждению соответствия могут быть из одной организации.

h) Оценщик должен быть независим и организационно независим от ролей менеджера проектов, менеджера требований, менеджера контроля качества, проектировщика, разработчика, интегратора, тестировщика, менеджера по проверке и менеджера по подтверждению соответствия.

Однако возможны следующие варианты:

i) менеджер требований, проектировщик, разработчик, интегратор и тестировщик могут быть одним и тем же человеком;

j) менеджер по подтверждению соответствия, менеджер по проверке и менеджер контроля качества могут также быть одним и тем же человеком;

k) лицо, являющееся менеджером по проверке, менеджером контроля качества или менеджером по подтверждению соответствия, не должно быть ни менеджером требований, ни проектировщиком, ни разработчиком.

5.1.2.13 Роли менеджера требований, проектировщика и разработчика для одного компонента могут выполнять роли тестировщика и интегратора для другого компонента.

5.1.2.14 Роли менеджера по проверке, менеджера контроля качества и менеджера по подтверждению соответствия должны быть определены на уровне проекта и должны остаться неизменными в процессе всей разработки проекта.

5.2 Компетентность персонала

5.2.1 Цели

Гарантировать, чтобы весь персонал, который несет ответственность за программное обеспечение, был компетентен разделить эту ответственность, демонстрируя возможность выполнить соответствующие задачи правильно, эффективно и последовательно с высоким качеством и в различных условиях.

5.2.2 Требования

5.2.2.1 Ключевые знания, необходимые для каждой роли при разработке программного обеспечения, определены в приложении В. Если для роли в жизненном цикле программного обеспечения будет требоваться дополнительный опыт, возможности или квалификации, то они должны быть определены в плане обеспечения качества программного обеспечения.

5.2.2.2 Документально оформленное доказательство компетентности персонала, включая технические знания, квалификации, соответствующий опыт и надлежащее обучение, должны поддерживаться организацией поставщика, чтобы демонстрировать надлежащее обеспечение безопасности в организации.

5.2.2.3 Если подтверждением оценщика или сертификацией было доказано, что для всего персонала, назначенного на различные роли, была продемонстрирована компетентность, то каждый человек должен продемонстрировать непрерывное развитие и повышение своей квалификации. Это может быть продемонстрировано, ведением регистрационного журнала, показывающего, что регулярно выполняемые работы правильны и в соответствии с ИСО 9001 и ИСО/МЭК 90003:2014, 6.2.2 «Компетентность, осведомленность и обучение» реализуется дополнительное обучение.

5.2.2.4 Организация должна поддерживать процедуры управления компетентностью персонала, удовлетворяя соответствующим ролям, согласно существующим стандартам качества.

5.3 Жизненный цикл и документация

5.3.1 Цели

5.3.1.1 Структурировать разработку программного обеспечения на определенные стадии и действия.

5.3.1.2 Фиксировать всю информацию, относящуюся к программному обеспечению, на всем жизненном цикле программного обеспечения.

5.3.2 Требования

5.3.2.1 Должна быть выбрана модель жизненного цикла для разработки программного обеспечения. Это должно быть подробно представлено в плане обеспечения качества программного обеспечения в соответствии с 6.5.

На рисунках 3 и 4 показаны два примера моделей жизненного цикла.

5.3.2.2 Модель жизненного цикла должна учитывать возможность итераций внутри стадий и между ними.

5.3.2.3 Процедуры обеспечения качества должны выполняться параллельно с действиями на жизненном цикле программного обеспечения и использовать ту же терминологию.

5.3.2.4 План обеспечения качества программного обеспечения, план проверки программного обеспечения, план подтверждения соответствия программного обеспечения и план управления конфигурацией программного обеспечения должны составляться в начале проекта и выполняться на всем жизненном цикле разработки программного обеспечения.

5.3.2.5 Все выполняемые на каждой стадии действия должны быть определены и спланированы до начала стадии.

5.3.2.6 Все документы должны быть структурированы, чтобы обеспечить их непрерывное увеличение в процессе разработки.

5.3.2.7 Для каждого документа должна быть обеспечена прослеживаемость с помощью уникального ссылочного номера, а также определенного и документально оформленного отношения с другими документами.

5.3.2.8 Каждый термин, условное обозначение или сокращение должен иметь одно и то же значение в каждом документе. Если по историческим причинам это будет невозможно, то различные значения должны быть перечислены и даны ссылки.

5.3.2.9 За исключением документов, касающихся существующего ранее программного обеспечения (см. 7.3.4.7), каждый документ должен быть подготовлен по следующим правилам:

- он должен содержать или реализовывать все применимые условия и требования предыдущего документа, с которым он иерархически связан;
- он не должен противоречить предыдущему документу.

5.3.2.10 Каждый элемент или понятие должно быть названо одним и тем же именем или одинаково описанием во всех документах.

5.3.2.11 Содержание всех документов должно быть оформлено в виде, подходящем для манипулирования, обработки и хранения.

5.3.2.12 Если документы, которые выполнены независимыми ролями, будут объединены в единый документ, то связь частей, выполненных любой независимой ролью, должна быть прослежена в документе.

5.3.2.13 Документы могут быть объединены или разделены в соответствии с 5.3.2.12. Некоторые шаги разработки могут быть объединены, разделены или, если обоснованы, устранены по решению руководителя проекта и с согласия менеджера по подтверждению соответствия.

5.3.2.14 Любой жизненный цикл и выбранная структура документации должны удовлетворять всем целям и требованиям настоящего стандарта.

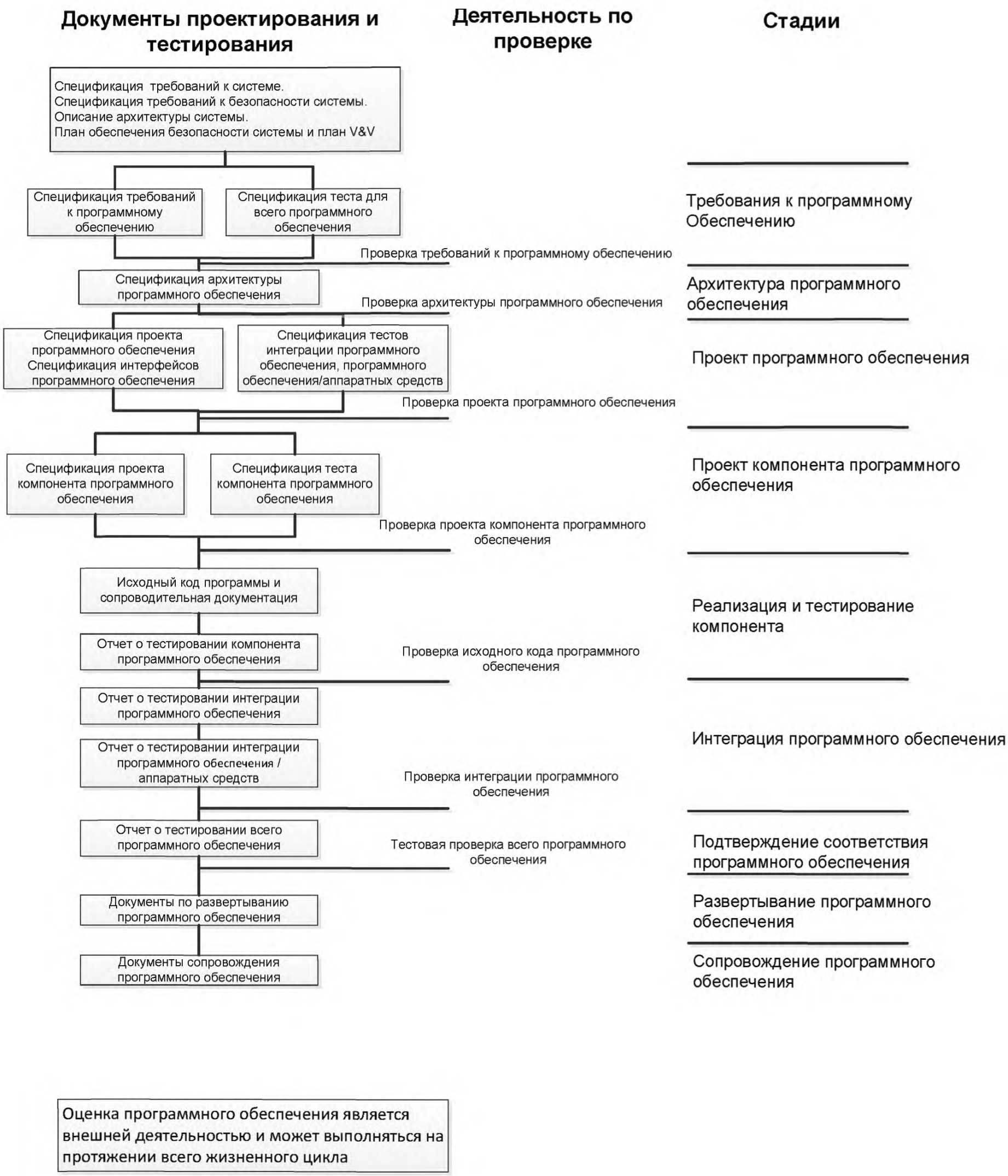

Рисунок 3 — Пример жизненного цикла разработки 1

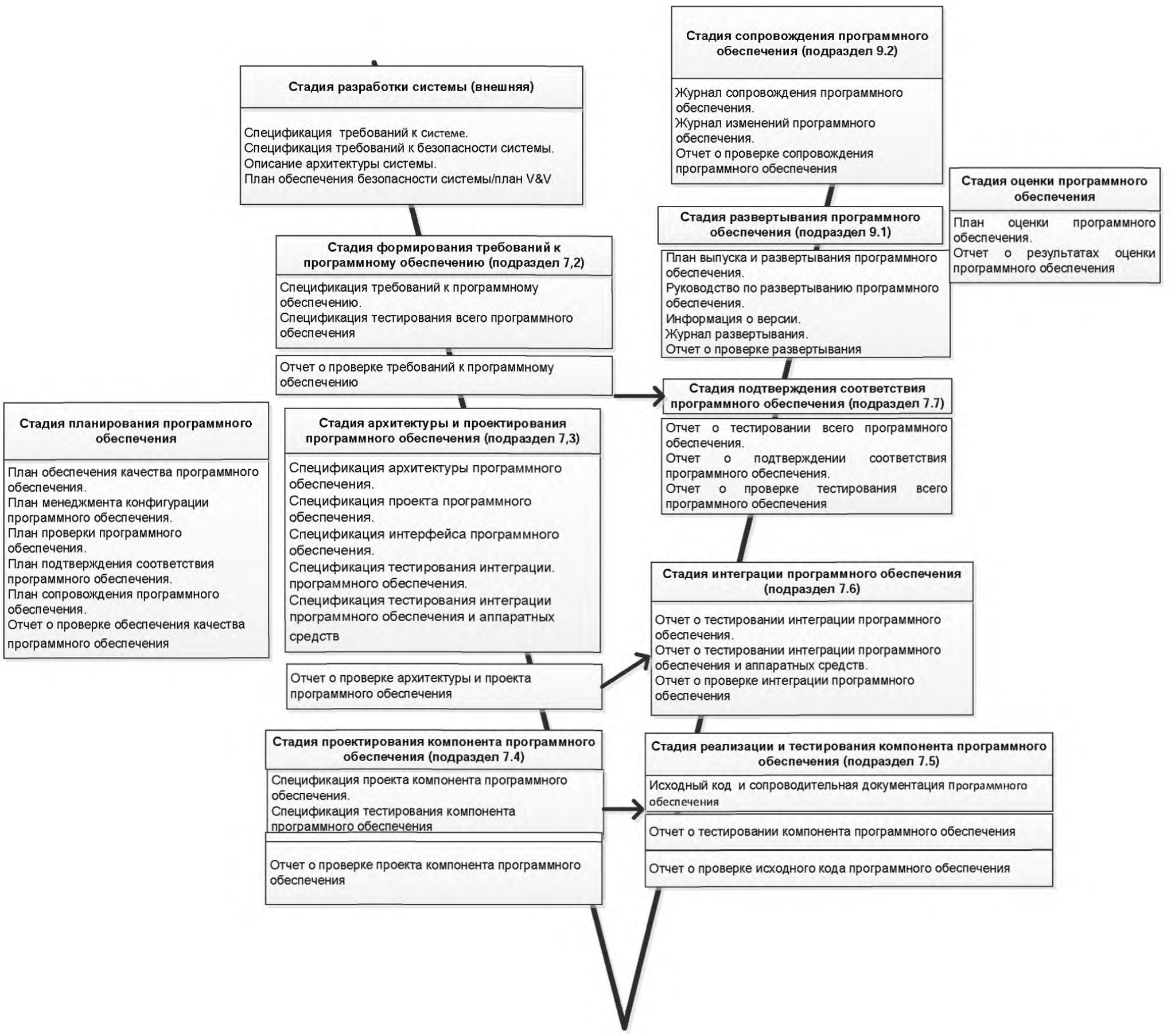

Рисунок 4 — Пример жизненного цикла разработки 2

6 Гарантированное программное обеспечение

6.1 Тестирование программного обеспечения

6.1.1 Цель

Цель тестирования программного обеспечения, выполняемого как тестировщиком, так и/или интегратором, состоит в установлении того, что поведение или функциональные характеристики программного обеспечения обеспечиваются соответствующей спецификацией тестирования до значения, достигаемого выбранным тестовым охватом.

6.1.2 Входные документы

Вся необходимая документация по системе, аппаратным средствам и программному обеспечению, как определено в плане проверки программного обеспечения.

6.1.3 Выходные документы

a) Спецификация тестирования всего программного обеспечения.

b) Отчет об испытаниях всего программного обеспечения.

c) Спецификация тестирования интеграции программного обеспечения.

d) Отчет об испытаниях интеграции программного обеспечения.

e) Спецификация тестирования интеграции программного обеспечения/аппаратных средств.

f) Отчет об испытаниях интеграции программного обеспечения/аппаратных средств.

g) Спецификация тестирования компонента программного обеспечения.

h) Отчет об испытаниях компонента программного обеспечения.

6.1.4 Требования

6.1.4.1 Тесты, выполненные другими сторонами, такими как менеджер требований, проектировщик или разработчик, если они полностью документально оформлены и соответствуют последующим требованиям, могут быть приняты менеджером по проверке.

6.1.4.2 Измерительное оборудование, используемое для тестирования, должно быть соответственно калибровано. Для любых инструментальных средств, аппаратных средств или программного обеспечения, используемых для тестирования, должно быть показано их пригодность реализуемой цели.

6.1.4.3 При тестировании программного обеспечения документально оформляется спецификация тестирования и отчет об испытаниях, как определено ниже.

6.1.4.4 Документ для каждой спецификации тестирования должен содержать следующее:

a) цели испытаний;

b) тестовые сценарии, данные испытаний и ожидаемые результаты;

c) типы тестов, которые будут выполнены;

d) окружение, инструменты, конфигурацию и программы испытаний;

e) критерии испытаний, по которым будет завершен тест;

f) критерии и уровень тестового охвата, который должен быть достигнут;

g) роли и обязанности персонала, участвующего в процессе испытаний;

h) требования, которые охвачены спецификацией тестов;

i) выбор и использование оборудования для испытания программного обеспечения.

6.1.4.5 Отчет об испытаниях должен быть представлен следующим образом:

a) отчет об испытаниях должен содержать имена, выполняющих тестирование, изложить результаты испытаний и проверить выполнены ли представленные в спецификации тестирования цели испытаний и критерии испытаний. Неудачи должны быть документально оформлены и обобщены;

b) тестовые сценарии и их результаты должны быть документально оформлены, предпочтительно в машиночитаемой форме для последующего анализа;

c) тесты должны быть воспроизводимыми и, если это реально, должны быть выполнены автоматическими средствами;

d) тестовые сценарии для автоматического выполнения испытаний должны быть проверены;

e) должны быть документально оформлены идентичность и конфигурация всех используемых элементов (используемых аппаратных средств, используемого программного обеспечения, используемого оборудования, калибровка оборудования, информацию о версии тестируемого программного обеспечения, а также информацию о версии спецификации испытаний);

f) оценка тестового охвата и завершение теста должны быть обеспечены, а любые отклонения зафиксированы.

6.2 Проверка программного обеспечения

6.2.1 Цель

Цель проверки программного обеспечения состоит в том, чтобы исследовать и получить обоснование, основанное на доказательствах, что выходные элементы (процесс, документация, программное обеспечение или применение) конкретной стадии разработки удовлетворяют требованиям и планам относительно полноты, правильности и согласованности. Этими действиями управляет менеджер по проверке.

6.2.2 Входные документы

Вся необходимая документация на систему, аппаратные средства и программное обеспечение.

6.2.3 Выходные документы

a) План проверки программного обеспечения.

b) Отчет(ы) о проверке программного обеспечения.

c) Отчет о проверке обеспечения качества программного обеспечения.

6.2.4 Требования

6.2.4.1 Проверка должна быть документально оформлена, по крайней мере, план проверки программного обеспечения и один или более (связанный с процессом) отчет о проверке.

6.2.4.2 План проверки программного обеспечения должен быть подготовлен в письменном виде под руководством менеджера по проверке на основе необходимой документации.

Требования 6.2.4.3—6.2.4.9 относятся к плану проверки программного обеспечения.

6.2.4.3 План проверки программного обеспечения должен описывать действия, которые необходимо выполнить для обеспечения надлежащей проверки, а также, чтобы были предусмотрены действия для определенного проекта или других потребностей проверки соответственно.

6.2.4.4 В процессе разработки (и в зависимости от размера системы) план может быть разделен на ряд более мелких входящих в него документов, поскольку становятся более ясными необходимые детали проверки.

6.2.4.5 План проверки программного обеспечения должен содержать все критерии, методы и инструменты, которые будут использоваться в процессе проверки. План проверки программного обеспечения должен включать методы и меры, выбранные из таблиц А.5—А.8. Выбранная комбинация должна быть обоснована как набор, удовлетворяющий 4.8—4.10.

6.2.4.6 План проверки программного обеспечения должен описать действия, которые будут выполнены, чтобы гарантировать правильность и согласованность информации на входе стадий анализа, тестирования и интеграции.

6.2.4.7 На каждой стадии разработки необходимо показать, что выполнены функциональные, эксплуатационные требования и требования безопасности.

6.2.4.8 Результаты каждой проверки должны быть сохранены в формате, который определен или на который имеется ссылка в плане проверки программного обеспечения.

6.2.4.9 План проверки программного обеспечения должен обеспечить следующее:

a) выбор стратегий проверки и методов (чтобы избежать излишней сложности при оценке проверки и тестирования, предпочтение должно быть дано выбору методов, которые сами по себе легко поддаются анализу);

b) выбор методов из таблиц А.5—А.8;

c) выбор и документальное оформление действий проверки;

d) оценка пользы, извлекаемой из результатов проверки;

e) оценка требований безопасности и устойчивости;

f) роли и обязанности персонала, участвующего в процессе проверки;

g) требуемый уровень тестового охвата функционала, который должен быть достигнут;

h) структура и содержание каждого шага проверки, особенно для проверки требований к программному обеспечению (7.2.4.22), проверки архитектуры и проекта программного обеспечения (7.3.4.41, 7.3.4.42), проверки компонентов программного обеспечения (7.4.4.13), проверки исходного кода программного обеспечения (7.5.4.10) и проверки интеграции (7.6.4.13) в таком представлении, которое облегчает анализ плана проверки программного обеспечения.

6.2.4.10 Отчет о проверке обеспечения качества программного обеспечения должен быть написан, под руководством менеджера по проверке, на основе входных документов из 6.2.2.

Требование в 6.2.4.12 ссылается к отчету о проверке обеспечения качества программного обеспечения.

6.2.4.11 Менеджер по проверке должен удостовериться, что проверка плана проверки программного обеспечения выполнена человеком, который компетентен в этом вопросе. Чтобы соответствовать хорошей практике и правилам независимости настоящего стандарта, менеджер по проверке не должен проверять план проверки программного обеспечения сам, если он является автором этого плана.

6.2.4.12 Если план проверки программного обеспечения создан, то проверка должна выяснить:

a) отвечает ли план проверки программного обеспечения общим требованиям удобочитаемости и прослеживаемости из 5.3.2.7—5.3.2.10 и из 6.5.4.14—6.5.4.17, а также конкретным требованиям 6.2.4.3—6.2.4.9;

b) внутреннюю согласованность плана проверки программного обеспечения.

Результаты должны быть зарегистрированы в отчете о проверке обеспечения качества программного обеспечения.

6.2.4.13 Любые отчеты о проверке программного обеспечения должны быть написаны, под руководством менеджера по проверке, на основе входных документов. Эти отчеты могут быть разделены

на части для ясности и удобства и должны следовать плану проверки программного обеспечения. Требование 6.2.4.14 ссылается на отчеты о проверке программного обеспечения.

6.2.4.14 Каждый отчет о проверке программного обеспечения должен содержать следующее:

a) идентификацию и конфигурацию проверенных элементов, а также имена проверяющих;

b) элементы, которые не соответствуют техническим требованиям;

c) компоненты, данные, структуры и алгоритмы, плохо адаптированные к проблеме;

d) обнаруженные ошибки или неточности;

e) выполнение или отклонение от плана проверки программного обеспечения (в случае отклонения отчет о проверке должен объяснить, важно ли отклонение или нет);

f) предположения если таковые имеются;

g) резюме результатов проверки.

6.3 Подтверждение соответствия программного обеспечения

6.3.1 Цель

6.3.1.1 Цель подтверждения соответствия программного обеспечения состоит в том, чтобы продемонстрировать, что процессы и их результаты таковы, что программное обеспечение имеет определенный уровень полноты безопасности программного обеспечения, удовлетворяет требованиям к программному обеспечению и пригодно для его применения по назначению. Эта деятельность выполняется менеджером по подтверждению соответствия.

6.3.1.2 Основные действия подтверждения соответствия должны продемонстрировать, используя анализ и/или тестирование, что все требования к программному обеспечению определены, реализованы, проверены и выполнены в соответствии с применяемым УПБ, а также оценить критичность для безопасности всех отклонений и несоответствий на основе результатов экспертиз, исследований и тестов.

6.3.2 Входные документы

Документация на все системы, аппаратные средства и программное обеспечение, как определено в настоящем стандарте.

6.3.3 Выходные документы

a) План подтверждения соответствия программного обеспечения.

b) Отчет о подтверждении соответствия программного обеспечения.

6.3.4 Требования

6.3.4.1 Действия по подтверждению соответствия программного обеспечения должны быть разработаны и выполнены с оценкой их результатов менеджером по подтверждению соответствия с соответствующим уровнем независимости, как определено в 5.1.

6.3.4.2 Подтверждение соответствия должно быть документально оформлено, по крайней мере, вместе с планом подтверждения соответствия программного обеспечения и отчетом о подтверждении соответствия программного обеспечения, как определено ниже.

6.3.4.3 План подтверждения соответствия программного обеспечения должен быть подготовлен в письменном виде под руководством менеджера по подтверждению соответствия, на основе входных документов.

Требования 6.3.4.4—6.3.4.6 относятся к плану подтверждения соответствия программного обеспечения.

6.3.4.4 План подтверждения соответствия программного обеспечения должен включать общее обоснование выбранной стратегии подтверждения соответствия. В соответствии с необходимым уровнем полноты безопасности программного обеспечения результатом этого обоснования должны быть рекомендации на применение:

a) ручных или автоматизированных методов или тех и других,

b) статических или динамических методов или тех и других,

c) аналитических или статистических методов или тех и других,

d) тестирования в реальном или моделируемом окружении или в том и другом.

6.3.4.5 План подтверждения соответствия программного обеспечения должен определить последовательность действий, обеспечивающих демонстрацию для любой спецификации программного обеспечения соответствие требованиям безопасности, представленным в спецификации требований к безопасности системы.

6.3.4.6 План подтверждения соответствия программного обеспечения должен определить последовательность действий, необходимых для демонстрации того, что полная спецификация испытаний

программного обеспечения соответствует тестированию спецификации требований программного обеспечения.

6.3.4.7 Отчет о подтверждении соответствия программного обеспечения должен быть подготовлен в письменном виде под руководством менеджера по подтверждению соответствия, на основе входных документов.

Требования 6.3.4.8—6.3.4.11 относятся к отчету о подтверждении соответствия программного обеспечения.

6.3.4.8 Результаты подтверждения соответствия должны быть документально оформлены в отчете о подтверждении соответствия программного обеспечения.

6.3.4.9 Менеджер по подтверждению соответствия должен проверить, что процесс проверки завершен.

6.3.4.10 В отчете о подтверждении соответствия программного обеспечения должно быть полностью описано универсальное программное обеспечение, для которого было выполнено подтверждение соответствия.

6.3.4.11 В отчете о подтверждении соответствия должны быть полностью определены все известные недостатки в программном обеспечении и то влияние, которое они могут оказать на использование программного обеспечения.

6.3.4.12 Лицо, ответственное за подтверждение соответствия, должно предоставить анализ документов, перечисленных в 6.3.3.

6.3.4.13 После того, как сформирован план подтверждения соответствия программного обеспечения, анализ этого документа должен быть направлен на то, чтобы:

a) план подтверждения соответствия программного обеспечения отвечал общим требованиям удобочитаемости и прослеживаемости, представленным в 5.3.2.7—5.3.2.10 и 6.5.4.14—6.5.4.17, а также требованиям, определенным в 6.3.4.4—6.3.4.6;

b) обеспечить внутреннюю согласованность плана подтверждения соответствия программного обеспечения.

6.3.4.14 После того, как сформирован отчет о подтверждении соответствия программного обеспечения, анализ этого документа должен быть направлен на то, чтобы:

a) отчет о подтверждении соответствия программного обеспечения отвечал общим требованиям удобочитаемости и прослеживаемости, представленным в 5.3.2.7—5.3.2.10 и 6.5.4.14—6.5.4.17, а также требованиям, определенным в 6.3.4.8—6.3.4.11 и 7.7.4.8—7.7.4.12;

b) обеспечить внутреннюю согласованность отчета о подтверждении соответствия программного обеспечения.

6.3.4.15 Менеджер по подтверждению соответствия должен быть уполномочен, чтобы потребовать или выполнить дополнительные рассмотрения, исследования и тесты.

6.3.4.16 Программное обеспечение должно быть выпущено для работы только после разрешения менеджера по подтверждению соответствия.

6.3.4.17 Эмуляция и моделирование могут использоваться в качестве дополнения к процессу подтверждения соответствия.

6.4 Оценка программного обеспечения

6.4.1 Цель

6.4.1.1 Оценить, что процессы жизненного цикла и их результаты таковы, что программное обеспечение соответствует определенным уровням (от 1 до 4) полноты безопасности программного обеспечения и пригодно для его применения по назначению.

6.4.1.2 Для программного обеспечения с УПБ 0 должны быть выполнены требования настоящего стандарта, но если для него имеется сертификат соответствия с ИСО 9001, то оценка такого программного обеспечения не требуется.

6.4.2 Входные документы

a) Спецификация требований к безопасности системы.

b) Спецификация требований к программному обеспечению.

c) Все другие документы, необходимые для выполнения процесса оценки.

6.4.3 Выходные документы

a) План оценки программного обеспечения.

b) Отчет о результатах оценки программного обеспечения.

6.4.4 Требования

6.4.4.1 Оценка программного обеспечения должна быть выполнена независимым оценщиком, как описано в 5.1.2.6 и 5.1.2.7.

6.4.4.2 Программное обеспечение с отчетом о результатах его оценки от другого оценщика не должно быть объектом новой оценки. Оценщик должен проверить, что программное обеспечение пригодно для его надлежащего использования в намеченном окружении, и что прежняя оценка установила, что программное обеспечение достигло уровня полноты безопасности, по крайней мере, равного требуемому уровню.

6.4.4.3 У оценщика должен быть доступ ко всей проектной документации процесса разработки.

6.4.4.4 План оценки программного обеспечения должен быть подготовлен в письменном виде под руководством оценщика на основе входных документов, перечисленных в 6.4.2. Где это уместно, может использоваться существующий документально оформленный общий план или процедура оценки программного обеспечения. Требования 6.4.4.5 относятся к плану оценки программного обеспечения.

6.4.4.5 План оценки программного обеспечения должен включать следующее:

a) аспекты, с которыми имеет дело оценивание;

b) действия в течение процесса оценки и их последовательные ссылки к техническим действиям;

c) документы, которые должны быть учтены;

d) описание критериев прошел/не прошел оценку и действия в случае, если оценка не удовлетворяет критерию;

e) требования к содержанию и форме отчета по результатам оценки программного обеспечения.

6.4.4.6 Лицо, ответственное за оценку, должно выполнить анализ оценки на основе входных документов, перечисленных в 6.4.2.

Требование 6.4.4.7 относится к внутренней проверке или независимой экспертизе оценки.

6.4.4.7 После того, как сформирован план оценки программного обеспечения, должна быть выполнена его проверка для того, чтобы:

a) план оценки программного обеспечения отвечал общим требованиям удобочитаемости и прослеживаемости, представленным в 5.3.2.7—5.3.2.10 и 6.5.4.14—6.5.4.17, а также требованиям, определенным в 6.4.4.5;

b) обеспечить внутреннюю согласованность плана оценки программного обеспечения.

6.4.4.8 Оценщик должен оценить, что программное обеспечение системы соответствует его предназначенной цели и корректно отрабатывает угрозы безопасности, полученные из спецификации требований безопасности к системе.

6.4.4.9 Оценщик должен оценить, правильно ли надлежащий набор методов из приложения А, подходящий для намеченной разработки, был выбран и применен согласно требуемому уровню полноты безопасности.

Кроме того, оценщик должен рассмотреть степень применения каждого метода из приложения А, т. е. применен ли он ко всему или только к части программного обеспечения, и также найти доказательство, что метод применен должным образом.

6.4.4.10 Оценщик должен оценить систему управления конфигурацией и управления изменениями и доказательства на ее использование и применение.

6.4.4.11 Оценщик должен рассмотреть доказательство компетентности проектной группы согласно приложению В и должен оценить организацию разработки программного обеспечения согласно 5.1.

6.4.4.12 Для любого программного обеспечения, использующегося в связанных с безопасностью применениях, оценщик должен проверить замеченные отклонения, несоблюдения требованиям и зарегистрированные несоответствия, если они оказывают влияние на безопасность, и обосновать, приемлемо ли их устранение из проекта. Результат должен быть утвержден в отчете по результатам оценки.

6.4.4.13 Оценщик должен оценить действия по проверке и подтверждению соответствия и подтверждающие их доказательства.

6.4.4.14 Оценщик должен согласовать объем и содержание плана подтверждения соответствия программного обеспечения. Это соглашение должно также содержать положение о присутствии оценщика во время тестирования.

6.4.4.15 Оценщик может выполнять аудиты и контроль (например, наблюдение за выполнением тестов) в течение всего процесса разработки. Оценщик может попросить выполнить дополнительные работы по проверке и подтверждению соответствия.

Примечание — Раннее включение оценщика в проект имеет преимущество.

6.4.4.16 Отчет по результатам оценки программного обеспечения должен быть подготовлен в письменном виде под руководством оценщика. Требования 6.4.4.17—6.4.4.19 относятся к отчету о результатах оценки программного обеспечения.

6.4.4.17 Отчет о результатах оценки программного обеспечения должен соответствовать требованиям плана оценки программного обеспечения и содержать заключение и рекомендации.

6.4.4.18 Оценщик должен записывать свои действия как согласованную основу для отчета о результатах оценки программного обеспечения. Они должны быть собраны в отчете о результатах оценки программного обеспечения.

6.4.4.19 Оценщик должен выявлять и оценивать любое несоответствие с требованиями настоящего стандарта и оценивать его влияние на конечный результат. Эти несоответствия и их оценки должны быть перечислены в отчете о результатах оценки программного обеспечения.

6.5 Обеспечение качества программного обеспечения

6.5.1 Цели

6.5.1.1 Следует определять, контролировать и управлять всеми техническими и организационными действиями, которые необходимы, чтобы гарантировать, что программное обеспечение достигает требуемого качества. Это необходимо, чтобы обеспечить требуемую качественную защиту от систематических сбоев и гарантировать, что может быть создан контрольный журнал, позволяющий выполнять действия по проверке и подтверждению соответствия более эффективно.

6.5.1.2 Представить свидетельства, что вышеупомянутые действия были выполнены.

6.5.2 Входные документы

Все документы, доступные на каждом этапе жизненного цикла.

6.5.3 Выходные документы

a) План обеспечения качества программного обеспечения.

b) План управления конфигурацией программного обеспечения, если не доступен на уровне системы.

c) Отчет о проверке обеспечения качества программного обеспечения.

Примечание — Отчет по обеспечению качества программного обеспечения включен в отчет управления качеством, определенный в МЭК 62425.

6.5.4 Требования

6.5.4.1 Все планы должны быть выпущены в начале проекта и обновляться в течение жизненного цикла.

6.5.4.2 Организации, участвующие в разработке программного обеспечения, должны реализовать и использовать у себя систему обеспечения качества, совместимую с ИСО 9000, чтобы поддерживать требования настоящего стандарта. Сертификация на соответствие требованиям ИСО 9001 настоятельно рекомендуется.

6.5.4.3 План обеспечения качества программного обеспечения должен быть подготовлен в письменном виде под руководством менеджера контроля качества, на основе входных документов, перечисленных в 6.5.2.

Требования 6.5.4.4—6.5.4.6 относятся к плану обеспечения качества программного обеспечения.

6.5.4.4 План обеспечения качества программного обеспечения должен быть подготовлен в письменном виде и должен быть определен для конкретного проекта. Он должен реализовывать требования 6.5.4.5.

6.5.4.5 Как минимум, следующие элементы должны быть определены или на них сделаны ссылки в плане обеспечения качества программного обеспечения.

a) Определение модели жизненного цикла:

1) действия и элементарные задачи, согласованные с планами, например, с планом обеспечения безопасности, которые были установлены на уровне системы;

2) критерии входа и выхода каждого действия;

3) входы и выходы каждого действия;

4) основные действия по обеспечению качества;

5) лицо, ответственное за каждое действие.

b) Структура документации.

c) Управление документацией:

1) роли, включенные для записи, проверки и принятия;

2) область распределения;

3) архивирование.

d) Отслеживание и трассировка отклонений.

e) Методы, меры и инструментальные средства для обеспечения качества в соответствии с распределенными уровнями полноты безопасности (см. приложение А).

f) Обоснования, как определено в 4.7 к 4.9, что каждая комбинация методов или мер, выбранных согласно приложению А, соответствует определенному уровню полноты безопасности программного обеспечения.

Часть необходимой информации для плана обеспечения качества программного обеспечения может содержаться в других документах, таких как отдельный план управления конфигурацией программного обеспечения, план сопровождения, план проверки программного обеспечения и план подтверждения соответствия программного обеспечения. Подразделы плана обеспечения качества программного обеспечения должны ссылаться на документы, в которых содержится информация. В любом случае содержание каждого подраздела плана обеспечения качества программного обеспечения должно быть определено либо непосредственно, либо ссылкой на другой документ.

Документы, на которые делаются ссылки, должны быть проанализированы, чтобы гарантировать, что они предоставляют всю требуемую информацию и что они полностью удовлетворяют требования настоящего стандарта.

6.5.4.6 Мероприятия по обеспечению качества, действия, документы и т. д., требуемые всеми нормативными подпунктами настоящего стандарта, должны быть определены или на них должны быть даны ссылки в плане обеспечения качества программного обеспечения и адаптированы в соответствии с конкретным проектом.

6.5.4.7 Отчет о проверке обеспечения качества программного обеспечения должен быть подготовлен в письменном виде под руководством менеджера по проверке на основе входных документов, перечисленных в 6.5.2.

Требование 6.5.4.8 относится к отчету о проверке обеспечения качества программного обеспечения.

6.5.4.8 После того, как сформирован план обеспечения качества программного обеспечения, должна быть выполнена его проверка для того, чтобы:

a) план обеспечения качества программного обеспечения отвечал общим требованиям удобочитаемости и прослеживаемости, представленным в 5.3.2.7—5.3.2.10 и 6.5.4.14—6.5.4.17, а также требованиям, определенным в 6.5.4.4—6.5.4.6;

b) обеспечить внутреннюю согласованность плана обеспечения качества программного обеспечения.

Эти результаты должны быть оформлены в отчете о проверке обеспечения качества программного обеспечения.

6.5.4.9 Каждый документ, описывающий план, должен содержать информацию, подробно описывающую все его изменения в процессе выполнения всего проекта: дата изменения, ответственное лицо, порядок выполнения.

6.5.4.10 Каждый документ, связанный с программным обеспечением, и поставляемый программный продукт должны быть охвачены процедурой управления конфигурацией с момента времени их первого выпуска.

6.5.4.11 Изменения во всех элементах, охваченных процедурой управления конфигурацией, должны быть разрешены и зарегистрированы.

6.5.4.12 В дополнение к разработке программного обеспечения система управления конфигурацией должна также охватить окружение разрабатываемого программного обеспечения, используемую в процессе всего его жизненного цикла.

Это расширение, необходимое для воспроизводимости разработки и действий по его поддержке, должно включать все инструментальные средства, трансляторы, файлы данных и тестов, файлы параметризации и используемые платформы аппаратных средств.

6.5.4.13 Поставщик должен установить, документально оформить и выполнять процедуры управления внешними поставщиками, включая:

- методы и соответствующие документы, гарантирующие, что программное обеспечение внешних поставщиков соответствует установленных требованиям. Должно быть гарантировано, что ранее разработанное программное обеспечение соответствует требуемому уровню полноты безопасности программного обеспечения и надежности. Новое программное обеспечение должно разрабатываться

и поддерживаться в соответствии с планом обеспечения качества программного обеспечения поставщика или с конкретным планом обеспечения качества программного обеспечения, подготовленным внешним поставщиком в соответствии с планом обеспечения качества программного обеспечения поставщика;

- методы и соответствующие документы, гарантирующие, что требования, предоставленные внешнему поставщику, соответствуют и завершены.

6.5.4.14 Прослеживаемость требований должна быть важным аспектом при выполнении подтверждения соответствия связанной с безопасностью системы, а также должны быть обеспечены средства, позволяющие демонстрацию прослеживаемости требований на всех стадиях жизненного цикла.

6.5.4.15 В контексте настоящего стандарта и, в известной степени, подходящая заданному уровню полноты безопасности программного обеспечения, должна быть, в частности, выполнена:

a) прослеживаемость требований к проекту или к другим объектам, которые выполняют эти требования,

b) прослеживаемость объектов проектирования к объектам реализации, которые их выполняют,

c) прослеживаемость требований и объектов проектирования к тестам (компонент, интеграции, общему тесту) и анализу, которые обеспечивают их проверку.

Прослеживаемость должна быть предметом управления конфигурацией.

6.5.4.16 В особых случаях, например, в уже существующем программном обеспечении или в экспериментальном программном обеспечении прослеживаемость может быть установлена после реализации и/или документального оформления кода, но до проверки/подтверждения соответствия. В этих случаях нужно показать, что проверка/подтверждение соответствия столь же эффективно, как это было бы с прослеживаемостью на всех стадиях.

6.5.4.17 Для объектов требований, проекта или реализации, которые не могут быть адекватно прослежены, должно быть продемонстрировано отсутствие их влияния на безопасность или целостность системы.

6.6 Управление модификациями и изменениями

6.6.1 Цели

6.6.1.1 Гарантировать, что программное обеспечение, выполненное в соответствии с требованиями, сохраняет полноту безопасности программного обеспечения и надежность при изменении программного обеспечения.

6.6.1.2 Этими целями управляет менеджер по конфигурации.

6.6.2 Входные документы

a) План обеспечения качества программного обеспечения.

b) План управления конфигурацией программного обеспечения.

c) Вся соответствующая документация по проектированию, разработке и анализу.

d) Запросы на изменение.

e) Анализ влияния и разрешение на изменение.

6.6.3 Выходные документы

a) Все измененные входные документы.

b) Журнал изменений программного обеспечения (см. 9.2.4.11).

c) Новые конфигурационные записи.

6.6.4 Требования

6.6.4.1 В процессе управления изменениями необходимо определить, по крайней мере, следующее:

a) документацию, необходимую для информирования о проблеме и/или корректирующих действиях, с целью предоставления ее отвечающему за это руководству;

b) вид анализа информации, собранной в отчетах о проблеме, для идентификации ее причины;

c) методы, которые будут использоваться для создания отчетов, прослеживания и решения проблем, идентифицированных как на стадии разработки, так и во время поддержки программного обеспечения;

d) конкретные организационные обязанности, связанные с разработкой и поддержкой программного обеспечения;

e) как применить средства управления, чтобы гарантировать, что корректирующие действия приняты и они эффективны;

f) анализ влияния результата изменений на разрабатываемый компонент программного обеспечения или уже поставленный;

g) анализ влияния должен предусмотреть повторную проверку, повторное подтверждение соответствия и повторное оценивание, необходимые для изменения;

h) если применены многократные изменения, то необходимо рассмотреть влияние кумулятивного эффекта.

Примечание — Несколько изменений, могут кумулятивно потребовать полного повторного тестирования;

i) разрешение на изменение перед реализацией.

6.6.4.2 Все изменения должны инициировать возврат к надлежащей стадии жизненного цикла. Затем все последующие стадии должны быть выполнены в соответствии с процедурами, определенными для конкретных стадий в соответствии с требованиями настоящего стандарта.

6.7 Инструментальные средства поддержки и языки

6.7.1 Цели

Представить свидетельства о том, что возможные отказы инструментальных средств не оказывают негативное влияние на выход интегрированного комплекса инструментальных средств в его связанном с безопасностью применении, которые не выявляются техническими и/или организационными мерами вне инструментальных средств. С этой целью инструментальные средства для программного обеспечения разделены в три класса Т1, Т2 и Т3 (см. определения в 3.1).

Если инструментальные средства используются вместо операций, выполняемых вручную, то доказательство полноты выхода инструментальных средств может быть представлено теми же этапами процесса, как будто этот выход был получен в результате операции, выполненной вручную. Эти этапы процесса могли бы быть заменены альтернативными методами, если дана аргументация о полноте выхода инструментальных средств, и уровень полноты программного обеспечения в результате такой замены не снижен.

6.7.2 Входные документы

Спецификация инструментальных средств или руководство по инструментальным средствам.

6.7.3 Выходные документы

Отчет о подтверждении соответствия инструментальных средств (при необходимости см. 6.7.4.4 или 6.7.4.6).

6.7.4 Требования

6.7.4.1 Инструментальные средства для программного обеспечения должны выбираться как согласованная часть действий по разработке программного обеспечения.

Примечание — Соответствующие инструментальные средства, поддерживающие разработку программного обеспечения, используются для увеличения полноты программного обеспечения, снижая вероятность введения или не обнаружения сбоев во время разработки. Примеры инструментальных средств, относящихся к стадиям жизненного цикла разработки программного обеспечения, включают:

a) инструментальные средства преобразования или трансляции, которые конвертируют программное обеспечение или представление проекта (например, текст или схему) от одного уровня абстракции до другого: инструментальные средства усовершенствования проекта, компиляторы, ассемблеры, компоновщики, редакторы связей, загрузчики и инструментальные средства генерации кода;

b) инструментальные средства проверки и подтверждения соответствия такие, как статические анализаторы кода, мониторы тестового охвата, «помощники» средств доказательства теорем, средства моделирования и средства проверки моделей;

c) инструментальные средства диагностики, используемые для поддержки и контроля программного обеспечения в условиях эксплуатации;

d) инструментальные средства инфраструктуры, такие как системы поддержки разработки;

e) инструментальные средства управления конфигурацией, такие как инструментальные средства управления версиями;

f) инструментальные средства данных приложения, которые производят или поддерживают данные, требующиеся для определения параметров и создания системных функций, например, параметры функции, диапазоны инструментов, уровни уставок и блокировок аварийной сигнализации, состояния выхода, которые будут определены как отказ, географическое расположение.

Выбранные инструментальные средства в состоянии взаимодействовать. В этом контексте инструментальные средства взаимодействуют, если выходы одного инструментального средства имеют соответствую-

щее содержание и формат для автоматической передачи на вход к последующему инструментальному средству, таким образом, минимизируя возможность введения ошибки человеком при доработке промежуточных результатов.

Инструментальные средства обычно выбираются и демонстрируются так, чтобы они были совместимыми с потребностями приложения.

Также рассматривается доступность подходящих инструментальных средств, предоставляющих определенные сервисы, которые необходимы на протяжении всего жизненного цикла программного обеспечения.

6.7.4.2 Выбор инструментальных средств для классов Т2 и Т3 должен быть обоснован (см. 7.3.4.12). Обоснование должно включать идентификацию возможных отказов, которые могут оказаться в выходном результате инструментальных средств, и меры для предотвращения или обработки таких отказов.

6.7.4.3 Все инструментальные средства для классов Т2 и Т3 должны иметь спецификацию или руководство, которое четко определяет поведение этого инструментального средства и любые команды или ограничения на его использование.

6.7.4.4 Для каждого инструментального средства в классе Т3 доказательство должно быть доступным, то есть либо выход инструментального средства соответствует спецификации выхода, или обнаружены отказы в выходе. Доказательство может основываться на тех же шагах, которые необходимы для выполняемого процесса вручную вместо инструментального средства, и представленном объяснении, если эти шаги заменены альтернативными (например на подтверждении соответствия инструментального средства). Доказательство может также основываться на:

a) подходящей комбинации истории успешного использования в аналогичном окружении и для подобных применений (в данной или других организациях);

b) подтверждении соответствия инструментального средства, как определено в 6.7.4.5;

c) разнообразном избыточном коде, который позволяет обнаружить и управлять отказами, полученными в результате сбоев, внесенных инструментальным средством;

d) соответствии с уровнями полноты безопасности, полученными из анализа рисков процесса и процедур, включая инструментальные средства;

e) других подходящих методах для предотвращения или обнаружения и управления отказами, внесенными инструментальными средствами.

Примечания

1 История версий может гарантировать зрелость инструментального средства и записи ошибок / неоднозначностей, связанных с его использованием в некотором окружении.

2 Доказательство, перечисленное для Т3, может также использоваться для инструментальных средств Т2 при оценке правильности их результатов.

6.7.4.5 Результаты подтверждения соответствия инструментального средства должны быть документально оформлены, включая следующие результаты:

a) запись действий, выполняемых для подтверждения соответствия;

b) версия используемого руководства инструментального средства;

c) функции инструментального средства, подтверждение соответствия которых было выполнено;

d) используемые инструменты и оборудование;

e) результаты действия, выполняемого для подтверждения соответствия, документально оформленные результаты подтверждения соответствия должны установить, что либо программное обеспечение проходит подтверждение соответствия, либо определены причины его отказа;

f) тестовые сценарии и их результаты для последующего анализа;

g) несоответствия между ожидаемыми и фактическими результатами.

6.7.4.6 Если доказательство соответствия 6.7.4.4 недоступно, то должны быть эффективные меры управления отказами исполнимого, связанного с безопасностью программного обеспечения, которые следуют из сбоев, относящихся к инструментальному средству.

Примечания

1 Примером является генерация разнообразного избыточного кода, который позволяет обнаружение и управление отказами, к которым приводят сбои, внесенные транслятором.

2 Например, пригодность для определенной цели не доверенного компилятора может быть обоснована следующим образом.

Объектный код, произведенный компилятором, был подвергнут комбинации тестов, проверок и исследований, которые способны обеспечить правильность кода до такой степени, что он соответствует целевому уровню полноты безопасности. В частности ко всем тестам, проверкам и исследованиям применяется следующее:

- должно быть показано, что тестирование имеет достаточно высокий охват реализованного кода. Если существует какой-либо код, недостижимый тестами, то должно быть показано проверками или исследованиями, что функция, в реализации которой применяется этот код, выполняется правильно, если код достигнут на цели;

- проверки и исследования были применены к объектному коду и было показано, что они способны к обнаружению типов ошибок, которые могли появиться в результате дефекта в компиляторе;

- после тестирования, проверки и анализа транслятор и компилятор не применялись;

- если будет выполнена еще одна компиляция или трансляция, то все тесты, проверки и исследования будут выполнены повторно.

6.7.4.7 Выбранное представление программного обеспечения или проекта (включая язык программирования) должно:

a) иметь транслятор, который был оценен на пригодность для определенной цели, включая, где это необходимо, несоответствие с международными или национальными стандартами;

b) соответствовать характеристикам применения;

c) содержать функции, которые упрощают обнаружение ошибок в проекте или в программе;

d) поддерживать функции, которые соответствуют методу разработки.

Язык программирования — это одно из класса представлений программного обеспечения или проекта. Транслятор преобразует представление программного обеспечения или проекта (например, текст или схему) из одного уровня абстракции к другому уровню. Примеры трансляторов включают: инструментальные средства усовершенствования проекта, компиляторы, ассемблеры, компоновщики, редакторы связей, загрузчики и инструментальные средства генерации кода.

Оценка транслятора может быть выполнена для конкретного проекта применения или для класса применений. В последнем случае вся необходимая информация об инструментальном средстве, связанная с намеченным и надлежащим использованием инструментального средства, должна быть доступна пользователю инструментального средства. Оценка инструментального средства для конкретного проекта может в таком случае быть сокращена до проверки общей пригодности инструментального средства для проекта и соответствия со «спецификацией или руководством» (т. е. надлежащее использование инструментального средства). Надлежащее использование могло бы включать дополнительные действия проверки в конкретном проекте.

Для подтверждения соответствия может использоваться набор тестов, чтобы оценить пригодность для определенной цели некоторого транслятора согласно заданным критериям, которые должны включать функциональные и нефункциональные требования. Для функциональных требований транслятора основным методом подтверждения соответствия может быть динамическое тестирование. Если возможно, должен использоваться автоматический набор тестов.

6.7.4.8 Если требования 6.7.4.7 не могут быть полностью выполнены, то пригодность для определенной цели языка и любые дополнительные меры, которые разрешают вопросы с любыми идентифицированными недостатками языка, должны быть обоснованы и оценены.

Примечание — См. 6.7.4.6, примечание 2.

6.7.4.9 Если выполняется автоматическая генерация кода или подобная автоматическая трансляция, то пригодность автоматического транслятора для разработки связанного с безопасностью программного обеспечения должна быть оценена на стадии жизненного цикла разработки, где выбираются инструментальные средства обеспечения разработки.

6.7.4.10 Управление конфигурацией должно гарантировать, что для инструментальных средств классов Т2 и Т3 используются только обоснованные версии.

6.7.4.11 Каждая новая версия используемого инструментального средства должна быть обоснована (см. таблицу 1). Это обоснование может полагаться на доказательство, предусмотренное для более ранней версии, если используются достаточные доказательства того, что:

a) функциональные различия (если таковые имеются) не будут влиять на совместимость инструментального средства с остальной частью комплекса инструментальных средств;

b) новая версия вряд ли будет содержать существенно новые, неизвестные отказы.

Примечание — Доказательство того, что новая версия вряд ли будет содержать существенно новые, неизвестные отказы, может основываться на достоверной идентификации сделанных изменений, и на анализе выполненных действий по проверке и подтверждению соответствия.

6.7.4.12 Отношение между классами инструментальных средств и применяемыми подпунктами настоящего стандарта определено в таблице 1.

Таблица 1 — Связь между классами инструментальных средств и применяемыми подпунктами настоящего стандарта

Класс инструментальных средств	Применяемые подпункты
Т1	6.7.4.1
Т2	6.7.4.1, 6.7.4.2, 6.7.4.3, 6.7.4.10, 6.7.4.11
Т3	6.7.4.1, 6.7.4.2, 6.7.4.3, 6.7.4.4, 6.7.4.5 (или 6.7.4.6), 6.7.4.7, 6.7.4.8, 6.7.4.9, 6.7.4.10, 6.7.4.11

Примечание — Выбор и применение инструментальных средств обычно согласуется с полнотой безопасности разрабатываемых изделий. Отношение между классами инструментальных средств и методологиями обеспечения их достоверности, соответствующими конкретным УПБ, для разрабатываемых изделий, где используются такие инструментальные средства, представлено в таблице 2.

Таблица 2 — Отношение между классом инструментальных средств и УПБ изделия

Класс инструментальных средств	Методология обеспечения достоверности инструментального средства	УПБ разрабатываемого изделия
Т1	Не требуется	Нет необходимости
Т2	1 Доказательство успешного использования в аналогичном окружении. или 2 Управление библиотекой тестовых сценариев с детерминированными результатами для установления функциональной полноты	1—2
Т3 и Т3	1 Выполнение всех требований настоящего стандарта, подходящих для конкретного УПБ применения к разработке или приобретению инструментального средства. или 2 Управление библиотекой широко признанных тестовых случаев/наборов тестов с детерминированными результатами для установления функциональной полноты. или 3 Применение разнообразных инструментальных средств к системе и сравнение производительности разрабатываемого изделия, проверка различий	3—4

7 Разработка универсального программного обеспечения

7.1 Жизненный цикл и документация универсального программного обеспечения

7.1.1 Цели

Предоставлять описание самого программного обеспечения, начиная с более высоких уровней абстракции и доходя до подробных уточнений, чтобы создать общую структуру для демонстрации достигнутой безопасности, а также для будущих действий по сопровождению.

7.1.2 Требования

7.1.2.1 В объеме требований уровня полноты безопасности программного обеспечения для комплексного программного обеспечения должны быть представлены документы, перечисленные в таблице А.1.

7.1.2.2 Последовательность поставляемых документов, как они описаны в таблице А.1, отражает идеальную линейную потоковую модель. Однако на эту модель не следует ссылаться, как на строгую последовательность действий и описание связи между ними, поскольку обычно трудно достигнуть ее строгого соответствия с практикой. Стадии могут пересекаться, но действия по проверке и подтверждению соответствия должны демонстрировать согласованность входов и выходов (документов и программного обеспечения) внутри и между стадиями.

Однако основная цель предусмотренной документации состоит в том, чтобы предоставить описание самого программного обеспечения, начиная с более высоких уровней абстракции и доходя до подробных уточнений, чтобы создать общую структуру для демонстрации достигнутой безопасности, а также для будущих действий по поддержке.

7.2 Требования к программному обеспечению

7.2.1 Цели

7.2.1.1 Описать полный набор требований к программному обеспечению, удовлетворяющих всей системе, и требования к безопасности, относящиеся к программному обеспечению, а также обеспечить исчерпывающий набор документов для каждой последующей стадии.

7.2.1.2 Описать спецификацию тестов для всего программного обеспечения.

7.2.2 Входные документы

a) Спецификация требований к системе.

b) Спецификация требований к безопасности системы.

c) Описание архитектуры системы.

d) Спецификации внешних интерфейсов (например, программное обеспечение/спецификация интерфейса программного обеспечения, программное обеспечение/спецификация интерфейса аппаратных средств).

e) План обеспечения качества программного обеспечения.

f) План подтверждения соответствия программного обеспечения.

7.2.3 Выходные документы

a) Спецификация требований к программному обеспечению.

b) Спецификация тестирования ко всему программному обеспечению.

c) Отчет о проверке требований к программному обеспечению.

7.2.4 Требования

7.2.4.1 Спецификация требований к программному обеспечению должна быть подготовлена в письменном виде под руководством менеджера требований, на основе входных документов, представленных в 7.2.2.

Требования 7.2.4.2—7.2.4.15 относятся к спецификации требований к программному обеспечению.

7.2.4.2 Спецификация требований к программному обеспечению должна содержать требуемые свойства разрабатываемого программного обеспечения. Эти свойства, все (кроме безопасности) из которых определены в комплексе ИСО/МЭК 25010, должны включать:

a) функциональность (включая производительность и время отклика);

b) устойчивость и пригодность для обслуживания;

c) безопасность (включая функции безопасности и связанные с ними уровни полноты безопасности программного обеспечения);

d) эффективность;

e) практичность;

f) мобильность.

7.2.4.3 Уровень полноты безопасности программного обеспечения должен быть определен в соответствии с 4 и включен в спецификацию требований к программному обеспечению.

7.2.4.4 В объеме требований уровня полноты безопасности программного обеспечения спецификация требований к программному обеспечению должна быть сформирована и структурирована таким способом, чтобы она была:

a) подробной, понятной, точной, однозначной, поддающейся проверке, тестируемой, удобной в сопровождении и выполнимой;

b) прослеживаемой в обратном направлении ко всем входным документам.

7.2.4.5 Спецификация требований к программному обеспечению должна включать способы выражения и описаний, которые понятны ответственному персоналу, реализующему процессы жизненного цикла программного обеспечения.

7.2.4.6 Спецификация требований к программному обеспечению должна определить и документально оформить все интерфейсы с любой другой системой, с управляемым оборудованием либо внутри, либо извне его, включая интерфейсы операторов, везде, где существует или запланировано прямое подключение.

7.2.4.7 Все соответствующие режимы работы должны быть детализированы в спецификации требований к программному обеспечению.

7.2.4.8 Все соответствующие режимы поведения программируемой электроники, при определенном поведении отказа, должны быть документально оформлены или на них должна быть предусмотрена ссылка (например, на документацию уровня системы) в спецификации требований к программному обеспечению.

7.2.4.9 Любые ограничения между аппаратными средствами и программным обеспечением должны быть документально оформлены или на них должна быть предусмотрена ссылка (например, на документацию уровня системы) в спецификации требований к программному обеспечению.

7.2.4.10 В объеме требований к описанию документации на систему, спецификация требований к программному обеспечению должна рассмотреть самопроверку программного обеспечения и проверку программным обеспечением аппаратных средств. Самопроверка программного обеспечения состоит из обнаружения и создания отчетов по отказам и ошибкам самого программного обеспечения.

7.2.4.11 Спецификация требований к программному обеспечению должна включать требования для периодического испытания функций в объеме требований спецификации требований безопасности системы.

7.2.4.12 Спецификация требований к программному обеспечению должна включать требования, обеспечивающие тестируемость всех функций безопасности во время всего периода работы системы в объеме требований спецификации требований безопасности системы.

7.2.4.13 Все функции, которые будут выполнены программным обеспечением, особенно связанные с достижением требуемого уровня полноты безопасности системы, должны быть четко определены в спецификации требований к программному обеспечению.

7.2.4.14 Любые функции, не связанные с безопасностью, для выполнения которых требуется программное обеспечение, должны быть четко определены в спецификации требований к программному обеспечению.

7.2.4.15 Спецификация требований к программному обеспечению должна поддерживаться методами и мерами, представленными в таблице А.2. Выбранная комбинация должна быть обоснована как набор, удовлетворяющий 4.8 и 4.9.

7.2.4.16 Спецификация тестирования всего программного обеспечения должна быть подготовлена в письменном виде под руководством тестировщика на основе спецификации требований к программному обеспечению.

Требования 7.2.4.17—7.2.4.19 относятся к спецификации тестирования всего программного обеспечения.

7.2.4.17 Спецификация тестирования всего программного обеспечения представляет собой описание тестов, которые будут выполнены для завершенного программного обеспечения.

7.2.4.18 Для спецификации тестирования всего программного обеспечения должны быть выбраны методы и меры, представленные в таблице А.7. Выбранная комбинация должна быть обоснована как набор, удовлетворяющий требованиям 4.8 и 4.9.

7.2.4.19 Спецификация тестирования всего программного обеспечения должна определять для каждой требуемой функции тестовые сценарии, включая:

a) требуемые входные сигналы, их последовательности и их значения;

b) ожидаемые выходные сигналы, их последовательности и их значения;

c) критерии прохождения теста, включая вопросы производительности и качества.

7.2.4.20 Отчет о проверке требований к программному обеспечению должен быть подготовлен в письменном виде под руководством менеджера по проверке на основе спецификации требований безопасности к системе, спецификации требований к программному обеспечению, спецификации тестирования всего программного обеспечения и плана обеспечения качества программного обеспечения.

Требования 7.2.4.21, 7.2.4.22 относятся к отчету о проверке требований к программному обеспечению.

7.2.4.21 Отчет о проверке требований к программному обеспечению должен быть подготовлен в соответствии с общими требованиями, установленными для всех отчетов о проверке (см. 6.2.4.14).

7.2.4.22 После того, как сформирована спецификация требований к программному обеспечению, должна быть выполнена проверка:

a) адекватности спецификации требований к программному обеспечению при выполнении требований, установленных в спецификации требований к системе, спецификации требований к безопасности системы и плане обеспечения качества программного обеспечения;

b) того, что спецификация требований к программному обеспечению удовлетворяет общим требованиям удобочитаемости и прослеживаемости, представленным в 5.3.2.7—5.3.2.10 и в 6.5.4.14—6.5.4.17, а также требованиям, определенным в 7.2.4.2—7.2.4.15;

c) адекватности спецификации тестирования всего программного обеспечения в виде теста спецификации требований к программному обеспечению;

d) определения любого дополнительного действия, чтобы продемонстрировать корректный охват нетестируемых требований;

e) внутренней непротиворечивости спецификации требований к программному обеспечению;

f) достоверности спецификации требований к программному обеспечению при выполнении или учете ограничений между аппаратными средствами и программным обеспечением.

Эти результаты должны быть включены в отчет о проверке требований к программному обеспечению.

7.3 Архитектура и проектирование

7.3.1 Цели

7.3.1.1 Разработать архитектуру программного обеспечения, которая удовлетворяет требованиям к программному обеспечению.

7.3.1.2 Определить и оценить значение взаимодействия аппаратных средств/программного обеспечения для безопасности.

7.3.1.3 Выбрать метод разработки, если он не был ранее определен.

7.3.1.4 Спроектировать программное обеспечение определенного уровня полноты безопасности программного обеспечения на основе входных документов.

7.3.1.5 Гарантировать, что результирующая система и ее программное обеспечение будут готовы к тестированию с самого начала. Поскольку проверка и тест будут критическим элементом при подтверждении соответствия, то определенное внимание должно быть уделено проверке и тестовым потребностям в течение реализации.

7.3.2 Входные документы

Спецификация требований к программному обеспечению.

7.3.3 Выходные документы

a) Спецификация архитектуры программного обеспечения.

b) Спецификация проекта программного обеспечения.

c) Спецификации интерфейса программного обеспечения.

d) Спецификация теста интеграции программного обеспечения.

e) Спецификация теста интеграции программного обеспечения/аппаратных средств.

f) Отчет об архитектуре программного обеспечения и о проверке проекта.

7.3.4 Требования

7.3.4.1 Спецификация архитектуры программного обеспечения должна быть подготовлена в письменном виде под руководством проектировщика, на основе спецификации архитектуры программного обеспечения.

Требования 7.3.4.2—7.3.4.14 относятся к спецификации архитектуры программного обеспечения.

7.3.4.2 Предложенная архитектура программного обеспечения должна быть определена и детально описана в спецификации архитектуры программного обеспечения.

7.3.4.3 Спецификация архитектуры программного обеспечения должна рассмотреть возможность выполнимости спецификации требований к программному обеспечению для требуемого уровня полноты безопасности программного обеспечения.

Примечание — Архитектура программного обеспечения стремится минимизировать размер и сложность связанной с безопасностью части в приложении.

7.3.4.4 Спецификация архитектуры программного обеспечения должна определить, проанализировать и детализировать значение всех взаимодействий аппаратных средств/программного обеспечения.

7.3.4.5 В спецификации архитектуры программного обеспечения должны быть определены все компоненты программного обеспечения и для этих компонентов определены:

a) новые ли эти компоненты или существующие;

b) было ли для этих компонентов ранее выполнено подтверждение соответствия и если было, то их условия подтверждения соответствия;

c) уровень полноты безопасности программного обеспечения компонента.

7.3.4.6 Компоненты программного обеспечения должны быть:

a) охвачены определенным подмножеством требований к программному обеспечению;

b) четко идентифицированы и иметь независимые версии в системе управления конфигурацией.

7.3.4.7 Использование существующего ранее программного обеспечения должно удовлетворять следующим ограничениям.

a) Для всех уровней полноты безопасности программного обеспечения выравнивается, следующая информация должна четко быть определена и документально оформлена:

- требования, которые существующее ранее программное обеспечение должно выполнить;
- предположения об окружении существующего ранее программного обеспечения;
- интерфейсы с другими частями программного обеспечения.

b) Для всех уровней полноты безопасности программного обеспечения выравнивается, существующее ранее программное обеспечение должно быть включено в процесс подтверждения соответствия всего программного обеспечения.

c) Для уровней полноты безопасности программного обеспечения УПБ 3 или УПБ 4 должны быть выполнены следующие предупредительные меры:

- должен быть выполнен анализ возможных отказов существующего ранее программного обеспечения и их последствий для всего программного обеспечения;
- должна быть определена стратегия, чтобы обнаружить отказы в существующем ранее программном обеспечении и защитить систему от этих отказов;
- процесс проверки и подтверждения соответствия должен гарантировать то, что:

1) существующее ранее программное обеспечение выполняет выделенные требования,

2) отказы существующего ранее программного обеспечения обнаружены и система, в которую интегрировано существующее ранее программное обеспечение, защищена от этих отказов,

3) предположения об окружении существующего ранее программного обеспечения выполнены.

d) Существующее ранее программное обеспечение должно сопровождаться достаточно точным (например, ограничиваться используемыми функциями), и полным описанием (т. е. функции, ограничения и доказательство). Описание должно включать ограничения на аппаратные средства и/или программное обеспечение, которые интегратор должен знать и учитывать во время применения. В частности это формирует механизм для информирования интегратора о том, для чего было разработано программное обеспечение, его свойства, поведение и характеристики.

Примечание — В стратегии подтверждения соответствия существующего ранее программного обеспечения могут быть использованы статистические данные.

7.3.4.8 Использованию в проекте существующих проверенных компонентов программного обеспечения, разработанных в соответствии с настоящим стандартом, должно быть по мере возможности отдано предпочтение.

7.3.4.9 Если программное обеспечение состоит из компонентов с различными значениями уровней полноты безопасности программного обеспечения, то все компоненты программного обеспечения нужно рассматривать как принадлежащие самому высокому из этих уровней, если не имеется доказательства независимости между компонентами программного обеспечения с более высокими уровнем полноты безопасности и компонентами программного обеспечения с более низким уровнем полноты безопасности. Это доказательство должно быть включено в спецификацию архитектуры программного обеспечения.

7.3.4.10 Спецификация архитектуры программного обеспечения должна содержать описание стратегии разработки программного обеспечения в объеме требований определенного уровня полноты безопасности программного обеспечения. Спецификация архитектуры программного обеспечения должна быть представлена и структурирована так, чтобы она была:

a) завершенной, непротиворечивой, ясной, точной, однозначной, поддающейся проверке, тестируемой, удобной в сопровождении и выполнимой;

b) прослеживаемой к исходной спецификации требований к программному обеспечению.

7.3.4.11 Для достижения баланса между стратегиями предотвращения сбоев и устранения сбоев в спецификацию архитектуры программного обеспечения должны быть включены меры для обработки сбоев.

7.3.4.12 Спецификация архитектуры программного обеспечения должна содержать обоснование, что эти методы, меры и инструменты выбраны из набора, который удовлетворяет спецификации требований к программному обеспечению для требуемого уровня полноты безопасности программного обеспечения.

7.3.4.13 Спецификация архитектуры программного обеспечения должна учитывать требования 8.4.8, если программное обеспечение будет сконфигурировано данными или алгоритмами приложений.

7.3.4.14 Спецификация архитектуры программного обеспечения должна содержать методы и меры, выбранные из таблицы А.3. Выбранная комбинация должна быть обоснована как набор, удовлетворяющий требованиям 4.8 и 4.9.

7.3.4.15 Размер и сложность разработанной архитектуры программного обеспечения должны быть сбалансированы.

7.3.4.16 Чтобы выявить требования или получить более подробное представление о требованиях и их последствиях, на любой стадии может использоваться прототипирование.

7.3.4.17 Код из прототипа может использоваться в целевой системе, только если продемонстрировано, что код, его разработка и документация на него удовлетворяют требованиям настоящего стандарта.

7.3.4.18 Спецификация интерфейса программного обеспечения для всех интерфейсов между компонентами программного обеспечения и внешних интерфейсов всего комплекса программного обеспечения должна быть подготовлена письменно под руководством проектировщика, на основе спецификации требований к программному обеспечению и спецификации архитектуры программного обеспечения.

Требование в 7.3.4.19 относится к спецификации интерфейса программного обеспечения.

7.3.4.19 Описание интерфейсов должно содержать:

a) (пред)постусловия;

b) определение и описание всех граничных значений для всех определенных данных;

c) выполняемые действия в случае превышения граничного значения;

d) выполняемые действия в случае достижения граничного значения;

e) для критичных по времени входных и выходных данных:

1) временные ограничения и требования для корректной работы,

2) управление исключениями;

f) информацию о выделенной памяти для интерфейсных буферов и механизмах обнаруживающих, о том, что память не может быть выделена или все буферы полны, где это применимо;

g) информацию о существовании механизмов синхронизации между функциями [см. перечисление e)].

Все данные от и до интерфейсов должны быть определены для всего диапазона значений, определенных типом данных, включая диапазоны, которые не используются, после обработки функциями;

h) определение и описание всех классов эквивалентности для всех определенных данных и каждой функции программного обеспечения, использующих их;

i) определение неиспользованных или запрещенных классов эквивалентности.

Примечание — Тип данных включает в себя описание следующих данных:

a) входные параметры и выходные результаты функций и/или процедур;

b) данные, определенные в телеграммах или коммуникационных пакетах;

c) данные от аппаратных средств.

7.3.4.20 Спецификация проекта программного обеспечения должна быть подготовлена в письменном виде под руководством проектировщика, на основе спецификации требований к программному обеспечению, спецификации архитектуры программного обеспечения и спецификации интерфейса программного обеспечения.

Требования 7.3.4.21—7.3.4.24 относятся к спецификации проекта программного обеспечения.

7.3.4.21 Входные документы должны быть доступны, хотя и не обязательно в завершенном виде, до начала процесса проектирования.

7.3.4.22 В спецификации проекта программного обеспечения должен быть описан проект программного обеспечения, его декомпозиция на компоненты, и каждый компонент должен иметь спецификацию проекта компонента программного обеспечения и спецификацию тестирования компонента программного обеспечения.

7.3.4.23 Спецификация проекта программного обеспечения должна рассматривать:

a) компоненты программного обеспечения, прослеженные до архитектуры программного обеспечения, и их уровень полноты безопасности;

b) интерфейсы компонентов программного обеспечения с окружением;

c) интерфейсы между компонентами программного обеспечения;

d) структуры данных;

e) распределение и прослеживание требований к компонентам;

f) основные алгоритмы и управление выполнением;

g) механизмы сообщения об ошибках.

7.3.4.24 Спецификация проекта программного обеспечения должна содержать методы и меры, выбранные из таблицы А.4. Выбранная комбинация должна быть обоснована как набор, удовлетворяющий требованиям 4.8 и 4.9.

7.3.4.25 Должны быть разработаны стандарты кодирования, определяющие:

a) хорошую практику программирования, как определено в таблице А.12;

b) меры для предотвращения или обнаружения ошибок, которые могут быть сделаны во время применения языка и не обнаружены во время проверки (см. 7.5 и 7.6). Такие отказы выводят в результате анализа всех функций языка;

c) процедуры для формирования документации на исходный код.

7.3.4.26 Выбор стандарта кодирования должен быть обоснован в объеме требований, который соответствует уровню полноты безопасности программного обеспечения.

7.3.4.27 Стандарты кодирования должны использоваться для разработки всего программного обеспечения и на них должны быть ссылки в плане обеспечения качества программного обеспечения.

7.3.4.28 В соответствии с требуемым уровнем полноты безопасности программного обеспечения выбранный метод его разработки должен обладать функциями, которые упрощают:

a) абстрагирование, декомпозицию на модули и другие функции, которые управляют сложностью;

b) ясное и точное выражение:

1) функциональности,

2) информационного потока между компонентами,

3) управления выполнением и соответствующей временной информации,

4) параллелизма,

5) структуры и свойств данных;

c) понимание человеком;

d) проверка и подтверждение соответствия;

e) поддержку программного обеспечения.

7.3.4.29 Спецификация тестирования интеграции программного обеспечения должна быть подготовлена в письменном виде под руководством интегратора на основе спецификации требований к программному обеспечению, спецификации архитектуры программного обеспечения, спецификации проекта программного обеспечения и спецификаций интерфейса программного обеспечения.

Требования 7.3.4.30—7.3.4.32 относятся к спецификации тестирования интеграции программного обеспечения.

7.3.4.30 Спецификация тестирования интеграции программного обеспечения должна быть представлена в соответствии с общими требованиями, установленными для спецификации тестирования (см. 6.1.4.4).

7.3.4.31 Спецификация тестирования интеграции программного обеспечения должна включать следующее:

a) необходимо показать, что каждый компонент программного обеспечения реализует определенные интерфейсы для других компонентов, обеспечивая совместное выполнение компонент;

b) необходимо показать, что программное обеспечение ведет себя надлежащим способом, когда на входе интерфейса появляются не специфицированные данные;

c) требуемые входные данные с их последовательностями и их значениями должны быть основой тестовых сценариев;

d) ожидаемые выходные данные с их последовательностями и их значениями должны быть основанием тестовых сценариев;

e) необходимо показать, какие результаты теста компонента (см. 7.5.4.5 и 7.5.4.7) предполагают повторное применение теста интеграции программного обеспечения.

7.3.4.32 Спецификация тестирования интеграции программного обеспечения должна содержать методы и меры, выбранные из таблицы А.5. Выбранная комбинация должна быть обоснована как набор, удовлетворяющий требованиям 4.8 и 4.9.

7.3.4.33 Спецификация тестирования интеграции программного обеспечения/аппаратных средств должна быть подготовлена в письменном виде под руководством интегратора, на основе описания проекта системы, спецификации требований к программному обеспечению, спецификации архитектуры программного обеспечения и спецификации проекта программного обеспечения.

Требования 7.3.4.34—7.3.4.39 относятся к спецификации тестирования интеграции программного обеспечения/аппаратных средств.

7.3.4.34 Спецификация тестирования интеграции программного обеспечения/аппаратных средств должна быть создана на ранних стадиях разработки в жизненном цикле, чтобы тестирование интеграции могло быть должным образом направлено и чтобы требования определенного проекта или другой интеграции могли быть соответственно предусмотрены. В зависимости от размера системы спецификация тестирования интеграции программного обеспечения/аппаратных средств может быть разделена во время разработки на ряд дочерних документов и естественным образом добавлена, поскольку аппаратные и программные проекты развиваются и более детальные требования интеграции становятся более понятными.

7.3.4.35 Спецификация тестирования интеграции программного обеспечения/аппаратных средств должна различить те действия, которые могут быть выполнены поставщиком на его территории и теми, которые требуют доступа к месту расположения пользователя.

7.3.4.36 Спецификация тестирования интеграции программного обеспечения/аппаратных средств должна включать следующее:

a) необходимо показать, что программное обеспечение работает на аппаратных средствах надлежащим способом, используя аппаратные средства через определенные аппаратные интерфейсы;

b) необходимо показать, что в случае необходимости программное обеспечение может обработать сбои аппаратных средств;

c) требуемые согласование по времени и производительность должны быть продемонстрированы;

d) требуемые входные данные с их последовательностями и их значениями должны быть основой тестовых сценариев;

e) ожидаемые выходные данные с их последовательностями и их значениями должны быть основой тестовых сценариев;

f) необходимо показать, какие результаты тестирования компонента (см. 7.5.4.5) и тестирования интеграции программного обеспечения (см. 7.6.4.3) предполагают повторное применение тестирования интеграции программного обеспечения/аппаратных средств.

7.3.4.37 Спецификация тестирования интеграции программного обеспечения/аппаратных средств должна содержать следующие документы, описывающие:

a) тестовые сценарии и данные тестирования;

b) типы выполняемых тестов;

c) тестовое окружение, включая инструментальные средства, программное обеспечение поддержки и описание конфигурации;

d) тестовые критерии, по которым будет оценено завершение теста.

7.3.4.38 Спецификация тестирования интеграции программного обеспечения/аппаратных средств должна быть представлена в соответствии с общими требованиями, установленными для спецификации тестирования (см. 6.1.4.4).

7.3.4.39 Спецификация тестирования интеграции программного обеспечения/аппаратных средств должна содержать методы и меры, выбранные из таблицы А.5. Выбранная комбинация должна быть обоснована как набор, удовлетворяющий требованиям 4.8 и 4.9.

7.3.4.40 Отчет о проверке архитектуры и проекта программного обеспечения должен быть подготовлен в письменном виде под руководством менеджера по проверке, на основе спецификации

требований к программному обеспечению, спецификации архитектуры программного обеспечения, спецификации проекта программного обеспечения, спецификации тестирования интеграции программного обеспечения и спецификации тестирования интеграции программного обеспечения/аппаратных средств.

Требования 7.3.4.41—7.3.4.43 относятся к отчету о проверке архитектуры и проекта программного обеспечения.

7.3.4.41 Отчет о проверке архитектуры и проекта программного обеспечения должен быть подготовлен в соответствии с общими требованиями, установленными для отчета о проверке (см. 6.2.4.14).

7.3.4.42 После определения архитектуры программного обеспечения и спецификаций проекта и интерфейса должна быть выполнена проверка, устанавливающая:

a) внутреннюю непротиворечивость архитектуры программного обеспечения, спецификаций интерфейса и проекта;

b) непротиворечивость и полноту спецификаций архитектуры программного обеспечения, проекта и интерфейса, адекватно реализующих спецификацию требований к программному обеспечению;

c) что спецификация архитектуры программного обеспечения удовлетворяет общим требованиям удобочитаемости и прослеживаемости, представленным в 5.3.2.7—5.3.2.10 и в 6.5.4.14—6.5.4.16, а также требованиям, определенным в 7.3.4.1—7.3.4.14;

d) что спецификация интерфейса программного обеспечения удовлетворяет общим требованиям удобочитаемости и прослеживаемости, представленным в 5.3.2.7—5.3.2.10 и в 6.5.4.14—6.5.4.16, а также требованиям, определенным в 7.3.4.18, 7.3.4.19;

e) что спецификация проекта программного обеспечения удовлетворяет общим требованиям удобочитаемости и прослеживаемости, представленным в 5.3.2.7—5.3.2.10 и в 6.5.4.14—6.5.4.16, а также требованиям, определенным в 7.3.4.20—7.3.4.24;

f) соответствие спецификации архитектуры программного обеспечения и спецификации проекта программного обеспечения при учете ограничений аппаратных средств и программного обеспечения.

Данные результаты должны быть включены в отчет о проверке архитектуры и проекта программного обеспечения.

7.3.4.43 После выполнения интеграции программного обеспечения и установления спецификации тестирования интеграции программного обеспечения/аппаратных средств должна быть выполнена проверка, устанавливающая:

a) что спецификация тестирования интеграции программного обеспечения удовлетворяет общим требованиям удобочитаемости и прослеживаемости, представленным в 5.3.2.7—5.3.2.10 и в 6.5.4.14—6.5.4.16, а также требованиям, определенным в 7.3.4.29—7.3.4.32;

b) что спецификация тестирования интеграции программного обеспечения/аппаратных средств удовлетворяет общим требованиям удобочитаемости и прослеживаемости, представленным в 5.3.2.7—5.3.2.10 и в 6.5.4.14—6.5.4.16, а также требованиям, определенным в 7.3.4.33—7.3.4.39. Результаты должны быть включены в отчет о проверке архитектуры и проекта программного обеспечения.

7.4 Проектирование компонент

7.4.1 Цели

7.4.1.1 Разработать проект компонента программного обеспечения, который удовлетворяет спецификации проекта программного обеспечения в объеме требований уровня полноты безопасности программного обеспечения.

7.4.1.2 Разработать спецификацию теста компонента программного обеспечения, которая удовлетворяет спецификации проекта компонента программного обеспечения в объеме требований определенного уровня полноты безопасности программного обеспечения.

7.4.2 Входные документы

Спецификация проекта программного обеспечения.

7.4.3 Выходные документы

a) Спецификация проекта компонента программного обеспечения.

b) Спецификация тестирования компонента программного обеспечения.

c) Отчет о проверке проекта компонента программного обеспечения.

7.4.4 Требования

7.4.4.1 Для каждого компонента спецификация проекта компонента программного обеспечения должна быть подготовлена в письменном виде под руководством проектировщиком, на основе спецификации проекта программного обеспечения.

Требования 7.4.4.2—7.4.4.6 относятся к спецификации проекта компонента программного обеспечения.

7.4.4.2 Для каждого компонента программного обеспечения должна быть доступна следующая информация:

- автор,
- история конфигурации,
- краткое описание.

История конфигурации должна включать точную идентификацию текущей и всех предыдущих версий компонента, определяя версию, дату и автора, а также описание изменений, внесенных в предыдущую версию.

7.4.4.3 Спецификация проекта компонента программного обеспечения должна содержать:

a) идентификацию всех низкоуровневых модулей программного обеспечения (например, подпрограмм, методов, процедур), прослеженных до верхнего уровня;

b) подробное описание их интерфейсов с окружением и другими компонентами с подробным описанием входов и выходов;

c) их уровень полноты безопасности без дальнейшего распределения внутри самого компонента;

d) подробное описание их алгоритмов и структур данных.

Каждая спецификация проекта компонента программного обеспечения сама должна быть непротиворечива и позволять преобразовывать ее в код соответствующих компонентов.

7.4.4.4 Каждая спецификация проекта компонента программного обеспечения должна быть читаемой, понятной и тестируемой.

7.4.4.5 Размер и сложность каждого разработанного компонента программного обеспечения должны быть сбалансированы.

7.4.4.6 Спецификация проекта компонента программного обеспечения должна содержать методы и меры, выбранные из таблицы А.4. Выбранная комбинация должна быть обоснована как набор, удовлетворяющий требованиям 4.8 и 4.9.

7.4.4.7 Для каждого компонента спецификация тестирования компонента программного обеспечения должна быть подготовлена в письменном виде под руководством тестировщика, на основе спецификации проекта компонента программного обеспечения.

Требования 7.4.4.8—7.4.4.10 относятся к спецификации тестирования компонента программного обеспечения.

7.4.4.8 Спецификация тестирования компонента программного обеспечения должна быть подготовлена в соответствии с общими требованиями, установленными для спецификации тестирования (см. 6.1.4.4).

7.4.4.9 Должна быть сформирована такая спецификация тестирования компонента программного обеспечения, используя которую компонент должен быть протестирован. Эти тесты должны показать, что каждый компонент выполняет свою предназначенную функцию. Спецификация тестирования компонента программного обеспечения должна определить и обосновать требуемые критерии и степень тестового охвата в объеме требований уровня полноты безопасности программного обеспечения. Тесты должны быть разработаны для выполнения трех целей:

a) подтвердить, что компонент выполняет предназначенные для него функции (тестирование методом «черного ящика»);

b) проверить, как внутренние части компонента взаимодействуют для выполнения предназначенной функции (тестирование методом «черного/белого ящика»);

c) подтвердить, что все части компонента протестированы (тестирование методом «белого ящика»).

7.4.4.10 Спецификация тестирования компонента программного обеспечения должна содержать методы и меры, выбранные из таблицы А.5. Выбранная комбинация должна быть обоснована как набор, удовлетворяющий требованиям 4.8 и 4.9.

7.4.4.11 Отчет о проверке проекта компонента программного обеспечения должен быть подготовлен в письменном виде под руководством менеджера по проверке, на основе спецификации проекта программного обеспечения, спецификации проекта компонента программного обеспечения и спецификации тестирования компонента программного обеспечения.

Требования 7.4.4.12, 7.4.4.13 относятся к отчету о проверке проекта компонента программного обеспечения.

7.4.4.12 Отчет о проверке проекта компонента программного обеспечения должен быть подготовлен в соответствии с общими требованиями, установленными для отчета о проверке (см. 6.2.4.14).

7.4.4.13 После установления спецификации проекта для каждого компонента программного обеспечения должна быть выполнена проверка, устанавливающая, что:

a) выполнение спецификации проекта компонента программного обеспечения соответствует выполнению спецификации проекта программного обеспечения;

b) спецификация проекта компонента программного обеспечения удовлетворяет общим требованиям удобочитаемости и прослеживаемости, представленным в 5.3.2.7—5.3.2.10 и в 6.5.4.14—6.5.4.17, а также требованиям, определенным в 7.4.4.1—7.4.4.6;

c) спецификация тестирования компонента программного обеспечения соответствует набору тестовых сценариев для спецификации проекта компонента программного обеспечения;

d) спецификация тестирования компонента программного обеспечения удовлетворяет общим требованиям удобочитаемости и прослеживаемости, представленным в 5.3.2.7—5.3.2.10 и в 6.5.4.14—6.5.4.17, а также требованиям, определенным в 7.4.4.7—7.4.4.10;

e) декомпозиция спецификации проекта программного обеспечения на компоненты программного обеспечения и спецификация проекта компонента программного обеспечения приводят к:

1) достижению требуемой производительности,

2) тестируемости для дальнейшей проверки и

3) пригодности для сопровождения, обеспечивающего дальнейшее развитие.

Данные результаты должны быть включены в отчет о проверке проекта компонента программного обеспечения.

7.5 Реализация и тестирование компонент

7.5.1 Цели

Для программного обеспечения обеспечить анализируемость, тестируемость, проверяемость и сопровождаемость. Тестирование компонент также включено в эту стадию.

7.5.2 Входные документы

a) Спецификация проекта компонента программного обеспечения.

b) Спецификация тестирования компонента программного обеспечения.

7.5.3 Выходные документы

a) Исходный код программного обеспечения и сопроводительная документация.

b) Отчет об испытаниях компонента программного обеспечения.

c) Отчет о проверке исходного кода программного обеспечения.

7.5.4 Требования

7.5.4.1 Исходный код программного обеспечения должен быть подготовлен в письменном виде под руководством разработчика на основе спецификации проекта компонента программного обеспечения. Требования 7.5.4.2—7.5.4.4 относятся к исходному коду программного обеспечения.

7.5.4.2 Размер и сложность разработанного исходного кода должны быть сбалансированы.

7.5.4.3 Исходный код программного обеспечения должен быть читаемым, понятным и тестируемым.

7.5.4.4 Исходный код программного обеспечения должен быть охвачен процедурой управления конфигурацией перед началом документально оформленного тестирования.

7.5.4.5 Отчет об испытаниях компонента программного обеспечения должен быть подготовлен в письменном виде под руководством тестировщика на основе спецификации тестирования к компоненту программного обеспечения и исходного кода программного обеспечения.

Требования 7.5.4.6, 7.5.4.7 обращаются к отчету об испытаниях компонента программного обеспечения.

7.5.4.6 Отчет об испытаниях компонента программного обеспечения должен быть подготовлен в соответствии с общими требованиями, установленными для отчета об испытаниях (см. 6.1.4.5).

7.5.4.7 Отчет об испытаниях компонента программного обеспечения должен включать следующую информацию:

a) описание результатов испытаний с указанием для каждого компонента о том, удовлетворяет ли он требованиям своей спецификации проекта компонента программного обеспечения;

b) описание результатов тестового охвата должно быть представлено для каждого компонента с демонстрацией того, что требуемая степень тестового охвата была достигнута для всех требуемых кри-

териев. Для любого исполняемого кода, требуемая степень тестового охвата которого не достигается, должно быть выполнено обоснование в соответствии с выделенным значением УПБ, см. таблицу А.21.

7.5.4.8 Отчет о проверке исходного кода программного обеспечения должен быть подготовлен в письменном виде под руководством менеджера по проверке на основе спецификации проекта компонента программного обеспечения, спецификации тестирования компонента программного обеспечения и исходного кода программного обеспечения.

Требования 7.5.4.9, 7.5.4.10 относятся к отчету о проверке исходного кода программного обеспечения.

7.5.4.9 Отчет о проверке исходного кода программного обеспечения должен быть подготовлен в соответствии с общими требованиями, установленными для отчета о проверке (см. 6.2.4.14).

7.5.4.10 После реализации исходного кода программного обеспечения и подготовке отчета об испытаниях компонента программного обеспечения должна быть выполнена проверка, устанавливающая, что:

a) реализация исходного кода программного обеспечения соответствует спецификации проекта компонента программного обеспечения;

b) корректно использованы выбранные методы и меры из таблицы А.4 как набор, удовлетворяющий требованиям 4.8 и 4.9;

c) корректно определено применение стандартов кодирования;

d) исходный код программного обеспечения удовлетворяет общим требованиям для удобочитаемости и прослеживаемости, представленным в 5.3.2.7—5.3.2.10 и в 6.5.4.14—6.5.4.17, а также требованиям, определенным в 7.5.4.1—7.5.4.4;

e) отчет об испытаниях компонента программного обеспечения с описанием выполненных тестов соответствует спецификации тестирования компонента программного обеспечения.

Результаты должны быть включены в отчет о проверке исходного кода программного обеспечения.

7.6 Интеграция

7.6.1 Цели

7.6.1.1 Выполнить интеграцию программного обеспечения и программного обеспечения/аппаратных средств.

7.6.1.2 Продемонстрировать, что программное обеспечение и аппаратные средства взаимодействуют правильно, чтобы выполнить предназначенные для них функции.

7.6.2 Входные документы

a) Спецификация тестирования интеграции программного обеспечения/аппаратных средств.

b) Спецификация тестирования интеграции программного обеспечения.

7.6.3 Выходные документы

a) Отчет об испытаниях интеграции программного обеспечения.

b) Отчет об испытаниях интеграции программного обеспечения/аппаратных средств.

c) Отчет о проверке интеграции программного обеспечения.

7.6.4 Требования

7.6.4.1 Интеграция компонентов программного обеспечения должна быть процессом постепенного объединения отдельных и ранее протестированных компонентов в составное целое, чтобы интерфейсы компонентов и собранное программное обеспечение могли быть соответственно доказаны до интеграции системы и тестирования системы.

7.6.4.2 Во время интеграции программного обеспечения/аппаратных средств при любой модификации или изменении в интегрированной системе необходимо выполнить анализ влияния, который должен определить все компоненты, на которые было оказано влияние, а также необходимые действия для повторной проверки.

7.6.4.3 Отчет об испытаниях интеграции программного обеспечения должен быть подготовлен в письменном виде под руководством интегратора на основе спецификации тестирования интеграции программного обеспечения.

Требования 7.6.4.4—7.6.4.6 относятся к отчету об испытаниях интеграции программного обеспечения.

7.6.4.4 Отчет об испытаниях интеграции программного обеспечения должен быть подготовлен в соответствии с общими требованиями, установленными для отчета об испытаниях (см. 6.1.4.5).

7.6.4.5 Отчет об испытаниях интеграции программного обеспечения должен быть сформирован следующим образом:

a) отчет об испытаниях интеграции программного обеспечения должен содержать результаты испытаний, а также информацию о том, были ли реализованы цели и выполнены критерии спецификации тестирования интеграции программного обеспечения. Если имеется отказ, то все обстоятельства и причины отказа должны быть зарегистрированы;

b) должны быть зарегистрированы тестовые сценарии и их результаты предпочтительно в машиночитаемой форме для последующего анализа;

c) тесты должны быть повторяемыми и, если реально, то должны выполняться автоматическими средствами;

d) отчет об испытаниях интеграции программного обеспечения должен содержать документально оформленные идентификационные данные и конфигурацию всех включенных элементов.

7.6.4.6 Отчет об испытаниях интеграции программного обеспечения должен демонстрировать корректное использование выбранных методов и мер из таблицы А.6 как набор, удовлетворяющий 4.8 и 4.9.

7.6.4.7 Отчет об испытаниях интеграции программного обеспечения/аппаратных средств должен быть подготовлен в письменном виде под руководством интегратора на основе спецификации теста интеграции программного обеспечения/аппаратных средств.

Требования 7.6.4.8—7.6.4.10 относятся к отчету об испытаниях интеграции программного обеспечения/аппаратных средств.

7.6.4.8 Отчет об испытаниях интеграции программного обеспечения/аппаратных средств должен быть подготовлен в соответствии с общими требованиями, установленными для отчета об испытаниях (см. 6.1.4.5).

7.6.4.9 Отчет об испытаниях интеграции программного обеспечения/аппаратных средств должен быть сформирован следующим образом:

a) отчет об испытаниях интеграции программного обеспечения/аппаратных средств должен содержать результаты испытаний, а также информацию о том, были ли реализованы цели и выполнены ли критерии спецификации тестирования интеграции программного обеспечения/аппаратных средств. Если имеется отказ, то все обстоятельства и причины отказа должны быть зарегистрированы;

b) тестовые сценарии и их результаты должны быть зарегистрированы, предпочтительно в машиночитаемой форме для последующего анализа;

c) отчет об испытаниях интеграции программного обеспечения/аппаратных средств должен содержать документально оформленные идентификационные данные и конфигурацию всех включенных элементов.

7.6.4.10 Отчет об испытаниях интеграции программного обеспечения/аппаратных средств должен демонстрировать корректное использование выбранных методов и мер из таблицы А.6 как набор, удовлетворяющий 4.8 и 4.9.

7.6.4.11 Отчет о проверке интеграции программного обеспечения должен быть подготовлен в письменном виде под руководством менеджера по проверке, на основе спецификаций тестов интеграции программного обеспечения и программного обеспечения/аппаратных средств и соответствующих отчетов об испытаниях.

Требования 7.6.4.12, 7.6.4.13 относятся к отчету о проверке интеграции программного обеспечения.

7.6.4.12 Отчет о проверке интеграции программного обеспечения должен быть подготовлен в соответствии с общими требованиями, установленными для отчета о проверке (см. 6.2.4.14).

7.6.4.13 После подготовки отчета об испытаниях интеграции программного обеспечения и отчета об испытаниях интеграции программного обеспечения/аппаратных средств должна быть выполнена проверка, устанавливающая, что:

a) отчет об испытаниях интеграции программного обеспечения с описанием выполненных тестов соответствует спецификации тестирования интеграции программного обеспечения;

b) отчет об испытаниях интеграции программного обеспечения удовлетворяет общим требованиям для удобочитаемости и прослеживаемости, представленным в 5.3.2.7—5.3.2.10 и в 6.5.4.14—6.5.4.17, а также требованиям, определенным в 7.6.4.3—7.6.4.6;

c) отчет об испытаниях интеграции программного обеспечения/аппаратных средств с описанием выполненных тестов соответствует спецификации тестирования интеграции программного обеспечения/аппаратных средств;

d) отчет об испытаниях интеграции программного обеспечения/аппаратных средств удовлетворяет общим требованиям для удобочитаемости и прослеживаемости, представленным в 5.3.2.7—5.3.2.10 и в 6.5.4.14—6.5.4.17, а также требованиям, определенным в 7.6.4.7—7.6.4.10.

7.7 Тестирование всего программного обеспечения. Заключительное подтверждение соответствия

7.7.1 Цели

Проанализировать и протестировать интегрированные программное обеспечение и аппаратные средства, чтобы гарантировать соответствие со спецификацией требований к программному обеспечению, обращая особое внимание на функциональные аспекты и аспекты безопасности, согласно уровню полноты безопасности программного обеспечения, а также проверить, пригодны ли они для применения по назначению.

7.7.2 Входные документы

a) Спецификация требований к программному обеспечению.

b) Спецификация тестирования всего программного обеспечения.

c) План проверки программного обеспечения.

d) План подтверждения соответствия программного обеспечения.

e) Вся документация на аппаратные средства и программное обеспечение, включая промежуточные результаты проверки.

f) Спецификация требований к безопасности системы.

7.7.3 Выходные документы

a) Отчет об испытаниях всего программного обеспечения.

b) Отчет о проверке испытаний всего программного обеспечения.

c) Отчет о подтверждении соответствия программного обеспечения.

d) Информация о версии.

7.7.4 Требования

7.7.4.1 Отчет об испытаниях всего программного обеспечения должен быть подготовлен в письменном виде под руководством тестировщика, на основе спецификации тестирования всего программного обеспечения.

Требования 7.7.4.2—7.7.4.4 относятся к отчету о комплексных испытаниях программного обеспечения.

7.7.4.2 Отчет об испытаниях всего программного обеспечения должен быть подготовлен в соответствии с общими требованиями, установленными для отчета об испытаниях (см. 6.1.4.5).

7.7.4.3 Менеджер по подтверждению соответствия должен определить и выполнить дополнительные испытания по его усмотрению или предложить их выполнение тестировщику. В то время как испытания всего программного обеспечения, главным образом, основаны на структуре спецификации требований к программному обеспечению, менеджер по подтверждению соответствия должен внести приносящий практическую пользу вклад, добавляя испытания, которые нагружают систему сложными сценариями, отражающими фактические потребности пользователя.

7.7.4.4 Результаты всех испытаний и исследований должны быть включены в отчет об испытаниях всего программного обеспечения.

7.7.4.5 Отчет о проверке испытаний всего программного обеспечения должен быть подготовлен в письменном виде под руководством менеджера по проверке.

7.7.4.6 Программное обеспечение должно быть реализовано или в виде загружаемого файла в реальные элементы аппаратных средств или в фактические системы, с которыми оно взаимодействовало бы в процессе работы, или в виде программы моделирования, которая по входным сигналам управляет нагрузками с помощью своих выходных сигналов. Программное обеспечение должно быть реализовано при существующих условиях во время нормального функционирования, для ожидаемых событий и нежелательных условий, требующих реакции системы. Если используются моделируемые входы или нагрузки, то необходимо показать, что они значительно не отличаются от входов и нагрузок, которые встречаются в процессе реальной работы.

Примечание — Моделируемые входы или нагрузки могут заменить входы или нагрузки, которые присутствуют только на уровне системы или в режимах сбоев.

7.7.4.7 Отчет о подтверждении соответствия программного обеспечения должен быть подготовлен в письменном виде под руководством менеджера по подтверждению соответствия на основе плана подтверждении соответствия программного обеспечения.

Требования 7.7.4.8—7.7.4.12 относятся к отчету о подтверждении соответствия программного обеспечения.

7.7.4.8 Отчет о подтверждении соответствия программного обеспечения должен быть подготовлен в соответствии с общими требованиями, установленными для отчета о подтверждении соответствия (см. 6.3.4.7—6.3.4.11).

7.7.4.9 Если окончена интеграция и завершены испытания и анализ всего программного обеспечения, то в отчете о подтверждении соответствия программного обеспечения должно быть представлено следующее:

a) заключение о том, были ли выполнены цели и критерии плана подтверждения соответствия программного обеспечения. Отклонения от плана должны быть зарегистрированы и обоснованы;

b) итоговое заключение о результатах испытаний и о том, выполняются ли полным программным обеспечением на его целевой машине требования, изложенные в спецификации требований к программному обеспечению;

c) оценка тестового охвата в соответствии с требованиями спецификации требований к программному обеспечению;

d) оценка других действий проверки в соответствии с планом и отчетом о проверке программного обеспечения вместе с проверкой, показывающей, что прослеживание требований полностью охвачено и выполнено;

e) если менеджером по подтверждению соответствия выполнены собственные тестовые сценарии, не переданные тестировщику, то они должны быть документально оформлены в отчете о подтверждении соответствия программного обеспечения в соответствии с требованиями 6.3.4.7—6.3.4.11.

7.7.4.10 Отчет о подтверждении соответствия программного обеспечения должен содержать утверждение о том, что каждая комбинация методов или мер, выбранных согласно приложению А, соответствует определенному уровню полноты безопасности программного обеспечения. Отчет должно содержать оценку общей эффективности комбинации выбранных методов и мер с учетом размера и сложности выполненного программного обеспечения, а также принимая во внимание фактические результаты тестирования, проверки и подтверждения соответствия.

7.7.4.11 Отчет о подтверждении соответствия программного обеспечения должен содержать следующее:

a) документацию по идентификации и конфигурации программного обеспечения;

b) заключение о надлежащей идентификации программного обеспечения и оборудования технической поддержки;

c) заключение о надлежащей идентификации используемых моделей;

d) заключение о соответствии спецификации тестирования всего программного обеспечения;

e) информацию о наборе и прослеживании любых найденных отклонений;

f) результаты анализа и оценки всех отклонений с оценкой рисков (влияние);

g) заключение о том, что в процессе проектирования выполнена надлежащая обработка корректирующих действий, которая соответствует процессу и процедурам управления изменениями и выполняется с четкой идентификацией любых найденных несоответствий;

h) информацию об отклонениях каждого ограничения в процессе прослеживания;

i) заключение о том, пригодно ли программное обеспечение для применения по назначению, учитывая условия применения и ограничения.

7.7.4.12 Любые найденные несоответствия, включая обнаруженные ошибки и несоответствия с настоящим стандартом или с любым из требований к программному обеспечению или планом, а также ограничения и предельные значения должны быть четко идентифицированы в отдельном подразделе отчета о подтверждении соответствия программного обеспечения, оценены относительно уровня полноты безопасности и включены в какую-либо информацию о версии, которая сопровождает поставляемое программное обеспечение.

7.7.4.13 Информация о версии, которая сопровождает поставляемое программное обеспечение, должна включать все ограничения в использовании программного обеспечения. Эти ограничения получены в результате:

a) обнаруженных ошибок,

b) несоответствия с настоящим стандартом,

c) степени выполнения требований,

d) степени выполнения любого плана.

8 Разработка прикладных данных или алгоритмов. Системы, сконфигурированные прикладными данными или алгоритмами

8.1 Цели

8.1.1 Характерной особенностью во многих железнодорожных системах является необходимость проектировать каждую инсталляцию, удовлетворяющую индивидуальным требованиям для конкретного применения. Система, конфигурируемая прикладными данными и/или прикладными алгоритмами, позволяет принятое универсальное программное обеспечение настроить в соответствии с индивидуальными требованиями для каждого конкретного применения.

Целью разработки прикладных данных является корректное получение данных для данной инсталляции и проверка предназначенного поведения, сопровождаемая оценкой используемого процесса разработки этих прикладных данных.

Требования для разработки прикладных алгоритмов совпадают с требованиями для разработки универсального программного обеспечения, как описано в разделах 1—7 и 9.

Типичным примером является система, универсальное программное обеспечение которой предварительно сконфигурировано для универсального железнодорожного применения как набор прикладных алгоритмов, и которая далее конфигурируется для каждой конкретной инсталляции с помощью конкретизации и взаимосвязей прикладных алгоритмов, а также с помощью конфигурационных данных. Например, принципы сигнализации системы централизации (например, управление сигнализацией, управление пунктами сигнализации) могут быть реализованы набором прикладных алгоритмов.

Прикладные данные обычно принимают форму значений или описаний параметров (идентификационные данные, тип, расположение, и т. д.) внешних объектов. Прикладные алгоритмы могут принять форму, например, диаграммы функциональных блоков, диаграммы состояний и релейной логической схемы, которые определяют желаемый ответ системы согласно ее входам, ее текущего состояния и конкретных значений параметров. Прикладные алгоритмы включают логические соединения и выполняемые операции.

Прикладные данные/алгоритмы обычно создаются с помощью специальных инструментальных средств. Они могут быть выражены в форматах таблиц или диаграмм, которые могут быть интерпретированы или могут компилироваться в исполняемые коды часто после преобразования в исходные коды специализированных языков (обладающих синтаксисом и семантикой).

Используя возможности конфигурации, настройка систем дает проектировщику различные уровни управления при реализации детальной функциональности программного обеспечения.

8.1.2 Процедуры и инструментальные средства, используемые для их разработки, должны соответствовать надлежащему уровню полноты безопасности системы, как определено функцией, для которой они разработаны.

8.1.3 В 8.4 описаны требования для начальной разработки конфигурируемой системы и для последующей разработки каждого набора специализированных прикладных данных/алгоритмов.

8.2 Входные документы

a) Спецификация требований к универсальному программному обеспечению.

b) Спецификация архитектуры универсального программного обеспечения.

c) Условия применения инструментальных средств для универсального программного обеспечения и приложения.

d) Руководства пользователя инструментальных средств для универсального программного обеспечения и приложения.

8.3 Выходные документы

a) План подготовки приложения.

b) Спецификация требований к приложению.

c) Архитектура и проект приложения.

d) Спецификация тестирования приложения.

e) Отчет об испытаниях приложения.

f) Отчет о проверке подготовки приложения.

g) Исходный код данных/алгоритмов приложения.

h) Отчет о проверке данных/алгоритмов приложения.

8.4 Требования

8.4.1 Процесс разработки приложения

8.4.1.1 План подготовки приложения должен быть подготовлен в письменном виде под руководством менеджера требований или проектировщиком на основе входных документов, перечисленных в 8.2.

Требования 8.4.1.2—8.4.1.11 относятся к плану подготовки приложения.

8.4.1.2 Необходимо создать план подготовки приложения, в котором определить и подробно описать процесс разработки приложения, включая все действия, конечный результат и роли, отвечающие за него. Он может быть создан или для каждого конкретного приложения или для класса конкретных применений, т. е. для универсального приложения.

8.4.1.3 План подготовки приложения должен определить структуру документации для процесса подготовки приложения.

8.4.1.4 План подготовки приложения должен выбрать методы и меры из таблицы А.11. Выбранная комбинация должна быть обоснована как набор, удовлетворяющий требованиям 4.8 и 4.9.

8.4.1.5 План подготовки приложения должен определить процедуры и инструментальные средства приложения (с их классификацией на основе требований 6.7), чтобы использовать их в процессе разработки приложения.

8.4.1.6 План подготовки приложения должен включать действия по проверке и подтверждению соответствия, чтобы гарантировать, что прикладные данные/алгоритмы получены, корректны и совместимы друг с другом и с универсальным приложением, а также представить свидетельства о том, что прикладные условия универсального приложения соблюдаются. Эти действия по проверке и подтверждению соответствия и доказательства могут быть заменены проверкой и подтверждением соответствия, выполненных с помощью инструментальных средств, которые создают прикладные данные/алгоритмы. Результаты должны быть собраны в отчете о проверке подготовки приложения и отчете об испытаниях приложения.

8.4.1.7 План подготовки приложения должен включать действия по проверке и подтверждению соответствия, чтобы гарантировать, что инструментальные средства приложения и универсальное программное обеспечение совместимы друг с другом и с конкретным приложением, а также должен представить свидетельства, что условия их применения соблюдаются.

8.4.1.8 Должен быть выполнен анализ рисков для всего процесса разработки приложения, включая инструментальные средства приложения и процедуры, чтобы выполнить подтверждение соответствия плана подготовки приложения и обеспечить выполнение требуемого уровня полноты безопасности программного обеспечения. План подготовки приложения должен включать анализ рисков.

8.4.1.9 План подготовки приложения должен определить требования к независимости между сотрудниками, выполняющими проверку, подтверждение соответствия и задачи подготовки в соответствии с требованиями 5.1.

Примечание — Действия по подготовке данных выполняются проектировщиками приложения.

8.4.1.10 План подготовки приложения должен определять класс инструментального средства для любых инструментальных средств, как для аппаратных средств, так и для программного обеспечения, используемых в жизненном цикле подготовки приложения.

8.4.1.11 Где это возможно, план подготовки приложения должен использовать нотации для определения требований и проекта, которые знакомы инженерам приложений. Если введена новая нотация, то это должно быть отражено в пользовательской документации, а также выполнено обучение, где это необходимо.

8.4.1.12 Отчет о проверке данных/алгоритмов приложения должен быть подготовлен в письменном виде под руководством менеджера по проверке, на основе входных документов, перечисленных в 8.2.

Требование 8.4.1.13 относится к отчету о проверке данных/алгоритмов приложения.

8.4.1.13 После создания плана подготовки приложения должна быть выполнена проверка, устанавливающая:

a) что план подготовки приложения удовлетворяет общим требованиям удобочитаемости и прослеживаемости, представленным в 5.3.2.7—5.3.2.10 и в 6.5.4.14—6.5.4.17, а также требованиям, определенным в 8.4.1.2—8.4.1.11;

b) внутреннюю непротиворечивость плана подготовки приложения.

Результаты должны быть включены в отчет о проверке данных/алгоритмов приложения.

8.4.1.14 Для каждого конкретного приложения реализация плана подготовки приложения должна быть проверена и для нее должно быть выполнено подтверждение соответствия.

8.4.2 Спецификация требований к приложению

8.4.2.1 Спецификация требований к приложению должна быть подготовлена в письменном виде под руководством менеджера требований, на основе входных документов, перечисленных в 8.2.

Требования 8.4.2.2, 8.4.2.3 относятся к спецификации требований к приложению.

8.4.2.2 Требования для конкретного приложения должны включать требования, которые являются конкретными для рассматриваемой инсталяции (например, развязка путей, сигнальные установки, ограничения скорости для системы сигнализации), краткое перечисление или ссылку на условия применения универсального программного обеспечения и инструментальных средств приложения, а также стандарты, которым приложение должно соответствовать (например, принципам сигнализации для системы сигнализации).

8.4.2.3 На данном этапе должны быть определены требования, связанные с данными и алгоритмами приложения, обрабатываемые универсальным программным обеспечением системы.

8.4.2.4 Отчет о проверке данных/алгоритмов приложения должен быть подготовлен в письменном виде под руководством менеджера по проверке на основе входных документов, перечисленных в 8.2.

Требование 8.4.2.5 относится к отчету о проверке данных/алгоритмов приложения.

8.4.2.5 После подготовки спецификации требований к приложению должна быть выполнена проверка, устанавливающая:

a) что спецификации требований к приложению удовлетворяют общим требованиям для удобочитаемости и прослеживаемости, представленным в 5.3.2.7—5.3.2.10 и в 6.5.4.14—6.5.4.17, а также требованиям, определенным в 8.4.2.2, 8.4.2.3;

b) внутреннюю непротиворечивость спецификации требований к приложению.

Результаты должны быть включены в отчет о проверке данных/алгоритмов приложения.

8.4.3 Архитектура и проект

Должны быть определены количество и тип компонентов аппаратных средств и универсального программного обеспечения, которые будут использоваться в конкретном приложении. Должно быть определено размещение компонентов, данных и алгоритмов приложения в конкретной архитектуре приложения. На данном этапе должны быть разработаны данные и алгоритмы приложения, обрабатываемые универсальным программным обеспечением.

8.4.4 Разработка данных/алгоритмов приложения

8.4.4.1 Процесс разработки приложений должен включать создание и компиляцию исходного кода универсальных и конкретных данных/алгоритмов, а также действия по проверке и тестированию, связанные с этим созданием. Для создания исходного кода алгоритмов приложения рекомендуется использовать графические языки. См. таблицу А.16.

8.4.4.2 Отчет об испытаниях приложения должен быть подготовлен в письменном виде под руководством тестировщика на основе входных документов, перечисленных в 8.2.

Требование 8.4.4.3 относятся к отчету об испытаниях приложения.

8.4.4.3 Отчет об испытаниях приложения должен документально оформить корректное и полное выполнение тестов, определенных в спецификации тестирования приложения.

8.4.4.4 В отчете о проверке подготовки приложения должно быть:

a) документально оформлено каждое действие, выполненное, чтобы гарантировать правильность и полноту данных/алгоритма и их согласованность с принципами приложения и конкретной архитектурой приложения;

b) представлена оценка совместимости данных/алгоритмов с универсальным приложением.

8.4.4.5 Спецификация тестирования приложения должна быть подготовлена в письменном виде под руководством тестировщика, на основе входных документов, перечисленных в 8.2.

Требование 8.4.4.6 относится к спецификации тестирования приложения.

8.4.4.6 Спецификация тестирования приложения должна определить тесты, которые будут выполнены на промежуточной или заключительной стадии подготовки данных/алгоритмов, чтобы гарантировать:

a) согласованность и полноту данных/алгоритмов относительно принципов приложения;

b) согласованность и полноту данных/алгоритмов относительно конкретной архитектуры приложения.

8.4.4.7 Отчет о проверке данных/алгоритмов приложения должен быть подготовлен в письменном виде под руководством менеджера по проверке на основе входных документов, перечисленных в 8.2.

Требование 8.4.4.8 относится к отчету о проверке данных/алгоритмов приложения.

8.4.4.8 После подготовки спецификации тестирования приложения должна быть выполнена проверка, устанавливающая:

a) что спецификации тестирования приложения удовлетворяют общим требованиям для удобочитаемости и прослеживаемости, представленным в 5.3.2.7—5.3.2.10 и в 6.5.4.14—6.5.4.17, а также требованиям, определенным в 8.4.4.6,

b) внутреннюю непротиворечивость спецификации тестирования приложения.

Результаты должны быть включены в отчет о проверке данных/алгоритмов приложения.

8.4.5 Приемка интеграции и тестирования приложения

8.4.5.1 Для некоторых систем прикладные данные/алгоритмы могут быть интегрированы с аппаратными средствами и универсальным программным обеспечением для выполнения производственных испытаний перед установкой в целевую систему. Это можно не выполнять, если достаточный уровень уверенности может быть получен другими средствами. В этом случае приложение должно быть установлено в целевой системе и тесты интеграции должны быть выполнены для всей инсталляции. Наконец целевая система должна быть введена в эксплуатацию, как полностью работоспособная система, и должен быть выполнен процесс окончательной приемки целевой системы для полной инсталляции. В отчете об испытаниях приложения должны быть документально оформлено корректное и полное выполнение тестов, определенных в спецификации тестирования приложения. Отчет о проверке подготовки приложения должен содержать информацию о проверке полноты и правильности тестов, выполненных на полной инсталляции.

8.4.5.2 Спецификация тестирования приложения должна быть подготовлена в письменном виде под руководством тестировщика на основе входных документов, перечисленных в 8.2.

Требование 8.4.5.3 относится к спецификации тестирования приложения.

8.4.5.3 Спецификация тестирования приложения должна определить тесты, которые будут выполнены, чтобы гарантировать:

a) корректную интеграцию данных/алгоритмов на аппаратных средствах и универсальном программном обеспечении, в случае необходимости;

b) корректную интеграцию данных/алгоритмов в полной инсталляции.

8.4.5.4 Отчет о проверке данных/алгоритмов приложения должен быть подготовлен в письменном виде под руководством менеджера по проверке на основе входных документов, перечисленных в 8.2.

Требование 8.4.5.5 относится к отчету о проверке данных/алгоритмов приложения.

8.4.5.5 После подготовки спецификации тестирования приложения должна быть выполнена проверка, устанавливающая, что спецификации тестирования приложения удовлетворяют конкретным требованиям в 8.4.5.3.

8.4.6 Подтверждение соответствия и оценка

Действия по подтверждению соответствия и оценке должны проводить проверку выполнения каждой стадии жизненного цикла.

8.4.7 Процедуры и инструментальные средства подготовки приложения

8.4.7.1 Для каждого нового типа системы, сконфигурированной данными/алгоритмами приложения, должны быть разработаны конкретные процедуры и инструментальные средства, чтобы обеспечить применение процесса разработки приложений, определенного в 8.4.1, к инсталляциям новой системы. Разработка таких инструментальных средств должна быть выполнена в соответствии с настоящим стандартом вместе с универсальным программным обеспечением и аппаратными средствами для системы. Действия по проверке, подтверждению соответствия и оценке должны гарантировать, что инструментальные средства подготовки данных и универсальное программное обеспечение совместимы.

8.4.7.2 Для любого процесса компиляции должно быть выполнено подтверждение соответствия и оценка. Нужно отметить, что для преобразования алгоритма и данных обычно необходимы специализированные компиляторы.

8.4.7.3 Все прикладные данные/алгоритмы и соответствующая документация для каждого конкретного приложения должны подчиняться требованиям развертывания программного обеспечения, как определено в 9.1.

8.4.7.4 Все прикладные данные/алгоритмы и соответствующая документация должны подчиняться требованиям сопровождения программного обеспечения, определенным в 9.2.

8.4.7.5 Все прикладные данные/алгоритмы и соответствующую документацию должна охватывать процедура управления конфигурацией согласно требованиям, определенным в 6.5 и 6.7. Управление конфигурацией данных/алгоритмов приложения может быть отделено от управления конфигурацией части универсального программного обеспечения.

8.4.7.6 Отчет о проверке приложения демонстрирует охват и осуществление условий применения универсального программного обеспечения и инструментальных средств приложения.

8.4.8 Разработка универсального программного обеспечения

8.4.8.1 Разработка универсального программного обеспечения, которое поддерживает выполнение данных/алгоритмов приложения, должна соответствовать требованиям 7.1—7.7. Также должны выполняться следующие дополнительные требования.

8.4.8.2 Типы или классы функции, которые могут быть сконфигурированы данными/алгоритмами приложения в каждой системе и подсистеме, должны быть идентифицированы в документах спецификации требований к программному обеспечению для универсального программного обеспечения. Уровень полноты безопасности, распределенный функциям, определит стандарты, которые будут применены для последующей разработке данных/алгоритмов приложения для всех инсталляций системы.

8.4.8.3 Во время проектирования универсального программного обеспечения должны быть определены подробные интерфейсы между универсальным программным обеспечением и данными/алгоритмами приложения, если это не было уже определено на более ранней стадии жизненного цикла, например в результате требования об использовании существующего специального для приложения языка.

8.4.8.4 Должно быть осуществлено строгое разделение между универсальным программным обеспечением и данными/алгоритмами приложения, т. е. должна существовать возможность отдельно перекомпилировать и обновить или универсальное программное обеспечение, или прикладные данные/алгоритмы, если не было изменения в определенном интерфейсе между универсальным программным обеспечением и данными/алгоритмами приложения. Аналогично должны быть разделены конкретные данные/алгоритмы для приложения и данные/алгоритмы для универсального приложения.

8.4.8.5 Процедуры управления изменениями должны гарантировать, что любое изменение универсального программного обеспечения может быть реализовано только после того, как было установлено, что либо исправленное программное обеспечение совместимо с исходными прикладными данными/алгоритмами, либо прикладные данные/алгоритмы должны быть изменены.

8.4.8.6 Необходимо соблюдать осторожность на стадии тестирования процесса проверки и подтверждения соответствия универсального программного обеспечения, чтобы гарантировать, что рассматриваются все соответствующие комбинации данных и алгоритмов.

Если все соответствующие комбинации данных и алгоритмов не были рассмотрены в процессе проверки, тестирования и подтверждения соответствия универсального программного обеспечения, то должен быть строго определен предел использования универсального программного обеспечения. Если некоторые данные или алгоритмы будут определены вне этого предела, то должны быть выполнены дополнительные процессы проверки, тестирования и подтверждения соответствия универсального программного обеспечения.

8.4.8.7 Универсальное программное обеспечение должно быть разработано так, чтобы оно обнаруживало поврежденные прикладные данные/алгоритмы, где это возможно.

8.4.8.8 Проектировщики должны публиковать информацию о новой версии инструментальных средств универсального программного обеспечения и приложения на стадии тестирования всего программного обеспечения/заключительного подтверждения соответствия инструментальных средств универсального программного обеспечения и приложения. Содержание этих документов должно быть проверено, а также для него должно быть выполнено подтверждение соответствия.

Следующая информация должна быть представлена в документе «Условия применения инструментальных средств универсального программного обеспечения и приложения»:

a) ссылки на руководства пользователя инструментальных средств универсального программного обеспечения и приложения;

b) любые ограничения на прикладные данные/алгоритмы, например, заданная архитектура или правила кодирования, чтобы удовлетворить уровням полноты безопасности.

9 Развертывание и сопровождение программного обеспечения

9.1 Развертывание программного обеспечения

9.1.1 Цель

Гарантировать, что программное обеспечение выполняется в соответствии с требованиями, сохраняя требуемый уровень полноты безопасности программного обеспечения и надежность, если оно развернуто в окончательном окружении применения.

9.1.2 Входные документы

Все документы по проектированию, разработке и анализу, относящиеся к развертыванию.

9.1.3 Выходные документы

a) План выпуска и развертывания программного обеспечения.

b) Руководство по развертыванию программного обеспечения.

c) Информация о версии.

d) Журнал развертывании.

e) Отчет о проверке развертывания.

9.1.4 Требования

9.1.4.1 Развертывание должно быть выполнено под руководством руководителя проекта.

9.1.4.2 Перед поставкой готового программного обеспечения должно быть зарегистрировано универсальное программное обеспечение и обеспечено управление прослеживаемостью под управлением конфигурацией. Должны также быть включены существующее ранее программное обеспечение и программное обеспечение, разработанное в соответствии с предыдущей версией настоящего стандарта.

9.1.4.3 Готовое программное обеспечение в течение жизненного цикла его базовой конфигурации должно быть восстанавливаемым.

9.1.4.4 Информация о версии должна быть подготовлена в письменном виде под руководством проектировщика на основе входных документов, перечисленных в 9.1.2.

Требование 9.1.4.5 относится к информации о версии.

9.1.4.5 Информация о версии должна включать:

a) условия применения, которым необходимо придерживаться;

b) информацию о совместимости между компонентами программного обеспечения и между программным обеспечением и аппаратными средствами;

c) все ограничения при использовании программного обеспечения (см. 7.7.4.13).

9.1.4.6 Руководство по развертыванию программного обеспечения должно быть подготовлено в письменном виде на основе входных документов, перечисленных в 9.1.2.

Требование 9.1.4.7 относится к руководству по развертыванию программного обеспечения.

9.1.4.7 Руководство по развертыванию программного обеспечения должно определить процедуры, чтобы правильно идентифицировать и установить готовое программное обеспечение.

9.1.4.8 В случае поэтапного развертывания (т. е. развертывание отдельных компонентов) настоятельно рекомендовано для УПБ 3 и УПБ 4 и рекомендуется для УПБ 1 и УПБ 2, чтобы разработанное программное обеспечение включало средства, которые гарантируют, что активация несовместимых версий компонентов программного обеспечения исключена.

9.1.4.9 Управление конфигурацией должно гарантировать, что никакого вреда не следует из совместного присутствия различных версий одних и тех же компонентов программного обеспечения, где этого нельзя избежать.

9.1.4.10 При установке новой версии программного обеспечения должна быть доступна процедура «отката» (т. е. возможность возвратиться к предыдущей версии).

9.1.4.11 Программное обеспечение должно иметь встроенные механизмы самоидентификации, позволяющие его идентификацию в процессе загрузки и после загрузки в целевую систему. Механизм самоидентификации должен указать информацию о версии программного обеспечения и любые данные о конфигурации, а также идентификационные данные продукта.

Примечание — Данные в коде, содержащие информацию о версии программного обеспечения, могут быть защищены посредством кодирования (см. таблицу А.3 «Коды с обнаружением ошибок»).

9.1.4.12 Журнал развертывания должен быть подготовлен в письменном виде на основе входных документов, перечисленных в 9.1.2.

Требование 9.1.4.13 относится к журналу развертывания программного обеспечения.

9.1.4.13 Журнал развертывания должен дать доказательство того, что программное обеспечение было загружено под контролем встроенных механизмов самоидентификации в (см. 9.1.4.11). Этот журнал должен быть сохранен в поставляемой документации, связанной с системой, как и другие документы о проверках, и является частью документации, необходимой для ввода в действие и принятия в эксплуатацию.

9.1.4.14 Развернутое программное обеспечение должно быть прослеживаемым в поставляемых установках.

Примечание — Это имеет особое значение, когда обнаружены критические отказы, и они должны быть исправлены более чем в одной установке.

9.1.4.15 Программное обеспечение должно предоставлять диагностическую информацию, являющуюся частью для контроля сбоев.

9.1.4.16 Отчет о проверке развертывания должен быть подготовлен в письменном виде под руководством менеджера по проверке на основе входных документов, перечисленных в 9.1.2.

Требования 9.1.4.17—9.1.4.19 относятся к отчету о проверке развертывания.

9.1.4.17 После подготовки руководства по развертыванию программного обеспечения должна быть выполнена проверка, устанавливающая:

a) что руководство по развертыванию программного обеспечения удовлетворяет общим требованиям для удобочитаемости и прослеживаемости, представленным в 5.3.2.7—5.3.2.10 и в 6.5.4.14—6.5.4.17, а также требованиям, определенным в 9.1.4.7,

b) внутреннюю непротиворечивость руководства по развертыванию программного обеспечения.

9.1.4.18 После подготовки журнала развертывания должна быть выполнена проверка, устанавливающая:

a) что журнал развертывания удовлетворяет общим требованиям для удобочитаемости и прослеживаемости, представленным в 5.3.2.7—5.3.2.10 и в 6.5.4.14—6.5.4.17, а также требованиям, определенным в 9.1.4.13,

b) внутреннюю непротиворечивость журнала развертывания.

9.1.4.19 После подготовки информации о версии должна быть выполнена проверка, устанавливающая:

a) что информация о версии удовлетворяет общим требованиям для удобочитаемости и прослеживаемости, представленным в 5.3.2.7—5.3.2.10 и в 6.5.4.14—6.5.4.17, а также требованиям, определенным в 9.1.4.5,

b) внутреннюю непротиворечивость информации о версии.

Результаты должны быть включены в отчет о проверке развертывания.

9.1.4.20 В пакет программного обеспечения должны быть включены меры предотвращающие или обнаруживающие ошибки, происходящие во время хранения, передачи или дублирования исполняемого кода или данных. Рекомендуется, чтобы исполнимый код был закодирован (см. таблицу А.3 «Коды с обнаружением ошибок») для проверки его целостности в процессе загрузки.

9.2 Сопровождение программного обеспечения

9.2.1 Цель

Гарантировать, что программное обеспечение выполняется в соответствии с требованиями, сохраняя требуемый уровень полноты безопасности программного обеспечения и надежность, при реализации исправлений, улучшений или адаптациях самого программного обеспечения. См. также 6.6 «Управление модификациями и изменениями» настоящего стандарта и стадию 13 «Модификация и модернизация» в МЭК 62278.

9.2.2 Входные документы

Все соответствующие документы по проектированию, разработке и анализу.

9.2.3 Выходные документы

a) План сопровождения программного обеспечения.

b) Журнал изменений программного обеспечения.

c) Журнал сопровождения программного обеспечения.

d) Отчет о проверке сопровождения программного обеспечения.

9.2.4 Требования

9.2.4.1 Несмотря на то, что настоящий стандарт не предназначен для ретроспективного использования и применяется, прежде всего, к новым разработкам и полностью применяется только к уже суще-

ствующему программному обеспечению, если оно подвергнуто серьезным модификациям, настоящий подраздел рассматривает сопровождение программного обеспечения и применяется ко всем изменениям, даже к незначительным. Поэтому настоятельно рекомендуется полностью применять настоящий стандарт в процессах обновлений и сопровождения существующего программного обеспечения.

9.2.4.2 Для любого уровня полноты безопасности программного обеспечения поставщик перед началом выполнения любого изменения должен решить, приведет ли оно к снижению или повышению действий по сопровождению системы или подойдут ли существующие методы сопровождения системы. Решение должно быть обосновано и зарегистрировано поставщиком и должно быть оценено оценщиком.

9.2.4.3 Сопровождение должно выполняться в соответствии с инструкциями, содержащимися в ИСО/МЭК 90003.

9.2.4.4 Должна быть разработана пригодность для сопровождения как аспект, присущий программному обеспечению, в частности, следуя требованиям 7.3, 7.4 и 7.5. Также должен использоваться комплекс ИСО/МЭК 25010, чтобы реализовать и проверить минимальный уровень пригодности для сопровождения.

9.2.4.5 План сопровождения программного обеспечения должен быть представлен в письменном виде на основе входных документов, перечисленных в 9.2.2.

Требование 9.2.4.6 относится к плану сопровождения программного обеспечения.

9.2.4.6 Процедуры сопровождения программного обеспечения должны быть установлены и включены в план сопровождения программного обеспечения. Эти процедуры также должны включать:

a) управление отчетами об ошибке, журналами регистрации ошибок, журналами сопровождения, авторизацией изменения и конфигурацией программного обеспечения/системы, а также методами и мерами из таблицы А.10;

b) проверку, подтверждение соответствия и оценку любой модификации;

c) определение полномочного органа, который принимает измененное программное обеспечение;

d) авторизацию для модификации.

9.2.4.7 Журнал сопровождения программного обеспечения должен быть подготовлен в письменном виде на основе входных документов, перечисленных в 9.2.2.

Требование 9.2.4.8 относится к журналу сопровождения программного обеспечения.

9.2.4.8 Журнал сопровождения программного обеспечения должен быть подготовлен для каждого элемента программного обеспечения перед его первым выпуском, и он должен сопровождаться. В дополнение к требованиям ИСО/МЭК 90003:2014 «Журналы и отчеты сопровождения» (см. ИСО/МЭК 90003:2014, «Сопровождение»), данный журнал должен также включать:

a) ссылки на все журналы изменений программного обеспечения для этого элемента программного обеспечения;

b) оценку влияния изменения;

c) тестовые сценарии для компонентов, включая данные повторного подтверждения соответствия и регрессного тестирования, и

d) историю конфигурации программного обеспечения.

9.2.4.9 Журнал изменений программного обеспечения должен быть подготовлен в письменном виде на основе входных документов, перечисленных 9.2.2.

Требование 9.2.4.10 относятся к журналу изменений программного обеспечения.

9.2.4.10 Журнал изменений программного обеспечения должен включать информацию по каждой операции по сопровождению. Этот отчет должен включать:

a) запрос на модификацию или изменение, версию, природу сбоя, требуемое изменение и источник изменения;

b) результаты анализа влияния на всю систему действий по сопровождению, включая аппаратные средства, программное обеспечение, взаимодействие с человеком и окружение, а также на возможные взаимодействия;

c) подробную спецификацию выполняемой модификации или изменения, и

d) результаты повторного подтверждения соответствия, регрессного тестирования, и повторной оценки модификации или изменения в объеме требований уровня полноты безопасности программного обеспечения. Ответственность за повторное подтверждение соответствия может меняться от проекта к проекту, согласно уровню полноты безопасности программного обеспечения. Кроме того, влияние модификации или изменения на процесс повторного подтверждения соответствия может быть ограничено на различных уровнях системы (только измененные компоненты, все компоненты, затронутые изме-

нением, вся система). Поэтому план подтверждения соответствия программного обеспечения должен решить обе проблемы, в соответствии с уровнем полноты безопасности программного обеспечения. Степень независимости выполнения повторного подтверждения соответствия должна быть такой же, как и для первоначального повторного подтверждения соответствия.

9.2.4.11 Отчет о проверке сопровождения программного обеспечения должен быть подготовлен в письменном виде под руководством менеджера по проверке на основе входных документов, перечисленных в 9.2.2.

Требования 9.2.4.12—9.2.4.14 относятся к отчету о проверке сопровождения программного обеспечения.

9.2.4.12 После подготовки плана сопровождения программного обеспечения должна быть выполнена проверка, устанавливающая:

a) что план сопровождения программного обеспечения удовлетворяет общим требованиям для удобочитаемости и прослеживаемости, представленным в 5.3.2.7—5.3.2.10 и в 6.5.4.14—6.5.4.17, а также требованиям, определенным в 9.2.4.6,

b) внутреннюю непротиворечивость плана сопровождения программного обеспечения.

9.2.4.13 После подготовки отчета о сопровождении программного обеспечения должна быть выполнена проверка, устанавливающая:

a) что отчет о сопровождении программного обеспечения удовлетворяет общим требованиям для удобочитаемости и прослеживаемости, представленным в 5.3.2.7—5.3.2.10 и в 6.5.4.14—6.5.4.17, а также требованиям, определенным в 9.2.4.8,

b) внутреннюю непротиворечивость отчета о сопровождении программного обеспечения.

9.2.4.14 После подготовки отчета об изменении программного обеспечения должна быть выполнена проверка, устанавливающая:

a) что отчет об изменении программного обеспечения удовлетворяет общим требованиям для удобочитаемости и прослеживаемости, представленным в 5.3.2.7—5.3.2.10 и в 6.5.4.14—6.5.4.17, а также требованиям, определенным в 9.2.4.10,

b) внутреннюю непротиворечивость отчета об изменении программного обеспечения.

9.2.4.15 Действия по сопровождению должны быть выполнены, следуя плану сопровождения программного обеспечения.

9.2.4.16 Методы и меры должны быть выбраны из таблицы А.10. Выбранная комбинация должна быть обоснована как набор, удовлетворяющий требованиям 4.8 и 4.9.

9.2.4.17 Сопровождение должно быть выполнено, по крайней мере, с тем же уровнем опыта, инструментальных средств, документации, планирования и управления, как и при первоначальной разработке программного обеспечения. Этот принцип должен применяться также и к управлению конфигурацией, управлению изменениями, управлению документацией и к независимости участвующих сторон.

9.2.4.18 Управлением внешнего поставщика, созданием отчетов задач и корректирующими действиями необходимо управлять по тем же критериям, определенным в соответствующих пунктах 6.5 «Обеспечение качества программного обеспечения», которые используются для новой разработки программного обеспечения.

9.2.4.19 Для каждой известной проблемы или ее развития должен быть выполнен анализ влияния на безопасность.

9.2.4.20 Для сопровождаемого программного обеспечения должны быть приняты меры смягчения, пропорциональные идентифицированному риску, чтобы гарантировать общую целостность системы, пока эти известные проблемы исследуются и корректируются.

Приложение А
(обязательное)

Критерии выбора методов и мер

А.1 Общие положения

В данном приложении представлены связанные с определенными разделами настоящего стандарта таблицы (см. А.2, таблицы А.1—А.11), которые иллюстрируют средства достижения соответствия требованиям, перечисленным в этих разделах. Далее следуют также таблицы более низкого уровня, (см. А.3, таблицы А.12—А.23), которые подробно останавливаются на определенных записях в таблицах для разделов. Например, запись «Моделирование» в таблице А.2 имеет ссылку на таблицу А.17, в которой подробно рассматривается моделирование. Кроме того в этих таблицах для разделов имеются ссылки на справочное приложение D.

С каждым методом или мерой в таблицах связано требование их применения для каждого уровня полноты безопасности (УПБ) программного обеспечения. В настоящем стандарте требования для уровней полноты безопасности программного обеспечения 1 и 2 одинаковы для каждого метода. Точно так же каждый метод имеет одинаковые требования для уровней полноты безопасности программного обеспечения 3 и 4. Эти требования следующие:

'M' — этот символ означает, что использование метода обязательно;

'HR' — этот символ означает, что метод или мера настоятельно рекомендованы для этого уровня полноты безопасности. Если этот метод или мера не будут использоваться, то тогда должно быть дано детальное объяснение об использовании альтернативных методов в плане обеспечения качества программного обеспечения или в другом документе, на который ссылается план обеспечения качества программного обеспечения;

'R' — этот символ означает, что метод или мера рекомендуются для этого уровня полноты безопасности. Но степень обязательности рекомендации ниже, чем в случае рекомендации 'HR', и такие методы могут объединяться в пакеты методов;

'-' — для данного метода или средства рекомендации ни за, ни против не приводятся,

'NR' — этот символ означает, что данный метод или средство решительно не рекомендуется для этого уровня полноты безопасности. Если этот метод или мера будут использоваться, то должно быть дано подробное объяснение о его использовании в плане обеспечения качества программного обеспечения или в другом документе, на который ссылается план обеспечения качества программного обеспечения.

Комбинация методов или мер должна быть утверждена в плане обеспечения качества программного обеспечения или в другом документе, на который ссылается план обеспечения качества программного обеспечения, вместе с одним или более выбранными методами или мерами, если примечания к таблице не содержат иных требований. Эти примечания могут включать ссылку на одобренные методы или одобренные комбинации методов. Если такие методы или комбинации методов, включая все соответствующие обязательные методы, будут использоваться, то оценщик должен принять их как допустимые и должен только внимательно следить за их правильным применением. Если используется различный набор методов и он может быть обоснован, то оценщик может считать это приемлемым.

А.2 Таблицы разделов

Таблица А.1 — Проблемы жизненного цикла и документация (см. 5.3)

Документация	УПБ 0	УПБ 1	УПБ 2	УПБ 3	УПБ 4
Планирование					
1 План обеспечения качества программного обеспечения	HR	HR	HR	HR	HR
2 Отчет о проверке обеспечения качества программного обеспечения	HR	HR	HR	HR	HR
3 План управления конфигурацией программного обеспечения	HR	HR	HR	HR	HR
4 План проверки программного обеспечения	HR	HR	HR	HR	HR
5 План подтверждения соответствия программного обеспечения	HR	HR	HR	HR	HR
Требования к программному обеспечению					
6 Спецификация требований к программному обеспечению	HR	HR	HR	HR	HR
7 Спецификация тестирования всего программного обеспечения	HR	HR	HR	HR	HR

Продолжение таблицы А.1

Документация	УПБ 0	УПБ 1	УПБ 2	УПБ 3	УПБ 4
8 Очет о проверке требований к программному обеспечению	HR	HR	HR	HR	HR
Архитектура и проект программного обеспечения					
9 Спецификация архитектуры программного обеспечения	HR	HR	HR	HR	HR
10 Спецификация проекта программного обеспечения	HR	HR	HR	HR	HR
11 Спецификации интерфейса программного обеспечения	HR	HR	HR	HR	HR
12 Спецификация тестирования интеграции программного обеспечения	HR	HR	HR	HR	HR
13 Спецификация тестирования интеграции программного обеспечения/ аппаратных средств	HR	HR	HR	HR	HR
14 Отчет о проверке архитектуры и проекта программного обеспечения	HR	HR	HR	HR	HR
Проект компонента программного обеспечения					
15 Спецификация проекта компонента программного обеспечения	R	HR	HR	HR	HR
16 Спецификация тестирования компонента программного обеспечения	R	HR	HR	HR	HR
17 Отчет о проверке проекта компонента программного обеспечения	R	HR	HR	HR	HR
Реализация и тестирование компонента					
18 Исходный код программного обеспечения и сопроводительная документация	HR	HR	HR	HR	HR
19 Отчет об испытаниях компонента программного обеспечения	R	HR	HR	HR	HR
20 Отчет о проверке исходного кода программного обеспечения	HR	HR	HR	HR	HR
Интеграция					
21 Отчет об испытаниях интеграции программного обеспечения	HR	HR	HR	HR	HR
22 Отчет об испытаниях интеграции программного обеспечения/аппаратных средств	HR	HR	HR	HR	HR
23 Отчет о проверке интеграции программного обеспечения	HR	HR	HR	HR	HR
Тестирование всего программного обеспечения/ заключительное подтверждение соответствия					
24 Отчет об испытаниях всего программного обеспечения	HR	HR	HR	HR	HR
25 Отчет о проверке испытаний всего программного обеспечения	HR	HR	HR	HR	HR
26 Отчет о подтверждении соответствия программного обеспечения	HR	HR	HR	HR	HR
27 Отчет о подтверждении соответствия инструментальных средств	R	HR	HR	HR	HR
28 Информация о версии	HR	HR	HR	HR	HR
Системы, сконфигурированные прикладными данными или алгоритмами					
29 Спецификация требований к приложению	HR	HR	HR	HR	HR
30 План подготовки приложения (см. примечание 2)	HR	HR	HR	HR	HR
31 Спецификация тестирования приложения (см. примечание 2)	HR	HR	HR	HR	HR
32 Архитектура и проект приложения (см. примечание 2)	HR	HR	HR	HR	HR
33 Отчет о проверке подготовки приложения	HR	HR	HR	HR	HR
34 Отчет об испытаниях приложения	HR	HR	HR	HR	HR

Окончание таблицы А.1

Документация	УПБ 0	УПБ 1	УПБ 2	УПБ 3	УПБ 4
35 Исходный код данных/алгоритмов приложения	HR	HR	HR	HR	HR
36 Отчет о проверке данных/алгоритмов приложения	HR	HR	HR	HR	HR
Развертывание программного обеспечения					
37 План выпуска и развертывания программного обеспечения	R	HR	HR	HR	HR
38 Руководство по развертыванию программного обеспечения	R	HR	HR	HR	HR
39 Информация о версии	HR	HR	HR	HR	HR
40 Журнал развертывания	R	HR	HR	HR	HR
41 Отчет о проверке развертывания	R	HR	HR	HR	HR
Сопровождение программного обеспечения					
42 План сопровождения программного обеспечения	R	HR	HR	HR	HR
43 Журнал изменений программного обеспечения	HR	HR	HR	HR	HR
44 Журнал сопровождения программного обеспечения	R	HR	HR	HR	HR
45 Отчет о проверке сопровождения программного обеспечения	R	HR	HR	HR	HR
Оценка программного обеспечения					
46 План оценки программного обеспечения	R	HR	HR	HR	HR
47 Отчет по результатам оценки программного обеспечения	R	HR	HR	HR	HR

Примечания

1 Согласно 5.3.2.11 и 5.3.2.12 документы могут быть объединены по-другому.

2 Указание значений ‘HR’ или ‘R’ для документов 30—32 зависит от важности, определяемой в процессе и в случае выполнения проверки. Например, для проверки данных может быть только необходимо их тестирование в предметной области системы, в то время как более функциональные свойства должны быть и протестированы и проверены. В этом случае используется ‘HR’, но может быть дополнительно и ‘R’.

3 О плане и отчете результатов оценки программного обеспечения см. 6.4.1.2.

4 Отчет об обеспечении качества программного обеспечения включен в отчет управления качеством, определенный в МЭК 62425.

Таблица А.2 — Спецификация требований к программному обеспечению (см. 7.2)

Методы/меры	Ссылка	УПБ 0	УПБ 1	УПБ 2	УПБ 3	УПБ 4
1 Формальные методы (математические подходы)	D.28	—	R	R	HR	HR
2 Моделирование	Таблица А.17	R	R	R	HR	HR
3 Структурная методология	D.52	R	R	R	HR	HR
4 Таблица решений	D.13	R	R	R	HR	HR

Требования

a) Спецификация требований к программному обеспечению должна включать описание проблемы на естественном языке и в любой необходимой формальной или полуформальной нотации.

b) Настоящая таблица ясно и точно отражает дополнительные требования для определения спецификации. Чтобы удовлетворить используемому уровню полноты безопасности программного обеспечения должен быть выбран один или более из этих методов.

Таблица А.3 — Архитектура программного обеспечения (см. 7.3)

Методы/меры	Ссылка	УПБ 0	УПБ 1	УПБ 2	УПБ 3	УПБ 4
1 Программирование с защитой	D.14	—	HR	HR	HR	HR
2 Обнаружение и диагностика ошибок	D.26	—	R	R	HR	HR
3 Коды с коррекцией ошибок	D.19	—	—	—	—	—
4 Коды с обнаружением ошибок	D.19	—	R	R	HR	HR
5 Программирование с проверкой ошибок	D.24	—	R	R	HR	HR
6 Методы «подушки безопасности»	D.47	—	R	R	R	R
7 Многовариантное программирование	D.16	—	R	R	HR	HR
8 Блок восстановления	D.44	—	R	R	R	R
9 Восстановление предыдущего состояния	D.5	—	NR	NR	NR	NR
10 Прямое исправление	D.30	—	NR	NR	NR	NR
11 Механизмы повторного восстановления после ошибок	D.46	—	R	R	R	R
12 Сохранение достигнутых состояний	D.36	—	R	R	HR	HR
13 Искусственный интеллект. Коррекция ошибок	D.1	—	NR	NR	NR	NR
14 Динамическое реконфигурирование программного обеспечения	D.17	—	NR	NR	NR	NR
15 Анализ влияния ошибок в программном обеспечении	D.25	—	R	R	HR	HR
16 Постепенное отключение функций	D.31	—	R	R	HR	HR
17 Ограничение доступа информации	D.33	—	—	—	—	—
18 Инкапсуляция информации	D.33	R	HR	HR	HR	HR
19 Полностью определенный интерфейс	D.38	HR	HR	HR	M	M
20 Формальные методы	D.28	—	R	R	HR	HR
21 Моделирование	Таблица А.17	R	R	R	HR	HR
22 Структурная методология	D.52	R	HR	HR	HR	HR
23 Моделирование, поддержанное инструментальными средствами спецификации и автоматизированного проектирования	Таблица А.17	R	R	R	HR	HR

Требования

a) Принятые комбинации методов для УПБ 3 и 4 программного обеспечения следующие:

1) 1, 7, 19, 22 и один из 4, 5, 12 или 21;

2) 1, 4, 19, 22 и один из 2, 5, 12, 15 или 21.

b) Принятые комбинации методов для УПБ 1 и 2 программного обеспечения следующие: 1, 19, 22 и один из 2, 4, 5, 7, 12, 15 или 21.

c) Некоторые из этих проблем могут быть определены на уровне системы.

d) Коды с обнаружением ошибок могут использоваться в соответствии с требованиями ЕН 50159.

Примечание — Метод/мера 19 применяется для внешних интерфейсов.

Таблица А.4 — Проект и реализация программного обеспечения (см. 7.4)

Методы/меры	Ссылка	УПБ 0	УПБ 1	УПБ 2	УПБ 3	УПБ 4
1 Формальные методы	D.28	—	R	R	HR	HR
2 Моделирование	Таблица А.17	R	HR	HR	HR	HR
3 Структурная методология	D.52	R	HR	HR	HR	HR
4 Модульный подход	D.38	HR	M	M	M	M
5 Компоненты	Таблица А.20	HR	HR	HR	HR	HR
6 Стандарты проектирования и кодирования	Таблица А.12	HR	HR	HR	M	M
7 Анализируемые программы	D.2	HR	HR	HR	HR	HR
8 Строго типизированный язык программирования	D.49	R	HR	HR	HR	HR
9 Структурное программирование	D.53	R	HR	HR	HR	HR
10 Язык программирования	Таблица А.15	R	HR	HR	HR	HR
11 Подмножество языка программирования	D.35	—	—	—	HR	HR
12 Объктно-ориентированное программирование	Таблица А.22 D.57	R	R	R	R	R
13 Процедурное программирование	D.60	R	HR	HR	HR	HR
14 Метапрограммирование	D.59	R	R	R	R	R
Требования a) Принятая комбинация методов для УПБ 3 и 4 программного обеспечения: 4, 5, 6, 8 и один из 1 или 2. b) Принятая комбинация методов для УПБ 1 и 2 программного обеспечения: 3, 4, 5, 6 и один из 8, 9 или 10. c) Метапрограммирование должно быть ограничено созданием исходного кода программного обеспечения перед компиляцией.						

Таблица А.5 — Проверка и тестирование (см. 6.2, 7.3, 7.5)

Методы/меры	Ссылка	УПБ 0	УПБ 1	УПБ 2	УПБ 3	УПБ 4
1 Формальное доказательство	D.29	—	R	R	HR	HR
2 Статический анализ	Таблица А.19	—	HR	HR	HR	HR
3 Динамический анализ и тестирование	Таблица А.13	—	HR	HR	HR	HR
4 Метрики	D.37	—	R	R	R	R
5 Прослеживаемость	D.58	R	HR	HR	M	M
6 Анализ влияния ошибок в программном обеспечении	D.25	—	R	R	HR	HR
7 Тестовый охват для кода	Таблица А.21	R	HR	HR	HR	HR
8 Функциональное тестирование или тестирование методом «черного ящика»	Таблица А.14	HR	HR	HR	M	M
9 Тестирование производительности	Таблица А.18	—	HR	HR	HR	HR
10 Тестирование интерфейса	D.34	HR	HR	HR	HR	HR
Требования a) Принятая комбинация методов для УПБ 3 и 4 программного обеспечения: 3 5, 7, 8 и один из 1, 2 или 6. b) Принятая комбинация методов для УПБ 1 и 2 программного обеспечения: 5 вместе с одним из 2, 3 или 8. Примечания 1 Методы/меры 1, 2, 4, 5, 6 и 7 для действий по проверке. 2 Методы/меры 3, 8, 9 и 10 для действий по тестированию.						

Таблица А.6 — Интеграция (см. 7.6)

Методы/меры	Ссылка	УПБ 0	УПБ 1	УПБ 2	УПБ 3	УПБ 4
1 Функциональное тестирование или тестирование методом «черного ящика»	Таблица А.14	HR	HR	HR	HR	HR
2 Тестирование производительности	Таблица А.18	—	R	R	HR	HR

Таблица А.7 — Тестирование всего программного обеспечения (см. 6.2 и 7.7)

Методы/меры	Ссылка	УПБ 0	УПБ 1	УПБ 2	УПБ 3	УПБ 4
1 Тестирование производительности	Таблица А.18	—	HR	HR	M	M
2 Функциональное тестирование или тестирование методом «черного ящика»	Таблица А.14	HR	HR	HR	M	M
3 Моделирование	Таблица А.18	—	R	R	R	R
Требование Принятая комбинация методов для УПБ 1 и 2 программного обеспечения: 1 или 2.						

Таблица А.8 — Методы анализа программного обеспечения (см. 6.3)

Методы/меры	Ссылка	УПБ 0	УПБ 1	УПБ 2	УПБ 3	УПБ 4
1 Статический анализ программного обеспечения	D.13, D.37, Таблица А.19	R	HR	HR	HR	HR
2 Динамический анализ программного обеспечения	Таблица А.13, Таблица А.14	—	R	R	HR	HR
3 Причинно-следственные диаграммы	D.6	R	R	R	R	R
4 Анализ дерева событий	D.22	—	R	R	R	R
5 Анализ влияния ошибок в программном обеспечении	D.25	—	R	R	HR	HR
Требование Чтобы удовлетворить используемому уровню полноты безопасности программного обеспечения, из этих методов должен быть выбран один или более методов.						

Таблица А.9 — Контроль программного обеспечения (см. 6.5)

Методы/меры	Ссылка	УПБ 0	УПБ 1	УПБ 2	УПБ 3	УПБ 4
1 Сертифицировано по ИСО 9001	7.1	R	HR	HR	HR	HR
2 Соотвествует ИСО 9001	7.1	M	M	M	M	M
3 Соотвествует ИСО 90003	7.1	R	R	R	R	R
4 Система качества компании	7.1	M	M	M	M	M
5 Управление конфигурацией программного обеспечения	D.48	M	M	M	M	M
6 Таблица контрольных проверок	D.7	R	HR	HR	HR	HR
7 Прослеживаемость	D.58	R	HR	HR	M	M
8 Регистрация и анализ данных	D.12	HR	HR	HR	M	M
Требование Данная таблица должна применяться к различным ролям на всех стадиях.						

Таблица А.10 — Сопровождение программного обеспечения (см. 9.2)

Методы/меры	Ссылка	УПБ 0	УПБ 1	УПБ 2	УПБ 3	УПБ 4
1 Анализ влияния	D.32	R	HR	HR	M	M
2 Регистрация и анализ данных	D.12	HR	HR	HR	M	M

Таблица А.11 — Методы подготовки данных (см. 8.4)

Методы/меры	Ссылка	УПБ 0	УПБ 1	УПБ 2	УПБ 3	УПБ 4
1 Табличные методы спецификации	D.68	R	R	R	R	R
2 Специализированный язык	D.69	R	R	R	R	R
3 Моделирование процесса	D.42	R	HR	HR	HR	HR
4 Функциональное тестирование	D.42	M	M	M	M	M
5 Таблицы контрольных проверок	D.7	R	HR	HR	M	M
6 Оценка программного обеспечения по Фагану	D.23	—	R	R	R	R
7 Формальный анализ проекта	D.56	R	HR	HR	HR	HR
8 Формальное доказательство корректности (данных)	D.29	—	—	—	HR	HR
9 Сквозной контроль	D.56	R	R	R	HR	HR

Требования

a) Принятые комбинации методов для УПБ 1 и 2 программного обеспечения следующие: 1 и 2.

b) Принятые комбинации методов для УПБ 3 и 4 программного обеспечения следующие: 1, 4, 5 и 7 или 2, 3 и 6.

Примечание — Описание ссылки D.29 касается программ, в то время как метод 8 в данном контексте применяется к формальному доказательству правильности данных.

А.3 Подробные таблицы

Таблица А.12 — Стандарты кодирования

Методы/меры	Ссылка	УПБ 0	УПБ 1	УПБ 2	УПБ 3	УПБ 4
1 Стандарт кодирования	D.15	HR	HR	HR	M	M
2 Руководство по стилю кодирования	D.15	HR	HR	HR	HR	HR
3 Отсутствие динамических объектов	D.15	—	R	R	HR	HR
4 Отсутствие динамических переменных	D.15	—	R	R	HR	HR
5 Ограниченное использование указателей	D.15	—	R	R	R	R
6 Ограниченное использование рекурсий	D.15	—	R	R	HR	HR
7 Отсутствие безусловных переходов	D.15	—	HR	HR	HR	HR
8 Ограниченный размер и сложность функций, подпрограмм и методов	D.38	HR	HR	HR	HR	HR
9 Стратегия точки входа/выхода для функций, подпрограмм и методов	D.38	R	HR	HR	HR	HR
10 Ограниченное количество параметров подпрограммы	D.38	R	R	R	R	R
11 Ограниченное использование глобальных переменных	D.38	HR	HR	HR	M	M

Требование

Признано, что методы 3, 4 и 5 могут присутствовать в рамках прошедшего подтверждение соответствия компилятора или транслятора.

Таблица А.13 — Динамический анализ и тестирование

Методы/меры	Ссылка	УПБ 0	УПБ 1	УПБ 2	УПБ 3	УПБ 4
1 Формирование тестового сценария при анализе граничных значений	D.4	—	HR	HR	HR	HR
2 Формирование тестового сценария при предположении ошибок	D.20	R	R	R	R	R
3 Формирование тестового сценария при введении ошибок	D.21	—	R	R	R	R
4 Моделирование реализации	D.39	—	R	R	HR	HR
5 Классы эквивалентности и тестирование разбиением входных данных	D.18	R	R	R	HR	HR
6 Структурное тестирование	D.50	—	R	R	HR	HR
Требование Анализ тестовых сценариев выполняется на уровне подсистемы и основывается на ее спецификации и/или на ее спецификации и ее коде.						

Таблица А.14 — Функциональное тестирование или тестирование методом «черного ящика»

Методы/меры	Ссылка	УПБ 0	УПБ 1	УПБ 2	УПБ 3	УПБ 4
1 Формирование тестовых сценариев из причинно-следственных диаграмм	D.6	—	—	—	R	R
2 Прототипирование/анимация	D.43	—	—	—	R	R
3 Анализ граничных значений	D.4	R	HR	HR	HR	HR
4 Классы эквивалентности и тестирование разбиением входных данных	D.18	R	HR	HR	HR	HR
5 Моделирование процесса	D.42	R	R	R	R	R
Требование Полнота моделирования будет зависеть от значения уровня полноты безопасности программного обеспечения, степени сложности и приложения.						

Таблица А.15 — Текстовые языки программирования

Методы/меры	Ссылка	УПБ 0	УПБ 1	УПБ 2	УПБ 3	УПБ 4
1 ADA	D.54	R	HR	HR	HR	HR
2 MODULA-2	D.54	R	HR	HR	HR	HR
3 PASCAL	D.54	R	HR	HR	HR	HR
4 C или C++	D.54, D.35	R	R	R	R	R
5 PL/M	D.54	R	R	R	NR	NR
6 BASIC	D.54	R	NR	NR	NR	NR
7 Ассемблер	D.54	R	R	R	R	R
8 C#	D.54, D.35	R	R	R	R	R
9 JAVA	D.54, D.35	R	R	R	R	R
10 Список операторов	D.54	R	R	R	R	R

Окончание таблицы А.15

Требования

a) Выбор языков должен основываться на требованиях, представленных в 6.7 и 7.3.

b) Не существует никакого требования к обоснованию решения по исключению конкретных языков программирования.

Примечания

1 Для получения информации об оценке пригодности языка программирования см. D.54, «Выбор подходящего языка программирования».

2 Если конкретный язык в таблице отсутствует, то он не исключается автоматически. Он, как предполагается, будет соответствовать D.54.

3 Системы времени выполнения, связанные с выбранными языками, необходимыми для выполнения прикладных программ, должны быть все же обоснованы для использования с соответствующим уровнем полноты безопасности программного обеспечения.

Таблица А.16 — Языки диаграмм для прикладных алгоритмов

Методы/меры	Ссылка	УПБ 0	УПБ 1	УПБ 2	УПБ 3	УПБ 4
1 Функциональные блок-схемы	D.63	R	R	R	R	R
2 Последовательные функциональные схемы	D.61	—	HR	HR	HR	HR
3 Лестничные диаграммы	D.62	R	R	R	R	R
4 Диаграммы состояний	D.64	R	HR	HR	HR	HR

Таблица А.17 — Моделирование

Методы/меры	Ссылка	УПБ 0	УПБ 1	УПБ 2	УПБ 3	УПБ 4
1 Моделирование данных	D.65	R	R	R	HR	HR
2 Диаграммы потоков данных	D.11	—	R	R	HR	HR
3 Диаграммы потоков управления	D.66	R	R	R	HR	HR
4 Конечные автоматы или диаграммы изменения состояний	D.27	—	HR	HR	HR	HR
5 Временные сети Петри	D.55	—	R	R	HR	HR
6 Таблицы истинности/таблицы решений	D.13	R	R	R	HR	HR
7 Формальные методы	D.28	—	R	R	HR	HR
8 Моделирование реализации	D.39	—	R	R	HR	HR
9 Прототипирование/анимация	D.43	—	R	R	R	R
10 Структурные диаграммы	D.51	—	R	R	HR	HR
11 Диаграммы последовательностей	D.67	R	R	HR	HR	HR

Требования

a) Инструкция по моделированию должна быть определена и использоваться.

b) По крайней мере, один из методов HR должен быть выбран.

Таблица А.18 — Тестирование производительности

Методы/меры	Ссылка	УПБ 0	УПБ 1	УПБ 2	УПБ 3	УПБ 4
1 Проверка на критические нагрузки или стресс-тестирование	D.3	—	R	R	HR	HR
2 Синхронизация ответа и ограничения памяти	D.45	—	HR	HR	HR	HR
3 Требования к реализации	D.40	—	HR	HR	HR	HR

Таблица А.19 — Статический анализ

Методы/меры	Ссылка	УПБ 0	УПБ 1	УПБ 2	УПБ 3	УПБ 4
1 Анализ граничных значений	D.4	—	R	R	HR	HR
2 Таблицы контрольных проверок	D.7	—	R	R	R	R
3 Анализ потока команд управления	D.8	—	HR	HR	HR	HR
4 Анализ потока данных	D.10	—	HR	HR	HR	HR
5 Предположение об ошибках	D.20	—	R	R	R	R
6 Сквозной контроль или анализ проекта	D.56	HR	HR	HR	HR	HR

Таблица А.20 — Компоненты

Методы/меры	Ссылка	УПБ 0	УПБ 1	УПБ 2	УПБ 3	УПБ 4
1 Ограничение доступа информации	D.33	—	—	—	—	—
2 Инкапсуляция информации	D.33	R	HR	HR	HR	HR
3 Ограничение для числа параметров	D.38	R	R	R	R	R
4 Полностью определенный интерфейс	D.38	R	HR	HR	M	M

Требование

Если нет никакой общей стратегии доступа к данным, то настоятельно рекомендуются только методы ограничения доступа и инкапсуляции информации.

Примечание — Метод/мера 4 используется только для внутренних интерфейсов.

Таблица А.21 — Тестовый охват для кода

Критерий тестового охвата	Ссылка	УПБ 0	УПБ 1	УПБ 2	УПБ 3	УПБ 4
1 Оператор	D.50	R	HR	HR	HR	HR
2 Ветвление	D.50	—	R	R	HR	HR
3 Составное условие	D.50	—	R	R	HR	HR
4 Поток данных	D.50	—	R	R	HR	HR
5 Последовательность	D.50	—	R	R	HR	HR

Требования

a) Для каждого значения УПБ для реализуемого теста должна быть разработана количественная мера охвата. Она может обеспечить решение о доверии, полученном в результате тестирования, и необходимости применения дополнительных методов.

b) Для УПБ 3 или 4 тестовый охват на уровне компонентов должен быть измерен по следующим критериям:

- 2 и 3, или
- 2 и 4, или
- 5,

или тестовый охват на уровне интеграции должен быть измерен по одному или более критериев из 2, 3, 4 или 5.

c) Могут использоваться другие критерии тестового охвата, учитывая, что это может быть обосновано. Эти критерии зависят от архитектуры программного обеспечения (см. таблицу А.3) и языка программирования (см. таблицы А.15 и А.16).

d) Для любого кода, который протестировать не реально, должно быть продемонстрировано его корректное использование с помощью подходящего метода, например, методом статического анализа из таблицы А.19.

Окончание таблицы А.21

Примечания

1 Охват операторов автоматически достигается, используя критерии 2—5.

2 Критерии тестового охвата в данной таблице используются для структурного тестирования (основанного на коде, методом белого ящика). Методы/меры для функционального тестирования (основанного на спецификации, методом черного ящика) даны в таблице А.14.

3 Высокий процент охвата обычно трудно достигнуть. Использование выполнение тестового сценария из метода анализа граничных значений (D.4) и метода классов эквивалентности и тестирования разбиением входных данных (D.18) может позволить достичь достаточный охват с меньшим числом тестов.

4 Различие между 2 и 3 на практике зависит от уровня языка программирования и использования составных условий. Если используются только единичные условия, например, в результате компиляции, то критерии 2 и 3 считаются идентичными.

Таблица А.22 — Архитектура объектно-ориентированного программного обеспечения

Методы/меры	Ссылка	УПБ 0	УПБ 1	УПБ 2	УПБ 3	УПБ 4
1 Прослеживаемость концепции прикладной предметной области к классам архитектуры	—	R	R	R	HR	HR
2 Использование подходящих фреймов, общеиспользуемых комбинаций классов и шаблонов разработки	—	R	R	R	HR	HR
3 Детальное объектно-ориентированное проектирование	Таблица А.23	R	R	R	HR	HR

Требование

При использовании существующих фреймов и шаблонов разработки, к ним применяются требования существующего ранее программного обеспечения.

Примечания

1 Объектно-ориентированный подход представляет информацию не так, как это делается в процедурных подходах. Далее следует список рекомендаций, требующих конкретного рассмотрения:

- понимание иерархий классов и идентификации функции(й) программного обеспечения, которые будут выполняться при вызове данного метода (включая случай использования существующей библиотеки классов);
- структурное тестирование (таблица А.13).

Прослеживаемость от прикладной предметной области до архитектуры класса менее важна.

2 В рамках намеченного программного обеспечения может существовать фрейм в уже существующем программном обеспечении, который успешно решает подобную задачу, и это хорошо известно разработчикам. Использование этого фрейма считают хорошей практикой.

Таблица А.23 — Детальное объектно-ориентированное проектирование

Методы/меры	Ссылка	УПБ 0	УПБ 1	УПБ 2	УПБ 3	УПБ 4
1 Классы должны иметь только одну цель	—	R	R	R	HR	HR
2 Наследование используется, только если производный класс является уточнением своего базового класса	—	R	HR	HR	HR	HR
3 Глубина наследования ограничена стандартами кодирования	—	R	R	R	HR	HR
4 Переопределение операций (методов) строго контролируется	—	R	R	R	HR	HR
5 Множественное наследование, используется только для интерфейсных классов	—	R	HR	HR	HR	HR
6 Наследование от неизвестных классов	—	—	—	—	HR	HR

Требования

a) Один класс характеризуется, как имеющий одну ответственность, т. е. он отвечает за тесно связанные данные и операции на этих данных.

b) Требуется быть внимательным, чтобы избежать циклических зависимостей между объектами.

Приложение В
(обязательное)

Основные роли и обязанности специалистов в области программного обеспечения

Таблица В.1 — Ролевая спецификация менеджера требований

Роль. Менеджер требований
Обязанности: 1) отвечает за определение требований к программному обеспечению; 2) полностью контролирует спецификацию требований к программному обеспечению; 3) устанавливает и сопровождает прослеживаемость к и от требований уровня системы; 4) гарантирует, что спецификации и требования к программному обеспечению являются объектами управления изменениями и управления конфигурацией, включая состояние, версию и состояние авторизации; 5) гарантирует непротиворечивость и полноту спецификации требований к программному обеспечению (для требований пользователя и окончательного окружения приложения); 6) разрабатывает и сопровождает документы с требованиями к программному обеспечению.
Основные компетенции. Он должен: 1) быть компетентен в разработке требований; 2) быть экспертом в прикладной предметной области; 3) быть экспертом в характеристиках безопасности прикладной предметной области; 4) полностью понимать роль системы и окружения приложения; 5) разбираться в аналитических методах и в их результатах; 6) понимать применяемые инструкции; 7) понимать требования настоящего стандарта.

Таблица В.2 — Ролевая спецификация проектировщика

Роль. Проектировщик
Обязанности: 1) преобразует определенные требования к программному обеспечению в приемлемые решения; 2) полностью контролирует архитектуру и нисходящие решения; 3) определяет или выбирает инструментальные средства поддержки и методы разработки; 4) применяет надлежащие принципы разработки и стандарты; 5) разрабатывает спецификации на компоненты, где это необходимо; 6) сопровождает прослеживаемость к и от указанных требований к программному обеспечению; 7) разрабатывает и сопровождает проектную документацию; 8) гарантирует, что документы проекта являются объектами управления изменениями и управления конфигурацией.
Основные компетенции. Он должен: 1) быть компетентным проектировщиком в соответствующей прикладной области; 2) быть компетентным специалистом по реализации принципов безопасности; 3) быть компетентным в методологиях анализа проекта и тестирования проекта; 4) быть в состоянии работать с ограничениями проекта в заданном окружении; 5) быть компетентным в понимании проблемной области; 6) понимать все ограничения, наложенные аппаратной платформой, операционной системой и взаимодействующими через интерфейс системами; 7) понимать соответствующие разделы/подразделы настоящего стандарта.

Таблица В.3 — Ролевая спецификация разработчика

Роль. Разработчик
Обязанности: 1) преобразует проектные решения в данные/исходный код/другие проектные представления; 2) преобразует исходный код в исполняемый код/другое представление проекта; 3) применяет принципы разработки систем безопасности; 4) применяет конкретные стандарты по подготовке данных/кодированию; 5) выполняет анализ для проверки промежуточных результатов; 6) интегрирует программное обеспечение на целевой машине; 7) разрабатывает и сопровождает документы реализации, включающие прикладные методы, типы данных и листинги; 8) сопровождает прослеживаемость к и от проекта; 9) сопровождает сгенерированные или измененные данные/код под управлением изменениями и управлением конфигурацией.
Основные компетенции. Он должен: 1) быть компетентным разработчиком в соответствующей прикладной области; 2) быть компетентным специалистом по языку реализации и инструментальным средствам поддержки; 3) быть способным применять конкретные стандарты кодирования и стили программирования; 4) понимать все ограничения, наложенные аппаратной платформой, операционной системой и взаимодействующими через интерфейс системами; 5) понимать соответствующие разделы/подразделы настоящего стандарта.

Таблица В.4 — Ролевая спецификация тестировщика

Роль. Тестировщик
Обязанности: 1) гарантирует, что тестовые действия планируются; 2) разрабатывает спецификацию тестирования (цели и сценарии); 3) гарантирует прослеживаемость тестовых целей к соответствующим указанным требованиям к программному обеспечению, и тестовых сценариев к указанным тестовым целям; 4) гарантирует, что запланированные тесты реализованы и конкретные тесты выполнены; 5) идентифицирует отклонения от ожидаемых результатов и фиксирует их в отчетах об испытаниях; 6) передает отклонения с соответствующей информацией по управлению изменениями для оценки и принятия решения; 7) собирает результаты в отчетах; 8) выбирает оборудование для испытаний программного обеспечения.
Основные компетенции. Он должен: 1) быть компетентным в предметной области, где выполняется тестирование, например, требования к программному обеспечению, данные, код, и т. д.; 2) быть компетентным в различных и подходах/методологиях тестирования и проверки и быть в состоянии идентифицировать самый подходящий метод для данного контекста; 3) быть способным получать тестовые сценарии из заданных спецификаций; 4) иметь способность к аналитическому мышлению и хорошие навыки наблюдения; 5) понимать соответствующие разделы/подразделы настоящего стандарта.

Таблица В.5 — Ролевая спецификация менеджера по проверке

Роль. Менеджер по проверке
Обязанности: 1) разрабатывает план проверки программного обеспечения (который может включать качественные вопросы), устанавливающий, какая нужна проверка и как реализуется (например, с помощью обзора, анализа и т. д.), а также требуемый тест, как доказательство; 2) проверяет соответствие (полноту, непротиворечивость, правильность, уместность и прослеживаемость) задокументированного доказательства из анализа, интеграции и тестирования со специфицированными целями проверки; 3) идентифицирует аномалии, оценивает их в терминах риска (влияния), записывает и передает их в соответствующий орган по управлению изменениями для оценки и принятия решения; 4) управляет процессом проверки (анализ, интеграция и тестирование) и гарантирует требуемую независимость действий; 5) разрабатывает и сопровождает журналы о действиях проверки; 6) разрабатывает отчет о проверке, утверждая результаты действий проверки.
Основные компетенции. Он должен: 1) быть компетентным в предметной области, где выполняется проверка, например, в требованиях к программному обеспечению, в данных, в кодах, и т. д.; 2) быть компетентным в различных подходах/методологиях проверки и быть в состоянии определить самый подходящий метод или комбинацию методов для заданного контекста; 3) быть способным выводить тип проверки из заданных спецификаций; 4) иметь аналитическое мышление и хорошие навыки наблюдения; 5) понимать соответствующие разделы/подразделы настоящего стандарта.

Таблица В.6 — Ролевая спецификация интегратора

Роль. Интегратор
Обязанности: 1) управляет процессом интеграции, используя базовые конфигурации программного обеспечения; 2) разрабатывает спецификации тестирования интеграции программного обеспечения и программного обеспечения/аппаратных средств для компонентов программного обеспечения на основе спецификаций компонентов и архитектуры, сформированных проектировщиком, устанавливая необходимые входные компоненты, последовательность действий для интеграции и результирующие интегрированные компоненты; 3) разрабатывает и сопровождает записи о действиях по интеграции; 4) идентифицирует аномалии интеграции, записывает и передает их в соответствующий орган по управлению изменениями для оценки и принятия решения; 5) разрабатывает отчет об интеграции компоненты и всей системы, утверждая результат интеграции.
Основные компетенции. Он должен: 1) быть компетентным в предметной области, где выполняется интеграция, например, в соответствующих языках программирования, интерфейсах программного обеспечения, операционных системах, данных, платформах, кодах и т. д.; 2) быть компетентным в различных и подходах/методологиях интеграции и быть в состоянии идентифицировать самый подходящий метод или комбинацию методов для данного контекста; 3) быть компетентным в понимании проекта и функциональности, требуемой на различных промежуточных уровнях; 4) быть способным получать типы интеграционных тестов из набора интегрированных функций; 5) иметь способность к аналитическому мышлению и хорошие навыки наблюдения, желательно на уровне системы; 6) понимать соответствующие разделы/подразделы настоящего стандарта.

Таблица В.7 — Ролевая спецификация менеджера по подтверждению соответствия

Роль. Менеджер по подтверждению соответствия
Обязанности: 1) разрабатывает понимание системы программного обеспечения в намеченном окружении; 2) разрабатывает план подтверждения соответствия и определяет существенные задачи и действия для подтверждения соответствия программного обеспечения и согласует этот план с оценщиком; 3) анализирует требования к программному обеспечению с намеченным окружением/использованием; 4) анализирует программное обеспечение и требования к программному обеспечению, чтобы гарантировать их выполнение; 5) оценивает соответствие процесса создания программного обеспечения и разработанного программного обеспечения требованиям настоящего стандарта, учитывая заданный УПБ; 6) рассматривает правильность, непротиворечивость и соответствие проверки и тестирования; 7) проверяет правильность, непротиворечивость и соответствие тестовых сценариев и выполняемых тестов; 8) должен гарантировать, что все действия плана подтверждения соответствия выполнены; 9) рассматривает и классифицирует все отклонения в терминах риска (влияния), записывает и передает их в соответствующий орган по управлению изменениями и принятию решения; 10) дает рекомендацию о пригодности программного обеспечения для надлежащего использования и указывает любые ограничения приложения по необходимости; 11) собирает отклонения от плана подтверждения соответствия; 12) выполняет аудиты, инспекции или анализ всего проекта (как конкретных реализаций общего процесса разработки) на различных стадиях разработки по необходимости; 13) рассматривает и анализирует отчеты о выполнении подтверждения соответствия, касающиеся предыдущих приложений по необходимости; 14) рассматривает, что разработанные решения прослеживаемы до требований к программному обеспечению; 15) гарантирует, что зарегистрированные опасности и остающиеся несоответствия рассмотрены, и все опасности ликвидированы надлежащим способом посредством их устранения или с помощью мер управления/преобразования рисков; 16) разрабатывает отчет по подтверждению соответствия; 17) дает согласие/несогласие на поставку программного обеспечения.
Основные компетенции. Он должен: 1) быть компетентным в предметной области, в которой выполняется подтверждение соответствия; 2) иметь опыт работы с параметрами безопасности приложений предметной области; 3) быть компетентным в различных подходах/методологиях подтверждения соответствия и быть в состоянии идентифицировать самый подходящий метод или комбинацию методов для данного контекста; 4) быть способным определять типы доказательства подтверждения соответствия из данных спецификаций, учитывая предполагаемое применение; 5) быть способным объединять различные источники и типы доказательств и синтезировать общее представление об их пригодности для цели или условий и ограничений приложения; 6) иметь способность к аналитическому мышлению и хорошие навыки наблюдения; 7) понимать и объективно воспринимать все программное обеспечение, включая понимание окружения приложения; 8) понимать требования настоящего стандарта.

Таблица В.8 — Ролевая спецификация оценщика

Роль. Оценщик
Обязанности: 1) развивает системное понимание программного обеспечения в предполагаемом окружении приложения; 2) разрабатывает план оценки и взаимодействует с уполномоченным органом по безопасности и организацией заказчика (договорной отдел оценщика); 3) оценивает процесс программы и разработанное программное обеспечение на соответствие требований настоящего стандарта, учитывая заданный УПБ; 4) оценивает компетентность проектной группы и организации по разработке программного обеспечения; 5) оценивает действия по проверке и подтверждению соответствия и поддерживающее обоснование; 6) оценивает системы управления качеством, адаптированные для разработки программного обеспечения; 7) оценивает систему управления конфигурацией и управления изменениями и доказательство ее использования и применения; 8) идентифицирует и оценивает в терминах риска (влияние) любые отклонения от требований к программному обеспечению в отчете по результатам оценки; 9) гарантирует, что план оценки реализован; 10) выполняет аудиты безопасности и процедуры контроля для всего процесса разработки при необходимости на различных стадиях разработки; 11) высказывает профессиональное мнение о пригодности разработанного программного обеспечения для его надлежащего использования, детализируя любые ограничения, условия применения и соображения по управлению риском при необходимости; 12) разрабатывает отчет по результатам оценки и ведет формуляр процесса оценки.
Основные компетенции. Он должен: 1) быть компетентным в технологиях/предметной области, где выполняется оценка; 2) иметь разрешение/лицензию от признанного уполномоченного органа по безопасности; 3) иметь/стремиться постоянно повышать свой уровень в области реализации принципов обеспечения безопасности и применять эти принципы для приложений данной предметной области; 4) быть компетентным проверить, были ли применены подходящий метод или комбинация методов в данном контексте; 5) быть компетентным в понимании соответствующей безопасности, человеческих ресурсов, технических процессов и процессов управления качеством при выполнении требований настоящего стандарта; 6) быть компетентным в подходах/методологиях оценки; 7) иметь способность к аналитическому мышлению и хорошие навыки наблюдения; 8) быть способным объединять различные источники и типы доказательств и синтезировать общее представление об их пригодности для цели или условий и ограничений приложения; 9) иметь полное понимание и перспективу программного обеспечения, включая понимание окружения приложения; 10) быть в состоянии оценить соответствие всех процессов разработки (таких, как процессы управления качеством, управления конфигурацией, подтверждения соответствия и проверки); 11) понимать требования настоящего стандарта.

Таблица В.9 — Ролевая спецификация менеджера проекта

Роль. Менеджер проекта
Обязанности: 1) гарантирует, что система управления качеством и независимость ролей согласно 5.1 реализованы для проекта и проект выполняется в соответствии с планами; 2) выделяет достаточное число компетентных ресурсов для проекта, чтобы выполнить важные задачи, включая действия по обеспечению безопасности, учитывая необходимую независимость ролей; 3) гарантирует, что для выполнения проекта был назначен подходящий менеджер по подтверждению соответствия, как определено в настоящем стандарте; 4) является ответственным за доставку и развертывание программного обеспечения и гарантирует, что требования безопасности от заинтересованных сторон также выполнены и реализованы в поставке; 5) выделяет достаточное количество времени для надлежащей реализации и выполнения задач обеспечения безопасности; 6) подтверждает частичную и полную безопасность поставляемых продуктов на основании процесса разработки; 7) гарантирует, что связанное с безопасностью принятие решений сопровождают необходимые журналы и прослеживаемость.
Основные компетенции. Он должен: 1) понимать требования настоящего стандарта к качеству, компетенциям, организации и управлению; 2) понимать требования процесса безопасности; 3) быть в состоянии оценить различные варианты и понимать, что влияет на эффективность защиты решения или выбранных вариантов; 4) понимать требования процесса разработки программного обеспечения; 5) понимать требования других соответствующих стандартов.

Таблица В.10 — Ролевая спецификация менеджера конфигураций

Роль. Менеджер конфигураций
Обязанности: 1) является ответственным за план управления конфигурацией программного обеспечения; 2) является владельцем системы управления конфигурацией; 3) устанавливает, что все компоненты программного обеспечения четко определены и имеют независимые имеющие версии в системе управления конфигурацией; 4) готовит информацию о версии, которая включает несовместимые версии компонентов программного обеспечения.
Основные компетенции. Он должен: 1) быть компетентным в управлении конфигурацией программного обеспечения; 2) понимать требования настоящего стандарта.

Таблица В.11 — Ролевая спецификация менеджера контроля качества

Роль. Менеджер контроля качества
Обязанности: 1) является ответственным за план обеспечению качества программного обеспечения; 2) в данном проекте применяет систему управления качеством; 3) гарантирует, что может быть установлен контрольный журнал, чтобы эффективно предпринимались действия по проверке и подтверждению соответствия; 4) определяет цели качества в сотрудничестве с руководителем проекта; 5) управляет процессом обеспечения качества программного обеспечения, как определено в плане обеспечения качества программного обеспечения; 6) разрабатывает отчет по обеспечению качества программного обеспечения, утверждая результаты действий по обеспечению качества, как запланировано в плане обеспечения качества программного обеспечения. Примечание — Отчет по обеспечению качества программного обеспечения включен в отчет управления качеством, определенный в МЭК 62425.
Основные компетенции. Он должен: 1) быть компетентным в управлении качеством программного обеспечения; 2) понимать в управлении качеством в данном проекте; 3) быть компетентным в адаптации управления качеством в данном проекте; 4) быть компетентным в действиях обеспечения качества программного обеспечения, как требуется в ИСО 9001, ИСО/МЭК 90003, ИСО/МЭК 25010 и настоящем стандарте; 5) быть компетентным в системе управления качеством компании; 6) быть компетентным в методах улучшения процесса создания и эксплуатации программного обеспечения; 7) иметь общее понимание об управлении конфигурацией программного обеспечения, управлении документами и прослеживаемости, регистрации данных и анализе; 8) быть компетентным в выполнении аудитов; 9) иметь общее понимание о разработке программного обеспечения; 10) иметь общее понимание об управлении проектами; 11) иметь способность к аналитическому мышлению и хорошие навыки наблюдения; 12) иметь хорошие коммуникационные возможности; 13) понимать требования настоящего стандарта.

Таблица В.12 — Ролевая спецификация рецензента

Роль. Рецензент
Обязанности: 1) выполняет экспертизу выходного документа согласно определенным критериям под контролем специалиста по проверке; 2) обеспечивает запись результатов экспертизы, такую как те, которые касаются 7.2.4.22; 3) не может быть тем же человеком, который разработал документ.
Основные компетенции. Он должен: 1) быть компетентным в предметной области, где выполняется экспертиза, например, требования к программному обеспечению, проект и тестирование; 2) быть компетентным в выполнении целей, которые выбраны для экспертизы.

Приложение С
(справочное)

Контроль прохождения технической документации

Таблица С.1 — Контроль прохождения технической документации

Стадия	Документ	Подписывает	1-я проверка	2-я проверка
Планирование	1 План обеспечения качества программного обеспечения	QAM	VER	VAL
	2 Отчет о проверке обеспечения качества программного обеспечения	VER		VAL
	3 План управления конфигурацией программного обеспечения	CGM	VER	VAL
	4 План проверки программного обеспечения	VER		VAL
	5 План подтверждения соответствия программного обеспечения	VAL	VER	
Требования к программному обеспечению	6 Спецификация требований к программному обеспечению	RQM	VER	VAL
	7 Спецификация тестирования всего программного обеспечения	TST	VER	VAL
	8 Очет о проверке требований к программному обеспечению	VER		VAL
Архитектура и проект программного обеспечения	9 Спецификация архитектуры программного обеспечения	DES	VER	VAL
	10 Спецификация проекта программного обеспечения	DES	VER	VAL
	11 Спецификации интерфейса программного обеспечения	DES	VER	VAL
	12 Спецификация тестирования интеграции программного обеспечения	INT	VER	VAL
	13 Спецификация тестирования интеграции программного обеспечения/аппаратных средств	INT	VER	VAL
	14 Отчет об архитектуре программного обеспечения и о проверке проекта	VER		VAL
Проект компонента программного обеспечения	15 Спецификация проекта компонента программного обеспечения	DES	VER	VAL
	16 Спецификация тестирования компонента программного обеспечения	TST	VER	VAL
	17 Отчет о проверке проекта компонента программного обеспечения	VER		
Реализация и тестирование компонента	18 Исходный код программного обеспечения и сопроводительная документация	IMP	VER	VAL
	19 Отчет об испытаниях компонента программного обеспечения	TST	VER	VAL
	20 Отчет о проверке исходного кода программного обеспечения	VER		VAL
Интеграция	21 Отчет об испытаниях интеграции программного обеспечения	INT	VER	VAL
	22 Отчет об испытаниях интеграции программного обеспечения/аппаратных средств	INT	VER	VAL
	23 Отчет о проверке интеграции программного обеспечения	VER		

Окончание таблицы С.1

Стадия	Документ	Подписывает	1-я проверка	2-я проверка
Тестирование всего программного обеспечения/заключительное подтверждение соответствия	24 Отчет об испытаниях всего программного обеспечения	TST	VER	VAL
	25 Отчет о проверке испытаний всего программного обеспечения	VER		VAL
	26 Отчет о подтверждении соответствия программного обеспечения	VAL	VER	
	27 Отчет о подтверждении соответствия инструментальных средств	a)	VER	
	28 Отчет об обеспечении качества программного обеспечения	QAM	VER	
	29 Информация о версии	CGM	VER	VAL
Системы, сконфигурированные прикладными данными или алгоритмами	30 Спецификация требований к приложению	RQM	VER	VAL
	31 План подготовки приложения	RQM или DES	VER	VAL
	32 Спецификация тестирования приложения	TST	VER	VAL
	33 Архитектура и проект приложения	DES	VER	VAL
	34 Отчет о проверке подготовки приложения	VER		
	35 Отчет об испытаниях приложения	TST	VER	VAL
	36 Исходный код данных/алгоритмов приложения	DES	VER	VAL
	37 Отчет о проверке данных/алгоритмов приложения	VER		VAL
Развертывание программного обеспечения	38 План выпуска и развертывания программного обеспечения	a)	VER	VAL
	39 Руководство по развертыванию программного обеспечения	a)	VER	VAL
	40 Информация о версии	DES	VER	VAL
	41 Журнал развертывания	a)	VER	VAL
	42 Отчет о проверке развертывания	VER		
Сопровождение программного обеспечения	43 План сопровождения программного обеспечения	a)	VER	VAL
	44 Журнал изменений программного обеспечения	a)	VER	VAL
	45 Журнал сопровождения программного обеспечения	a)	VER	VAL
	46 Отчет о проверке сопровождения программного обеспечения	VER		VAL
Оценивание программного обеспечения	47 План оценки программного обеспечения	ASR		
	48 Отчет о результатах оценки программного обеспечения	ASR		

a) Никакая конкретная роль не определена.

Примечание — Отчет об обеспечении качества программного обеспечения включен в отчет по управлению качеством, определенный в МЭК 62425.

Приложение D
(справочное)

Цель и описание методов

D.1 Исправление сбоев методами искусственного интеллекта

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. Определенное прогнозирование (вычисление тенденций), исправление ошибок, обслуживание и контролирующие действия могут достаточно эффективно поддерживаться системами, основанными на методах искусственного интеллекта. 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 безопасности и надежности.

D.2 Анализируемые программы

Цель. Проектировать программу так, чтобы ее анализ был легко выполним. Поведение программы должно быть полностью тестируемым на основе анализа.

Описание. Необходимо создавать программы, которые легко проанализировать, используя методы статического анализа. Чтобы этого достигнуть, необходимо следовать правилам структурного программирования, например:

- поток команд управления компонента должен быть составлен из структурированных конструкций, таких как последовательностей, итераций и ветвлений;
- компоненты должны быть небольшими;
- число возможных путей через компонент небольшое;
- отдельные части программы должны быть разработаны так, чтобы они были не связаны по управлению в максимально возможной степени;
- отношение между параметрами ввода и вывода должно быть максимально простым;
- вычисления условий для выполнения операторов цикла и ветвления должны быть простыми;
- условия операторов цикла и ветвления должны быть просто связаны с входными параметрами компонента;
- границы между различными типами преобразований должны быть простыми.

D.3 Проверка на критические нагрузки/стресс-тестирование

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.

D.4 Анализ граничных значений

Цель. Устранение ошибок в программном обеспечении при предельных и граничных значениях параметров.

Описание. Предметная входная область программы делится на множество входных классов. 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;
- нераспечатываемые управляющие символы;
- пустой стек или элемент списка;
- нулевая 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.

D.5 Восстановление предыдущего состояния

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.

D.6 Причинно-следственные диаграммы

Цель. Моделирование последовательности событий 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, причинно-следственный граф просматривается в прямом и обратном направлениях. 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 распространения 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 вычисления вероятности появления определенных критических последовательностей.

D.7 Таблица контрольных проверок

Цель. Создать стимул для критической оценки всех аспектов системы, а не установить определенные требования.

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.

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.

D.8 Анализ потоков управления

Цель. Обнаружение низкокачественных и потенциально некорректных структур программ.

Описание. Анализ потока управления выявляет подозреваемые области программы, которые не соответ-

ствуют 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;
- запутанного кода, который является хорошо структурированным кодом, имеющим управляющий граф, который допускает сокращение путем его последовательного сокращения до одного узла. В отличие от этого плохо структурированный код может быть сокращен только до группы, состоящей из нескольких узлов.

D.9 Анализ отказов по общей причине

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. Компьютерные системы, ориентированные на безопасность объекта, часто используют избыточность аппаратных средств и мажоритарный принцип голосования. Этот подход исключает случайные отказы в компонентах аппаратных средств, которые могут помешать корректной обработке данных в компьютерных системах.

Однако некоторые отказы могут оказаться общими для нескольких компонентов. Например, если компьютерная система установлена в одном помещении, то недостатки вентиляции могут снизить преимущества избыточности. Это может оказаться верным и для других внешних влияний на компьютерную систему (например пожар, затопление, электромагнитные влияния, трещины в панелях и землетрясение). Компьютерная система может быть также подвержена воздействиям, относящимся к ее функционированию и поддержке. Поэтому важно, чтобы в рабочих инструкциях были предусмотрены адекватные и хорошо задокументированные процедуры по функционированию и поддержке системы, а обслуживающий персонал был хорошо обучен.

Внутренние причины также вносят большой вклад в отказы по общей причине. 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 в разных каналах избыточных компьютерных систем используются разные программы, то 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.

D.10 Анализ потока данных

Цель. Обнаружение низкокачественных и потенциально некорректных структур программ.

Описание. Анализ потока данных объединяет 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. Вполне вероятно, что это является ошибкой, и конечно считается плохой практикой программирования;
- 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.

Существует расширение анализа потока данных известное, как анализ потока информации, где фактические потоки данных (и внутри и между процедурами) сравниваются с целью проекта. Обычно это выполняется с компьютеризированным инструментальным средством, где предполагаемые потоки данных определены, используя структурированный комментарий, который может быть прочитан этим инструментальным средством.

D.11 Диаграммы потоков данных

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.

Основными компонентами диаграммы потока данных являются:

- функции, представленные кружками;
- потоки данных, представленные стрелами;
- накопители данных, представленные открытыми блоками;
- ввод/вывод, представленный специальными видами блоков.

Диаграммы потока данных описывают, как входные данные преобразуются в выходные. Они не включают в себя и не должны включать управляющую информацию или информацию о последовательности процесса. Каждый 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 диаграмм потока данных является то, что они показывают преобразования, не предполагая, как они реализуются.

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.

D.12 Регистрация и анализ данных

Цель. Облегчить совершенствование процесса создания программного обеспечения документируя, выполняя подтверждение соответствия и анализируя соответствующие данные из отдельных проектов и полученные у специалистов. Соответствие данных определено стратегическими целями организации. Цели могут быть связаны с оценкой отдельного метода разработки программного обеспечения относительно требований к данному методу, например, относительно эффективности предотвращения дефекта.

Описание. Документирование и анализ данных составляют основную часть совершенствования процесса программного обеспечения. Документирование данных, для которых выполнено подтверждение соответствия, представляет важную часть более глубокого знакомства с процессом разработки программного обеспечения и оценки альтернативных методов разработки программного обеспечения.

В процессе проектирования формируются подробные документы, как на проект в целом, так и отдельные документы. Например, инженер обязан вести документацию, которая может включать:

- ресурсы, затраченные на отдельные компоненты;
- тестирование, выполненное для каждого компонента;
- решения и их разумные обоснования;
- достижение основных этапов проекта;
- 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.

Из-за плохого планирования документы часто имеют тенденцию быть слишком объемными и не вполне конкретными. Этого можно избежать, следуя принципу о том, что документирование данных должно отвечать целям, вопросам и метрикам, относящимся к тому, что стратегически важно для организации.

Для достижения желаемой точности, процесс документирования данных и их подтверждения соответствия должен продолжиться одновременно с разработкой, например, как часть процесса управления конфигурацией.

D.13 Таблицы решений (таблицы истинности)

Цель. Обеспечение ясных и согласующихся спецификаций и анализа сложных логических комбинаций и их отношений.

Описание. Данные методы использует бинарные таблицы для точного описания логических отношений между булевыми переменными программы.

Использование таблиц и точность методов позволили применить их в качестве средства анализа сложных логических комбинаций, выраженных в бинарных кодах.

Оба метода потенциально выполнимы, если используются в качестве средства спецификации систем.

D.14 Программирование с защитой

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. Встроенные методы защиты от ошибок проектируются в программном обеспечении для преодоления недостатков в процессе создания этих программных средств. Эти недостатки могут быть обусловлены очевидными ошибками при проектировании или кодировании либо ошибочными требованиями. Ниже перечислены некоторые из методов защиты:

- 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 в терминах программных функций, так и в терминах 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 проектируются 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.

D.15 Стандарты кодирования и руководство по стилю

Цель. Гарантировать стандартную структуру проектной документации и произведенного кода, осуществлять непротиворечивое программирование и выполнять стандартный метод разработки, который избегает ошибок.

Описание. Стандарты кодирования — это правила и ограничения на данный язык программирования, применение которых помогает избежать возможные сбои, которые могут быть сформированы при использовании этого языка.

Содержание стандарта кодирования может включать:

- обоснование языка;
- область применения и базовый стандарт, если доступен.

Примечание — Для предметно-ориентированных языков базовые стандарты могут быть не доступными;

- процедура по изменению стандарта кодирования;
- анализ возможных сбоев и рекомендуемая их обработка;
- ограничения, чтобы избежать сбои;
- мобильность.

Инструкции по стилю рассматривают такие вопросы, как форматирование и соглашения о присвоении имен и хотя это может быть очень субъективно, больше, чем что-либо, стиль влияет на удобочитаемость создаваемого кода. Установление общего и согласованного стиля для проекта упростит понимание и сопровождение кода, разработанного более чем одним программистом, и упростит сотрудничество нескольких специалистов, участвующих в разработке одной и той же программы.

D.16 Многовариантное программирование

Цель. Обнаружение и наложение маски при выполнении программ на не выявленные на этапах проектирования 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.

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 соответствия (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 соответствие, 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, однако он является средством для обнаружения и маскирования ошибок, прежде чем они смогут повлиять на безопасность.

К сожалению, эксперименты и аналитические исследования показывают, что *N*-вариантное программирование не всегда столь эффективно, как хотелось бы. Даже если используются различные алгоритмы, то разнообразные версии программного обеспечения слишком часто перестают работать на одинаковых входах.

Двумя альтернативами *N*-вариантному программированию является разнообразие проекта и функциональное разнообразие. Разнообразие проекта включает использование множества компонентов, каждый из которых разработан отличным от других способом, но реализует ту же функцию. Функциональное разнообразие включает решение одной и той же проблемы функционально различными способами. Независимо от подхода в настоящее время нет эффективного метода, оценивающего уровень разнообразия.

D.17 Динамическая реконфигурация

Цель. Обеспечение функциональности системы, несмотря на внутренний сбой.

Описание. Логическая архитектура системы должна быть такой, чтобы она могла быть отображена 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.

Несмотря на то, что данный метод традиционно применялся к аппаратным средствам, он разрабатывается для применения к программному обеспечению и, таким образом, к полной системе. Данный метод должен рассматриваться на первом этапе проектирования системы.

D.18 Классы эквивалентности и тестирование разбиением входных данных

Цель. Адекватное тестирование программных средств с использованием минимума тестируемых данных. Тестируемые данные образуются путем выбора разбиений 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 с целью 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.

D.19 Коды обнаружения и исправления ошибок

Цель. Обнаружение и исправление ошибок в чувствительной к ним информации.

Описание. Для информации, состоящей из *n* битов, генерируется закодированный блок из *k* битов, который позволяет обнаруживать и исправлять *r* ошибок. Примерами кодов могут быть:

- коды Хэмминга;
- циклические коды;
- полиномиальные коды;
- хэш-коды;
- криптографические коды.

D.20 Предположение ошибок

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.

D.21 Введение ошибок

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.

Если используется подсеивание ошибок, то расположение всех ошибок должно быть зарегистрировано, и специалист по подтверждению соответствия должен гарантировать, что все подсеянные ошибки были удалены перед выпуском программного обеспечения.

D.22 Анализ дерева событий

Цель. Моделирование с помощью диаграмм последовательности событий, которая может возникнуть в системе после появления инициализирующего события и указать на возможные опасные последствия.

Описание. В верхней части диаграммы записывается последовательность условий, относящихся к формированию последовательности событий, следующих за инициализирующим событием, являющегося целью анализа. Начиная с инициализирующего события 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. Дерево событий может быть использовано для вычисления вероятностей различных последовательностей, основываясь на значениях вероятностей условий и их числе в последовательности.

D.23 Инспекция программ по Фагану

Цель. Выявить ошибки на всех стадиях разработки программы.

Описание. «Формальный» аудит документов обеспечения качества нацелен на нахождение ошибок и пропусков. Инспекционный процесс состоит из пяти стадий; планирование, подготовка, инспекция, доработка и контроль результатов. У каждой из этих стадий существует своя собственная отдельная цель. Разработка всей системы (спецификация, проект, кодирование и тестирование) должна быть проинспектирована.

D.24 Программирование с проверкой ошибок

Цель. Обнаружение ошибок, оставшихся при проектировании программных средств, в процессе выполнения программ с целью 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>;

D.25 Анализ влияния ошибок программного обеспечения (SEEA)

Цель. Идентифицировать компоненты программного обеспечения, их критичность; предложить средства для обнаружения ошибок программного обеспечения и улучшения устойчивости программного обеспечения; оценить объем усилий по подтверждению соответствия, необходимый для различных компонентов программного обеспечения.

Описание. Анализ включает три фазы.

- Идентификация жизненно важных компонентов программного обеспечения.

Определение глубины анализа (на уровне отдельной командной строки, группы команд, компонента, и т. д.), необходимого для каждого компонента программного обеспечения из его спецификации.

- Анализ ошибок программного обеспечения.

Результатом этой фазы является таблица, содержащая следующую информацию:

- имя компонента,
- рассматриваемую ошибку,
- последствия ошибки на уровне модуля,
- последствия на уровне системы,
- нарушенный критерий безопасности,
- критичность ошибки,
- предложенные средства обнаружения ошибки,
- нарушенный критерий, если средства обнаружения реализованы,
- остаточная критичность, если средства обнаружения реализованы.

- Синтез.

Синтез идентифицирует остающиеся небезопасные сценарии и усилия по подтверждению соответствия, необходимые для данной критичности каждого модуля.

SEEA, будучи всесторонним анализом, выполненным независимой командой, является мощным методом нахождения серьезных программных ошибок.

D.26 Обнаружение и диагностика сбоев

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 том, чтобы предотвратить появление неверных результатов. Система, действующая 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) 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).

D.27 Конечные автоматы/диаграммы переходов

Цель. Определить или реализовать структуры управления системы.

Описание. Многие системы могут быть описаны в терминах их состояний, входов и действий. Таким образом, находясь в состоянии S1 и при получении входа I, система может выполнить действие A и перейти в состояние S2. 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.

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) и достижимость (возможно или нет перейти из одного состояния в другое с помощью некоторой последовательности входов). Эти свойства являются важными для критических систем, так как легко разработать инструменты для обеспечения таких проверок. Существуют также алгоритмы, позволяющие автоматически генерировать тестовые примеры для верификации реализаций конечных автоматов или анимации модели конечного автомата.

Для улучшения описания поведения сложной системы были разработаны несколько расширений конечных автоматов. К так называемым диаграммам состояний добавляют иерархию, композицию (параллелизм), межуровневые переходы, историю состояний, и т. д. Особенно полезная функция — вложенность внутренних состояний и переходов, что дает возможность показать или скрыть внутренние состояния, если требуется. Диаграммы состояний являются частью UML (унифицированный язык моделирования), поэтому поддерживаются многими коммерческими инструментальными средствами.

D.28 Формальные методы

D.28.1 Общие положения

Цель. «Формальные методы» относятся к математически строгим методам и инструментальным средствам для спецификации, проектирования и проверки систем программного обеспечения и аппаратных средств.

Описание. «Математически строгий» означает, что спецификации, используемые в формальных методах, являются правильно построенными операторами в математической логике, а формальные проверки — строгие выводы в этой логике (т. е. каждый шаг выполняется по правилам вывода и, следовательно, может быть проверен с помощью механического процесса). Важность формальных методов в том, что они обеспечивают средства символьного исследования всего пространства состояний цифрового проекта (или аппаратных средств или программного обеспечения) и установливают свойство правильности или безопасности, которое является истиной для всех возможных входов. Однако в настоящее время это редко реализуется на практике (за исключением критических компонентов критических систем безопасности) из-за большой сложности реальных систем.

Используется несколько подходов, чтобы преодолеть астрономический размер пространства состояний, связанного с реальными системами:

- применяются формальные методы к требованиям и проектам высокого уровня, где от большинства деталей выполнено абстрагирование;
- применяются формальные методы только к самым критическим компонентам;
- анализируются модели программного обеспечения и аппаратных средств, в которых значения переменных делают дискретными, а их диапазоны существенно уменьшают;
- анализируются модели систем иерархическим способом, который реализует принцип «разделяй и властвуй»;
- насколько это возможно, максимально автоматизируются проверки.

Несмотря на то, что математическая логика используется во всех формальных методах, нет никакого единственного лучшего «формального метода». Каждая прикладная предметная область требует различных методов моделирования и различных подходов доказательства. Кроме того, даже в определенной прикладной предметной области, различные стадии жизненного цикла могут быть лучше всего выполнены с использованием других инструментальных средств и методов. Например, программа автоматического доказательства теоремы могла бы лучше всего использоваться для анализа правильности описания на уровне межрегистровых пересылок схемы быстрого преобразования Фурье, тогда как алгебраические методы вывода могли бы лучше всего использоваться для анализа правильности усовершенствований проекта, представленного на уровне логических элементов. Поэтому во всем мире разработано большое количество формальных методов.

В следующих пунктах описаны несколько примеров формальных методов. Список примеров здесь не исчерпывающий. Описаны формальные методы: CSP, CCS, HOL, LOTOS, OBJ, временная логика, VDM, Z Метод, B Метод и метод проверки моделей.

D.28.2 Взаимодействующие последовательные процессы (CSP)

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

Описание. Метод CSP обеспечивает язык для спецификации процессов системы и доказательства соответствия реализации процессов их спецификациям (описанным как трасса, то есть в виде допустимых последовательностей событий).

Система моделируется в виде сети независимых процессов. Каждый независимый процесс описывается в терминах всех его возможных поведений. Моделируемая система состоит из последовательных или параллельных процессов. 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[1)], а язык OCCAM[2)] позволил непосредственно реализовывать на сетях транспьютеров системы, специфицированных в языке CSP.

D.28.3 Расчет взаимодействующих систем (CCS)

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

Описание. Аналогично CSP, CCS также является математическим аппаратом, описывающим 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 по-

---

[1)] INMOS была британской полупроводниковой компанией, которая произвела в 80-х инновационную архитектуру микропроцессора, предназначенную для параллельной обработки, названной транспьютером. Позже INMOS стала частью SGS-Thomson, затем STMicroelectronics.

[2)] OCCAM — параллельный язык программирования, который называют в честь Уильяма из Ockham, известного из-за Бритвы Оккама. Это — собственный язык программирования транспьютера INMOS.

шаговое 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.

D.28.4 Логика высшего порядка (HOL)

Цель. Формальный язык, предназначенный в качестве основы для спецификации и верификации 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).

D.28.5 LOTOS

Цель. LOTOS является средством для описания и анализа 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.

D.28.6 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[3)], 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.

D.28.7 Временная 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 временные интервалы и ограничения во временной логике явно не обрабатываются. Абсолютное время обрабатывается путем образования дополнительных временных состояний, что является частью описания состояния.

D.28.8 Метод разработки Vienna (VDM)

Цель. Систематическая спецификация и реализация последовательных 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 пост-условий в терминах системных состояний. Операции могут проверяться на сохранение системных инвариантов.

[3)] Ада — структурированный, статически типизированный, императивный, с широким спектром применения, объектно-ориентированный язык программирования высокого уровня, расширенный от Паскаля и других языков.

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

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.

D.28.9 Z

Цель. Z — это нотация языка спецификаций для последовательных систем и метод проектирования, позволяющий проектировщику 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 пост-условий в терминах системных состояний. Операции допускается проверять на сохранение системных инвариантов для демонстрации их согласованности. Формальная часть спецификации разделяется на схемы, которые обеспечивают возможность структурирования спецификаций путем их уточнения.

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.

D.28.10 Метод B

Цель. Как и VDM, цель метода B состоит в том, чтобы формально промоделировать систему или программное обеспечение и доказать, что поведение системы или программного обеспечения соответствует свойствам, которые были выявлены во время моделирования.

Описание. Моделирование в методе B использует математические элементы из теории множеств. С одной стороны, инварианты (т. е. предикаты) определяют статические свойства модели. С другой стороны, операции устанавливают постусловия, определяя, таким образом, ее динамическое поведение. Существует возможность спецификации сложной системы или программного обеспечения, декомпозируя модель в «машины», связанные ссылками с различной семантикой.

Различают две основные категории моделирования с B формализмом:

- первый (исторически, первый) стремится разрабатывать программное обеспечение. В этом случае цель состоит в том, чтобы создать программу, которая отвечает ее спецификации. Модель состоит из абстрактных машин (не обязательно детерминированных) и последовательное уточнение этих машин, приводит к детерминированным реализациям, записанным в псевдокоде, названном «B0». Этот псевдокод затем может быть автоматически транслирован в целевой язык программирования;

- последний стремится моделировать системы и в этом случае мы говорим о «Событии B». Цель состоит в том, чтобы однозначно и связно определить систему, которая выполняет явно определенные свойства. Модель учитывает саму систему и ее окружение. Динамика системы моделируется с помощью «событий», а метод уточнения используется для уточнения взаимодействий между системой и ее окружением.

Автоматически генерируется набор доказанных обязательных свойств (логических утверждений, которые должны быть формально доказаны из гипотез, которые были получены из формальной модели B). Эти доказанные обязательные свойства гарантируют:

- существование данных, которые реализуют статические и динамические свойства модели;
- что операции (динамическое поведение модели) соответствуют инварианту;
- что уточнение данных и операции (и псевдокод B0, если необходимо) не противоречит спецификации, записанной в абстрактных машинах;
- что каждая операция вызывается в контексте ее предусловия;
- что в случае моделирования программного обеспечения программа действительно завершается (в частности, каждый цикл завершается).

Также генерируются другие доказанные обязательные свойства, например проверка целочисленного переполнения или потери разрядов.

D.28.11 Проверка модели

Цель. Для заданной модели системы автоматически проверить, удовлетворяет ли эта модель заданной спецификации.

Описание. Проверка модели — это процесс проверки, определяющий является ли данная структура моделью заданной логической формулы. Это общая концепция и применяется ко всем типам логик и подходящих структур. Простой задачей проверки модели является определение, является ли данная структура моделью заданной логической формулы в логике высказываний.

Для алгоритмической проверки формальных систем был разработан важный класс методов проверки модели. Это достигается проверкой, удовлетворяет ли структура, часто получаемая из проекта аппаратных средств или программного обеспечения, формальной спецификации, обычно представленной формулой во временной логике.

Проверка модели чаще всего применяется в проектах аппаратных средств. Для программного обеспечения из-за неразрешимости (см. теорию вычислимости) этот подход не может быть полностью алгоритмизован; обычно данный подход не может доказать или опровергнуть заданное свойство.

Структура обычно задается в виде описания в исходном коде языка описания промышленных аппаратных средств или языка специального назначения. Такая программа соответствует конечному автомату, т. е. направленному графу, состоящему из узлов (или вершин) и дуг. С каждым узлом связан набор атомарных высказываний, обычно устанавливающих, какие элементы памяти с ним связаны. Узлы представляют состояния системы, дуги представляют возможные переходы, которые могут изменить состояние, а атомарные высказывания представляют основные свойства, которые характеризуют точку выполнения.

Формально, проблема может быть представлена следующим образом: задается желаемое свойство, выраженное в виде формулы p во временной логике, и структура M с начальным состоянием s, и если M имеет конечное значение, как это выполняется для аппаратных средств, то проверка модели сводится к поиску на графе.

D.29 Формальное доказательство

Цель. Используя теоретические и математические модели и правила возможно доказать правильность программы или модели, не выполняя ее.

Описание. Ряд операторов включаются в различные места программы и они используются в качестве пред- и постусловий для различных путей в программе. Доказательство заключается в демонстрации того, что программа преобразует предусловия в постусловия согласно ряду логических правил, и что программа завершается.

В настоящем приложении описано несколько формальных методов, например, CCS, CSP, HOL, LOTOS, OBJ, Временная логика, VDM и Z. Их описания можно найти в D.28.

D.30 Прямое исправление

Цель. Обеспечить корректное функционирование в присутствии одного или более сбоев.

Описание. Если был обнаружен сбой, то текущее состояние системы обрабатывается, чтобы получить состояние, которое будет непротиворечивым спустя некоторое время. Эта концепция особенно подходит для систем реального времени с небольшой базой данных и с высокой скоростью изменения внутреннего состояния. Предполагается, что, по крайней мере, часть состояния системы может быть окружением, и только часть состояний системы находится под влияние (принуждением) окружения.

D.31 Постепенное отключение функций

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.

Другим примером была бы система сигнализации, где в случае потери связи с центром управления локальное линейное путевое оборудование автоматически устанавливает доступные маршруты для направления с трафиком наивысшего приоритета. Будет происходить постепенное отключение функций, потому что поезда приоритетных маршрутов будут в состоянии пройти через область, где произошла потеря связи с центром управления, но другие движения, такие как маневровое передвижение, будет невозможно.

D.32 Анализ влияния

Цель. Определение влияния изменения или расширения в программном обеспечении, которое оказывается на другие программные модули этого программного обеспечения, а также на другие системы.

Описание. Перед выполнением модификации или расширения программного обеспечения следует выполнить анализ, чтобы определить влияние модификации или расширения 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 измененных программных модулей;
- повторная проверка всех подвергнувшихся воздействию программных модулей;
- повторная проверка всей системы.

D.33 Ограничение доступа/инкапсуляция информации

Цель. Увеличение устойчивости и пригодности для обслуживания программного обеспечения.

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 для 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.

Эта концепция абстрактных типов данных непосредственно поддерживается в ряде языков программирования, но основной принцип может быть применен для любого используемого языка программирования.

D.34 Тестирование интерфейса

Цель. Продемонстрировать, что интерфейсы подпрограмм не содержат ошибок или любых ошибок, которые приводят к отказам в определенном приложении программного обеспечения, или обнаружить все ошибки, которые могут быть в интерфейсах подпрограмм.

Описание. Возможны несколько уровней детализации или полноты тестирования. К наиболее важным уровням относится тестирование:

- всех переменных интерфейса с их предельными значениями;
- всех переменных интерфейса по отдельности с их предельными значениями с переменными других интерфейсов с их нормальными значениями;
- всех значений предметной области каждой переменной интерфейса с другими переменными интерфейса с их нормальными значениями;
- всех значений всех переменных в разных комбинациях (возможно только для небольших интерфейсов);
- при заданных условиях тестирования, относящихся к каждому вызову каждой подпрограммы.

Эти тесты особенно важны, если интерфейсы не содержат операторов, обнаруживающих неправильные значения параметров. Такие тестирования также важны при генерации новых конфигураций ранее существовавших подпрограмм.

D.35 Подмножество языка

Цель. Снижение вероятности внесения программных ошибок и повышение вероятности обнаружения любых оставшихся ошибок.

Описание. Язык исследуется для определения программных конструкций, подверженных ошибкам либо сложных для анализа, например, используя методы статического анализа. После этого определяется языковое подмножество, которое исключает такие конструкции.

D.36 Сохранение информации о выполнении программы

Цель. Безопасное прекращение работы программы в случае, если она попытается выполнить неразрешенное действие.

Описание. Все соответствующие подробные сведения о каждом разрешенном выполнении программы документируются. Во время нормального режима работы каждое выполнение программы сравнивается с набором разрешенных выполнений. Если появляется расхождение, то предпринимаются действия по безопасности.

Документация о выполненных операциях может содержать последовательность индивидуальных шагов «от решения к решению» (DDpaths) или последовательность отдельных обращений к массивам, записям или томам, либо к тому и другому.

Возможны различные методы хранения сведений о последовательностях шагов выполнения программы. Чтобы отобразить последовательность выполнения на отдельное большое число или на последовательность чисел, могут использоваться методы хэш-кодирования. При нормальном режиме работы перед выполнением выходной операции значения чисел, отображающих последовательности шагов выполнения программы, должны быть сопоставлены со значениями, сохраненными в памяти.

Учитывая, что число возможных комбинаций таких последовательностей шагов от решения к решению у одной программы очень велико, полная интерпретация программы может оказаться невыполнимой. В этом случае данный метод может быть применен на уровне компонентов.

D.37 Метрики

Цель. Прогнозирование характеристик программ, исходя из свойств самих программ, а не из информации об их разработке или тестовой истории.

Описание. Данные модели оценивают некоторые структурные свойства программного обеспечения и связывают их с требуемой характеристикой, такой как сложность. Для оценки большинства мер требуются программные инструментальные средства.

Некоторые применяющиеся метрики перечислены ниже:

- графо-теоретическая сложность: эта мера может быть применена на ранних стадиях жизненного цикла, чтобы оценить компромиссные решения, и основывается на величине сложности графа управления программы, представленной ее значением цикломатической сложности;
- число способов активизации определенного компонента (доступность): чем больше к компоненту выполняется обращений, тем более качественно он должен быть отлажен;
- меры сложности Холстеда: эта мера вычисляет длину программы, считая число операторов и операндов. Она обеспечивает меру сложности и оценивает ресурсы, необходимые для разработки;
- число входов и выходов на компонент: уменьшение числа точек входа/выхода является ключевой характеристикой методов структурного проектирования и программирования.

D.38 Модульный подход

Цель. Декомпозиция программного обеспечения на небольшие законченные модули с целью сокращения сложности системы.

Описание. Модульный подход или модульность включает в себя несколько различных правил для этапов проектирования, кодирования и сопровождения проекта программного обеспечения. Эти правила меняются в соответствии с реализуемым методом проектирования. Большинство методов подчиняются следующим правилам:

- программный модуль (компонент) должен выполнять одну четко сформулированную задачу или функцию;
- связи между программными модулями (компонентами) 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 (компонент);
- 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 (компоненты) должны взаимодействовать с другими программными модулями (компонентами) через свои интерфейсы. Если используются глобальные или общие переменные, то они 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 (компонент) должны быть полностью документально оформлены;
- все интерфейсы программных модулей (компонент) должны содержать только минимальное число параметров, необходимых для их функционирования;
- обычно наиболее подходящее ограничение для числа параметров равно 5.

D.39 Моделирование реализации

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 для средних и 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 времена задержки и откликов 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.

D.40 Требования к реализации

Цель. Установить, что требования к реализации программного обеспечения были удовлетворены.

Описание. Выполняется анализ, как системы, так и спецификаций требований программного обеспечения с целью спецификации всех общих и конкретных, явных и неявных требований к функционированию.

Каждое требование к реализации анализируется по очереди для того, чтобы определить:

- критерии успешности результата, который следует получить;
- возможность получения меры критерия успешности;
- возможную точность таких результатов измерения;
- стадии проектирования, на которых эти результаты измерения могут быть оценены;
- стадии проектирования, на которых могут быть получены эти результаты измерений.

Затем анализируется целесообразность каждого требования к реализации для получения списка требований к реализации, критериев успешности результата и возможных результатов измерений. Основными целями являются:

- связь каждого требования к реализации, 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 реализации и критериев успешности результата;
- использование (по возможности) преимуществ применения одной меры для нескольких требований к реализации.

D.41 Вероятностное тестирование

Цель. Получение количественных показателей надежности исследуемой программы. Эти показатели могут быть получены с учетом относительных уровней доверия и значимости и должны иметь следующий вид:

- вероятность отказа по запросу;
- вероятность отказа в течение определенного периода времени;
- вероятность последствий ошибки.

Из этих показателей могут быть получены другие показатели, например:

- вероятность безотказной работы;
- вероятность сохранения работоспособности;
- доступность;
- MTBF или частота отказов;
- вероятность безопасного исполнения.

Описание. Вероятностные соображения основываются либо на статистических испытаниях, либо на опыте эксплуатации. Обычно количество тестовых примеров или наблюдаемых практических примеров очень велико.

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. Например, общее управление тестированием (систематические ошибки) 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 подробными 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.

D.42 Моделирование процесса

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.

D.43 Макетирование/анимация

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 результате этой оценки.

D.44 Блок восстановления

Цель: Повышение вероятности выполнения программой, в конечном счете, своих заданных функций.

Описание: Некоторые различные разделы программы, которые часто пишутся независимо, предназначены для выполнения одной требуемой функции. Из таких разделов конструируется окончательная программа. Сначала выполняется первый раздел, называемый первичным. Далее происходит тестирование его результатов. Если проверка проходит успешно, результат принимается и передается следующим разделам программы. Если проверка дает отрицательный результат, то все побочные эффекты первого сбрасываются и выполняется второй раздел, называемый первой альтернативой. Далее следует тестирование второго раздела, которое выполняется аналогично первому. При необходимости могут быть предусмотрены вторая, третья и т. д. альтернативы.

D.45 Ограничения на время ответа и объем памяти

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. Выполняется анализ для определения распределения запросов при средних и наихудших условиях. Такой анализ 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.

D.46 Повторный запуск механизмов восстановления после ошибок

Цель. Пытаться обеспечить функциональное восстановление в условиях обнаруженного сбоя с помощью повторного запуска механизмов восстановления.

Описание. В случае обнаружения сбоя или ошибочного условия предпринимаются попытки восстановления ситуации путем повторного выполнения того же кода. Восстановление с помощью повторной попытки может быть полным в виде перезагрузки и повторного запуска процедуры, либо небольшим в виде перепланирования и повторного запуска задачи после выполнения блокировки по времени программы или управляющего действия задачи. Методы повторного запуска широко используются при коммуникационных сбоях или при восстановлении после ошибок, и условия повторного запуска могут быть отделены флажками от ошибки протокола связи (контрольная сумма и т. д.) или от подтверждающего ответа блокировки по времени коммуникации.

D.47 Методы «подушки безопасности»

Цель: Защита от необнаруженных на этапах спецификации и реализации ошибок в программных средствах, которые неблагоприятно влияют на их безопасность.

Описание: Метод «подушки безопасности» представляет собой использование внешнего монитора, реализованного на независимом компьютере с другой спецификацией. Данный метод касается исключительно гарантии того, чтобы главный компьютер выполнял безопасные, не обязательно корректные, действия. «Подушка безопасности» непрерывно контролирует главный компьютер и предотвращает вхождение системы в опасное состояние. Кроме того, если обнаружится, что главный компьютер вошел в возможно опасное состояние, система должна возвратиться обратно в безопасное состояние с помощью либо «подушки безопасности», либо главного компьютера.

D.48 Управление конфигурацией программного обеспечения

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. 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 проектировщику определять, как влияет на другие поставки изменение в предыдущей поставке (особенно одного из ее элементов). В частности, системы или подсистемы могут надежно компоноваться (конфигурироваться) из согласованных наборов версий элементов.

D.49 Строго типизированные языки программирования

Цель. Снижение вероятности ошибок путем использования языка, который компилятором обеспечивает высокий уровень проверки.

Описание. Подобные языки обычно позволяют определять установленные пользователем типы данных на основе типов данных базового языка (например целое число, действительное число). Затем эти типы могут быть использованы так же, как и базовые типы, но вводятся строгие проверки, гарантирующие правильность используемого 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 хорошо структурированным программам.

Типичными примерами строго типизированных языков являются Pascal, Ada и Modula 2.

D.50 Структурное тестирование

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. Проверяемые элементы программы, в зависимости от требуемого уровня строгости, могут быть различными:

- утверждения — это наименее строгий тест, поскольку можно выполнить все закодированные утверждения без анализа обеих ветвей условного утверждения;
- ветвления — обе стороны каждого ветвления следует проверять. Это может оказаться непрактичным для некоторых типов кодов защиты;
- составные условия — анализируется каждое условие в составном условном переходе (например, связанное оператором И/ИЛИ);
- LCSAJ (последовательность линейного кода и переход) — представляет собой любую линейную последовательность закодированных утверждений, включая условные утверждения, заканчивающиеся переходом. Многие возможные подпоследовательности могут оказаться невыполнимыми благодаря ограничениям, 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;
- граф вызовов — программа, состоящая из подпрограмм, которые могут быть вызваны из других подпрограмм. Граф вызовов представляет собой дерево вызовов подпрограмм в программе. Тесты должны охватывать все вызовы в дереве;
- все последовательности — выполняются все возможные последовательности кодированием. Полное тестирование обычно неосуществимо из-за очень большого количества возможных последовательностей.

D.51 Структурные диаграммы

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 кругом, с именем данных, передаваемых в направлении элементов структурной диаграммы и обратно (обычно такая стрелка изображается параллельно линиям, соединяющим прямоугольники схемы).

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

Структурные диаграммы, полученные на основании диаграмм потоков данных, представляют первый уровень структуры системы, где каждый блок на структурной диаграмме представляет кружок на диаграмме потоков данных. Естественно, более низкие уровни могут быть описаны, используя такой же подход.

D.52 Структурная методология

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), которые определяют и позволяют документально оформлять требования и возможности реализации в логической последовательности и структурированным способом.

Описание. Существует достаточно много структурных методов. Некоторые из них, такие как SSADM, LBMS, созданы для выполнения традиционных функций обработки данных и транзакций, другие (MASCOT, JSD, метод

Йордона в реальном времени) 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).

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 окружение, 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.

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 (математически) формальные нотации (например, JSD использует регулярные выражения; метод Йордона, SOM и SDL используют 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 наличии у него большого объема, хотя и неполной информации.

D.53 Структурное программирование

Цель. Проектирование и реализация компонента программного обеспечения с использованием практического анализа компонента программного обеспечения без его выполнения. Такой анализ должен быть способен обнаружить все существенное поведение компонента.

Описание. Компонент программного обеспечения должен обладать минимальной структурной сложностью. Сложные ветвления должны быть исключены. По возможности, ограничения цикла и ветвление должны быть просто связаны с входными параметрами. Компонент программного обеспечения должен быть разделен на подходящие, небольшие модули, взаимодействие между которыми должно быть точно специфицировано. Следует использовать свойства языков программирования, которые способствуют данному методу, предпочитая их другим свойствам, которые (как утверждают) более эффективны, за исключением случаев, когда эффективность приобретает абсолютный приоритет (например некоторые критичные к безопасности системы).

D.54 Выбор подходящего языка программирования

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 компонент;
- ограничение доступа к данным в определенных компонентах;
- определение поддиапазонов переменных;
- любые другие типы конструкции, ограничивающие ошибки.

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.

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.

D.55 Моделирование во времени сетями Петри

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

Описание. 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.

В модели сети Петри потенциальные опасности могут быть представлены в виде конкретных состояний (маркировок). 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.

D.56 Сквозной контроль/анализ проектов

Цель. Обнаружить ошибки в процессе разработки некоторого изделия с высокой оперативностью и экономичностью

Описание. В МЭК 61160 представлено руководство по формальному анализу проектов, которое содержит общее описание формального анализа проектов, его цели, подробные сведения о различных типах анализа проекта, состав группы анализа проекта, и относящиеся к ним обязанности и ответственности. МЭК 61160 содержит также общие руководящие материалы по планированию и выполнению формального анализа проектов, а также конкретные подробные сведения, относящиеся к роли независимых специалистов в группе по анализу проекта.

МЭК 61160 рекомендует, чтобы «формальный анализ проекта проводился для всех новых изделий/процессов, новых применений и при пересмотрах существующих изделий и производственных процессов, влияющих на функции, производительность, безопасность, надежность, способность анализировать обслуживание, доступность, способность к экономичности и другие характеристики, влияющие на конечные изделия/процессы, пользователей или стоящих рядом лиц».

Для выполнения сквозного контроля кода необходимы группы сквозного контроля, выбирающая небольшой набор изложенных на бумаге тестовых примеров, представительные наборы входных данных и соответствующих предполагаемые выходы для программы. После этого тестовые данные вручную пропускаются через логику программы.

D.57 Объектно-ориентированное программирование

Цель. Обеспечить быстрое прототипирование, более легко повторно использовать существующие компоненты программного обеспечения, достигнуть сокрытия информации, уменьшить вероятность ошибок в течение всего жизненного цикла, снизить необходимые усилия во время стадии сопровождения, преобразовать сложные проблемы в небольшие, достаточно легко управляемые проблемы, уменьшить зависимости между компонентами программного обеспечения, создать более легко расширяемые приложения.

Описание. Объектно-ориентированное программирование — это существенно новый подход в программировании, основанный на абстракциях, которые существуют в реальном мире, а не на вычислительных абстракциях. Объектно-ориентированное программирование позволяет организовать программное обеспечение как набор объектов, которые включают и структуру данных и поведение. Объектно-ориентированный подход отличается от стандартного программирования, где структура данных и поведение только соединены не жестко.

Объект: Объект состоит из закрытой области данных и набора операций — так называемых методов — для этого объекта. Методы могут быть общедоступными или частными. Никакому другому компоненту программного обеспечения не позволено считать или изменить частные данные объекта непосредственно. Любой компонент программного обеспечения должен использовать общедоступные методы для этого объекта, чтобы считать или записать данные в частной области данных объекта.

Класс объекта. Определяя класс объекта (часто в форме определения типа) Вы обеспечиваете инстанциирование многочисленных объектов того же класса, т. е. у всех инстанциаций частная область данных и методы, определенные в этом классе объекта.

Наследование (множественное). Класс объекта может наследовать частную область данных и методы одного или более суперклассов (классов объектов выше его в иерархии классов), а также может добавлять некоторые частные данные, добавлять некоторые методы или изменять реализации наследованных методов. Используя наследование, могут быть созданы деревья множественных экземпляров классов объектов.

Полиморфизм. Одна и та же операция может выполняться по-другому на различных классах объектов, например, операция записи для терминального объекта записывает символы в этот терминал, и операция записи в файловый объект записывает символы в этот файл.

Недостаток. Объектно-ориентированные языки программирования могут потребовать дополнительные ресурсы, что может негативно повлиять на производительность системы.

D.58 Прослеживаемость

Цель. Цель прослеживаемости состоит в обеспечении гарантии, что для всех требований можно показать, что они должным образом удовлетворены и что какой-либо материал о непрослеживаемости отсутствует.

Описание. Прослеживаемость требований должна быть важным соображением при выполнении подтверждения соответствия системы, и должны быть обеспечены средства, позволяющие продемонстрировать это по всем стадиям жизненного цикла.

Прослеживаемость должна выполняться как для функциональных, так и для нефункциональных требований и должна, в частности, осуществлять:

a) прослеживаемость требований к проекту или другим объектам, которые их выполняют;

b) прослеживаемость объектов проектирования к объектам реализации, которые их конкретизируют;

c) прослеживаемость требований и объектов проектирования к объектам эксплуатации и сопровождения, которые требуется применить в безопасном и надлежащем использовании системы;

d) прослеживаемость требований проекта, реализации, эксплуатации и объектов сопровождения к планам проверки и тестирования, а также к спецификациям, которые определят их приемлемость,

e) прослеживаемость планов проверки и тестирования, а также спецификаций к отчетам о проверке или другим отчетам, которые записывают результаты их применения.

Если требования, проект или другие объекты представлены как набор отдельных документов, то прослеживаемость должна быть выполнена в структурах документов иерархически.

Результат процесса прослеживаемости должен быть предметом формального управления конфигурацией.

D.59 Метапрограммирование

Цель. Метапрограммирование позволяет программистам выполнить больший объем работы за тоже количество времени, которое они потратили бы на подготовку всего кода вручную.

Описание. Метапрограммирование — это запись компьютерных программ, которые пишут или обрабатывают другие программы (или делают это сами), как их данные, или которые выполняют часть работы в течение времени компиляции, которая иначе выполняется во время выполнения.

Язык, на котором пишется метапрограмма, называют метаязыком. Язык программ, которыми управляют, называют объектным языком. Возможность языка программирования быть его собственным метаязыком вызывают рефлексией или рефлексивностью.

Рефлексия — это ценная функция языка, упрощающая метапрограммирование. Иметь сам язык программирования, как первоклассный тип данных (как в Lisp), также очень полезно. Абстрактное программирование вызывает средства метапрограммирования внутри языка в тех языках, которые его поддерживают.

Метапрограммирование обычно работает одним из двух способов. Первый способ состоит в том, чтобы предоставить внутренние механизмы исполнения кода программы через прикладные программные интерфейсы (API). Второй способ — динамическое выполнение строковых выражений, которые содержат команды программирования. Таким образом, «программы могут написать программы». Несмотря на то, что оба подхода могут использоваться, большинство языков имеет тенденцию склоняться к одному или другому.

D.60 Процедурное программирование

Цель. При определении шагов программа должна взять, чтобы достигнуть требуемого состояния.

Описание. Процедурное программирование основано на понятии вызова процедуры. Процедуры, также известные как программы, подпрограммы, методы или функции (не путать с математическими функциями, а также с используемыми в функциональном программировании) просто содержат последовательности вычислительных шагов, которые будут выполняться. Любую данную процедуру можно вызвать в любой точке в процессе выполнения программы, включая другие процедуры или ее саму.

D.61 Последовательные функциональные схемы (SFC)

Цель. Описание алгоритмов программы с помощью диаграмм.

Описание. Элементы SFC позволяют разделить совокупность прикладных алгоритмов на ряд шагов и переходов, соединенных направленными связями. С каждым шагом связан набор действий, а с каждым переходом связано условие перехода. Так как элементы SFC обеспечивают хранение информации о состоянии, то только совокупности прикладных алгоритмов, которые могут быть структурированы, используя эти элементы, являются функциональными блоками.

См. МЭК 61131-3:2013, 6.7

D.62 Релейно-контактные схемы (LD)

Цель. Описание программ с помощью диаграмм.

Описание. См. МЭК 61131-3:2013, 8.2

D.63 Диаграммы функциональных блоков (FBD)

Цель. Описание функции между входными переменными и выходными переменными с помощью диаграмм.
Описание. См. МЭК 61131-3:2013, 8.3

D.64 Диаграмма состояний / диаграммы переходов состояний

Цель. Описание поведения системы с помощью диаграмм.

Описание. Диаграммы состояний или диаграммы переходов состояний используются, чтобы описать поведение системы. Диаграммы состояний могут описать возможные состояния объекта, поскольку имеют место события. Каждая диаграмма обычно представляет объекты одного класса и отслеживает различные состояния его объектов в системе.

Диаграмма состояний может использоваться для графического представления конечных автоматов. Это было предложено Тейлором Бутом в его книге 1967 года «Последовательные машины и теория автоматов». Другое возможное представление — таблица переходов состояний.

Классическая форма диаграммы состояний для конечного автомата — направленный граф.

D.65 Моделирование данных

Цель. Создание модели данных.

Описание. Моделирование данных в информатике — процесс создания модели данных, применяя формальные описания модели данных, используя методы моделирования данных.

Модель данных при разработке программного обеспечения — абстрактная модель, которая описывает, как данные представлены и как к ним может быть получен доступ. Модели данных формально определяют объекты данных и отношения между объектами данных для интересуемой предметной области. Некоторые типичные применения моделей базы данных включают поддержку разработки баз данных и обмена данными для определенной сферы интересов. Модели данных определены на языке моделирования данных.

D.66 Диаграммы потоков управления/граф потока управления

Цель. Описание поведения системы с помощью диаграмм.

Описание. В информатике, диаграмма потока управления или граф потока управления (CFG) является использующим нотацию графа представлением всех путей, которые могли бы быть реализованы в программе во время ее выполнения. Каждый узел графа представляет базовый блок, т. е. линейную часть кода без любых переходов или меток перехода; меток перехода в начале блока, и переходов в конце блока. Для представления переходов в потоке управления используются ориентированные ребра. В большинстве представлений существует два специально определяемых блоках: входной блок, через который управление входит в блок-схему, и блок выхода, через который весь поток управления уходит.

CFG важен для многих оптимизаций компилятора и инструментальных средств статического анализа.

Достижимость — другое свойство графа, полезное при оптимизации. Если вход блока/подграфа не соединен с подграфом, содержащим входной блок, то этот блок недостижим во время любого выполнения программы, и поэтому его код недостижим; этот блок можно безопасно удалить. Если выходной блок недостижим из входного блока, то это указывает на бесконечный цикл. Кроме того, недостижимый код и некоторые бесконечные циклы возможны, даже если программист явно не кодировал такой путь: оптимизации подобные подстановке значений констант и сворачиванию констант, за которыми следует переход к многопоточной обработке, могут свернуть множество основных блоков в один, что ведет к удалению ребер из CFG, и т. д., и, таким образом, возможна потеря части графа.

Терминология — эти термины обычно используются при обсуждении графа потока управления;

входной блок — блок, от которого начинается любой путь;

выходной блок — блок, на котором завершаются все пути;

обратное ребро — ребро, которое указывает на предка при обходе графа в глубину (DFS);

критическое ребро — ребро, которое не является ни единственным ребром, выходящим из исходного блока графа, ни единственным ребром, входящим в целевой блок графа. Эти ребра должны быть разделены (должен быть создан новый блок в середине ребра), чтобы вставить вычисления на ребре;

аномальное ребро — ребро, назначение которого неизвестно. Эти ребра препятствуют оптимизации. Они могут появляться, например, после преобразования конструкций обработки исключений;

невозможное ребро — (также известное, как ложное ребро) ребро, которое было добавлено к графу исключительно для того, чтобы сохранить свойство графа о постдоминировании выходным блоком любого другого узла. Это ребро никогда не может быть пройдено;

доминатор (обязательный предшественник) — узел *M* доминирует над блоком *N*, если любой путь, достигающий *N*, обязан предварительно пройти блок *M*. Входной узел доминирует над всеми остальными блоками графа;

постдоминатор — узел *M* постдоминирует над блоком *N*, если любой путь от *N* к выходному блоку обязан пройти через узел *M*. Выходной блок постдоминирует все блоки графа.

непосредственный доминатор — блок *M* непосредственно доминирует над блоком *N*, если *M* доминирует над *N*, и нет никакого промежуточного блока *P* такого, что *M* доминирует над *P* и *P* доминирует над *N*. Другими словами, *M* — последний доминатор на любом пути от входа до *N*. У каждого блока есть уникальный непосредственный доминатор, если у него есть кто-либо вообще;

непосредственный постдоминатор — аналогичный непосредственному доминатору;

дерево доминаторов — структура вспомогательных данных, изображающая отношения доминаторов. Существует ребро от блока *M* к блоку *N*, если *M* — непосредственный доминатор *N*. Этот граф является деревом, так как у каждого блока есть уникальный непосредственный доминатор. Корнем этого дерева является входной блок;

дерево постдоминаторов — аналог дерева доминаторов. Корнем этого дерева является выходной блок;

заголовок цикла (иногда называют точкой входа цикла) — доминатор, который является назначением обратных ребер, образующих цикл. Доминирует над всеми блоками тела цикла;

предзаголовок цикла — предположим, что блок *M* является доминатором с несколькими входными ребрами, некоторые из которых являются обратными (таким образом, *M* — заголовок цикла). Для некоторых стадий оптимизации выгодно, чтобы блок *M* был разделен на два блока, M_{pre} и M_{loop}. Содержимое *M* и обратные ребра относятся к M_{loop}, остальные ребра относятся к M_{pre}, и создается новое ребро от M_{pre} к M_{loop} (таким образом, M_{pre} стал непосредственным доминатором M_{loop}). Сразу после преобразования M_{pre} будет пустым, но прохождения подобные изменению кода постоянной цикла могут пополнить его. M_{pre} называется предзаголовком цикла, а M_{loop} становится заголовком цикла.

D.67 Диаграммы последовательностей

Цель. Описание взаимодействия между процессами или компонентами с помощью диаграмм.

Описание. Диаграммы последовательностей — это вид диаграмм взаимодействия, которые показывают, как процессы или компоненты взаимодействуют друг с другом и в каком порядке.

D.68 Табличный метод спецификации

Цель. Обеспечить стандартизированное и хорошо структурированное средство определения управляемых данными функций системы.

Описание. Табличные нотации, такие как у таблиц управления сигнализацией, являются хорошо установленным методом документального оформления установления определенных требований для железнодорожной системы сигнализации.

Данный метод подходит там, где типы отношений между элементами системы стандартизированы.

Преимущество его состоит в том, что форматы таблицы и возможных записей в каждом поле могут служить в качестве контрольного списка в процессе проверки.

D.69 Специализированный язык

Цель. Обеспечить средство определения функциональности управляемой данными системы, используя понятия и терминологию, которые легко воспринимаются инженерами приложений, которые могут быть не знакомы со стандартными языками программирования.

Описание. Специализированный язык обычно объединяет конструкции управления, которые подобны стандартным высокоуровневым языкам программирования, с операторами, которые определены для рассматриваемого типа системы.

Данный метод подходит там, где необходимо использовать булевы выражения, но может также быть применен и в других случаях.

Преимуществом является гибкость, позволяющая формировать данные для необычных обстоятельств, которые не могли быть предсказаны, когда система была первоначально разработана.

D.70 Универсальный язык моделирования (UML)

Цель. Представлять программы и связанные с ней артефакты, обеспечивая сокращение сложности с помощью абстракции. Выполнение в UML моделирования существующего или запланированного проекта с использованием множества типов схем упрощает оценку основных характеристик проекта с помощью представлений на необходимых уровнях детализации. UML часто используется в так называемой разработке, управляемой моделью, поддерживаемой коммерческими продуктами. Такой стиль разработки улучшает качество программного обеспечения и повышает производительность разработчиков при помощи высокоуровневых языков моделирования.

Описание. UML — унифицированный и стандартизированный язык моделирования, использующий преимущества графически ориентированных языков спецификаций программного обеспечения и языков объектно-ориентированного программирования. Основываясь на этих традициях, UML использует многие понятия и методы его предшественников. Модели представляются в виде одного или нескольких типов диаграмм, которые можно разделить на структурные и поведенческие. Последние также включают четыре типа диаграмм, которые называются диаграммами взаимодействия.

Структурные диаграммы:

- диаграммы пакетов: показывают содержание и отношения между различными пакетами, каждый из которых содержит связанные с моделью элементы;
- диаграммы классов: определяют типы объектов с их различными свойствами и их отношениями с другими типами объектов, на основе адаптации традиционных диаграмм сущность-связь;
- диаграммы объектов: показывают, как различные объекты (экземпляры класса) связаны друг с другом;

- диаграммы составной структуры: показывают внутреннюю структуру классификатора (класса или компонента) и его точки взаимодействия с другими частями системы;
- диаграммы компонентов: показывают компоненты, которые составляют систему, их взаимосвязи, взаимодействия и внешние интерфейсы;
- диаграммы развертывания: определяют, как программное обеспечение распределено по платформе выполнения.

Поведенческие диаграммы:

- диаграммы деятельности: описывают алгоритмическое поведение, используя адаптацию традиционных блок-схем, которые позволяют моделировать передачи данных и параллельное выполнение;
- диаграммы состояний: описывают управляемое событиями поведение с помощью языка конечных автоматов;
- диаграммы вариантов использования: моделируют взаимодействия проектируемой системы с любыми внешними или внутренними объектами (пользователями, другими системами и т. п.) для достижения конкретных вариантов использования;
- диаграммы взаимодействия (диаграммы коммуникации, диаграммы обзора взаимодействия, диаграммы последовательности, диаграммы синхронизации): описывают сценарии, включающие действия, выполняемые, взаимодействующими объектами.

Хотя UML является универсальным языком моделирования, проблемно-ориентированные интерпретации могут быть реализованы с помощью профилей. Уточняя стандартные понятия UML, профили дают возможность выполнить такие интерпретации при помощи расширений определенных в профиле. Таким образом, UML используется в качестве основы для определения проблемно-ориентированных языков.

D.71 Предметно-ориентированный язык

Цель. Представлять программы и связанные с ней артефакты на языке, адаптированном к конкретной предметной области.

Описание. Предметно-ориентированный язык (DSL), являющийся языком программирования, спецификаций или моделирования, создан специально для решения задач конкретной прикладной предметной области или проблемы предметной области, или используя определенный метод. Язык основывается на понятиях и основных свойствах этой предметной области. Предметно-ориентированные языки также известны, как языки специального назначения, в отличие от универсальных языков программирования или языков моделирования таких, как Java и UML.

Одним из важных преимуществ предметно-ориентированных языков является возможность представлять и решать проблемы в определенной предметной области без привлечения знаний об универсальных средствах программирования, спецификации или моделирования. В результате программы, спецификации или модели могут быть выполнены на более высоком уровне, возможно, конечным пользователем. Обеспечивая конструкции, адаптированные к этой предметной области, и, возможно, средства для автоматизированной генерации кода, DSL в большинстве случаев также повышает производительность программиста и качество получающегося продукта. Генерация кода обычно реализуется как генератор прикладных программ, использующий DSL в качестве входного языка.

Приложение ДА
(справочное)

Сведения о соответствии ссылочных международных стандартов национальным стандартам

Таблица ДА.1

Обозначение ссылочного международного стандарта	Степень соответствия	Обозначение и наименование соответствующего национального стандарта
IEC 62278:2002	IDT	ГОСТ Р МЭК 62278—2008 «Определение и подтверждение надежности, эксплуатационной готовности, ремонтопригодности и безопасности (RAMS) на железных дорогах»
ISO/IEC 90003:2014	IDT	ГОСТ Р ИСО/МЭК 90003—2014 «Разработка программных продуктов. Руководящие указания по применению ИСО 9001:2008 при разработке программных продуктов»
ISO/IEC 25010	IDT	ГОСТ Р ИСО/МЭК 25010—2015 «Информационные технологии. Системная и программная инженерия. Требования и оценка качества систем и программного обеспечения (SQuaRE). Модели качества систем и программных продуктов»
ISO 9000	IDT	ГОСТ Р ИСО 9001—2008 «Системы менеджмента качества. Основные положения и словарь»
ISO 9001:2008	IDT	ГОСТ Р ИСО 9000—2008 «Системы менеджмента качества. Требования»

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

- IDT — идентичный стандарт.

Библиография

[1] IEC 61131-3:2013, Programmable controllers — Part 3: Programming languages

[2] IEC 61158-2:2014, Industrial communication networks — Fieldbus specifications — Part 2: Physical layer specification and service definition

[3] IEC 62280, Railway applications — Communication, signalling and processing systems — Safety-related communication in transmission systems

[4] IEC 62425:2007, Railway applications — Communication, signalling and processing systems — Safety related electronic systems for signalling

УДК 62-783:614.8:331.454:006.354 ОКС 13.110 Т51

Ключевые слова: машины землеройные, риск, защитные устройства, управляющее устройство, элементы систем управления, принципы конструирования, полнота безопасности

Редактор *Д.Е. Титов*
Технический редактор *В.Ю. Фотиева*
Корректор *С.В. Смирнова*
Компьютерная верстка *Е.А. Кондрашовой*

Сдано в набор 19.12.2016. Подписано в печать 13.01.2017. Формат 60×84⅛. Гарнитура Ариал.
Усл. печ. л. 11,63. Уч.-изд. л. 10,46. Тираж 29 экз. Зак. 50.

Подготовлено на основе электронной версии, предоставленной разработчиком стандарта

Издано и отпечатано во ФГУП «СТАНДАРТИНФОРМ», 123995 Москва, Гранатный пер., 4.
www.gostinfo.ru info@gostinfo.ru